MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Machine Learning und Künstliche Intelligenz im Mathematikunterricht der Sekundarstufe“

verfasst von / submitted by

Eva-Theres Mölk, BEd

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Education (MEd)

Wien, 2023 / Vienna 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 199 510 520
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Lehramt Sek (AB) Unterrichtsfach Geographie und Wirtschaftskunde Unterrichtsfach Mathematik
Betreut von / Supervisor:	Dr. Franz Embacher

Danksagung

Ich möchte mich bei meinen Eltern bedanken, die mich während meiner ganzen Studienzeit immer unterstützt haben, wo sie nur konnten. Danke für eure Geduld, Hilfe und Bestärkung. Ihr habt mein Studium erst möglich gemacht.

Danke meiner eigenen kleinen Familie, meinem Ehemann und meinen beiden kleinen Söhnen. Ihr seid wunderbar und gebt mir jeden Tag einen Grund, mich anzustrengen. Meinem Mann gilt ein besonderer Dank für seine Geduld und die vielen Gespräche. Du bist meine Stütze und mein Fangnetz, mein Seelenmensch und bester Freund.

Zum Abschluss möchte ich mich bei meinem Betreuer Dr. Franz Embacher besonders bedanken. Danke für Ihre Geduld, Ihre fachliche Unterstützung sowie Ihr immer hilfreiches Feedback. Ich habe ihre Betreuung sehr wertgeschätzt!

Wien, Jänner 2023

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Themen wie Machine Learning (ML) und Künstliche Intelligenz (KI) beeinflussen indirekt sowie direkt immer mehr unser tägliches Leben. Sie sind aus unserem Alltag und aus der Lebenswelt der Schüler*innen nicht mehr wegzudenken. Das Mobiltelefon mit Gesichtserkennungstechnologie, die Personalisierung in Sozialen Medien, der Spam-Filter des E-Mail-Postfaches oder der digitale Sprachassistent sind Beispiele dafür, wo wir der Künstlichen Intelligenz täglich begegnen. Überdies wird kontrovers über KI diskutiert, von manchen wird sie als Bedrohung für die Menschheit gesehen und sie steht im Zentrum einer großen Zahl von Science Fiction-Filmen. Viele Alltagstechnologien arbeiten heute mit Künstlicher Intelligenz, hinter der wiederum pure Mathematik steckt.

Auf viele Schüler*innen wirkt besonders die höhere Mathematik, die in der Sekundarstufe II unterrichtet wird, zu abstrakt und zu weit weg von ihrem Leben. Sie fragen sich, warum sie Kurvendiskussionen, Integral- und Wahrscheinlichkeitsrechnung lernen sollen und finden diese Themen wenig nützlich. Mathematiklehrer*innen streben danach, ihre Schüler*innen gut durch die Matura zu bringen, Interesse für ihr Fach zu wecken, ja vielleicht sogar, die Kinder und Jugendlichen dafür zu begeistern. Sie wollen den Lernenden zeigen, dass die Mathematik wichtig ist, dass sie zwar meist gut versteckt ist, aber sehr wohl gebraucht wird.

Daher erscheint es sinnvoll, die Themen Machine Learning und Künstliche Intelligenz in die Schulen zu holen. Weil es einige Anknüpfungspunkte zu den Mathematiklehrplänen der Sekundarstufe gibt, könnte dieser Unterrichtsgegenstand dafür prädestiniert sein, Grundlagen zu ML und KI zu vermitteln. Einerseits ergibt sich damit die Möglichkeit, anhand von anwendungsorientierten und praktischen Problemstellungen, den Einsatz der höheren Mathematik sehr anschaulich zu demonstrieren, was das Interesse und die Akzeptanz dieses Gegenstandes fördern

kann. Darüber hinaus erscheinen ML und KI geeignet, die Anwendung der Mathematik an modernsten Themen zu zeigen. Obgleich hier ein Potential zu liegen scheint, werden Künstliche Intelligenz und Machine Learning von den gängigen Schulbüchern bisher noch nicht oder sehr wenig aufgegriffen.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist die Beantwortung der Frage, inwieweit Inhalte aus dem Gebiet des Machine Learning und der Künstlichen Intelligenz im Rahmen des aktuell gültigen Mathematiklehrplans Anwendung im Unterricht der Sekundarstufe finden können. Dabei werden ausschließlich die Schultypen AHS, HAK und HTL näher betrachtet. Darüber hinaus sollen exemplarisch konkrete Aufgaben entwickelt werden, mit deren Hilfe ein Vorschlag zur praktischen Umsetzung gemacht wird.

1.3 Aufbau der Arbeit

Diese Arbeit gliedert sich in sechs Kapitel, wobei das erste Kapitel eine Motivation sowie die Formulierung der Fragestellung umfasst. Im zweiten Kapitel findet sich neben einem Einstieg in die Thematik ein geschichtlicher Teil sowie die grundlegende Arbeitsweise von ML und KI. Das dritte Kapitel ist der Datenvorverarbeitung gewidmet. Hier werden zwei im Verlauf der Arbeit immer wieder verwendete Datasets (Titanic Disaster Dataset und Amerikanischer Gebrauchtwagenmarkt) zum ersten Mal vorgestellt. Darüber hinaus gibt es Teile zur Dateinvisualisierung und zum Umgang mit nicht aufbereiteten Daten. Im vierten Kapitel werden spezielle ML-Algorithmen vorgestellt sowie Einsatzmöglichkeiten derselben im Schulunterricht. Das fünfte Kapitel schließlich befasst sich mit neuronalen Netzen und der Funktionsapproximation durch ein neuronales Netz auf Schulniveau. Im sechsten Kapitel findet sich ein Fazit zur Arbeit und die Beantwortung der Fragestellung.

1.4 Verwendung der Inhalte im Unterricht

Der überwiegende Teil der Arbeit beschäftigt sich mathematischen Inhalten, die an geeigneter Stelle im Mathematikunterricht der Sekundarstufe verwendet werden können. Zur Abrundung der Arbeit wurden auch konkrete Pythonskripts aufgenommen. Diese sind für das Verständnis der mathematischen Inhalte nicht notwendig und wurden daher in den Anhang ausgelagert. Es ist also nicht erforderlich, die Python-Quellcodes im Anhang B zu verstehen und/oder durchzuarbeiten. Sollten die Inhalte aber fächerübergreifend mit Informatik Verwendung finden, so

bieten diese Skripts eine Hilfestellung für die Umsetzung im Unterricht. Speziell für berufsbildende Schulen mit Informatikschwerpunkt sollten diese Skript eine gute Grundlage sein.
Für die Umsetzung im Mathematikunterricht ist Geogreba ein ansprechendes Tool, die in dieser Arbeit verwendeten Anwendungen mit Geogebra sind im Wesentlichen in Anhang C zusammengestellt.
Die Arbeit enthält auch Vorschläge für konkrete anwendungsorientierte Beispiele, mit deren Hilfe bei verschiedenen Lehrinhalten die Themen dieser Arbeit angesprochen werden können. Zur Motivation und Orientierung der interessierten Leser*innen wird hier eine Übersicht gegeben, welche Lehrinhalte (nicht aufgegliedert nach AHS, HTL oder HAK) den einzelnen Abschnitten dieser Arbeit zugeordnet werden können.

1. Unterrichtsidee für den Einstieg in das Thema (vgl. Abschnitt 2.2)
   Diskussion zur künstlichen Intelligenz.

2. Overfitting und Underfitting (vgl. Abschnitt 2.3.6)
   Polynomfunktionen unterschiedlicher Ordnung als Trendlinien.

3. Interpretation von Historgammen und Boxplots (vgl. Abschnitt 3.3)
   Beschreibende Statistik - Histogramme, Boxplots.

4. Behandlung nicht numerischer Daten (vgl. Abschnitt 3.5)
   Beschreibende Statistik - Nominalskalen, Ordinalskalen.
   Beispiel zur linearen Abhängigkeit - One Hot Encoding.

5. Featurescaling (vgl. Abschnitt 3.7)
   Verschiebung, Skalierung von Achsen, Mittelwert, Standardabweichung.

6. Einfache Lineare Regression (vgl. Abschnitt 4.1)
   Lineare Regression, Geradengleichung.
   Extremwertaufgabe (ohne und mit Nebenbedingung).

7. Mehrdimensionale Lineare Regression (vgl. Abschnitt 4.1.3)
   Interpretation von Korrelationskoeffizienten.

8. Logistische Regression (vgl. Abschnitt 4.2)
   Transzendente Funktionen (logistisches Wachstum), Interpretation von Wahrscheinlichkeiten.

9. KNN - K nearest neighbors (vgl. Abschnitt 4.3)
   Euklidische Abstände (Betrag von Vektoren).

10. Naive Bayes (vgl. Abschnitt 4.4)

    Bedingte Wahrscheinlichkeiten, Satz von Bayes.

11. Das Neuron (vgl. Abschnitt 5.2)

    Skalares Produkt, funktionale Zusammenhänge.

12. Aktivierungfunktionen (vgl. Abschnitt 5.3)

    Stückweise definierte (lineare) Funktionen, Logistisches Wachstum.

13. Perzeptron (vgl. Abschnitt 5.5)

    Normalvektorform einer Geraden (Orientierung des Normalvektors), iterative Prozesse.

14. Arbeitsweise neuronaler Netze(vgl. Abschnitt 5.6)

    Funktionale Zusammenhänge (Costfunction).

    Ableitungen, iterative Prozesse (Gradientenabstiegsverfahren).

    Funktionen in mehreren Veränderlichen, Gradient (Gradientenabstiegsverfahren).

    Verkettete Funktionen, Kettenregel, partielle Ableitungen (Back Propagation).

15. Funktionsapproximation durch ein neuronales Netz (vgl. Abschnitt 5.7)

    Approximation einer Funktion durch stückweise definierte (lineare) Funktionen.

    Elementare Funktionstransformationen (Translation, Skalierung).

    Systematische Behandlung von Fallunterscheidungen.

Kapitel 2

Grundlagen

2.1 Einstieg in die Thematik

Künstliche Intelligenz, kurz KI oder AI vom englischen "Artificial Intelligence" ist ein Teilgebiet der Informatik und heute in immer mehr Bereichen unseres Lebens anzutreffen. Da erst seit etwa fünf Jahrzehnten Forschung dazu betrieben wird, kann sie als junge wissenschaftliche Disziplin angesehen werden. (vgl. (*1*), S. 11) Viele denken beim Begriff KI vielleicht an Roboter oder den Androiden aus dem letzten Science Fiction-Film. Was aber die Wissenschaft unter KI versteht und ab wann eine Maschine als "intelligent" angesehen wird, soll in diesem Abschnitt erläutert werden. John McCarthy war einer der Pioniere der KI. Er definierte den Begriff *Künstliche Intelligenz* im Jahr 1955 etwa so: "Ziel der KI ist es, Maschinen zu etwickeln, die sich verhalten, als verfügten sie über Intelligenz." (vgl. (*2*), S. 1) Diese Definition greift nach heutigem Verständnis allerdings zu kurz, denn ein Verhalten, das innerhalb bestimmter Grenzen intelligent wirkt, kann durch eine*n Programmierer*in mithilfe verschiedener Befehle in eine Maschine schlicht "hineinprogrammiert" sein (*Codierte Intelligenz*). So kann Intelligenz nur simuliert werden. (vgl. (*3*), S. 33) Als Beispiel stelle man sich fünfzehn kleine Roboterfahrzeuge vor, bei denen man verschiedene Verhaltensweisen beobachten kann. Manche formen kleine Grüppchen mit relativ wenig Bewegung, andere bewegen sich ruhig durch den Raum und weichen jeder Kollision aus. Andere folgen scheinbar einem Anführer und bei manchen kann man aggressives Verhalten beobachten. Nach McCarthy könnten wir diese Roboter als intelligent bezeichnen. Dieses scheinbar komplexe Verhalten lässt sich aber recht einfach mit elektrischen Verschaltungen erzeugen. Dabei reguliert ein Lichtsensor an der Vorderseite des Roboters seine Geschwindigkeit und je nach Programmierung bewegt sich ein Roboter von der Lichtquelle weg oder darauf zu. Andere Verhaltensweisen können mit weiteren kleinen Modifikationen erreicht werden. (vgl. (*2*), S. 1-2)

Künstliche Intelligenz ist der Versuch, menschliche Intelligenz auf Maschinen zu simulieren, um sie für den Menschen gewinn- und nutzbringend einzusetzen, aber die Simulation alleine reicht nicht aus. Unter Intelligenz versteht man die Fähigkeit, sich an neue Begebenheiten, Bedingungen und Aufgaben anzupassen. Aus Wahrnehmungen werden Schlüsse gezogen, woraufhin eine Reaktion folgt. Sollte die Reaktion unangemessen sein, wird daraus gelernt, um nächstes Mal eine passendere Reaktion zu erzielen. Intelligenz setzt sich zusammen aus Erkenntnisvermögen, Urteilsvermögen, Erfassen von Möglichkeiten, Begreifen von Zusammenhängen und Gewinnen von Einsichten. Damit ein Modell also zum Gebiet der Künstlichen Intelligenz zählen kann, muss es **lernfähig** sein. (vgl. (*3*), S. 29-32) Die Idee der Erschaffung einer Künstlichen Intelligenz trifft in der Bevölkerung auf gespaltene Gemüter. Dass Menschen etwas Menschenähnliches erschaffen wollen, stößt auf Ablehnung und auf Faszination gleichermaßen. Dabei wird angenommen, es gäbe zwischen Computer und Mensch keine kategoriale Differenz, sodass Softwaresysteme, die menschliches Verhalten, Urteilen und Entscheiden nachahmen, auch menschliche Eigenschaften aufweisen. In der heutigen Realität ist KI aber etwas viel Nüchterneres, da dabei lediglich Computerprogramme für Problembereiche entwickelt werden, die bislang nur von Menschen lösbar waren, indem Maschinen menschliche Wahrnehmungs- und Verstandesleistungen übernehmen. (vgl. (*4*), S. 2)

In Fachkreisen wird zwischen der schwachen und der starken KI unterschieden. Schwache KI (Artificial Narrow Intelligence) stellt den derzeitigen Stand der Technik dar. Hierzu zählen Maschinen, die aus Daten selbstständig lernen können und mithilfe gegebener Axiome durch Anwendung von Logik neue Daten und Modelle erzeugen können. Von dieser KI sind wir heute bereits allseits umgeben. Starke KI (Artificial General Intelligence) soll die menschliche Intelligenzstufe erreichen und in Zukunft Aufgaben erfüllen, die normalerweise nur von einem Menschen erbracht werden. Hierzu zählen Computer, die nicht nur intelligent sind, sondern einen eigenen Willen und ein eigenes Selbstbewusstsein herausbilden können. Artificial Superintelligence bezeichnet schließlich eine Form der KI, die intelligenter ist als der Mensch. Starke KI gibt es heute und in naher Zukunft allerdings nicht. Es gibt nicht einmal eine Theorie dazu, wie eine Starke KI erzeugt werden könnte. (vgl. (*3*), S. 34)

Heute unterstützen selbstlernende Programme die medizinische Diagnostik. In der Automobil- und Luftfahrtindustrie helfen sie beim Aufspüren von Materialproblemen und in der Finanzindustrie bei Investitionsentscheidungen und der Kreditvergabe. KI wird gebraucht beim Onlinemarketing, für automatisch fahrende Autos und Optimierungen in der Logistik. (vgl. (*4*), S. 1) Onlinepräsenzen von Unternehmen besitzen manchmal bereits lernfähige Chatbots, die bei Kundenfragen behilflich sind. Auch in der Psychologie, beispielsweise beim Aufspüren von De-

pressionen, oder im E-Learningbereich, werden Chatbots unterstützend verwendet. Hierbei ist es wichtig zu erwähnen, dass bei weitem nicht alle Chatbots, auf die man im Internet trifft, zur KI zählen. Künstliche Intelligenz entwickelt sich heutzutage mit solch einer rasanten Geschwindigkeit weiter, dass der momentane Forschungsstand gar nicht adäquat dargestellt werden kann. Zweifellos wird KI in Zukunft zu einem Wandel vieler Bereiche unserer Welt führen. So könnte Künstliche Intelligenz beispielsweise die Art, wie wir leben und spielen, eine Partnerin oder einen Partner suchen, wie wir unsere Kinder erziehen und alte Menschen pflegen, verändern. Bestimmte Betätigungsfelder am Arbeitsmarkt könnten durch sie überflüssig werden. (vgl. (*1*), S. 13) Eine KI ohne begleitende ethische Überlegungen einzuführen, wäre daher fahrlässig. Diese Technologie könnte genauso wie die Dampfmaschine, die Elektrizität und die Informationstechnologie eine grundlegende technische Revolution auslösen. Daher ist es notwendig und wichtig, sich mit ihrer Funktionsweise auseinanderzusetzen und zu erkennen, wozu sie in der Lage ist und wozu nicht. (vgl. (*3*), S. 35) Ein Ort, der bei der Vermittlung dieses Wissens eine tragende Rolle spielen kann, ist die Schule.

2.2 Geschichte und Meilensteine

Maschinen gibt es schon sehr lange. Bereits vor 2000 Jahren verfasste Heron von Alexandria das Buch der Maschinen (*Automata*), worin er unter anderem Tempeltüren beschrieb, die sich selbst öffnen konnten. Leonardo da Vinci (1452-1519) hat "Roboter" konstruiert, die ihre Arme und den Kopf bewegen und sich aufsetzen konnten. Ab dem 18. Jahrhundert wurde das Interesse an Maschinen immer größer. Es entstanden automatische Musikinstrumente und Webstühle, die zwar voll autonom waren, aber natürlich nicht intelligent, da sie nicht lernen konnten. Trotzdem ebneten diese Erfindungen den Weg zur Automatisierung, der heute in die sogenannte 4. industrielle Revolution mündet. (vgl. (*3*), S. 37)

Die KI als eigene Wissenschaft gibt es etwa seit Mitte des 20. Jahrhunderts, wobei sich mehrere Hauptströmungen herausbildeten. Wichtige Fundamente für die Logik und die theoretische Informatik wurden in den dreißiger Jahren des zwanzigsten Jahrhunderts durch Kurt Gödel, Alonso Church und Alan Turing gelegt. Besonders bedeutsam für die KI sind hierbei die Gödelschen Sätze wie der Vollständigkeitssatz, nach dem jede in der Prädikatenlogik formalisierte wahre Aussage beweisbar mit Hilfe der Schlussregeln eines formalen Kalküls ist, und der Unvollständigkeitssatz, nach dem es in Logiken höherer Stufe wahre Aussagen gibt, die nicht beweisbar sind.

In den vierziger Jahren wurden, basierend auf Ergebnissen aus der Hirnforschung, die ersten mathematischen Modelle für neuronale Netze entwickelt, wobei allerdings die Computer noch

nicht leistungsfähig genug waren, um einfache Gehirne simulieren zu können.

Seit den fünfziger Jahren gibt es programmierbare Rechenmaschinen und seit dieser Zeit gibt es die KI als praktische Wissenschaft. Seit 1985 werden sogenannte verteilte KI-Systeme erforscht. Dabei werden Parallelrechner zur Effizienzsteigerung benutzt, wobei sich gezeigt hat, dass die Verwendung "intelligenterer" Systeme gewinnbringender ist als die Parallelisierung. Daneben gibt es Modelle, die wie menschliche Teams bei der Bearbeitung ihrer Aufgaben kooperieren sollen. Es gibt viele Beispiele, bei denen ein einzelner Agent nicht in der Lage ist, ein Problem zu lösen, aber eine Zusammenarbeit im Team zu intelligentem Verhalten führt. Man kann sich die Funktionsweise wie die Arbeit eines Ameisenvolkes vorstellen, das in der Lage ist, komplexe und hohe Gebäude zu bauen, obwohl die einzelne Ameise kein Verständnis der globalen Zusammenhänge hat. Ein sehr aktuelles Forschungsgebiet ist das aktive Lernen von Robotern, die sich beispielsweise selbstständig beibringen, zu laufen.

In den Jahren 2006 bis 2016 gab es einen Durchbruch im Bereich neuronaler Netze (*Deep Learning*), indem einige Tricks herausgefunden wurden, durch die Maschinen beigebracht werden konnte, Objekte zu erkennen. Heute sind Serviceroboter für den Haushalt und autonomes Fahren in Reichweite. Die KI auf unseren Smartphones kann heute problemlos die Bilder ausgewählter Personen in der Foto-App finden und in der Medizin, bei der Diagnose von Krankheiten mit Hilfe von bildgebenden Verfahren wie CT und MRT, machen Maschinen mittlerweile sogar weniger Fehler als Ärzt*innen.

Die KI ist interdisziplinär, das bedeutet, in ihr finden sich viele Elemente unterschiedlichster Wissenschaften wieder, wie zum Beispiel Logik, Bildverarbeitung, Linguistik, Statistik, Philosophie, Psychologie und Neurobiologie. (vgl. (*2*), S. 6-13) In Bezug auf die Schule wäre so auch eine Erarbeitung des Themas in einem fächerübergreifenden Unterricht, beispielsweise in Form eines Projekts, denkbar.

Unterrichtsidee für den Einstieg in das Thema

Eine Idee für einen Einstieg in das Thema ist, die Schüler*innen mit einer Künstlichen Intelligenz chatten zu lassen. Es gibt beispielsweise den Chatbot *Woebot* (`https://woebothealth.com/`), mit dem man immer und überall über seine Gefühle sprechen kann und der mittels Künstlicher Intelligenz bei leichten Depressionen und Angstzuständen hilft (Abbildung 2.1). Es gibt dazu eine von Psycholog*innen von der Universität Stanford entwickelte kostenlose App für Apple und Android-Geräte, die 2017 auf den Markt gebracht wurde. Woebot tritt in Form eines kleinen Roboters mit sympathischer, lockerer Sprache auf, allerdings bisher nur auf Englisch. (vgl. (*5*), Online Ressource)

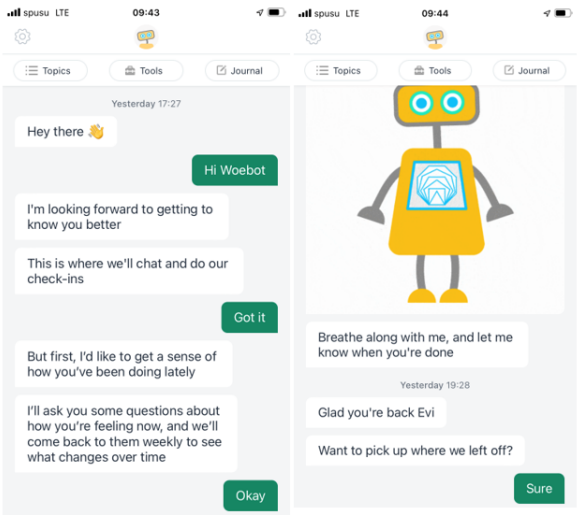


Abbildung 2.1: Chat mit Woebot

Folgende Fragen könnten dazu im Unterricht diskutiert werden:

- Wir sind bereits allseits von Künstlicher Intelligenz umgeben. Wo kommen Schüler*innen in ihrem Alltag mit KI in Berührung?
- Alle Schüler*innen laden sich *Woebot* aufs Smartphone oder Tablet. Wie lange wird es dauern, bis jede*r sich sicher ist, nicht mit einem Menschen zu chatten? Vor dem Start werden Schätzungen abgegeben, während dem Chatten stoppt jede*r die Zeit. Im Anschluss werden die Ergebnisse und Gedanken dazu im Plenum besprochen.
- Diskussion in der Klasse: KI - Gefahr oder Chance?

2.3 Arbeitsweise von ML und KI

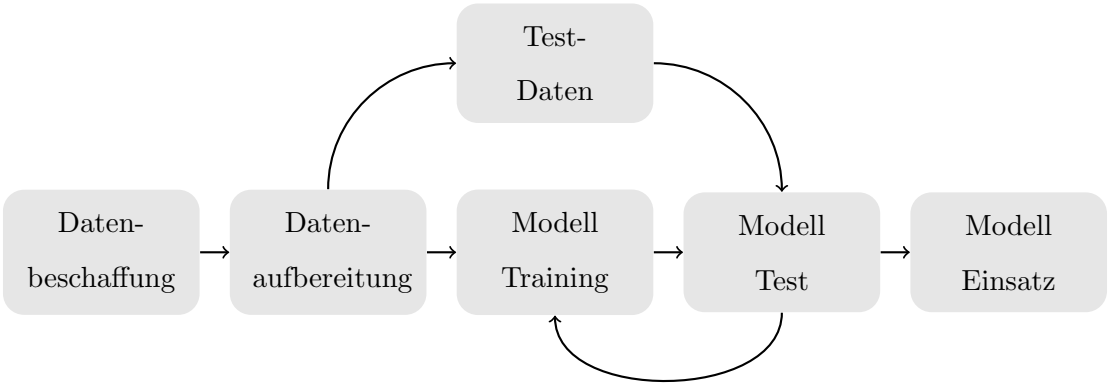


Abbildung 2.2: Ablauf eines ML-Prozesses

Wie Maschinen arbeiten und wie sie lernen wird in diesem Abschnitt erklärt. Weiters werden

gängige Lernstrategien beschrieben. Abbildung 2.2 veranschaulicht den typischen Machine Learning Prozess. Am Beginn des ML-Prozesses stehen immer eine Menge Daten. Im ersten Schritt (*Datenbeschaffung*) werden also Daten beschafft, die allerdings vor ihrer Verwendung noch analysiert und bereinigt werden müssen (*Datenaufbereitung*). Dabei werden fehlende Daten ergänzt, unnötige gelöscht und textuelle in numerische Daten umgewandelt, da Computer nur mit Zahlen umgehen können. Die Datenvorverarbeitung wird in Kapitel 3 genau beschrieben. Die bereinigten Daten werden nun nach bestimmten Regeln in Trainingsdaten und Testdaten geteilt (*Train-Test-Split*). (vgl. (*6*), Online Ressource) Jetzt kann anhand der Trainingsdaten der Prozess des Lernens (*Training*) gestartet werden, indem die Daten einem bestimmten ML-Algorithmus zugeführt werden. Als Ergebnis dieses Trainings erhält man ein *Modell*. Ist das Training abgeschlossen, kann anhand der dem Modell unbekannten Testdaten seine Güte festgestellt werden (*Testphase*). Ist das Ergebnis (noch) nicht zufriedenstellend, können Trainingsphase und Testphase wiederholt werden. (vgl. (*7*), S. 14) Die gängigsten ML-Strategien sind Supervised Learning (überwachtes Lernen), Unsupervised Learning (unüberwachtes Lernen), Semi-Supervised Learning, Reinforcement Learning (bestärkendes Lernen) und Transfer Learning [1].

2.3.1 Supervised Learning

Beim Supervised Learning ist die vorherzusagende Größe in den Daten enthalten. Anhand von sogenannten Eingangsfeatures $X = [x_1, x_2, ..., x_n]$ soll eine gekennzeichnete Größe y vorhergesagt werden (labeled data). Supervised Learning wird angewendet, wenn klar ist, wie die Ergebnisse aussehen, aber nicht wie die Daten zusammen hängen und miteinander in Beziehung stehen, damit ein bestimmtes Ergebnis eintritt. Die Aufgabe des ML-Algorithmus ist es hierbei, diese Beziehungen und die Zusammenhänge zu finden, die offenbar zu einem bestimmten Ergebnis führen. Das Modell kann damit durch die bereits mit y versehenen Trainingsdaten lernen, in Zukunft Vorhersagen für neue Daten zu treffen, bei denen der vorherzusagende Wert y unbekannt ist. (vgl. (*7*), S. 13)

Beispiele:

- **Klassifikation**

  Hier tritt y in Form von nominalem Charakter auf (ja/nein, Hund/Katze/Vogel/Fisch, ...). Das bekannte Titanic Disaster Dataset beschreibt eine Klassifikation, wobei mit Hilfe der Eingangsfeatures (Geschlecht, Alter, Anzahl von Angehörigen an Board, Ticketpreis, ...) vorhergesagt werden soll, ob eine bestimmte Person den Untergang der Titanic überlebt

[1] Diese Arbeit beschränkt sich auf Supervised Learning

(= 1) oder nicht (= 0). Dieses Dataset wird im weiteren Verlauf der Arbeit mehrfach verwendet. Eine nähere Beschreibung erfolgt im Abschnitt 3.2 (Datenvorverarbeitung).

- **Regression**

  Hier ist y ein stetiges Merkmal, kann also jeden beliebigen Zahlenwert in einem definierten Intervall annehmen.

  Ein Beispiel hierfür sind Preise von Gebrauchtwagen. Anhand von Eingangsfeatures (Jahr der Zulassung, Neupreise, Kilometerstand, Kraftstoffart, Anzahl Vorbesitzer, ...) kann so der Wert eines bestimmten Gebrauchtwagens ermittelt werden. Auch dieses Dataset wird in Abschnitt 3.2 im Detail vorgestellt und im weiteren Verlauf der Arbeit bei Klassifikationsverfahren verwendet.

2.3.2 Unsupervised Learning

In der Praxis sind Daten nicht immer mit einem Label, also dem erwarteten Ergebnis versehen oder das Ergebnis ist von vornherein unklar, weshalb es auch andere Lernstrategien geben muss. Beim Unsupervised Learning ist die vorherzusagende Größe in den Daten nicht enthalten (unlabeled data). Anhand von Eingangsfeatures $X = [x_1, x_2, ..., x_n]$ sollen Gruppen gefunden werden, die zusammengehörige Daten repräsentieren, während der Traingsphase gibt es aber kein vorgegebenes y. Die Lernleistung besteht darin, Daten zu analysieren und Muster in ihnen zu erkennen.

Beispiele

- **Clustering**

  Einteilung in Gruppen, zum Beispiel die Zuordnung von Kunden, die über Facebook mit einem Unternehmen verbunden sind, in vorher nicht näher definierte Käufer*innengruppen.

- **Rule Mining**

  Auffinden von Regeln, zum Beispiel die Identifikation von oft zusammen gekauften Gütern in einem Lebensmittelgeschäft.

- **Anomaly/Outlier Detection**

  Auffinden von Fehlern oder Anomalien unter den Input-Elementen, zum Beispiel die Suche nach verdächtigen oder missbräuchlichen Transaktionen bei Kreditkarten. (vgl. (7), S. 15-17)

2.3.3 Semi-Supervised Learning

Semi-Supervised Learning ist eine Mischung aus Unsupervised und Supervised Learning. Ein Anwendungsgebiet sind medizinische Bildgebungsverfahren wie Röntgenbilder oder CT-Scans. Hier sind eine Vielzahl an Daten (Bilder) verfügbar, aber nur wenige von ihnen sind mit einem Label versehen. In diesem Fall kann zuerst ein Unsupervised ML-Algorithmus die Bilder in Gruppen einteilen, die dann mit Labels versehen werden. Sobald eine große Anzahl von Daten mit Label existieren, kann ein Supervised ML-Algorithmus trainiert werden, um ein finales Modell zu erstellen, das genauere Vorhersagen über zukünftige Daten treffen kann. (vgl. (7), S. 17)

2.3.4 Reinforcement Learning

Reinforcement Learning ist eine Lernstrategie, im Zuge derer mit einem Belohnungs- und Bestrafungssystem gearbeitet wird. Eine Belohnung und/oder Bestrafung wird definiert und dem Algorithmus als Input zugeführt. Diese leiten den Algorithmus durch den Lernprozess, indem er seine Strategie durch Feedback nach und nach verbessert. Anwendungsgebiete für Reinforcement Learning finden sich im Bereich Robotik, Gaming, Navigation, aber auch in der Industrie. So kann ein Reinforcement Learning-Algorithmus die dynamische Preislegung von Gütern auf Basis von Angebot und Nachfrage mit dem Ziel der Profitmaximierung übernehmen. (vgl. (7), S. 18-20)

2.3.5 Transfer Learning

Beim Transfer Learning werden bereits für bestimmte Aufgaben erstellte Modelle benutzt, um mit ihrer Hilfe andere, verwandte Probleme zu lösen. Der Grund für diese Vorgehensweise ist die Tatsache, dass die Algorithmen zum Trainieren riesige Mengen an Daten benötigen, die in der Realität meist nicht verfügbar sind. Die Abbildung 2.3 zeigt, wie Transfer Learning für eine Aufgabe aus dem Bereich Objekterkennung benutzt werden kann. (vgl. (7), S. 20-21)
Die in Abbildung 2.3 vorgestellten Modelle sind neuronale Netze. Durch ein neuronales Netz werden Vorgänge im menschlichen Gehirn mathematisch modelliert. Sie haben mehrere Schichten (Layer), wobei jede Schicht etwas anderes lernt. In den ersten Schichten werden Ecken sowie Farben erkannt und erst in den letzten Schichten die charakteristischen Strukturen von Objekten und Lebewesen. Die Fähigkeit, Ecken und Farben zu erkennen kann allgemein genutzt werden. Modell A konnte diese Fähigkeit besser ausbilden als Modell B, weil Modell A mit einem größeren Dataset trainiert wurde. Modell B kann also durch einen Wissenstransfer verbessert werden (vgl. (7), S. 21).

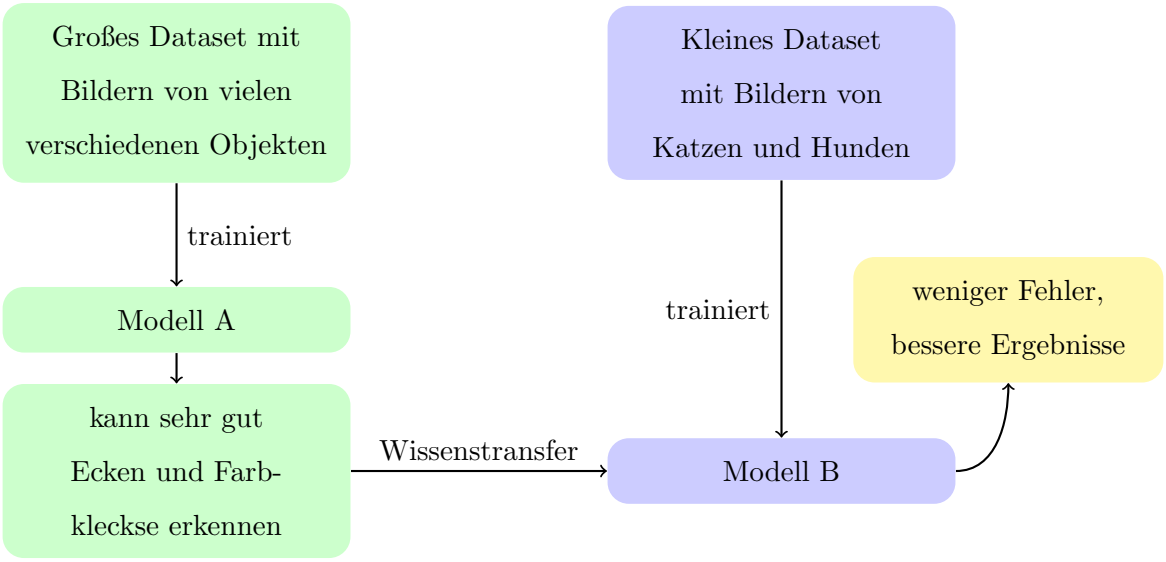


Abbildung 2.3: Bedeutung von Transfer Learning bei einer Bilderkennungsaufgabe

2.3.6 Overfitting und Underfitting

Es kann passieren, dass auf diese Weise erstellte Modelle keine gute Leistung erbringen. Ein Grund dafür könnte das sogenannte Underfitting oder das Overfitting sein. Underfitting liegt vor, wenn ein Modell nicht gut genug an die Daten angepasst wurde. In diesem Fall entstehen große Fehler sowohl bei Konfrontation des Modells mit den Trainingsdaten als auch mit den Testdaten. Overfitting liegt vor, wenn ein Modell die Trainingsdaten zu gut gelernt hat, sodass eine Generalisierung auf neue Daten nicht mehr funktioniert. Daten beinhalten immer natürliche Ausreißer oder Rauschen, die fälschlicherweise als "wahre Signale" gelesen werden können. Im Fall von Overfitting ist der Fehler bei Konfrontation mit den Trainingsdaten sehr klein, werden dem Modell aber die Testdaten eingespeist, erhält man große Fehler. Die Güte eines ML-Algorithmus kann über den sogenannten Score bestimmt werden, der auf Seite 36 genauer erklärt wird.

Zur Veranschaulichung von Over- und Underfitting wurden in Abbildung 2.4 zu gelabelten Trainingsdaten drei Modelle (Trendlinien) in Form von Polynomfunktionen ersten, zweiten und fünfzehnten Grades erstellt.

Für Modell A wurde eine lineare Funktion gewählt. Sie zeigt die Situation *Underfitting*. In Abbildung 2.5 werden die großen Fehler beim Testen veranschaulicht. Die großen Abstände der Datenpunkte zum Modell (siehe Abbildung 2.4) weisen darauf hin, dass auch bei Konfrontation mit den Trainingsdaten große Fehler entstehen würden.

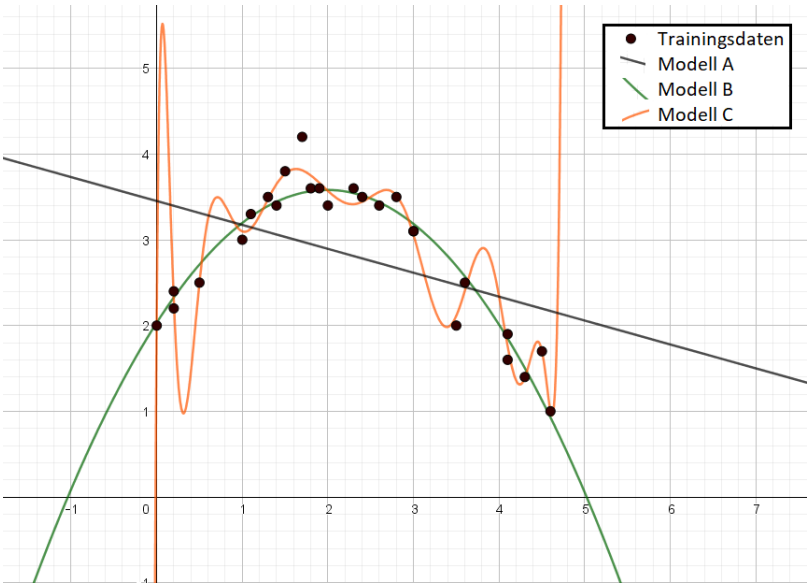


Abbildung 2.4: Modellerstellung mit den Trainingsdaten

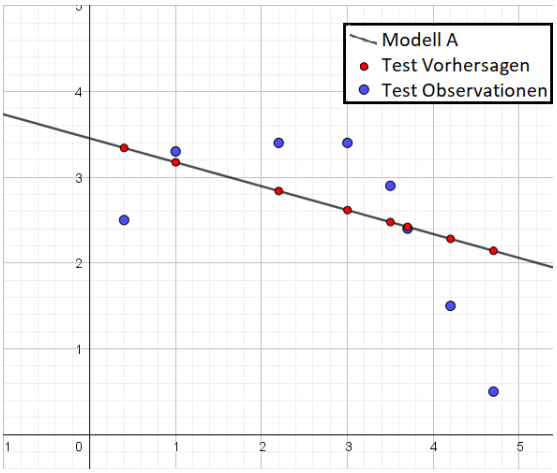


Abbildung 2.5: Modell A Test

Für Modell B wurde eine Polynomfunktion zweiten Grades gewählt. Sie folgt dem Trend der Daten und zeigt ein passendes Modell für diese Problemstellung.

Für Modell C wurde eine Polynomfunktion fünfzehnten Grades gewählt. Sie zeigt die Situation *Overfitting*. Die Kurve schmiegt sich bereits zu gut an die Trainingsdaten an, indem sie Ausreißer und Rauschen als wahre Werte annimmt (siehe Abbildung 2.4), wodurch große Fehler beim Testen entstehen.

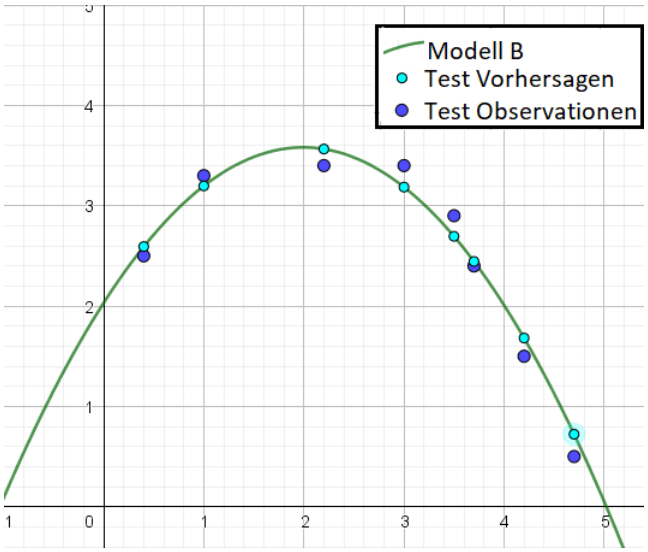


Abbildung 2.6: Modell B Test

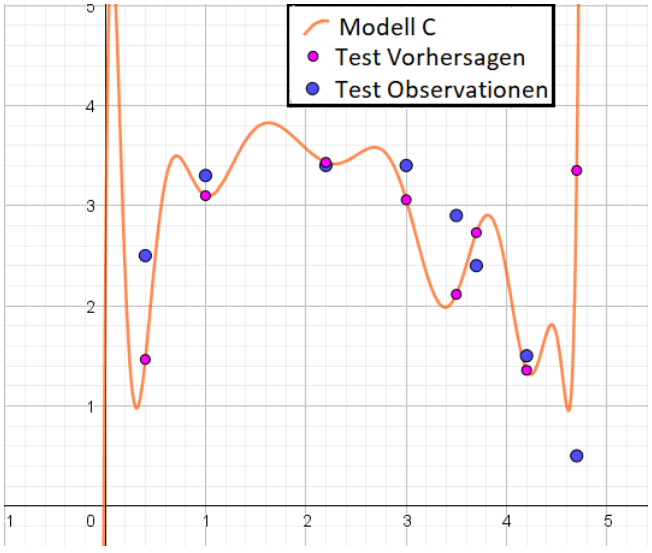


Abbildung 2.7: Modell C Test

Kapitel 3

Datenvorverarbeitung

Bei ML- und KI-Prozessen ist die Datenvorverarbeitung ein wesentlicher Schritt, da die Güte der Ergebnisse des Prozesses stark von der Güte des Dateninputs abhängt. Diese wichtige Tatsache postuliert sich in der Phrase "Garbage in, garbage out". (vgl. (*8*), Online Ressource) Die Datenvorverarbeitung gliedert sich im Wesentlichen in folgende Teilbereiche: (vgl. (*9*), S. 195ff)

- Visualisierung und Analyse der Daten
- Löschen bzw. Ergänzen fehlender Daten und Entfernen nicht relevanter Daten
- Behandlung nicht numerischer Daten, wobei zwischen ordinalen und kategorischen Daten zu unterscheiden ist
- Datennormalisierung oder -skalierung (Feature Scaling)

3.1 Lehrplanbezug

Die Datenvorverarbeitung weist zum Teil hohen Bezug zur Schulmathematik auf. Nach dem Lehrplan der AHS sollen bereits in der Sekundarstufe I beim Thema "Arbeiten mit Modellen, Statistik" Datenmengen unter Verwendung statistischer Kennzahlen (Mittelwert, Median, Quartil, relative Häufigkeit, Streudiagramm) untersucht werden. In der sechsten Klasse sollen die Schüler*innen Darstellungen und Kennzahlen der Beschreibenden Statistik kennen lernen und damit arbeiten können. Im Thema Funktionen sollen in der sechsten Klasse Veränderungen von Funktionsgraphen beschrieben werden können, wenn man von $f(x)$ zu $c \cdot f(x)$, $f(x)+c$, $f(x+c)$, bzw. $f(c \cdot x)$ übergeht. In der siebenten Klasse schließlich sollen Schüler*innen der AHS beim Thema diskrete Wahrscheinlichkeitsverteilungen unter anderem auch die Standardabweichung einer diskreten Zufallsvariablen kennen und deuten können. (vgl. (*10*))

In den Handelsakademien lernen die Schüler*innen in der vierten Klasse, die unterschiedlichen Datentypen (normalskaliert, ordinalskaliert, metrisch) zu beschreiben und erhobene Daten entsprechend zuzuordnen. Sie lernen, Daten zu erheben, Häufigkeitsverteilungen (absolute und relative Häufigkeiten) grafisch darzustellen und zu interpretieren. Überdies sollen sie die Auswahl einer bestimmten Darstellungsweise problembezogen argumentieren können. Die Lernenden sollen verschiedene Zentralmaße (arithmetisches Mittel, Median, Modus, geometrisches Mittel) berechnen, interpretieren und ihre Verwendung unter anderem im Bezug auf verschiedene Datentypen argumentieren können. Sie lernen, unterschiedliche Streumaße (Standardabweichung, Varianz, Spannweite, Quartile) zu berechnen und zu interpretieren. Median, Quartile und Spannweite sollen auch in einem Boxplot dargestellt und interpretiert werden können. In der fünften Klasse sollen die Schüler*innen der Handelsakademien den Begriff Standardabweichung erklären können und damit Argumentationen über die Normalverteilung anstellen können. Überdies sollen Schüler*innen im zweiten Jahrgang lernen, die Koeffizienten einer Funktion in Bezug auf den Verlauf des Graphen zu beschreiben und zu interpretieren. In der vierten Klasse ist der Korrelationskoeffizient nach Pearson ein Thema, seine Berechnung und Interpretation. (vgl. (*11*))

In den Höheren Technischen Lehranstalten lernen die Schüler*innen in der zweiten und / oder dritten Klasse, aus Stichprobenwerten Häufigkeitsverteilungen tabellarisch und graphisch darzustellen, Lage- und Streuungsmaße zu bestimmen und zu interpretieren. Dabei werden vor allem die Boxplots genannt. In der vierten Klasse treten wiederum der Mittelwert und die Standardabweichung im Zusammenhang mit der Normalverteilung auf. Ausgleichsfunktionen, der Korrelationskoeffizient sowie die lineare Regression sind in der fünften Klasse ein Thema. Beim Thema Funktionen sollen die Funktionsparameter interpretiert werden können (zweiter Jahrgang). (vgl. (*12*))

Die Unterscheidung von nominalen, ordinalen und metrischen Skalen ist Grundlage für die korrekte Verwendung von Lage- und Streuungsparametern. Die Datenskalierung ist eine anschauliche Anwendung für lineare Transformationen und vertieft das Verständnis für die Begriffe Mittelwert und Standardabweichung.

3.2 Titanic Disaster Dataset und Amerikanischer Gebrauchtwagenmarkt

Im weiteren Verlauf der Arbeit wird häufig auf zwei Datasets Bezug genommen, daher werden diese hier kurz vorgestellt. Beide wurden von der Plattform Kaggle (www.kaggle.com) heruntergeladen.

Titanic Disaster Dataset

Dieses Dataset beschreibt in Abhängigkeit von 11 Features, ob ein Passagier der Titanic deren Untergang überlebt hat oder nicht. (vgl. (*13*)), Online Ressource) Es handelt sich also um eine binäre Klassifikation.
Die folgende Grafik zeigt die ersten 10 von 891 Datensätzen:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

y

Abbildung 3.1: Titanic Disaster Dataset

Die folgende Übersicht wurde mit der Python-Bibliothek Pandas erstellt (vgl. (*14*), Python Library) und sie wurde um textuelle Beschreibungen der einzelnen Spalten ergänzt.

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
#   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64         Interne PassagierId
 1   Survived     891 non-null    int64         Vorhezusagender Wert - Label
 2   Pclass       891 non-null    int64         Passagierklasse Werte 1,2,3
 3   Name         891 non-null    object        Name des Passagiers
 4   Sex          891 non-null    object        Geschlecht des Passagiers
 5   Age          714 non-null    float64       Alter des Passagiers
 6   SibSp        891 non-null    int64         Anzahl der Geschwister/Cousins an Bord
 7   Parch        891 non-null    int64         Anzahl der Eltern / Kinder an Bord
```

```
 8   Ticket       891 non-null    object        Ticketnummer
 9   Fare         891 non-null    float64       Ticketpreis
10   Cabin        204 non-null    object        Kabinennummer
11   Embarked     889 non-null    object        Zustiegsstelle Werte S, C, Q
```

In dieser Übersicht lässt sich die Anzahl der fehlenden Daten pro Spalte ablesen. So sind etwa nur 204 Kabinennummern bekannt, es fehlen also 687 Kabinennummern. Außerdem lassen sich die Datentypen der einzelnen Features ablesen (Dtype), wobei `float64` einen Fließkommawert repräsentiert, `object` einen textuellen Eintrag darstellt und `int64` bedeutet, dass es sich um eine ganze Zahl handelt.

Amerikanischer Gebrauchwagenmarkt

Dieses Dataset stammt ebenfalls von Kaggle und beschreibt den Gebrauchtwagenmarkt in den USA. Hier soll der Verkaufspreis des entsprechenden Fahrzeuges vorhergesagt werden. Es handelt sich also um eine Regression. Wieder werden die ersten 10 Datensätze visualisiert.

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0
5	vitara brezza	2018	9.25	9.83	2071	Diesel	Dealer	Manual	0
6	ciaz	2015	6.75	8.12	18796	Petrol	Dealer	Manual	0
7	s cross	2015	6.50	8.61	33429	Diesel	Dealer	Manual	0
8	ciaz	2016	8.75	8.89	20273	Diesel	Dealer	Manual	0
9	ciaz	2015	7.45	8.92	42367	Diesel	Dealer	Manual	0

y

Abbildung 3.2: Dataset Gebrauchtwagen

Die folgende Übersicht zeigt wieder die verschiedenen Datenfelder inklusive der Datentypen und die Anzahl der pro Datenfeld vorhandenen Werte (hier fehlen keine Daten).

```
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
#   Column         Non-Null Count  Dtype
---  ------         --------------  -----
0   Car_Name       301 non-null    object          Name des Autos
1   Year           301 non-null    int64           Baujahr
2   Selling_Price  301 non-null    float64         Aktueller Verkaufspreis (Label)
3   Present_Price  301 non-null    float64         Aktueller Neupreis
4   Kms_Driven     301 non-null    int64           Gefahrene Kilometer
```

```
5   Fuel_Type       301 non-null    object          Antriebsart: 'Petrol', 'Diesel', 'CNG'
6   Seller_Type     301 non-null    object          Verkäufer: 'Dealer', 'Individual'
7   Transmission    301 non-null    object          Schaltung/Automatik: 'Manual', 'Automatic'
8   Owner           301 non-null    int64           Anzahl der Vorbesitzer: Werte 0,1,3
```

3.3 Datenvisualisierung und Analyse

Ein wesentlicher erster Schritt bei der Auswahl geeigneter ML-Verfahren ist die grafische Aufbereitung der Daten eines Datasets und die damit verbundene Analyse der Daten. In diesem Abschnitt sind einige Grafiken zu finden, die mit Hilfe der Python-Grafikbibliothek Seaborn (vgl. (*15*), Online Resource) erzeugt wurden. Der Quellcode zur Erzeugung der Grafiken wird im Anhang angegeben.

3.3.1 Histogramme

In Abbildung 3.3 werden Historamme dargestellt, welche die Verteilung des Alters der Passagiere der Titanic zeigen. Die linke Grafik zeigt eine Klasseneinteilung mit der Klassenbreite 10 Jahre über alle Passagiere, die rechte Grafik unterscheidet zwischen weiblichen (blau) und männlichen (orange) Reisenden. Daraus lässt sich z.B. erkennen, dass bei den Kindern zwischen 0 und 10 Jahren nur geringfügig mehr Buben als Mädchen an Bord waren, während sich in der Altersklasse 20 bis 30 mehr als doppelt so viele Männer als Frauen finden.

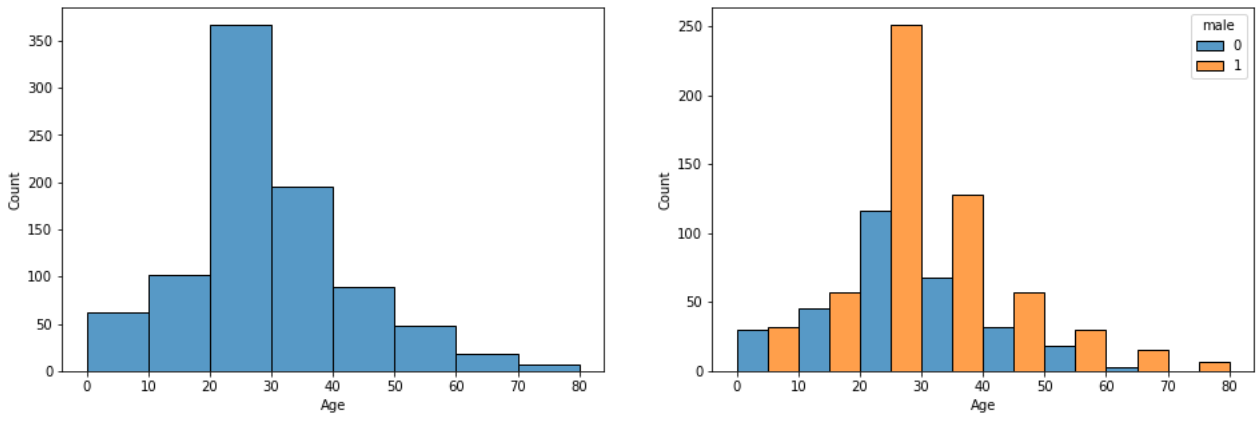


Abbildung 3.3: Histogramm - Titanic Verteilung des Alters

In der Schule könnte beispielsweise zum Thema gemacht werden, wie sich die Aussagekraft einer Datenvisualisierung verändert, je nachdem, wie die Darstellung erfolgt. Hierzu könnten etwa die beiden Histogramme aus Abb. 3.3 vergleichend betrachtet werden. Im Unterricht aller drei Schultypen könnten zur Grafik mit der Altersverteilung im Zusammenhang mit dem Überleben des Unterganges (Abb. 3.5) folgende Fragen gestellt werden:

- In welchen Altersklassen gibt es mehr Überlebende als Tote?
- In welcher Altersklasse ist der Prozentsatz der Überlebenden am kleinsten?

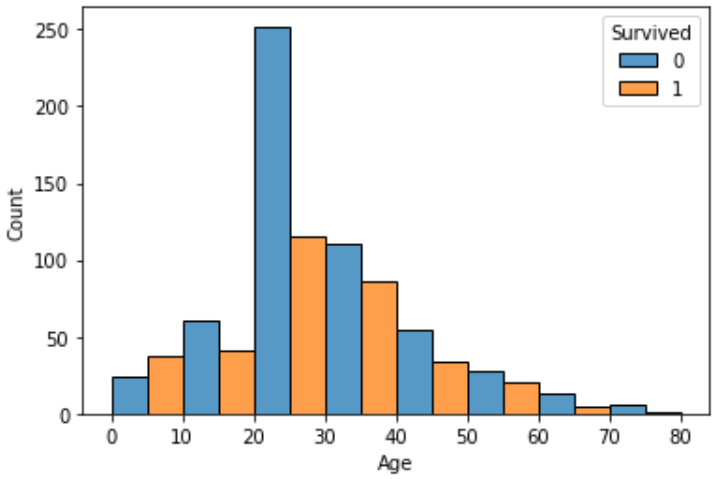


Abbildung 3.4: Histogramm - Titanic Verteilung des Alters nach überleben/nicht überleben

Da die zu bestimmende Größe *Survived* ist, ist natürlich auch der Anteil der Überlebenden je nach Passagierklasse bzw. Geschlecht interessant. Auch hier lassen sich, genauso wie zu den folgenden Boxplots, im Unterricht viele Interpretationsfragen stellen. Überdies können die Histogramme und Boxplots von den Schüler*innen per Hand oder mit Geogebra gezeichnet werden.

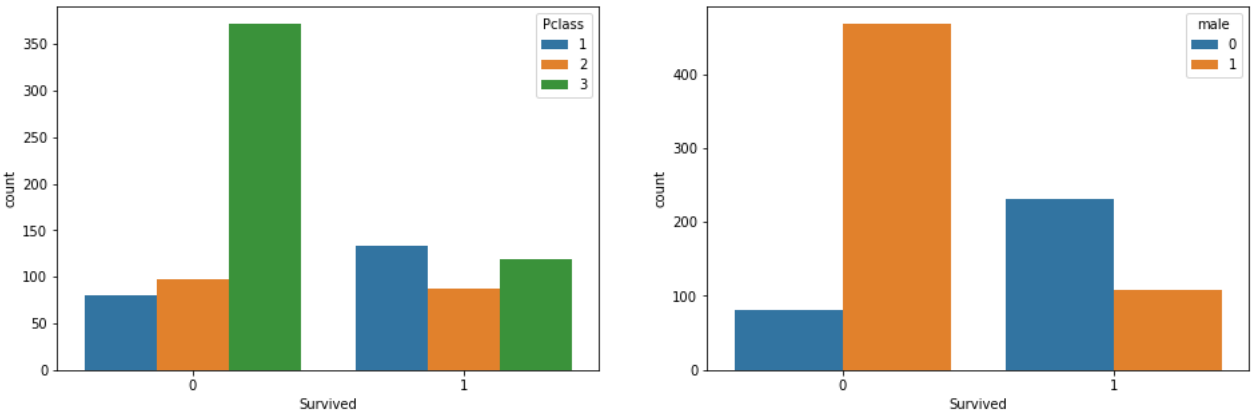


Abbildung 3.5: Histogramm - Titanic Überlebende nach Passagierklasse und Geschlecht

3.3.2 Boxplot

Boxplots findet man bereits im Lehrplan der Unterstufe. Sie können bei der Datenanalyse zum Vergleich einzelner Features bezogen auf ein zweites Feature gute Vergleiche liefern. Die Abbildung 3.6 zeigt Boxplots für das Alter der Passagiere der Titanic. Die linke Grafik verwendet alle bekannten Altersangaben, die rechte zeigt die Altersverteilung der Passagiere in Abhängigkeit

von der Passagierklasse. Hier sieht man, dass der Median des Alters in der teuersten Passagierklasse 1 am höchsten und in der billigsten Passagierklasse 3 am niedrigsten ist.

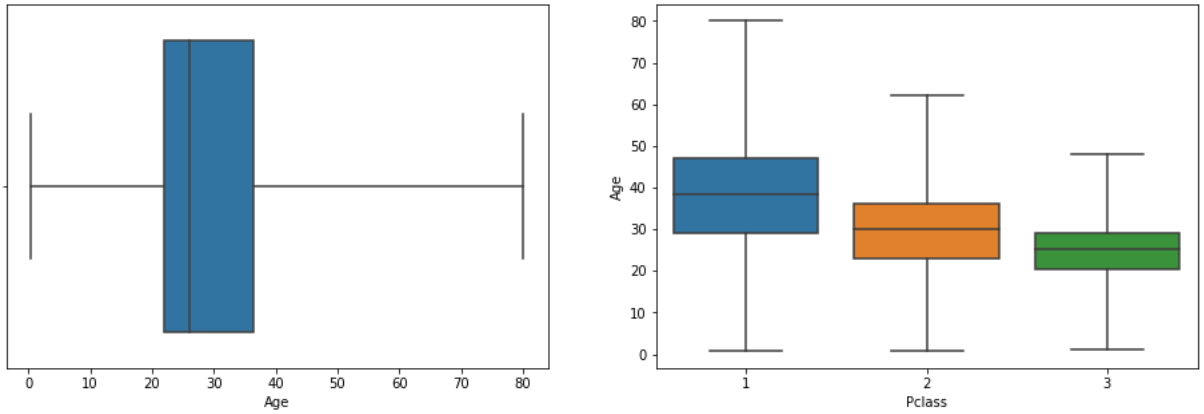


Abbildung 3.6: Boxplot - Titanic Analyse der Altersverteilung

3.3.3 Zusammenhänge zwischen Features - Scatterplot und Regression

Eine weitere Form der Datenvisualisierung bilden Punktwolken, auch Scatterplots, hier sogar in Verbindung mit einer Regressionsgeraden, denn oft ist es vorteilhaft, den Zusammenhang zwischen zwei Features zu visualisieren. Die Grafik 3.7 zeigt den Zusammenhang zwischen dem Neupreis (*Present_Price*) und dem Gebrauchtenwagenpreis (*Selling_Price*) beim amerikanischen Gebrauchtwagenmarkt:

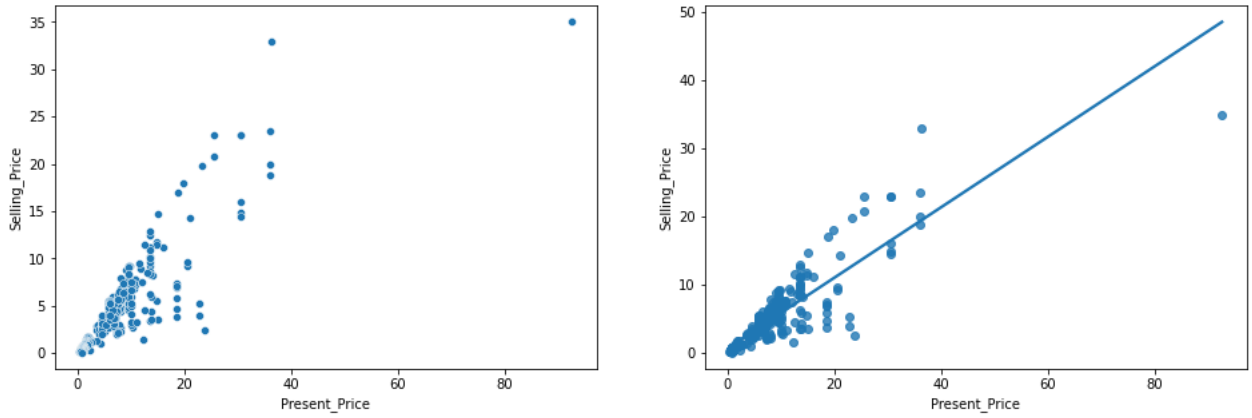


Abbildung 3.7: Scatter- und Regplot

Dabei visualisiert die linke Grafik nur die Punktwolke, während in der rechten Grafik auch die Regressionsgerade zwischen den beiden Features dargestellt ist. Hier lässt sich ein hoher Zusammenhang erkennen, was auch der Korrelationskoeffizient von 0,88 bestätigt. Außerdem sieht sowohl beim Scatterplot als auch an Hand der Steigung der Regressionsgeraden, dass mit wachsendem Neupreis wie zu erwarten auch der Gebrauchtwagenpreis steigt.

Möchte man mehrere Features eines Datasets miteinander vergleichen, so eignet sich der sogenannte Pairplot. Die Grafik 3.8 zeigt die Zusammenhänge zwischen Baujahr, Verkaufspreis, Neupreis und Kilometerstand. Ergänzend zu dieser Grafik können alle Korrelationskoeffizienten zwischen diesen Features berechnet werden. Dabei lassen sich vor allem bei der Interpretation der Korrelationskoeffizienten viele Schlüsse ziehen, wie gut die vorherzusagende Größe von den Features abhängig ist.

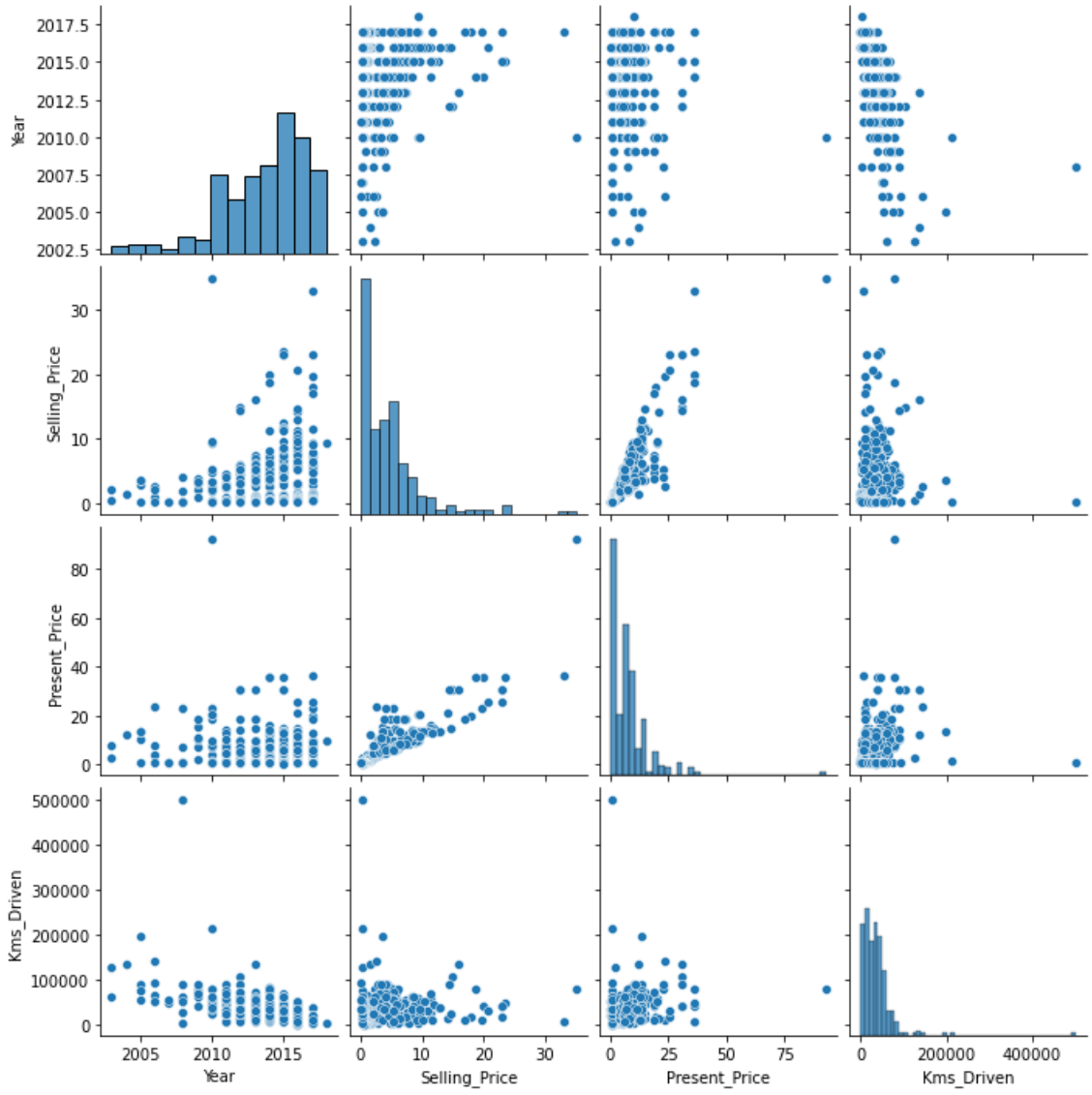


Abbildung 3.8: Scatter- und Regplot

	Year	Selling_Price	Present_Price	Kms_Driven
Year	1.000000	0.236141	-0.047584	-0.524342
Selling_Price	0.236141	1.000000	0.878983	0.029187
Present_Price	-0.047584	0.878983	1.000000	0.203647
Kms_Driven	-0.524342	0.029187	0.203647	1.000000

Hier sieht man, dass nur zwischen Neupreis und Verkaufspreis ein hoher Zusammenhang besteht und dass die Anzahl der gefahrenen Kilometer wenig Einfluss auf den Verkaufspreis hat. Überraschenderweise ist dieser Korrelationskoeffizient positiv, d.h. mit Anzahl der gefahrenen Kilometer steigt auch der Verkaufspreis. Eine mögliche Interpretation ist die Vermutung, dass bei großen und teuren Autos die Kilometeranzahl höher ist als bei Kleinwagen.

3.4 Löschen bzw. Ergänzen fehlender Daten, Entfernen nicht relevanter Daten

Oft fehlen in den zu analysierenden Daten einzelne Werte. ML-Algorithmen bzw. neuronale Netze können mit solchen fehlenden Daten nicht umgehen. Daher sind diese im Rahmen der Datenvorverarbeitung zu löschen bzw. zu ergänzen. (vgl. (*9*), S. 200ff)

Einzelne Datensätze mit fehlenden Daten können natürlich gelöscht werden. Dies ist aber insbesondere dann problematisch, wenn man dadurch zu viele Datensätze verliert. In diesem Fall ist die i.A. bessere Strategie das sinnvolle Ergänzen von Daten.

So fehlt etwa im Titanic Disaster Dataset bei 177 von 891 Datensätzen das Alter des/der Reisenden. Eine Strategie wäre, die fehlenden Altersangaben gemäß Abbildung 3.6 (links) durch den Mittelwert oder den Median aller zur Verfügung stehenden Altersangaben zu ersetzen. Besser erscheint jedoch gemäß dem rechten Teil von Abbildung 3.6 die fehlenden Werte für das Alter mit dem Median oder dem Mittelwert des Alters der jeweiligen Passagierklasse zu ergänzen, da hier ein eindeutiger Trend zu erkennen ist: Passagierklasse 1 hat den höchsten Altersmedian und Passagierklasse 3 den niedrigsten.

Das Ergänzen fehlender Daten ist ein wichtiger Prozess der Datenvorverarbeitung, von dem die Güte der erzielten Ergebnisse eines KI-Prozesses stark abhängig ist. Ebenso wichtig wie die Ergänzung fehlender Daten ist die Elimination von Daten, die für den vorherzusagender Wert unerheblich sind. So scheinen im Titanic-Disaster-Dataset die Features *PassengerId*, *Name* und *Ticket* keinen Einfuss auf das vorherzusagende Taget *Survived* zu haben. Diese Spalten sollten also gelöscht werden. Allenfalls könnte man das Feature *Name* auf darin vorkommende akademische Grade untersuchen und diese Spalte durch ein neues binäres Feature *IsAcademic* ersetzen. Dieses Feature hat möglicherweise Einfluss auf die Überlebenschance. Weiter fehlen in der Spalte *Cabin* 678 von 891 Werten. Hier erscheint es nicht sinnvoll, diese Daten zu ersetzen, sondern das Feature komplett zu entfernen.

3.5 Behandlung nicht numerischer Daten

ML-Algorithmen können nur numerische Daten verarbeiten. Solche Daten sind, wenn sie auf einer Verhältnisskala liegen, in der weiteren Verarbeitung unproblematisch, müssen aber abhängig vom verwendeten Algorithmus skaliert werden (vgl. Abschnitt 3.6, Feature Scaling). Nicht numerische (textuelle Daten) müssen in numerische Daten konvertiert werden. Dabei ist zwischen nominalen und ordinalen Features zu unterscheiden. (vgl. (*9*), S. 39ff)

3.5.1 Behandlung nominalskalierter Daten

Nominal skalierte Daten sind Daten, die sehr oft textuell beschrieben werden und die in keine natürliche Reihenfolge gebracht werden können. Beispiele dafür sind Familienstand (ledig, verheiratet, verwitwet, geschieden) oder das Geschlecht (männlich, weiblich, divers). Mathematisch ist hier nur die Bestimmung des Modalwertes sinnvoll.
Beim Titanic - Disaster - Dataset ist ein solches nominal skaliertes Merkmal das Feature *Embarked*. Es beschreibt die Einstiegstelle des betrachteten Passagiers. Hier kommen die Werte S (Southampton - England), C (Cherbourg - Frankreich) und Q (Queenstown - heute Chob in Irland) vor. Für solche Daten führt man ein sogenanntes One-Hot-Encoding durch. (vgl. (*16*), Online Resource) Dies bedeutet, dass man jeden Featurewert (hier also S, C und Q) in einem eigenen Feature binär codiert. Die Abbildung 3.9 demonstriert dies für die ersten 10 Datensätze des Titanic-Disaster Datasets.

	Embarked	C	Q	S
0	S	0	0	1
1	C	1	0	0
2	S	0	0	1
3	S	0	0	1
4	S	0	0	1
5	Q	0	1	0
6	S	0	0	1
7	S	0	0	1
8	S	0	0	1
9	C	1	0	0

Abbildung 3.9: Behandlung nicht numerischer Daten: One Hot Encoding

Nun ist natürlich noch das alte Feature *Embarked* aus den Daten zu entfernen. Außerdem ist zu berücksichtigen, dass die so erzeugten drei neuen Features linear abhängig sind. Für jeden Passagier ist die Summe der drei Werte gleich 1. Solche Abhängigkeiten sind unerwünscht, da Ml-Algorithmen oder Neuronale Netze diese Abhängigkeiten lernen könnten. Führt man auf

diese Art und Weise ein One-Hot-Encoding durch, muss also auch noch eines der neu erzeugten binären Features aus den Daten gelöscht werden. Löscht man etwa die Spalte C, so bedeutet Q=0 und S=0, dass der entsprechende Passagier in Cherbourg an Bord gegangen ist.

3.5.2 Behandlung ordinal skalierter Daten

Ordinal skalierte Daten haben eine natürliche Reihenfolge, stehen aber in keinem natürlichen Verhältnis zueinander. Ein hier häufig zitiertes Beispiel sind Schulnoten. Etwa ist ein Befriedigend (3) nicht drei mal so schlecht wie ein Sehr Gut (1). Bei solchen ordinal skalierten Daten sind Rechenoperationen wie z.B. die Berechnung des arithmetischen Mittelwerts nicht sinnvoll. Weitere Beispiele für ordinal skalierte Merkmale sind Güteklassen und die Bewertung von Konsumgütern mit beispielsweise 1 bis 5 Sternen.
Beim Gebrauchtwagen-Dataset könnte ein zusätzliches ordinal skaliertes Feature *Condition* mit den Werten "good", "like new", "excellent" und "fair" existieren. Diese müssten vor der Verarbeitung in die richtige Reihenfolge gebracht und durch numerische Werte bzw. Prozentwerte ersetzt werden: "like new" = 1, "excellent" = 2, "good" = 3, "fair" = 4

3.6 Feature Scaling

Bevor Daten von einem ML- bzw. KI-Algorithmus analysiert werden können, müssen sie in vielen Fällen skaliert werden. Die einzelnen Features von Rohdaten haben oft unterschiedliche Wertebereiche und dies ist speziell bei der Berechnung euklidischer Abstände problematisch. Aber auch bei der Optimierung von Gewichten bei neuronalen Netzen ist es notwendig, dass die einzelnen Features gleiche und kleine Wertebereiche aufweisen.
Zur Skalierung der Features stehen im Wesentlichen Normalisierung und Standardisierung zur Verfügung. (vgl. (*9*) S. 212f)

3.6.1 Normalisierung

Bei der Normalisierung werden die Werte einer Datenreihe mit Hilfe linearer Transforationen (Verschiebung, Skalierung) auf den Wertebereich $[0;1]$ abgebildet[1]. Dabei wird das Minimum der Werte in den Nullpunkt verschoben und die Spannweite wird auf den Wert 1 skaliert. Nach der Normalisierung liegen alle Werte im Intervall $[0;1]$.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.1}$$

[1] MinMax-Scaler verallgemeinern dieses Intervall oft auf $[min; max]$

3.6.2 Standardisierung

Bei der Standardisierung werden Mittelwert $\overline{x}$ und Standardabweichung s der Datenreihe bestimmt, der Mittelwert wird in den Nullpunkt verschoben und die Standardabweichung auf den Wert 1 skaliert. Dies bewirkt, dass die Daten abgesehen von Ausreißern im Wesentlichen auf den Wertebereich $(-3; 3)$ abgebildet werden.

$$x' = \frac{x - \overline{x}}{s} \tag{3.2}$$

Standardisierung und Normalisierung vgl. (*2*), S. 228f.

Konkretes Beispiel

Die Grafik 3.10 zeigt zweidimensionale mit 0 (rot) und 1 (schwarz) gelabelte Daten, bei denen eine schöne optische Trennung der beiden Klassen erkennbar ist. Die Wertebereiche der Features x_1 und x_2 unterscheiden sich jedoch stark voneinander (vgl. Tabelle 3.1).

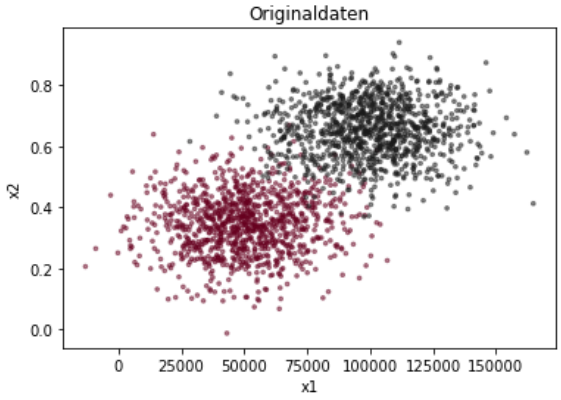


Abbildung 3.10: Feature Scaling: Rohdaten

	x_1	x_2
Mittelwert	75295.72	0.50
Standardabweichung	31491.01	0.18
Minimum	-10755.78	0.06
Maximum	150956.35	0.96

Tabelle 3.1: Feature Scaling: Parameter der Rohdaten

Führt man auf diesen Daten nach einem Train-Test-Split im Verhältnis 80:20 den Klassifikationsalgorithmus KNN mit 5 Nachbarn aus, so erhält man auf den Trainingsdaten einen Score von 90.88% und auf den Testdaten einen Score von 87.25%.
KNN steht für K-Nearest-Neighbors und wird im Abschnitt 4.3 ausführlich vorgestellt. Im We-

sentlichen werden hier zu einem Testpunkt k Nachbarn analysiert und der Testpunkt wird jener Klasse zugeordnet, die unter den Nachbarn am häufigsten auftritt.

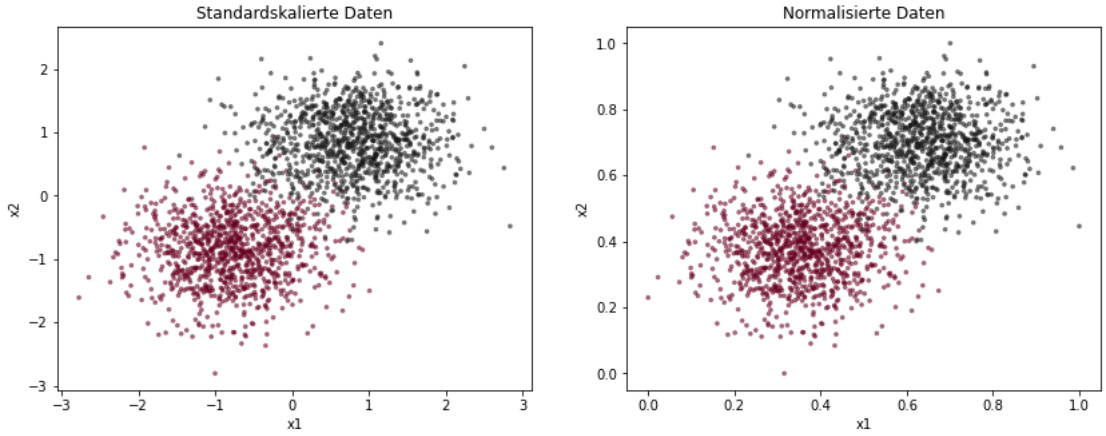


Abbildung 3.11: Feature Scaling: Standardisiert und Normalisierte Daten

Abbildung 3.11 zeigt die standardisierten und normalisierten Daten. Die Gestalt der Punktwolke ändert sich dabei nicht, da ja jeweils nur eine Verschiebung und eine Skalierung in Achsenrichtung durchgeführt werden. Die Tabelle 3.2 zeigt die Parameter der Datenreihen nach Normalisierung und Standardisierung:

	Standardisierung		Normalisierung	
	x_1	x_2	x_1	x_2
Mittelwert	0.00	0.00	0.49	0.54
Standardabweichung	1.00	1.00	0.18	0.19
Minimum	-2.77	-2.81	0.00	1.00
Maximum	2.83	2.41	0.00	1.00

Tabelle 3.2: Feature Scaling: Parameter der skalierten Daten

Führt man auf diesen transformierten Daten nach einem Train-Test-Split im Verhältnis 80:20 den Klassifikationsalgorithmus KNN der Pythonbibliothek `sklearn` mit 5 Nachbarn aus, so verbessert sich bei den Trainingsdaten der Score von 90.88% auf 97.81% bei Standardisierung und auf 98.00% bei Normalisierung. Bei den Testdaten konnte bei beiden Verfahren ein verbesserter Score von 97.75% erzielt werden.

Auf Grund der Formel 3.2 haben die standardisierten Datenreihen einen Mittelwert 0.0 und eine Standardabweichung 1.0, das Intervall wird abgesehen von Ausreißern in etwa den Bereich $(-3; 3)$ umfassen. Es wird empfohlen, die Standardisierung bei annähernd normalverteilten Daten anzuwenden.

Bei der Normalisierung hängen Mittelwert und Standardabweichung von der Verteilung der Daten ab, das Intervall ist aber immer $[0; 1]$. Für gleich verteilte Daten ist die Normalisierung

der Standardisierung vorzuziehen.
Gemäß ((2) Seite 230) ist die Datenskalierung bei allen Arten von neuronalen Netzen sowie bei Nearest-Neighbour Verfahren anzuwenden. Nicht notwendig (wenn auch nicht nachteilig) ist Featurescaling beim Naive Bayes Algorithmen.
Ein Nachteil von FeatureScaling ist, dass die Interpretierbarkeit der Daten verloren geht, was bei Algorithmen wie Entscheidungsbäumen die Nachvollziehbarkeit stört.

3.7 Ideen für den Unterricht

Anwendung elementarer Transformationen

Sowohl Normalisierung als auch Standardisierung basieren auf zwei linearen Transformationen (Translation und Skalierung) eindimensionaler Datenreihen, deren Verständnis grundlegend ist. Die Betrachtung aus verschiedenen Blickwinkeln fördert dieses Verständnis. Die Transformationen einer Datenreihe sind vergleichbar mit elementaren Funktionstransformationen in Richtung der unabhängigen Variablen und können auch so veranschaulicht werden.

Konkrete Aufgabenstellungen für den Unterricht

Die Datenreihe 2,2,3,3,5,6,7,7,10,20 ist zu normalisieren und zu standardisieren.
Es gilt $\overline{x} = 6.5$, $s = 5.4006$, $min(x) = 2$, Spannweite = 18. Daher ergibt sich für den ersten Wert $x_1 = 2$ bei Normalisierung $x_1' = \frac{2-2}{18} = 0$. Hier erscheint es wichtig darauf hinzuweisen, dass die Transformation $x_1 \to x_1 - 2$ eine Verschiebung der Datenreihe um 2 Einheiten nach links bedeutet, also die Datenreihe 0,0,1,1,3,4,5,5,8,18 erzeugt. Nun werden alle Werte durch die Spannweite 18 dividiert, was einer Stauchung um den Faktor $\frac{1}{18}$ entspricht. Analog lassen sich die Rechenschritte bei der Standardisierung interpretieren.
Die Ergebnisse werden in folgender Tabelle zusammengefasst:

Originaldaten	2	2	3	3	5	6	7	7	10	20
Normalisiert	0	0	0.06	0.06	0.17	0.22	0.28	0.28	0.44	1
Standardisiert	-0.83	-0.83	-0.65	-0.65	-0.27	-0.09	0.09	0.09	0.64	2.50

Tabelle 3.3: Feature Scaling: Beispiel für den Unterricht

Aufgrund dieser Ergebnisse lassen sich nun z.B. folgende Fragen erörtern:

- Welche Originalwerte sind nach der Standardisierung negativ, welche positiv?
- Warum sind die standardisierten Werte der Originalwerte 6 und 7 vom Betrag her gleich?

Kapitel 4

Spezielle Machine Learning Algorithmen

In diesem Abschnitt werden einige ausgewählte ML-Algorithmen und deren Bezug zu den Lehrplänen von AHS, HAK und HTL vorgestellt. Im Zuge dessen werden teilweise selbst generierte Testdaten verwendet, wobei die Generierung dieser Testdaten mit der Programmiersprache Python erfolgt. Dabei wird der Regressionsalgorithmus Lineare Regression auf den amerikanischen Gebrauchtwagenmarkt und alle vorgestellten Klassifikationsalgorithmen (Logistische Regression, KNN, Naive Bayes) auf das Titanic Disaster Dataset angewendet. Die entsprechenden Quellcodes und die dabei erzielten Ergebnisse werden im Anhang B zusammengefasst.

4.1 Lineare Regression

Die lineare Regression ist der grundlegendste und wichtigster ML-Algorithmus. Die Grundidee ist, aus einer Observation $\mathbf{x}$, also einem n-dimensionalen Vektor, einen reellen Wert y (Label) vorherzusagen (vgl. Abschnitt 2.3.1). Hier wird allerdings vorausgesetzt bzw. angenommen, dass zwischen den Eingangsfeatures und dem vorherzusagenden Wert ein annähernd linearer Zusammenhang besteht. Der für den Eingangsvektor $\mathbf{x}$ vom Modell vorhergesagte Wert $\hat{y}$ berechnet sich auf Grund des linearen Zusammenhangs über die Beziehung

$$\hat{y} = \mathbf{a} \cdot \mathbf{x} + b = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + b = \sum_{i=1}^{n} (a_i \cdot x_i) + b \tag{4.1}$$

wobei der Vektor $\mathbf{a}$ und das Skalar b die im Training zu lernenden Parameter des Modells sind. Im

Training werden diese Parameter zunächst mit zufälligen Werten vorbelegt und ein Fehlerwert ermittelt. (vgl. (*17*) S. 59ff) Zu diesem Zweck definiert man nach der Methode der kleinsten Quadrate eine Cost- oder Errorfunction

$$C(\mathbf{a}, b) = \sum_{j=1}^{m} (\hat{y}_j - y_j)^2 \tag{4.2}$$

wobei m die Anzahl der Datensätze im Trainingsdataset bezeichnet. Nun wird das Minimum dieser Costfunction bestimmt. Konkret handelt es hier also um eine Extremwertaufgabe für eine Funktion mit $n + 1$ Unbekannten. Im Rahmen des Machine Learnings wird das Minimum der Costfunktion in der Regel nicht analytisch sondern iterativ mit Hilfe des Gradientabstiegsverfahrens (vgl. Abschnitt 5.6.3) bestimmt. (vgl. (*18*), Online Ressource)

4.1.1 Lehrplanbezug

Der Mathematiklehrplan der AHS sieht in der fünften Klasse einiges zum Thema Funktionen vor. So sollen Schüler*innen lernen, Abhängigkeiten, die durch reelle Funktionen in einer Variablen erfassbar sind, mittels Termen, Tabellen und Graphen zu beschreiben und über den Modellcharakter von Funktionen zu reflektieren. Lineare Funktionen sollen beschrieben und untersucht werden. Auch Formeln sollen in Hinblick auf funktionale Aspekte untersucht werden. Überdies soll mit Funktionen in anwendungsorientierten Bereichen gearbeitet werden und Schüler*innen sollen lernen, Funktionen als mathematische Modelle aufzufassen. Die Lineare Regression ist kein Teil des Lehrplans der AHS. Im dritten Jahr sollen beim Thema Erweiterungen und Exaktifizierung der Differentialrechnung weitere Anwendungen der Differentialrechnung, insbesondere aus Wirtschaft und Naturwissenschaft durchgeführt werden. Dazu würden Extremwertaufgaben passen, sie sind jedoch nicht explizit im AHS-Lehrplan angeführt. (vgl. (*10*), Online Ressource) Auch die Schüler*innen der Handelsakademien müssen in der ersten Klasse lernen, Funktionen als Modelle zur Beschreibung von Zusammenhängen zwischen Größen zu verstehen und zu erklären. Sie lernen, Funktionen in einer Variablen in einem kartesischen Koordinatensystem darzustellen. Überdies sollen die Darstellungsformen linearer Funktionen interpretiert und erklärt werden können, insbesondere geht es hier um die Bedeutung der Parameter Steigung und Achsenabschnitt. Im vierten Jahr gibt es in den Handelsakademien einen Schwerpunkt "Optimierung und Regressionsrechnung". Hier soll die Idee der Optimierung erklärt und anhand von Hauptbedingung und Nebenbedingung modelliert und berechnet werden. Die Schüler*innen sollen das Prinzip der Methode der kleinsten Quadrate und die zugrundeliegenden Ideen erläutern und die Güte der Ergebnisse bewerten können. Mit Technologieeinsatz sollen für vorgegebene Modellfunktionen mit Hilfe der Methode der kleinsten Quadrate Funktionsgleichungen bestimmt

werden (vgl. (*11*), Online Ressource)

Die Schüler*innen der HTLs sollen in ihrem ersten Jahr Funktionen als Mittel zur Beschreibung von Zusammenhängen verstehen sowie Funktionen durch Wertetabellen und grafisch im rechtwinkligen Koordinatensystem, auch mit technischen Hilfsmitteln darstellen. Sie sollen die Gleichung einer Geraden aufstellen können, deren Parameter berechnen und interpretieren können. Im zweiten Jahr soll der Funktionsbegriff erklärt werden können, Eigenschaften von Funktionen sollen erkannt und an Beispielen veranschaulicht werden. Überdies ist die Lineare Regression ebenfalls ein Teil des Lehrplans mancher HTLs, allerdings erst in der fünften Klasse. Hier sollen Schüler*innen im Bereich Stochastik lernen, aus vorgegebenen Punkten eine passende Ausgleichsfunktion (speziell Lineare Regression) mittels Technologieeinsatz zu ermitteln und das Ergebnis zu interpretieren. Die Methode der kleinsten Quadrate findet sich in vielen HTLs ebenfalls im Lehrplan wieder. Auch Optimierungsaufgaben sind Teil der HTL-Lehrpläne. (vgl. (*12*), Online Ressource)

Auch wenn die Lineare Regression im Lehrplan der AHS keinen Platz gefunden hat, bietet sie eine gute Gelegenheit, das Verständnis für lineare Funktionen $f : \mathbb{R} \to \mathbb{R} : y = kx + d$ zu vertiefen. Besonders der Modellcharakter von Funktionen kann hier gut veranschaulicht werden und es ergibt sich die Möglichkeit, mit linearen Funktionen in einem anwendungsorientierten Bereich zu arbeiten. Eine Regressionsgerade kann dazu verwendet werden, das Verständnis für die Parameter Steigung und Achsenabschnitt zu vertiefen, beispielsweise könnte die Änderung der beiden Parameter, die bei Veränderung der Datenpunkte entsteht, thematisiert werden. Ist die Lineare Regression kein Teil des Mathematiklehrplans und soll daher auch nicht näher thematisiert und händisch gerechnet werden, kann sie dennoch mit Hilfe von Geogebra (vergleiche Anhang C.1) ganz einfach durchgeführt werden. In diesem Fall liegt das Hauptaugenmerk auf dem zeitgemäßen Technologieeinsatz und der Interpretation der Ergebnisse. Überdies könnte sie im Rahmen einer Extremwertaufgabe im Regelunterricht vorkommen (siehe Abschnitt 4.1.2).

4.1.2 Einfache lineare Regression

Bei der einfachen linearen Regression besteht jede Observation nur aus einem Feature. Dabei sind im Trainingsdatensatz m Punkte $P_j(x_j, y_j), j = 1, \ldots, m$ gegeben, wobei x_j die Features und y_j die entsprechenden Labels sind. Zeichnet man diese Punkte in ein Koordinatensystem, so erhält man eine Punktwolke. Ziel der linearen Regression ist es nun, eine Gerade zu finden, die nach der Methode der kleinsten Qaudrate möglichst gut an diese Punktwolke angepasst ist. Nun wird die Gerade $g : \hat{y} = ax + b$ wie folgt bestimmt: setzt man in die Gerade einen konkreten x-Wert x_j ein, so erhält man einen Schätzwert $\hat{y}_j = ax_j + b$. Die unbekannten Koeffizienten a

und b (Parameter oder Gewichte des ML-Modells) der Geraden werden nun so bestimmt, dass die Costfunktion $C(a, b)$ minimal wird.

$$C(a,b) = \sum_{j=1}^{m} (\hat{y}_j - y_j)^2 = \sum_{j=1}^{m} (ax_j + b - y_j)^2 \to \min. \tag{4.3}$$

Die notwendige Bedingung dafür lautet

$$\frac{\partial C}{\partial a} = 0 \quad \wedge \quad \frac{\partial C}{\partial b} = 0 \tag{4.4}$$

und liefert ein lineares Gleichungssystem in den Unbekannten a und b, dessen Lösung

$$a = \frac{\sum_{j=1}^{m} (x_j - \bar{x})(y_i - \bar{y})}{\sum_{j=1}^{m} (x_i - \bar{x})^2}, \quad b = \bar{y} - a\bar{x} \tag{4.5}$$

lautet (vgl. (*17*) S. 62).

Führt man dieses Verfahren für das Trainingsset $P_1(1,1)$, $P_2(2,4)$, $P_3(2.5,3)$, $P_4(4,3.5)$, $P_5(5,5)$, $P_6(5.5,4)$, $P_7(6,5)$ und $P_8(7,6.5)$ durch, so erhält man die Regressionsgerade $g : \hat{y} = 0.669x + 1.239$. Die folgende Abbildung visualisiert die Situation, wobei in der Grafik die Residuen $|\hat{y}_j - y_j|$ mit r_j beschriftet wurden.

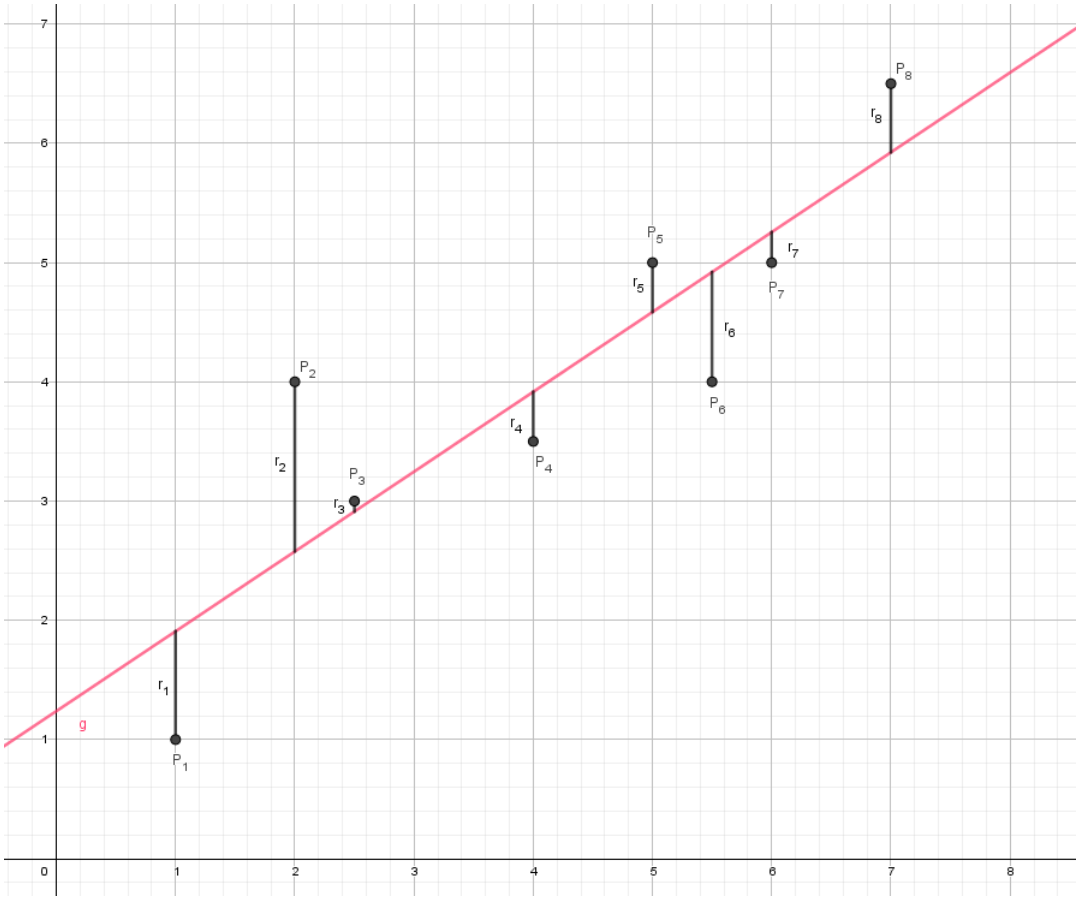


Abbildung 4.1: Einfache lineare Regression

Möglicher Einsatz im Unterricht

Obwohl die lineare Regression im AHS-Lehrplan nicht enthalten ist, kann das oben angegebene Beispiel auch im Rahmen einer Extremwertaufgabe verwendet werden. Legt man die Ausgleichsgerade durch den Ursprung ($\hat{y} = ax$), so erhält man eine Fehlerfunktion $C(a)$, von der das Minimum zu bestimmen ist.

$$C(a) = \sum_{j=1}^{8} (ax_j - y_j)^2 = (1 \cdot a - 1)^2 + (2 \cdot a - 4)^2 + \ldots + (7 \cdot a - 1)^2 \to \min$$

Für die Ausgleichsgerade $\hat{y} = ax + b$ kann auf Grund der Tatsache, dass die Regressionsgerade durch den Schwerpunkt $S(\bar{x}, \bar{y})$ der Punktwolke geht, eine Extremwertaufgabe mit Nebenbedingung modelliert werden:

HB: $C(a,b) = \sum_{j=1}^{8} (ax_j + b - y_j)^2 = (1 \cdot a + b - 1)^2 + (2 \cdot a + b - 4)^2 + \ldots + (7 \cdot a + b - 1)^2 \to \min$

NB: $\bar{y} = a \cdot \bar{x} + b$

4.1.3 Mehrdimensionale lineare Regression in der Praxis

Besteht eine Observation nicht nur aus einem Feature sondern aus n Werten, so berechnet sich $\hat{y}$ gemäß Formel 4.1. Geometrisch bestimmt man mit der Methode der kleinsten Quadrate also eine Hyperebene im $n + 1$ - dimensionalen Raum.

Im Folgenden wird mit Hilfe von Scikit-Learn eine lineare Regression mit 5 Eingangsfeatures durchgeführt und die Ergebnisse werden analysiert. Dabei werden auch Metriken vorgestellt, die zur Bestimmung der Güte des Modells dienen. Konkret werden Preise von Gebrauchtwagen (Abb. 3.2) aus den Eingangsfeatures Baujahr (*Year*), Neupreis (*Present_Price*) in 1000 Dollar, gefahrene Kilometer (*Kms_Driven*), Vorbesitzer (*Owner*) - (Werte 0,1,2,3) und Verkäufer (*Individual*) - (0...Dealer oder 1...Individual) prognostiziert.

Die praktische Umsetzung dieser linearen Regression mit Scikit-Learn findet man im Anhang (vgl. Listing B.2). Hier werden nur die Ergebnisse angegeben und interpretiert.

Eine Analyse der Features und des Labels mit Pandas ergibt folgende Übersicht. Das Feature *Seller_Type* wurde mit Hilfe von One-Hot-Encoding in das binäre Feature *Individual* konvertiert. Zunächst wird für jedes Eingangsfeature und für den Label die Anzahl der Beobachtungen (immer 301), Mittelwert, Standardabweichung, Minimum, Maximum und die Quartile angegeben.

	Year	Present_Price	Kms_Driven	Owner	Individual	Selling_Price
count	301.000000	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	7.628472	36947.205980	0.043189	0.352159	4.661296
std	2.891554	8.644115	38886.883882	0.247915	0.478439	5.082812
min	2003.000000	0.320000	500.000000	0.000000	0.000000	0.100000

25%	2012.000000	1.200000	15000.000000	0.000000	0.000000	0.900000
50%	2014.000000	6.400000	32000.000000	0.000000	0.000000	3.666666
75%	2016.000000	9.900000	48767.000000	0.000000	1.000000	6.000000
max	2018.000000	92.600000	500000.000000	3.000000	1.000000	35.000000

Zu einer weiteren Analyse der Daten ist es vorteilhaft, die Zusammenhänge zwischen den einzelnen Features und der vorherzusagenden Größe zu untersuchen. Die folgende Abbildung visualisiert dies für das Baujahr und die Anzahl der Vorbesitzer:

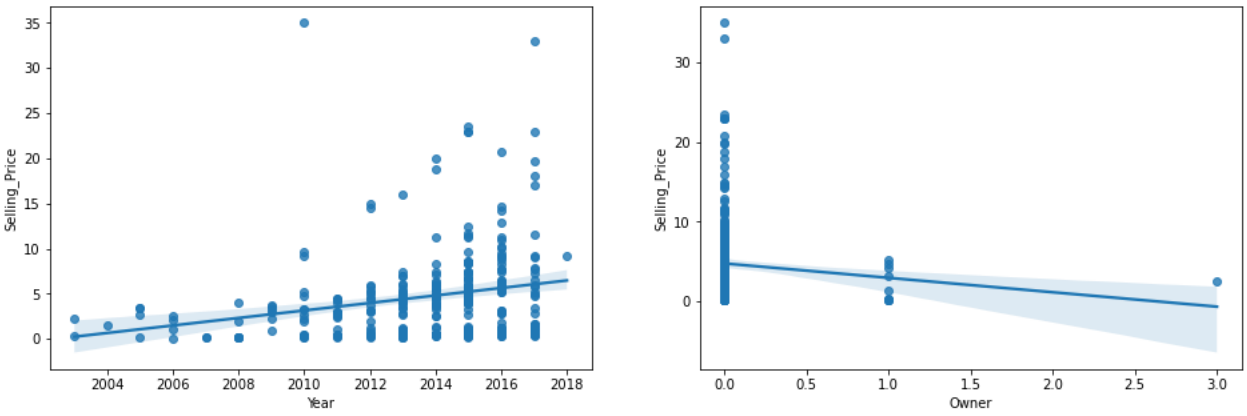


Abbildung 4.2: Regression: Zusammenhänge zwischen Features und Label

Aus den Grafiken ist zu erkennen, dass sich ein höheres Baujahr positiv und mehr Vorbesitzer negativ auf den Gebrauchtwagenpreis auswirken. Allerdings ergeben sich hier schwache bis sehr schwache Zusammenhänge, wie die berechneten Korrelationskoeffizienten zeigen. Der stärkste Zusammenhang ergibt sich zwischen Neupreis und Verkaufspreis:

```
Korrelation Year-Selling_Price          :  0.236
Korrelation Owner-Selling_Price         : -0.088
Korrelation Present_Price-Selling_Price:  0.879
```

Nach Durchführung der linearen Regression erhält man folgende Ergebnisse:

```
Koeffizienten :  [ 4.538e-01  4.934e-01 -1.869e-06 -6.844e-01 -1.148e+00]
Intercept     :  -912.4416701573135
Trainingsscore:  0.8653938775004216
Testscore     :  0.8388146354270003
```

Das Modell berechnet also für einen bestimmten Datensatz x_1 (Baujahr), x_2 (Neupreis), x_3 (Kilometerstand), x_4 (Anzahl der Vorbesitzer) und x_5 (Verkäufer) den Schätzwert $\hat{y}$ für den Verkaufspreis mit

$$\hat{y} = 0.4538x_1 + 0.4934x_2 - 1.869 \cdot 10^{-6}x_3 - 0.6844x_4 - 1.148x_5 - 912.442 \qquad (4.6)$$

Hier ist wieder abzulesen, dass sich höhere Werte von Baujahr und Neupreis positiv und höhere Werte der anderen drei Features negativ auf den Verkaufspreis auswirken. Insbesondere erzielen Privatverkäufe ($x_5 = 1$) einen geringfügig niedrigeren Verkaufspreis als Händlerverkäufe ($x_5 = 0$).

Um den Score zu ermitteln, der die Güte des ML-Verfahrens bestimmt, werden die Vorhersagen mit den tatsächlichen Ergebnissen verglichen. Diese sind zumindest für die Trainings- und Testdaten beim Supervised Learning ja bekannt. Bei Klassifikationen ist der Score in der Regel der Prozentwert der richtig vorhergesagten Klassenzugehörigkeiten, bei Regressionen wird im Allgemeinen das Bestimmtheitsmaß R^2 verwendet. (vgl. (*19*), Online Resource) Im Wesentlichen handelt es sich dabei um eine dimensionslose Maßzahl, die das Verhältnis der Varianz der Vorhersagewerte $\hat{y}_j$ und der Varianz der tatsächlichen Messwerte y_j beschreibt:

$$R^2 = \frac{\sum_{j=1}^{m}(\hat{y}_j - \bar{y})^2}{\sum_{j=1}^{m}(y_j - \bar{y})^2} = 1 - \frac{\sum_{j=1}^{m}(\hat{y}_j - y_j)^2}{\sum_{j=1}^{m}(y_j - \bar{y})^2} = 1 - \frac{\sum_{j=1}^{m} r_j^2}{\sum_{j=1}^{m}(y_j - \bar{y})^2} \tag{4.7}$$

Je näher R^2 bei 1 liegt, um so kleiner ist die Summe der Quadratfehler r_j^2 und umso besser ist die Güte des Regressionsmodells. Das Bestimmtheitsmaß kann im Gegensatz zum Korrelationskoeffizienten bei linearen Regressionen in beliebigen Dimensionen berechnet werden. Der Werte von 0.865 auf den Trainingsdaten und 0.839 auf den Testdaten in obigem Beispiel zeigt dass das Modell eine hohe Güte hat. Man könnte nun empirisch durch Hinzufügen weiter Features bzw. Weglassen verwendeter Features feststellen, ob man ein Modell mit höherer Güte findet.

Eine weitere Möglichkeit sich von der Güte des Modells zu überzeugen ist es, die Verteilung der Residuen $r_j = \hat{y}_j - y_j$ zu analysieren. Dabei ergeben sich folgende Resultate: `Mean: -1.218e-13, Std: 1.89` Hier sieht man, dass der Mittelwert praktisch bei Null liegt und die Standardabweichung 1.89 Geldeinheiten beträgt.

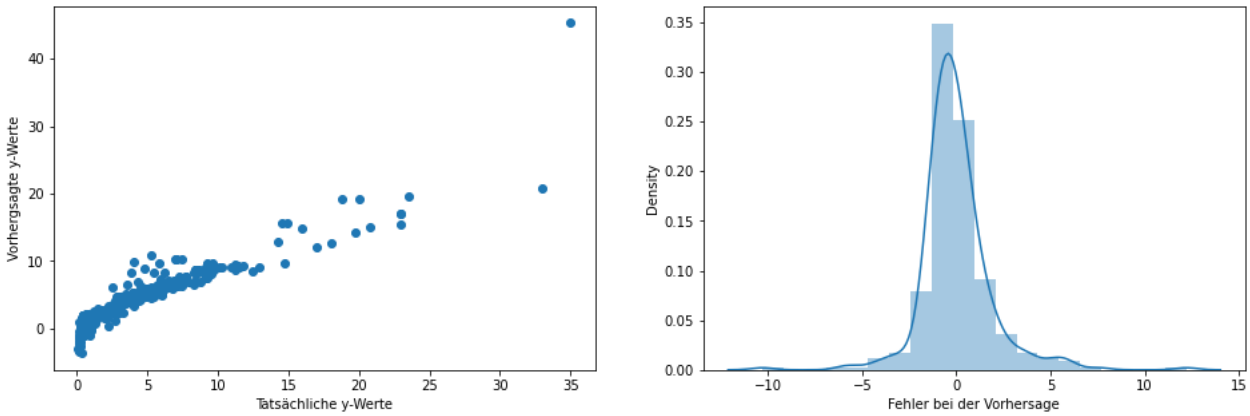


Abbildung 4.3: Lineare Regression - Grafische Analyse der Residuen

Die obige Abbildung 4.3 zeigt links ein Streudiagramm, wobei auf der waagrechten Achse die

tatsächlichen Label und auf der senkrechten Achse die Vorhersagewerte aufgetragen sind (je näher diese Punkte an der ersten Mediane liegen, umso besser ist das Modell) und rechts die Verteilung der Residuen. Der Korrelationskoeffizient zwischen echten und vorhergesagten Verkaufspreisen mit 0.928 zeigt erneut die zufriedenstellende Güte des Modells.

4.2 Logistische Regression

4.2.1 Lehrplanbezug

Im Lehrplan der AHS finden sich weder das logistische Wachstum, noch die logistische Regression. Allerdings werden in der sechsten Klasse Exponentialfunktionen behandelt. Im Rahmen der Differentialrechnung (7. Klasse) könnte beispielsweise die Sigmoid-Funktion (diese wird in Abschnitt 5.3.3 vorgestellt) als wichtige Exponentialfunktion aus der Praxis auch in der AHS besprochen und diskutiert werden. (vgl. (*10*), Online Ressource) Der Lehrplan der Handelsakademien sieht beim Thema Funktionale Zusammenhänge im dritten Lernjahr vor, Exponentialfunktionen zu beschreiben, sie graphisch darzustellen, sie als Modelle für Zu- und Abnahmeprozesse zu interpretieren und Berechnungen damit durchzuführen, verschiedene Modelle zu vergleichen und ihre Angemessenheit zu bewerten. (vgl. (*11*), Online Ressource) An den HTLs werden Funktionen, auch Exponentialfunktionen, als Mittel zur Beschreibung von Zusammenhängen vermittelt. Sie sollen graphisch und mit technischen Hilfsmitteln dargestellt werden. Im Rahmen der Differentialrechnung werden die Ableitungen der elementaren Funktionen sowie zusammengesetzter Funktionen bestimmt. (vgl. (*12*), Online Ressource)

4.2.2 Intuition und Grundidee

Die lineare Regression erlaubt es, einen kontinuierlichen Wert vorherzusagen. Für Klassifikationsprobleme ist dieses Verfahren nicht geeignet. Die Problematik wird zunächst an Hand eines einfachen Beispiels erklärt. Ausgehend vom Alter (unabhängige Variable) soll vorausgesagt werden, ob ein potentieller Käufer ein bestimmtes Produkt erwirbt (Klasse 1) oder nicht (Klasse 0). Die folgende Abbildung visualisiert Testdaten (30 Observationen) und das Ergebnis einer auf diesen Daten durchgeführten linearen Regression:
Wie die Grafik 4.4 zeigt ist die lineare Regression schlecht geeignet dafür, Klassenzugehörigkeiten 0 bzw. 1 vorherzusagen. Verkettet man das Ergebnis $\hat{z} = 0.042x - 1.26$ der linearen Regression mit der Sigmoidfunktion (vgl. Abschnitt 5.3.3)

$$\hat{y} = \frac{1}{1 + e^{-\hat{z}}} = \frac{1}{1 + e^{-0.042x+1.26}} \tag{4.8}$$

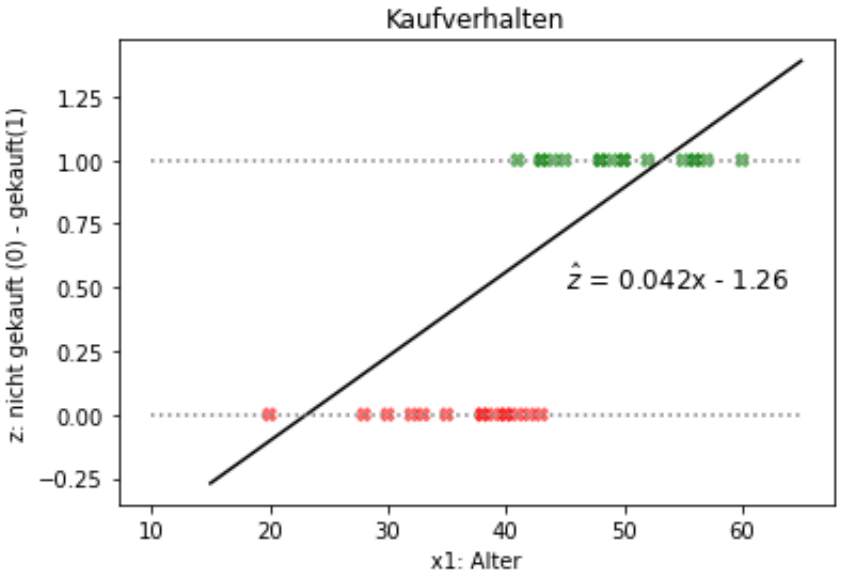


Abbildung 4.4: Logistische Regression - Problemstellung

so erhält man das folgende Ergebnis: (vgl. (*17*) S 133ff)

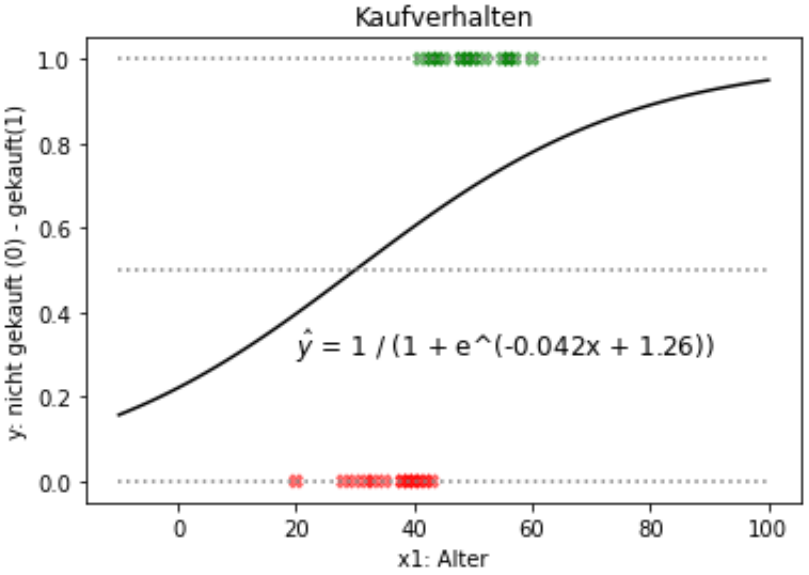


Abbildung 4.5: Logistische Regression - Sigmoidfunktion

Die entsprechenden Schätzwerte $\hat{y} \in (0, 1)$ können nun als Wahrscheinlichkeiten für die Klassenzugehörigkeit mit folgender Entscheidungsregel interpretiert werden. Ist $\hat{y}(x) \geq 0.5$, so wird x zur Klasse 1 gehörig prognostiziert, andernfalls zur Klasse 0 gehörig.

Die folgende Tabelle gibt für einige ausgewählte Trainingswerte x_i die tatsächliche Klassenzugehörigkeit y_i, die prognostizierte Wahrscheinlichkeit$\hat{y}_i$ der Zugehörigkeit zur Klasse 1 und die prognostizierte Klassenzugehörigkeit an:

Als Metrik für die Güte eines Klassifikationsmodells wird im Allgemeinen der Prozentwert der

x_i	20	30	40	50	60
y_i	0	0	0	1	1
$\hat{y}_i$	0.393	0.496	0.599	0.694	0.775
Prog. Klassenzugehörigkeit	0	0	1	1	1

Tabelle 4.1: Ergebnisse Logistische Regression

richtigen Klassifikationen gewählt.

Das hier vorgestellte Verfahren kann analog auch für Datensätze mit mehreren Eingangsfeatures verwendet werden, an Stelle der einfachen linearen Regression tritt hier die mehrdimensionale lineare Regression.

4.2.3 Praktische Umsetzung an Hand des Titanic Disaster Datasets

An Hand des in Abschnitt 3.2 vorgestellten Titanic Disaster Datasets wird die Logistischen Regression mit Hilfe von Scikit - Learn konkret umgesetzt. Dabei wird von bereinigten Daten ausgegangen:

Der Pythoncode (Listing B.3) liest die Daten ein, druckt die ersten 2 Observationen des Datasets und extrahiert die Matrix $\mathbf{X}$ der Features (jede Zeile eine Observation) und den Vektor $\mathbf{y}$ der Labels. Anschließend wird ein Train-Test-Split mit 20% Testdaten durchgeführt. Danach wird das Modell erzeugt und trainiert. Anschließend werden sowohl für die Trainings- als auch für die Testdaten die Scores berechnet und ausgegeben. Für die Trainingsdaten werden auch die vorhergesagten Wahrscheinlichkeiten und Klassenzugehörigkeiten berechnet, für die ersten 10 Testdatensätze ausgegeben und mit den tatsächlichen Labels verglichen. Abschließend wird die sogenannte Confusionmatrix berechnet und ausgegeben. Eine genaue Erklärung zu dieser Matrix folgt im Anschluss.

Der Programmlauf liefert folgende Ergebnisse:

```
   PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare  male  Q  S
0            1         0       3  22.0      1      0   7.2500     1  0  1
1            2         1       1  38.0      1      0  71.2833     0  0  0


Score auf den Trainingsdaten:  0.807
Score auf den Testdaten:       0.792


[ [0.90  0.10] [0.04  0.96] [0.24  0.76] [0.75  0.25] [0.03  0.97]
  [0.90  0.10] [0.70  0.30] [0.87  0.13] [0.08  0.92] [0.03 0.97]]
Vorhergesagte Klassenzugeörigkeit:  [0 1 1 0 1 0 0 0 1 1]
Tatsächliche Klassenzugehörigkeit:  [0 1 1 0 1 1 0 0 1 1]
```

Das Modell liefert auf den Trainingsdaten also 80.7% richtige Vorhersagen und auf den Testdaten 79.2%. Für den ersten Testdatensatz hat das Modell mit einer Wahrscheinlichkeit von 90% richtigerweise die Klassenzugehörigkeit 0 (nicht überlebt) vorhergesagt. Es gibt im Bereich der ersten 10 Testdatensätze nur eine falsche Vorhersage beim 6.Datensatz (diese fällt aber mit 90% für die falsche Klasse 0 sehr deutlich aus).

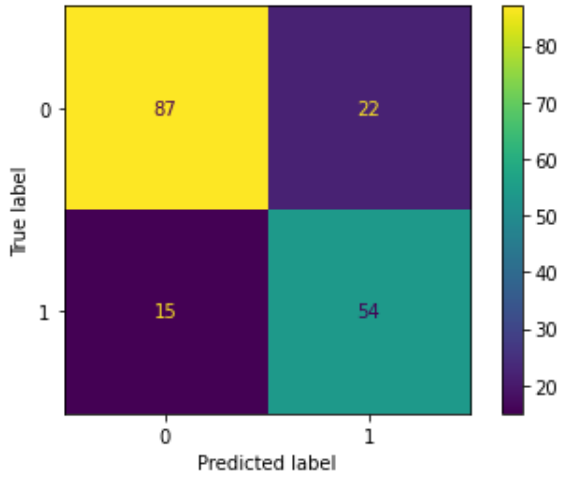


Abbildung 4.6: Logistische Regression - Titanic Disaster Dataset - Confusionmatrix

Letztendlich wird die Confusionmatrix (vgl. (*17*), S 148) für die Testdaten visualiert (vgl. Abbildung 4.6). Diese gibt einen vollständigen Überblick über die richtigen/falschen Vorhersagen und kann wie folgt interpretiert werden:

- Das Testset besteht aus insgesamt 87 + 22 + 15 + 54 = 178 Datensätzen
- Davon wurden 87 richtig mit 0 klassifiziert (true negativ)
- Davon wurden 22 falsch mit 1 klassifiziert (false positiv)
- Davon wurden 15 falsch mit 0 klassifiziert (false negativ)
- Davon wurden 54 richtig mit 1 klassifiziert (true positiv)

4.3 KNN (K-Nearest-Neighbors)

4.3.1 Lehrplanbezug

Bereits in der fünften Klasse der AHS lernen die Schüler*innen beim Thema Vektoren und analytische Geometrie im $\mathbb{R}^2$, Abstände zwischen zwei Punkten oder einem Punkt und einer Geraden zu ermitteln. In der sechsten Klasse werden die aus der fünften Klasse bekannten Methoden auf

den dreidimensionalen Fall übertragen. Darüber hinaus werden Vektoren und deren Rechenoperationen im n-dimensionalen Raum gelernt, wobei Schüler*innen diese in Anwendungskontexten interpretieren und verständig erklären können sollen. (vgl. (*10*), Online Ressource)
In den Handelsakademien hat die Vektorrechnung keinen Platz im Lehrplan gefunden. Da es bei KNN im Wesentlichen um das Berechnen von Abständen geht, könnte dies auch mit dem Satz von Pythagoras realisiert werden.
In den HTLs lernen die Schüler*innen im ersten Jahr neben weiteren Methoden aus der Vektorrechnung auch den Betrag von Vektoren kennen. Im zweiten Jahr werden die zuvor im $\mathbb{R}^2$ behandelten Themen nun auf den $\mathbb{R}^3$ ausgeweitet. (vgl. (*12*), Online Ressource)

4.3.2 Arbeitsweise

Bei diesem Klassifikationsalgorithmus handelt es sich um einen sehr einfach zu verstehenden Algorithmus. Da im Wesentlichen euklidische Abstände zwischen den Datenpunkten berechnet werden, ist es unbedingt notwendig, die Daten vor Anwendung des Algorithmus zu skalieren (vgl. Abschnitt 3.6).
Der Algorithmus im Detail: (vgl. (*20*) S 55ff)

1. Zunächst wird die Anzahl k der betrachteten Nachbarn gewählt.
2. Nun werden für den zu klassifizierenden Datenpunkt X jene k Nachbarn des Trainingsdatensatzes bestimmt, die ihm am nächsten liegen.
3. Für diese k Nachbarn wird die Anzahl der Zugehörigkeit zu jeder Klasse bestimmt.
4. Der neue Datenpunkt X wird jener Klasse zugeordnet, zu der die meisten Nachbarn gehören.

Das Ergebnis kann stark von k abhängen. Um immer eine eindeutige Entscheidung zu erhalten ist es vorteilhaft, wenn die Anzahl der Klassen und die Anzahl k der gewählten Nachbarn teilerfremd sind. Die folgende Abbildung 4.7 visualisiert den Algorithmus und zeigt mögliche verschiedene Ergebnisse in Abhängigkeit von k.
Klassifiziert man den grauen Datenpunkt mit $k = 3$, so wird er der durch die grünen Datenpunkte repräsentierten Klasse zugeordnet, bei $k = 5$ zu jener der roten Datenpunkte.

Möglicher Einsatz im Unterricht

Bereits im Rahmen Grundlagen der Vektorrechnung könnte man folgende Aufgabe stellen:
Gegeben ist ein Trainingsset für KNN bestehend aus 6 Punkten im dreidimensonalen Raum (3 Features). Die Punkte sind dabei mit rot/grün gelabelt. $P_1(0, 3, 0)$, $P_2(2, 0, 0)$, $P3(0, 1, 3)$,

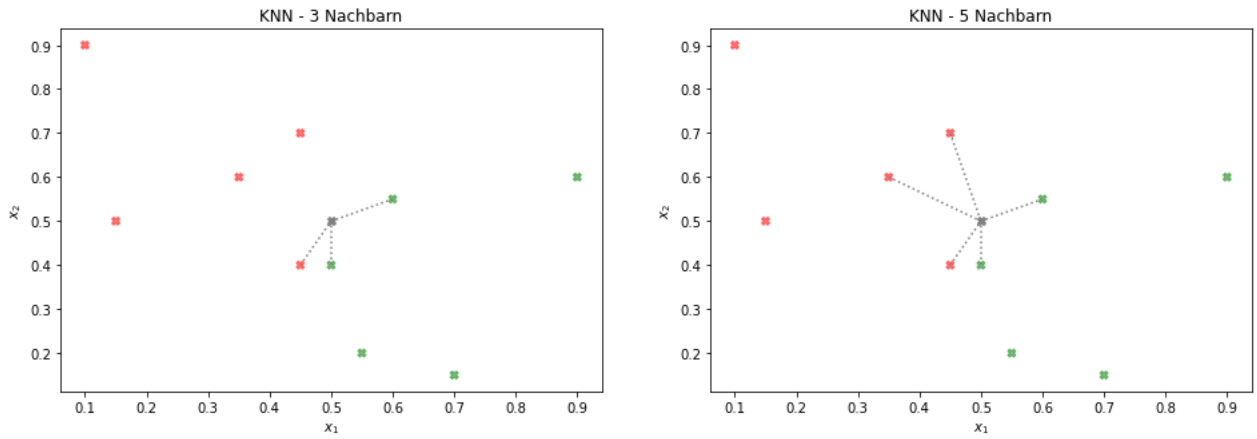


Abbildung 4.7: KNN - Ergebnis abhängig von k

$P_4(0,1,2)$, $P_5(-1,0,1)$, $P_6(1,1,1)$. Dabei gehören die Punkte P_4 und P_5 zur Klasse "grün" und die anderen Punkte zur Klasse "rot".

Aufgabe:

Bestimme für den Testpunkt $X(0,0,0)$ die euklidischen Abstände zu den Trainingspunkten und ermittle für X die Klassenzugehörigkeit für $k = 1$ bzw. $k = 3$.

4.3.3 Praktische Umsetzung an Hand des Titanic Disaster Datasets

Der Quellcode (Listing B.4) trainiert das Titanic Disaster Dataset mit KNN mit 5 Nachbarn. Dabei erhält man folgende Ergebnisse:

```
Score auf den Trainingsdaten:  0.8720112517580872
Score auf den Testdaten:       0.7696629213483146
[ [1.0  0.0] [0.0  1.0] [0.4  0.6] [0.6  0.4] [0.0  1.0]
  [0.8  0.2] [1.0  0.0] [1.0  0.0] [0.0  1.0] [0.0  1.0 ] ]
Vorhergesagte Klassenzugeörigkeit:  [0 1 1 0 1 0 0 0 1 1]
Tatsächliche Klassenzugehörigkeit:  [0 1 1 0 1 1 0 0 1 1]
```

Beim Vergleich von Trainings- und Testscore sieht man, dass hier das Modell besser an die Trainingsdaten angepasst ist als bei der Logistischen Regression und der Score auf den Testdaten etwas schlechter ausfällt.

Die Wahrscheinlichkeiten geben an, dass beim ersten Testdatensatz alle 5 Nachbarn (100%) zur Klasse 0 gehören und beim 3.Testdatensatz 2 Nachbarn (40%) zur Klasse 0 und 3 Nachbarn (60%) zur Klasse 1. Wieder wurde in diesem Bereich nur der 6.Testdatensatz falsch klassifiziert. Eine vollständige Analyse des Tests zeigt wieder die Confusionmatrix:

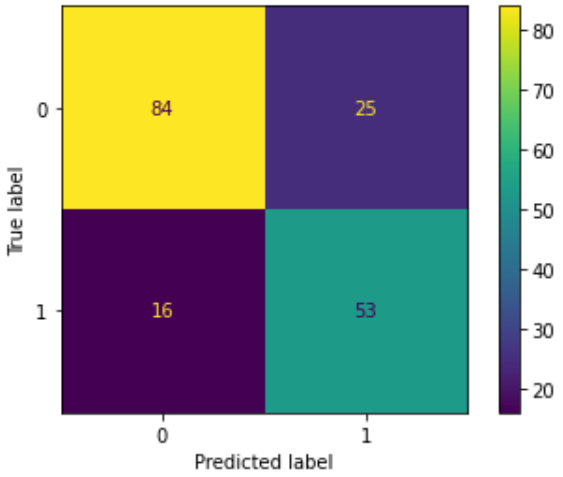


Abbildung 4.8: KNN - Titanic Disaster Dataset - Confusionmatrix

Die Grafik 4.9, visualisiert die erzielten Scores am Testset für die k-Werte von 1 bis 49. Der beste Score mit 82% ergibt sich bei 14 Nachbarn.

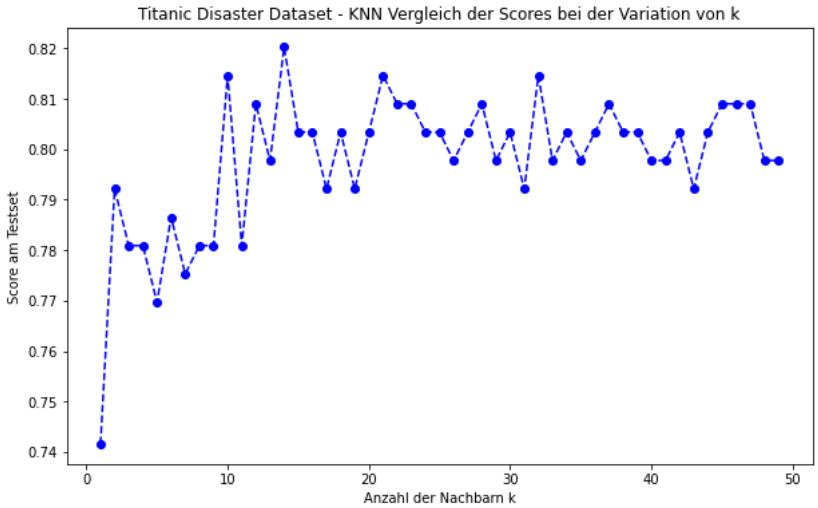


Abbildung 4.9: KNN - Titanic Disaster Dataset - Scores bei variablem k

4.4 Naive Bayes

Bei Naive Bayes handelt es sich um eine Anwendung der Formel von Bayes in der Praxis. Der Bayes-Klassifikator ist ein aus dem Satz von Bayes hergeleiteter Klassifikator, der jedes Objekt der Klasse zuordnet, der es mit größter Wahrscheinlichkeit angehört. Er kann vorteilhaft bei kleinen Datasets mit vielen Features eingesetzt werden. Bei Naive Bayes wird (in naiver Weise) davon ausgegangen, dass die einzelnen Features der zu klassifizierenden Observationen voneinander unabhängig sind, was jedoch in der Regel selten zutrifft. Trotzdem erzielt Naive

Bayes für gewöhnlich gute Ergebnisse, wenn die Annahme nur annähernd erfüllt ist. (vgl. (*20*) S 83ff)

4.4.1 Lehrplanbezug

Im Mathematiklehrplan der AHS-Oberstufe ist der Satz von Bayes bereits in der sechsten Klasse als Unterpunkt des Themas "Beschreibende Statistik; Wahrscheinlichkeit" zu finden. Schüler*innen sollen bedingte Wahrscheinlichkeiten und (stochastische) Unabhängigkeit von Ereignissen kennen sowie den Satz von Bayes kennen und anwenden können. (vgl. Lehrplan AHS Oberstufe 2021)

Laut den Lehrplänen der Handelsakademien müssen bedingte Wahrscheinlichkeit und der Satz von Bayes nicht im Unterricht vorkommen. (vgl. Lehrplan HAK und HAS 2021)

In den Mathematiklehrplänen der HTLs sollen Schüler:innen im vierten Lernjahr den "Additions- und Multiplikationssatz für einander ausschließende bzw. unabhängige Ereignisse" sowie die "Bedingte Wahrscheinlichkeit" lernen. Ebenfalls im vierten Lernjahr sollen laut Lehrplan "aus Stichprobenwerten Häufigkeitsverteilungen tabellarisch und grafisch dargestellt werden". (vgl. Lehrplan HTL 2021)

Die Arbeitsweise von Naive Bayes kann jedenfalls in der AHS und der HTL intuitiv als Aufgabe zur bedingten Wahrscheinlichkeit und zum Satz von Bayes erklärt werden (vgl. Abschnitt 4.4.2). In der Praxis werden die notwendigen Wahrscheinlichkeiten nicht wie im intuitiven Beispiel durch Abzählen, sondern über Häufigkeitsverteilungen ermittelt. Auch dies wird der Vollständigkeit halber beschrieben, geht aber inhaltlich zumindest über den AHS-Lehrplan hinaus. In der AHS kann dieser Teil im Zuge eines eventuell sogar fächerübergreifenden Projekts, eines Wahlfaches oder einer vorwissenschaftlichen Arbeit Anwendung im Unterricht finden.

4.4.2 Intuition im Unterricht als Anwendung des Theorems von Bayes

Die (vereinfachte) Erklärung für die Arbeitsweise des ML-Algorithmus Naive Bayes wird für den Unterrichtsgebrauch durch folgende konkreten Aufgabenstellung vorgestellt:

Anhand der beiden Eingangsfeatures *Alter* und *Einkommen* von potentiellen Konsument*innen soll klassifiziert werden, ob eine bestimmte Person ein Produkt kaufen wird oder nicht. Die Abbildung 4.10 zeigt das Kaufverhalten für 30 Beobachtungen (dabei bedeutet grün "gekauft" und rot "nicht gekauft").

Für einen bestimmten zu klassifizierenden Datenpunkt $X(x_1, x_2) = X(40, 2200)$ (Alter = 40, Monatseinkommen = 2200) gilt für die zu klassifizierende Eigenschaft A (nicht gekauft) gemäß

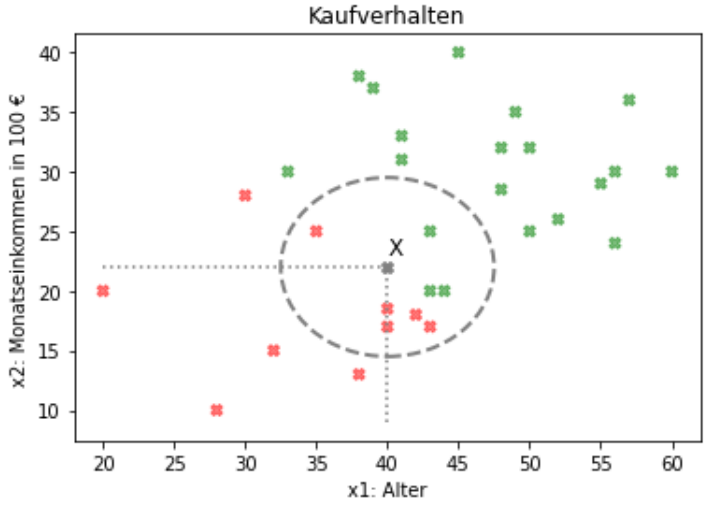


Abbildung 4.10: Naive Bayes Intuition

dem Theorem von Bayes (vgl. (*20*) S 51) folgender Zusammenhang:

$$P(A|X) = \frac{P(A) \cdot P(X|A)}{P(X)} \tag{4.9}$$

Analog gilt für die zu klassifizierende Eigenschaft B (gekauft):

$$P(B|X) = \frac{P(B) \cdot P(X|B)}{P(X)} \tag{4.10}$$

Wir berechnen also mit der Formel für $P(B|X)$ (P für B unter der Bedingung von X) die Wahrscheinlichkeit dafür, dass eine Person mit einem gewissen Alter und einem gewissen Monatseinkommen ein bestimmtes Produkt kauft. Analog ergibt die Formel für $P(A|X)$ (P für A unter der Bedingung von X) die Wahrscheinlichkeit, dass eine Person mit einem gewissen Alter und einem gewissen Monatseinkommen ein bestimmtes Produkt nicht kauft. Um diese Wahrscheinlichkeiten zu berechnen, wird um den neuen zu klassifizierenden Datenpunkt X ein Kreis mit geeignetem Radius (hier 15 Einheiten) gelegt. Wir sehen uns damit die Datenpunkte X in der Nähe des gesuchten Datenpunktes an, die innerhalb des Kreises liegen und zu denen deshalb in etwa ähnliche Einkommen und Alter gehören.

Nun lassen sich die Wahrscheinlichkeiten durch Abzählen mit Hilfe von relativen Häufigkeiten annähern. Bei der praktischen Umsetzung von Naive Bayes werden die Wahrscheinlichkeiten $P(A|X)$ und $P(B|X)$ nicht durch Abzählen, sondern über Häufigkeitsverteilungen berechnet. Siehe dazu Abschnitt 4.4.3.

Wie Abbildung 4.10 zu entnehmen ist, existieren 10 rote (A - nicht gekauft) und 20 grüne (B - gekauft) Datenpunkte. In dem betrachteten Kreis liegen insgesamt 8 Beobachtungen, davon 5 mit der Eigenschaft A und 3 mit der Eigenschaft B. Damit gilt:

$$P(A) = \frac{10}{30} \qquad P(X) = \frac{8}{30} \qquad P(X|A) = \frac{5}{10} \tag{4.11}$$

Nun lässt sich die gesuchte Wahrscheinlichkeit $P(A|X)$ mit Hilfe des Satzes von Bayes bestimmen:

$$P(A|X) = \frac{P(A) \cdot P(X|A)}{P(X)} = \frac{\frac{10}{30} \cdot \frac{5}{10}}{\frac{8}{30}} = \frac{5}{8} = 62.5\% \tag{4.12}$$

Analog gilt:

$$P(B) = \frac{20}{30} \qquad P(X) = \frac{8}{30} \qquad P(X|B) = \frac{3}{20} \tag{4.13}$$

$$P(B|X) = \frac{P(B) \cdot P(X|B)}{P(X)} = \frac{\frac{20}{30} \cdot \frac{3}{20}}{\frac{8}{30}} = \frac{3}{8} = 37.5\% \tag{4.14}$$

Also wird der neue Datenpunkt X wegen

$$P(A|X) > P(B|X)$$

als zur Klasse A (nicht gekauft) gehörig klassifiziert.
Diese Intuition ist eine schöne Anwendung für bedingte Wahrscheinlichkeiten und den Satz von Bayes und kann im Rahmen des Mathematikunterrichts als Beispiel verwendet werden.

4.4.3 Durchführung von Naive Bayes in der Praxis

Das im Abschnitt 4.4.2 vorgestellte Verfahren dient vor allem der Intuition und dem Verständnis für Schülerinnen und Schüler und wird in der Praxis so nicht durchgeführt. Zur Bestimmung der zum Vergleich notwendigen Wahrscheinlichkeiten $P(A|X)$ und $P(B|X)$ ist eine andere Vorgangsweise zielführend.
Da es bei den Wahrscheinlichkeiten $P(A|X)$ und $P(B|X)$ nur auf das Verhältnis ankommt, kann mit dem gemeinsamen Nenner $P(X)$ (vgl. 4.9 und 4.10) multipliziert werden und man erhält:

$$P(A|X) : P(B|X) = (P(A) \cdot P(X|A)) : (P(B) \cdot P(X|B)) \tag{4.15}$$

Da die Wahrscheinlichkeiten $P(A)$ und $P(B)$ durch die relativen Häufigkeiten für das Auftreten von A und B im Trainingsdataset angenähert werden können, müssen also nur noch die bedingten Wahrscheinlichkeiten $P(X|A)$ und $P(X|B)$ bestimmt werden. Konkret bedeutet dies für den zu klassifizierenden Punkt $X(x_1, x_2)$:

$$P(X|A) = P(X_1 = x_1 \wedge X_2 = x_2|A) \quad P(X|B) = P(X_1 = x_1 \wedge X_2 = x_2|B) \tag{4.16}$$

Setzt man naiv und in der Regel unrealistisch voraus[1], dass die Features X_1 und X_2 voneinander unabhängig sind, kann der vereinfachte Multiplikationssatz angewendet werden:

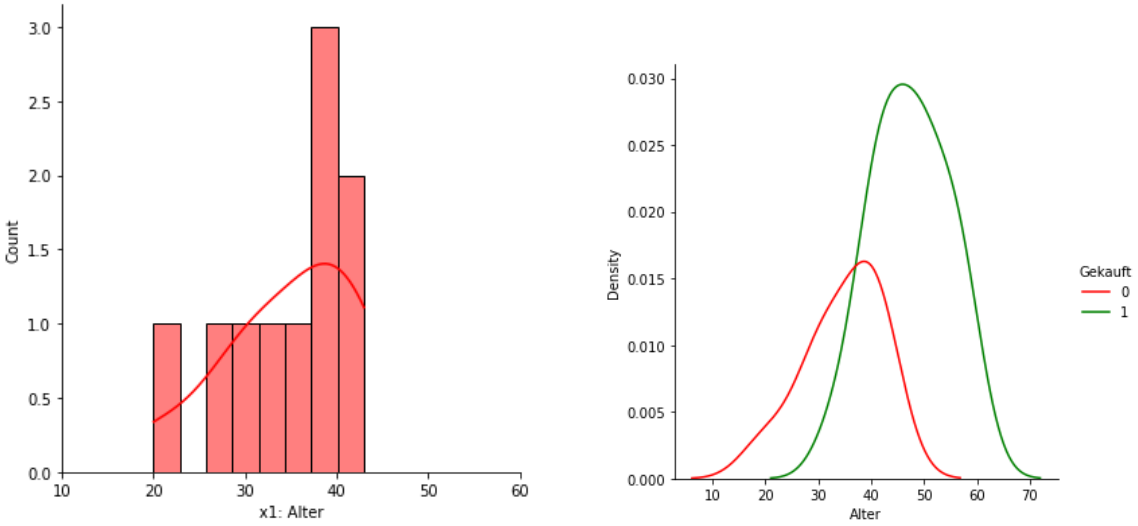
$$P(X_1 = x_1 \wedge X_2 = x_2|A) = P(X_1 = x_1|A) \cdot P(X_2 = x_2|A) \tag{4.17}$$

[1] Diese Voraussetzung ist für den Algorithmus namensgebend

Also gilt unter der Voraussetzung dieser Unabhängigkeit:

$$P(A) \cdot P(X|A) = P(A) \cdot P(X_1 = x_1|A) \cdot P(X_2 = x_2|A) \tag{4.18}$$

Diese Wahrscheinlichkeiten $P(X_1 = x_1|A)$ und $P(X_2 = x_2|A)$ können in der Praxis aus den Häufigkeitsverteilungen für X_1 und X_2, bei denen man sich auf die Daten für A bzw. B beschränkt, leicht ermittelt werden. Die Grundidee wird im Folgenden vorgestellt.
Die Abbildung 4.11 zeigt die Häufigkeitsverteilungen für das Alter X_1, wobei die in Abbildung 4.10 visualisierten Datenpunkte verwendet werden.



(a) (b)

Abbildung 4.11: Verteilung des Alters

Die linke Grafik zeigt die Häufigkeitsverteilung für das Alter in Gestalt eines Histogramms, wobei nur die Klasse A (nicht gekauft) verwendet wurde. Da das Histogramm zum Schätzen der Wahrscheinlichkeit $P(A|X_1 = x_1)$ ungeeignet ist (unstetig, kann Lücken haben) wird es durch eine stetige Dichtefunktion ersetzt. Ein Standardverfahren dafür ist der sogenannte *Kernel Density Estimator (KDE)*. (vgl (*21*), Online Resource) Dabei wird über jeden Datenpunkt ein sogenannter *Kern* gelegt, der eine geeignete Dichtefunktion darstellt. Der Kerndichteschätzer ergibt sich als Summe aller Kerne. In der rechten Teilabbildung von Grafik 4.11 werden die Kerndichteschätzer für das Alter bezogen auf die beiden Klassen A und B dargestellt.
Die obige Abbildung 4.12 zeigt einen Scatterplot der Daten mit den entsprechenden, durch KDE geschätzten, klassenabhängigen Wahrscheinlichkeitsverteilungen. Aus diesen Verteilungen lassen sich nun die benötigen bedingten Wahrscheinlichkeiten $P(X_1 = x_1|A)$ und $P(X_2 = x_2|A)$ bestimmen.
Das für zwei Eingangsfeatures vorgestellte Verfahren lässt sich leicht auf n Features und beliebig

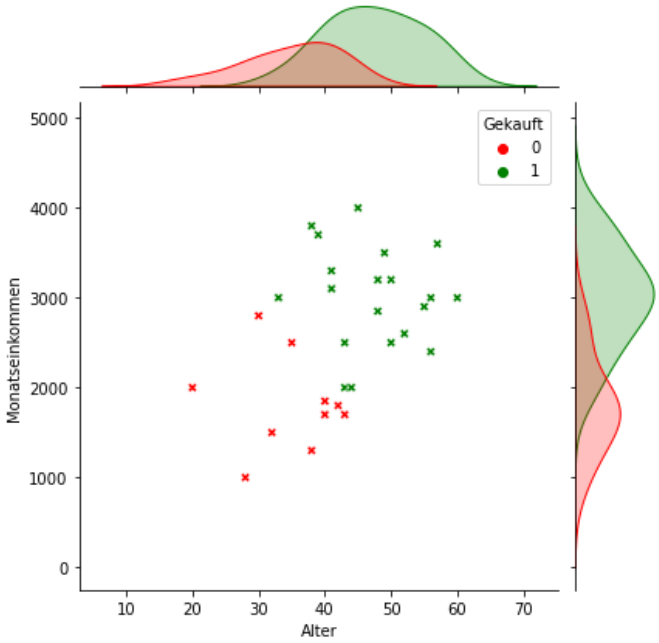


Abbildung 4.12: Scatterplot mit Kernel Density Estimator

viele Klassen verallgemeinern. Die Wahrscheinlichkeit, dass ein Datenpunkt $X(x_1, \ldots, x_n)$ zur Klasse K_j gehört, ergibt sich mit

$$P(K_j|X) = P(K_j) \cdot P(X|K_j) = P(K_j) \cdot \prod_{i=1}^{n} P(X_i = x_i|K_j) \tag{4.19}$$

Zur Klassifikation eines Datenpunktes müssen bei n Features also nur $n + 1$ Wahrscheinlichkeiten miteinander multipliziert werden. Dies macht den Algorithmus speziell bei vielen Features sehr effizient. Aus diesem Grund wird er bevorzugt bei Textklassifikation [2] eingesetzt. Typische Anwendungen sind Spamfilter oder Stimmungsanalysen in Texten.

Scikit-Learn stellt mehrere Modelle für Naive Bayes zur Verfügung. (vgl. (*22*), Online Resource) Die beiden wichtigsten sind:

1. **GaussianNB**

   Dabei werden für die Abbildung 4.12 visualisierten Dichtefunktionen (also zur Bestimmung der Wahrscheinlichkeiten für die einzelnen Features) Normalverteilungen verwendet. Diese Version ist dann vorteilhaft einzusetzen, wenn die einzelnen Features (annähernd) normalverteilt sind.

2. **MultinomialNB**

   Hier werden zur Bestimmung der Wahrscheinlichkeiten der einzelnen Features Multinomial-

[2] NLP - Natural Language Processing

verteilungen[3] eingesetzt. Dieses Modell kommt vorteilhaft dann zur Anwendung, wenn die einzelnen Features Zählergebnisse widerspiegeln. Haupteinsatz ist die Textklassifikation (die Features repräsentiern Worthäufigkeiten im Text).

4.4.4 Praktische Umsetzung an Hand des Titanic Disaster Datasets

Der Quellcode B.5 trainiert das Titanic Disaster Dataset mit GaussianNaiveBayes. Dabei erhält man folgende Ergebnisse:

```
Score auf den Trainingsdaten:  0.7960618846694796
Score auf den Testdaten:       0.7752808988764045
[ [0.950 0.050] [0.001 0.999] [0.131 0.869] [0.899 0.101] [0.012 0.988]
  [0.949 0.051] [0.895 0.105] [0.946 0.054] [0.001 0.999] [0.001 0.999] ]
Vorhergesagte Klassenzugeörigkeit:  [0 1 1 0 1 0 0 0 1 1]
Tatsächliche Klassenzugehörigkeit:  [0 1 1 0 1 1 0 0 1 1]
```

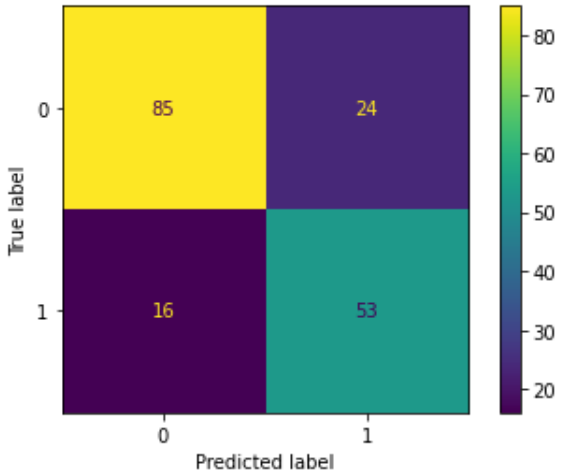


Abbildung 4.13: Gaussian Naive Bayes - Titanic Disaster Dataset - Confusionmatrix

[3] Verallgemeinerung der Binomialverteilung auf mehr als 2 Merkmale

Kapitel 5

Neuronale Netze

5.1 Grundidee und Architektur

Die Grundidee neuronaler Netze ist es, die Vorgänge im menschlichen Gehirn mathematisch zu modellieren. (vgl. (*2*) S 286f) Der Grundbaustein dabei ist das Neuron. Das biologische Neuron ist eine Zelle, die Signale in Form von elektrischen und chemischen Impulsen aufnimmt, diese verarbeitet und weiterleitet. Dabei unterstützen die Dendriten den Empfang dieser Impulse und das Axon (bestehend aus Schwan' schen Zellen) leitet elektrische Impulse von einem Neuron weg zum nächsten. Man geht heute davon aus, dass das menschliche Gehirn etwa 86 Milliarden Neuronen besitzt. (vgl (*23*), Online Ressource) Die Verbindungen zwischen den Neuronen werden Synapsen genannt.

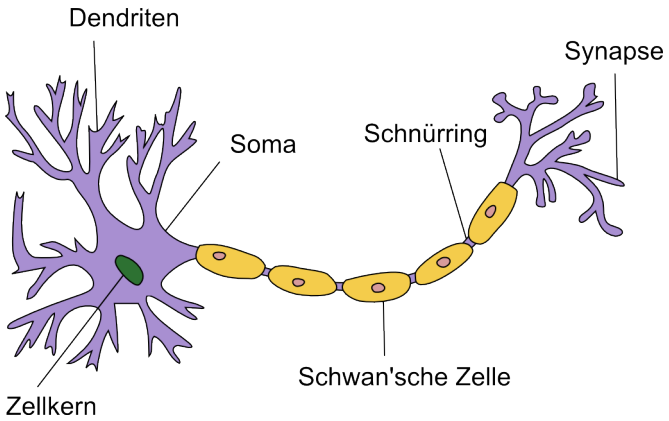


Abbildung 5.1: Biologisches Neuron (Quelle: https://abitur-wissen.org/)

Ein künstliches neuronales Netz (Artifical Neuronal Network - ANN) besteht aus künstlichen Neuronen, die in Schichten (Layer) angeordnet sind. Die Arbeitsweise eines künstlichen Neurons wird in Abschnitt 5.2 beschrieben. Zwischen dem Input Layer (Eingabeschicht) und dem Output Layer (Ausgabeschicht) werden beliebig viele Hidden Layer (Zwischenschichten) angeordnet.

Die Anzahl der Neuronen in den beiden Randschichten ist von der konkreten Problemstellung abhängig, die Anzahl der Neuronen in den Hidden Layern kann theoretisch frei gewählt werden. In dieser Arbeit werden nur neuronale Netze mit folgenden Eigenschaften behandelt:

1. **Feedforward - Netze**

   In einem Feedforward-Netz laufen die Signale immer in der Richtung vom Input Layer zum Output Layer, es gibt also keine Rückkopplungen. Netze mit solchen Rückkopplungen heißen Recurrent Neuronal Networks (RNN).

2. **Dense - Layer**

   Alle Layer sind sogenannte Dense-Layer, d.h. alle Neuronen eines Layers sind mit allen Neuronen des nächsten Layers durch Synapsen verbunden.

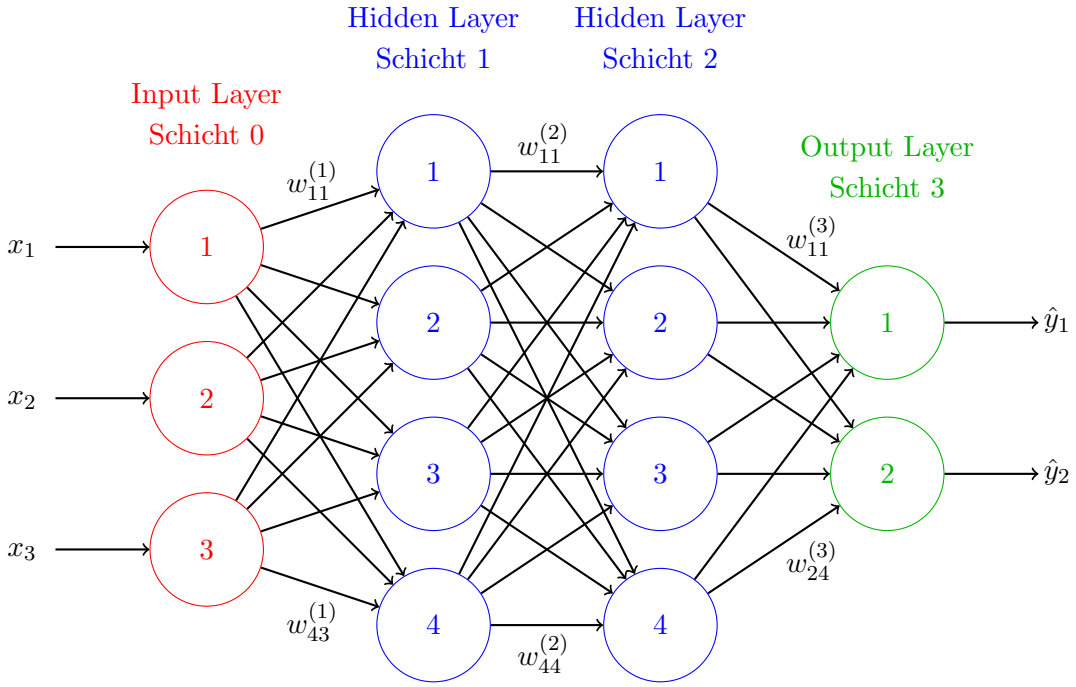


Abbildung 5.2: ANN - Achitektur

Die Abbildung 5.2 zeigt ein ANN mit zwei Hidden Layer mit je 4 Neuronen, das aus jedem Eingabevektor $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ einen Ausgabevektor $\hat{\mathbf{y}} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \end{pmatrix}$ berechnet.

Die genaue Arbeitsweise eines solchen Netzes wird im Abschnitt 5.6 beschrieben. Kurz zusammengefasst wird jeder Synapse zunächst ein zufälliges Gewicht $w_{kl}^{(s)}$ zugewiesen (dabei bezeichnet s die Nummer der Zielschicht, k die Nummer des das Signal empfangenden Neurons in der Ziel-

schicht und l die Nummer des das Signal feuernden Neurons in der vorherigen Schicht)[1] und der Ausgabevektor $\hat{\mathbf{y}}$ berechnet sich aus dem Eingabevektor $\mathbf{x}$ und allen zugewiesenen Gewichten. Nun werden im Training die Gewichte iterativ so optimiert, dass der berechnete Ausgabevektor möglichst gut mit dem erwarteten Output $\mathbf{y}$ übereinstimmt (Supervised Learning).

5.2 Das Neuron

Jedes Neuron (k-tes Neuron in der Schicht s) berechnet aus seinen n Eingangssignalen $x_1, x_2, \ldots, x_n$ und den entsprechenden Gewichten $w_{ki}^{(s)},\ i = 1; \ldots, n$ auf folgende Weise das Ausgangssignal $\hat{y}$:

1. Im ersten Schritt wird der Gewichtsvektor $\mathbf{w}$ mit dem Vektor $\mathbf{x}$ der Eingangssignale skalar multipliziert:

$$z = \mathbf{w} \cdot \mathbf{x} = \begin{pmatrix} w_{k1}^{(s)} \\ w_{k2}^{(s)} \\ \vdots \\ w_{kn}^{(s)} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sum_{i=1}^{n} w_{ki}^{(s)} \cdot x_i \tag{5.1}$$

2. Auf den so errechneten skalaren Wert z wird nun i.A. eine Aktivierungsfunktion $\sigma(z)$ angewandt. Obwohl in Schritt 1 eine eine rein lineare (affine) Abbildung vorliegt erlaubt die Anwendung einer nicht linearen Aktivierungsfunktion dem Neuron, auch nicht lineare Ergebnisse zu liefern. Typische Aktivierungsfunktionen werden in Abschnitt 5.3 vorgestellt.

$$\hat{y} = \sigma(z) = \sigma\left(\sum_{i=1}^{n} w_{ki}^{(s)} \cdot x_i\right) \quad \text{mit} \quad \sigma : \mathbb{R} \to \mathbb{R} \tag{5.2}$$

Da die in Schritt 1 definierte lineare Abbildung den Nullvektor stets auf den Wert Null abbildet und damit stets $\sigma(\mathbf{w} \cdot \mathbf{0}) = \sigma(0)$ gilt (die in Schritt 1 berechnete Hyperebene ϵ des (n+1)-dimensionalen Raumes geht stets durch den Ursprung) ist die Funktionalität des Neurons stark eingeschränkt. Hier schafft die Verwendung eines Bias b Abhilfe. Dabei werden der Gewichtsvektor $\mathbf{w}$ um die Koordinate b und der Inputvektor $\mathbf{x}$ um die Koodinate 1 ergänzt. Damit kann das Neuron auch bei einem Input von $\mathbf{0}$ ein von 0 verschiedenes Ausgangssignal feuern.

$$\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x}) = \sigma\left(\begin{pmatrix} w_{k1}^{(s)} \\ \vdots \\ w_{kn}^{(s)} \\ b \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix}\right) = \sigma\left(\sum_{i=1}^{n} w_{ki}^{(s)} \cdot x_i + b\right) = \sigma\left(z + b\right) \tag{5.3}$$

[1] Aus Gründen der Übersichtlichkeit wurden in Abb. 5.2 nur wenige Synapsen beschriftet.

Die folgende Abbildung 5.3 visualisiert die Arbeitsweise eines Neurons ohne und mit Bias. Mathematisch wird hier also die Hyperebene ϵ um b Einheiten in Richtung der y-Achse verschoben.

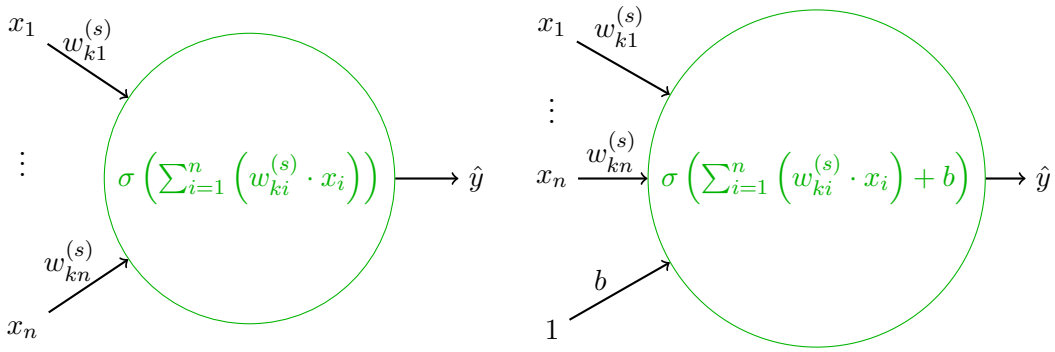


Abbildung 5.3: Neuron ohne (links) und mit (rechts) Bias

5.3 Aktivierungsfunktionen

In diesem Abschnitt werden die gebräuchlichsten Aktivierungsfunktionen (vgl. (*20*) S 154f) vorgestellt. Für den Schulunterricht ergeben sich hier Anwendungen für stückweise definierte Funktionen und die Sigmoidfunktion, die auch vom logistischen Wachstum her bekannt ist.

5.3.1 Threshold- oder Stepfunktion

Diese Aktivierungsfunktion bestimmt nur, ob ein Signal z kleiner oder größer gleich Null ist:

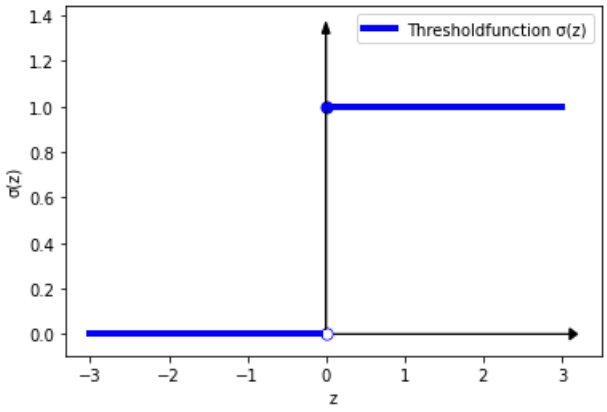


Abbildung 5.4: Threshod- oder Stepfunktion

$$\sigma : \mathbb{R} \to \{0, 1\}, \quad \sigma(x) = \begin{cases} 1 & \text{wenn } x \geq 0 \\ 0 & \text{wenn } x < 0 \end{cases} \tag{5.4}$$

Die Stepfunktion wird im Abschnitt 5.5 verwendet. Dort ist nur eine ja/nein-Entscheidung gefordert, nämlich ob ein bestimmter Punkt auf der positiven[2] oder negativen Seite einer Geraden liegt. Wahrscheinlichkeiten können damit nicht vorhergesagt werden, dafür eignet sich die Sigmoidfunktion.

5.3.2 Rectifier Funktion - RELU

Die RELU[3] - Aktivierungsfunktion ist eine der am meisten verwendeten und lässt Signale größer gleich Null unverändert, setzt aber negative Signale auf Null:

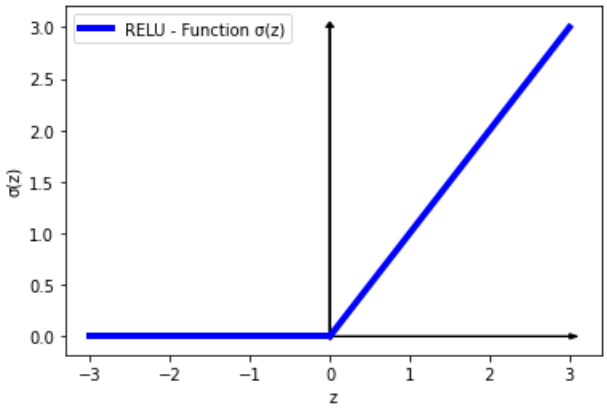


Abbildung 5.5: RELU - Linear Rectifier Units

$$\sigma : \mathbb{R} \to \mathbb{R}_0^+, \quad \sigma(x) = \max(0, x) = \begin{cases} x & \text{wenn } x \geq 0 \\ 0 & \text{wenn } x < 0 \end{cases} \tag{5.5}$$

Ein konkreter Anwendungsfall für diese Aktivierungsfunktion wird im Abschnitt 5.7 Funktionsapproximation durch ein neuronales Netz vorgestellt.

5.3.3 Sigmoid oder Logistische Funktion

Diese bereits von der Logistischen Regression bekannte Funktion eignet sich besonders, wenn ein Signal z in eine Wahrscheinlichkeit umgewandelt werden soll. Insbesondere also bei binären Klassifikationsproblemen.

$$\sigma : \mathbb{R} \to (0, 1), \quad \sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.6}$$

[2] Seite, auf die der Normalvektor zeigt

[3] Rectified Linear Unit

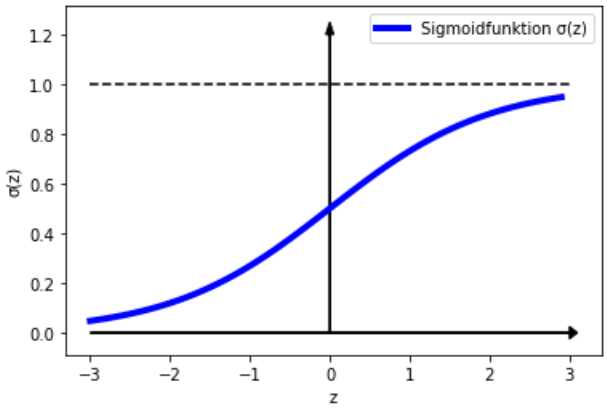


Abbildung 5.6: Sigmoidfunktion

5.4 Lehrplanbezug

Für das grundsätzliche Verständnis der Arbeitsweise eines neuronalen Netzes benötigen Schüler*innen Vektoren und Normalvektoren, das Skalarprodukt, sowie Funktionen und iterative Verfahren. Vor allem kommen stückweise definierte Funktionen und die Sigmoidfunktion vor. In der AHS und den HTLs werden diese Inhalte großteils, wie im Kapitel 4 schon ausführlich angeführt, im Zuge der Vektorrechnung sowie des Themas (Reelle) Funktionen in der neunten und zehnten Schulstufe vermittelt. In Höheren Technischen Lehranstalten gibt es überdies noch im vierten Lernjahr einen Schwerpunkt zu Matrizen, wobei Daten strukturiert in Vektoren und Matrizen zusammengefasst, sowie Berechnungen im Fachgebiet durchgeführt werden sollen. Schüler*innen der AHS lernen in der zwölften Schulstufe beim Thema Differenzen- und Differentialgleichungen, diskrete Veränderungen von Größen durch Differenzengleichungen zu beschreiben und zu deuten, sowie einfache dynamische Systeme mit Hilfe von Differenzengleichungen zu beschreiben und zu untersuchen. In den HTLs lernen die Schüler*innen im dritten Jahr Iterationsverfahren zur Berechnung von Nullstellen kennen. (vgl. (*12*) und (*10*), Online Ressourcen) In den Lehrplänen der Handelsakademien fehlt die Vektorrechnung, allerdings lernen die Schüler*innen im zweiten Jahrgang einiges zum Thema Matrizen. (vgl. (*11*), Online Ressource)

Für das Thema Forward- und Backpropagation (Abschnitt 5.6.4) werden überdies Ableitungen sowie die Kettenregel aus dem Bereich der Differentialrechnung benötigt, die in allen drei Schulformen in der elften (AHS und HTL) oder zwölften (HAK) Schulstufe behandelt wird.

5.5 Perzeptron, das einfachste neuronale Netz

In diesem Abschnitt wird die Arbeitsweise eines Neurons an Hand eines Perzeptrons (vgl. (*24*), Online Resource) vorgestellt. Ein Perzeptron stellt das einfachste neuronale Netz dar, es besteht

abgesehen vom Input Layer nur aus genau einem Neuron. Der Aufbau und der Lernprozess werden mittels einer einfachen binären Klassifikation erklärt und visualisiert. Das hier vorgestellte Beispiel eignet sich auch für den Unterrichtsgebrauch. Von sechs zweidimensionalen Datenpunkten $P_i, i = 1, \ldots, 6$ soll das Perzeptron eine Entscheidungsregel lernen. Die beiden Klassen sind mit 0 (rot) und 1 (grün) gelabelt, die Trainingsdaten haben die folgende Struktur:

$$P_1(1,1) \to 0 \quad P_2(1,2) \to 0 \quad P_3(1,3) \to 1 \quad P_4(2,1) \to 0 \quad P_5(2,2) \to 1 \quad P_2(2,3) \to 1$$

Die Daten können also wie folgt visualisiert werden:

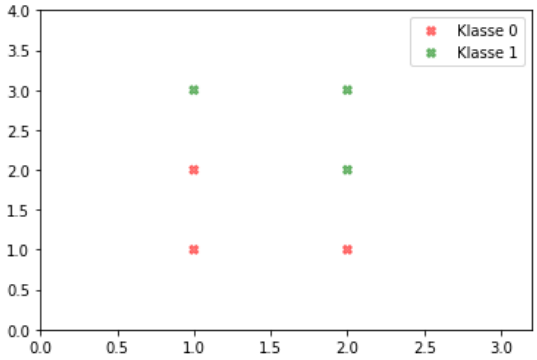


Abbildung 5.7: Perzeptron - Daten für das Training

Abbildung 5.8 zeigt die Arbeitsweise des Perzeptrons.

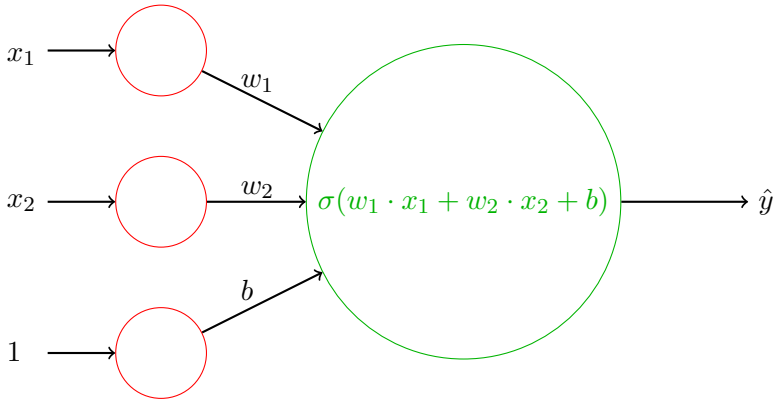


Abbildung 5.8: Perzeptron - Aufbau

Konkret wird hier aus jeder Observation $P(x_1, x_2)$ ein Vorhersagewert $\hat{y}$ nach folgender Regel errechnet. Ein zunächst beliebig angenommener Gewichtsvektor $\mathbf{w}$ wird mit dem Inputvektor $\mathbf{x}$ skalar multipliziert, woraus sich ein Skalar z ergibt:

$$\mathbf{w} \cdot \mathbf{x} = \begin{pmatrix} w_1 \\ w_2 \\ b \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} = w_1 \cdot x_1 + w_2 \cdot x_2 + b = z \tag{5.7}$$

Variiert man x_1 und x_2, so erhält man mit $w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0$ die implizite Form einer Geradengleichung. Ziel ist es, die Gewichte so anzupassen, dass wenn möglich diese Gerade die beiden Klassen linear trennt. Das Bias (vgl. Abschnitt 5.2) ist erforderlich, damit die Gerade nicht notwendigerweise durch den Ursprung geht (hier könnte man bei den gegebenen Daten keine lineare Trennung erzielen).

Auf den so erhaltenen Wert z wird nun eine Aktivierungsfunktion $\sigma(z)$ angewendet, hier im konkreten die Threshold- oder Stepfunction, vergleiche Abschnitt 5.3.1. Liegt der Eingabepunkt auf jener Seite der Geraden, auf die der Normalvektor $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ weist oder auf der Geraden selbst, so ergibt sich der Wert 1, andernfalls der Wert 0.

Zunächst werden die Gewichte zufällig gewählt und der Output bestimmt. Dieser Vorgang wird für die Anfangsgewichte $\mathbf{w} = \begin{pmatrix} 0.5 \\ 0.5 \\ -2.4 \end{pmatrix}$ und $\mathbf{x_1} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ zuerst nur für den Punkt $P_1(1,1)$ und dann mit Hilfe der Matrizenrechnung kompakt für alle Trainingspunkte in einem Rechenschritt durchgeführt:

$$\hat{y_1} = \sigma\left(\mathbf{w} \cdot \mathbf{x_1}\right) = \sigma\left(\begin{pmatrix} 0.5 \\ 0.5 \\ -2.4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}\right) = \sigma\left(-1.4\right) = 0 \tag{5.8}$$

Dies bedeutet, dass der Punkt P_1 richtigerweise als zur Klasse 0 (rot) gehörig klassifiziert wurde.

$$\begin{aligned} \hat{\mathbf{y}}^{\mathbf{T}} &= \sigma\left(\mathbf{w}^{\mathbf{T}} \cdot \mathbf{X}\right) \\ &= \sigma\left(\begin{pmatrix} 0.5 & 0.5 & -2.4 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 2 & 3 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}\right) \\ &= \sigma\begin{pmatrix} -1.4 & -0.9 & -0.4 & -0.9 & -0.4 & 0.1 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{5.9}$$

Hier ist nun ersichtlich, dass die ersten 5 Punkte als zur Klasse 0 gehörig klassifiziert wurden und nur der Punkt $P_6(2,3)$ als zur Klasse 1 gehörig. Falsch klassifiziert wurden die Punkte $P_3(1,3)$ und $P_5(2,2)$. Die folgende Abbildung 5.9 veranschaulicht dieses Ergebnis. Es wurden also 4 von 6 Trainingspunkten richtig klassifiziert, das ergibt einen Score von 66,6%.

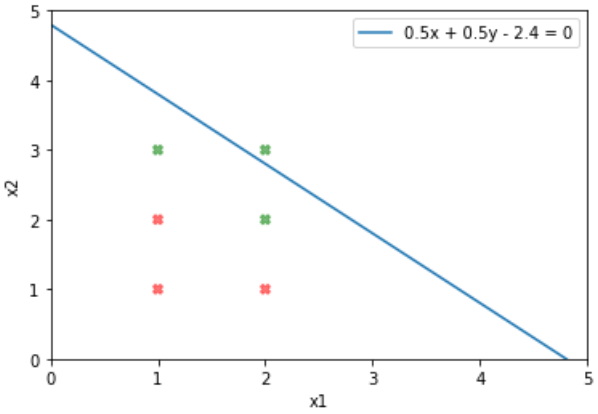


Abbildung 5.9: Perzeptron - Erste Iteration

Nun wird das Modell iterativ verbessert. Zu diesem Zweck wird zunächst eine Cost- oder Fehlerfunktion $C(\mathbf{w})$ definiert:

$$C(\mathbf{w}) = \mathbf{y} - \hat{\mathbf{y}} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{5.10}$$

Ziel ist es nun, die Gewichte Schritt für Schritt so zu verbessern, dass schließlich $C(\mathbf{w})$ gleich dem Nullvektor ist, d.h. dass alle Punkte richtig klassifiziert werden. Dieses Ziel ist natürlich nur bei linear trennbaren Daten zu erreichen.

Zur Optimierung der Costfunction wird hier die sogenannte Delta-Regel (vgl. (*25*), Online Resource) verwendet. Diese ist speziell bei Perzeptrons ein gängiges und zielführendes Verfahren. Eine Verallgemeinerung der Delta-Regel ist die Backpropagation-Regel, die auch auf mehrschichtige neuronale Netze anwendbar ist. Mathematisch gesehen handelt es sich bei der Deltaregel um ein Gradientenabstiegsverfahren. Anschaulich kann man das Gradientenabstiegsverfahren als "Verfahren des steilsten Abstiegs" beschreiben, d.h. man nähert sich iterativ einem (lokalen) Minimum, indem man sich schrittweise in Richtung fallender Funktionswerte bewegt. Das Gradientabstiegsverfahren wird in Abschnitt 5.6.3 im Detail beschrieben.

Die Delta-Regel kann formal mit

$$\mathbf{w_{neu}} = \mathbf{w_{alt}} + \boldsymbol{\Delta}\mathbf{w} \quad \text{mit} \quad \boldsymbol{\Delta}\mathbf{w} = \eta \cdot \mathbf{X} \cdot (\mathbf{y} - \hat{\mathbf{y}}) \tag{5.11}$$

beschrieben werden. η ist dabei die Lernrate und bestimmt, in welchen Schritten man sich dem Minimum der Costfunction anzunähern versucht. Die Delta-Regel kann dabei wie folgenderma-

ßen Interpretiert werden:

- Ist für ein konkretes Gewicht w_i der Output des Netzes $\hat{y}_i$ kleiner als der gewünschte Wert y_i (der Output muss also erhöht werden), so ist $y_i - \hat{y}_i$ positiv und damit ist Δw_i für positive Inputs größer Null und für negative Inputs negativ. Die Gewichte zu positiven Inputwerten werden also erhöht (gestärkt) und zu negativen Inputwerten verringert (geschwächt).

- Ist für ein konkretes Gewicht w_i der Output des Netzes $\hat{y}_i$ größer als der gewünschte Wert y_i (der Output muss also verringert werden), so ist $y_i - \hat{y}_i$ negativ und damit ist Δw_i für positive Inputs kleiner Null und für negative Inputs positiv. Die Gewichte zu positiven Inputwerten werden also verringert (geschwächt) und zu negativen Inputwerten erhöht (gestärkt).

- Stimmt für ein konkretes Gewicht w_i der Output des Netzes $\hat{y}_i$ mit dem gewünschte Wertn y_i überein (der Output ist also bereits richtig), so ist $y_i - \hat{y}_i = 0$ und die Delta-Regel verändert das entsprechende Gewicht nicht.

Führt man für das konkrete Beispiel die erste Iteration durch, so ergeben sich für $\eta = 0.1$ die folgenden Werte:

$$\mathbf{\Delta w} = 0.1 \cdot \begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 2 & 3 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.3 \\ 0.5 \\ 0.2 \end{pmatrix} \tag{5.12}$$

und damit

$$\mathbf{w_{neu}} = \mathbf{w_{alt}} + \mathbf{\Delta w} = \begin{pmatrix} 0.5 \\ 0.5 \\ -2.4 \end{pmatrix} + \begin{pmatrix} 0.3 \\ 0.5 \\ 0.2 \end{pmatrix} = \begin{pmatrix} 0.8 \\ 1.0 \\ -2.2 \end{pmatrix} \tag{5.13}$$

Berechnet man mit diesem neuen Gewichtsvektor wieder $\hat{\mathbf{y}}^{\mathbf{T}}$, so ergibt sich

$$\hat{\mathbf{y}}^{\mathbf{T}} = \sigma \begin{pmatrix} -0.4 & 0.6 & 1.6 & 0.4 & 1.4 & 2.4 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{5.14}$$

Bereits die nächste Iteration[4] liefert $\hat{\mathbf{y}}^{\mathbf{T}} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$ und damit $C(\mathbf{w}) = \mathbf{0}$. Also ergibt sich $(\mathbf{y} - \hat{\mathbf{y}}) = \mathbf{0}$ und weitere Iterationen verändern den Gewichtsvektor $\mathbf{w}$ nicht mehr. Die Abbildung 5.10 visualisiert die Ergebnisse:

[4] Mögliche Übungsaufgabe für Schüler:innen

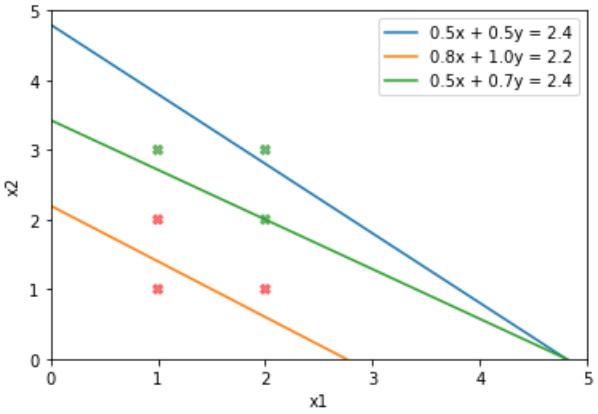
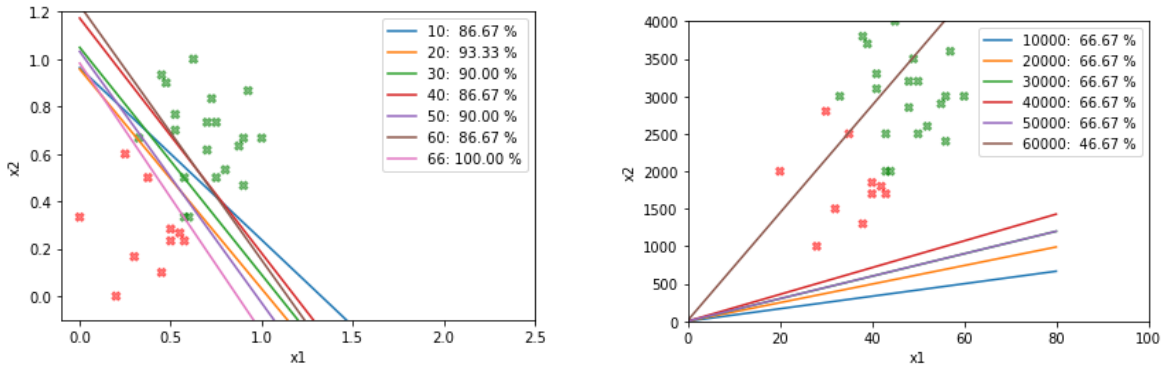


Abbildung 5.10: Perzeptron - Optimaler Gewichtsvektor

Führt man das oben vorgestellte Verfahren für das in Abschnitt 4.4.2 vorgestellte Klassifikationsproblem mit den Startgewichten $\mathbf{w} = \begin{pmatrix} 0.5 \\ 0.5 \\ -2.4 \end{pmatrix}$ durch, so erreicht man nach 65 Iterationen $C(\mathbf{w}) = \mathbf{0}$. Dabei ist aber wesentlich, dass vor dem Training ein Featurescaling gemäß Abschnitt 3.6 durchgeführt wird[5]. Andernfalls führen mit den gleichen Ausgangsgewichten auch 60000 Iterationen nicht annähernd zum Ziel.



(a) (b)

Abbildung 5.11: Perzeptron: Vergleich Normalisierte Daten versus Originaldaten

Möglicher Einsatz im Unterricht

In der oben beschriebenen Vorgangsweise werden die jeweiligen Gewichtsvektoren als die Koeffizienten einer Geradengleichung in allgemeiner Form interpretiert. Damit bekommen die aktuellen Gewichte des Netzes eine konkrete geometrische Bedeutung, sie bestimmen nämlich die aktuelle Lage der vom Perzeptron berechneten Grenzlinie zwischen den beiden Klassen. Hier kann ver-

[5]Hier wurden die Daten normalisiert

tieft werden, dass durch Einsetzen eines konkreten Punktes in diese Geradengleichung bestimmt werden kann, auf welcher Seite der Geraden der entsprechende Punkt liegt. Die Vorgangsweise kann auch auf den dreidimensionalen Raum verallgemeinert werden.

Konkrete Aufgabe:

Gegeben ist ein Trainingsset für ein Perzeptron bestehend aus 6 Punkten im dreidimensonalen Raum (3 Features). Die Punkte sind dabei mit rot/grün gelabelt. $P_1(0,3,0)$, $P_2(2,0,0)$, $P_3(0,1,3)$, $P_4(0,1,2)$, $P_5(-1,0,1)$, $P_6(1,1,1)$. Dabei gehören die Punkte P_4 und P_5 zur Klasse "grün" und die anderen Punkte zur Klasse "rot".

Stelle fest, wie viele Punkte durch die aktuellen Gewichte $w_1 = 0.2$, $w_2 = 0.4$, $w_3 = 0.1$, $b = -1.0$ richtig klassifiziert werden.

5.6 Arbeitsweise und Lernprozess neuronaler Netze

5.6.1 Intuitive Erklärung der Arbeitsweise

Wie bereits in Abschnitt 5.1 beschrieben transportiert ein Feed-Foreward-Netz Signale vom Input-Layer über den/die Hidden-layer zum Outputlayer. Dabei berechnet jedes Neuron aus allen Eingangsignalen und den zugehörigen Gewichten ein Ausgangssignal gemäß Abschnitt 5.2.

In diesem Abschnitt wird der Versuch unternommen, an Hand einer konkreten Aufgabenstellung die Funktionsweise eines neuronalen Netzes ohne tiefe Mathematik verständlich zu machen, wobei darauf hingewiesen wird, dass beim konkreten Training die einzelnen Neuronen andere Schwerpunkte setzen, die für den menschlichen Betrachter wesentlich abstraktere Features herausfiltern.

Zum Erreichen dieses Ziels wird ein einfaches neuronales Netzwerk angenommen, das an Hand der Eingangsfeatures

- x_1 ... Fläche in Quadratmeter
- x_2 ... Anzahl der Schlafzimmer
- x_3 ... Entfernung zum nächstgelegenen Stadtzentrum
- x_4 ... Alter der Immobilie

den Preis einer Immobilie vorhersagt. In der folgenden Grafik sind nicht alle Synapsen eingezeichnet, die Gewichte der fehlenden Synapsen sind Null oder fast Null und die entsprechenden Signale daher für das jeweilige Neuron im Hidden Layer nicht relevant.

In obiger Abbildung 5.12 verwendet das Neuron 1 im Hidden Layer zur Berechnung des Ausgangssignals nur die Eingangsfeatures x_1 (Fläche) und x_3 (Entfernung). Eine mögliche Interpretation ist, dass dass dieses Neuron also große Immobilien mit guter Anbindung zum nächsten

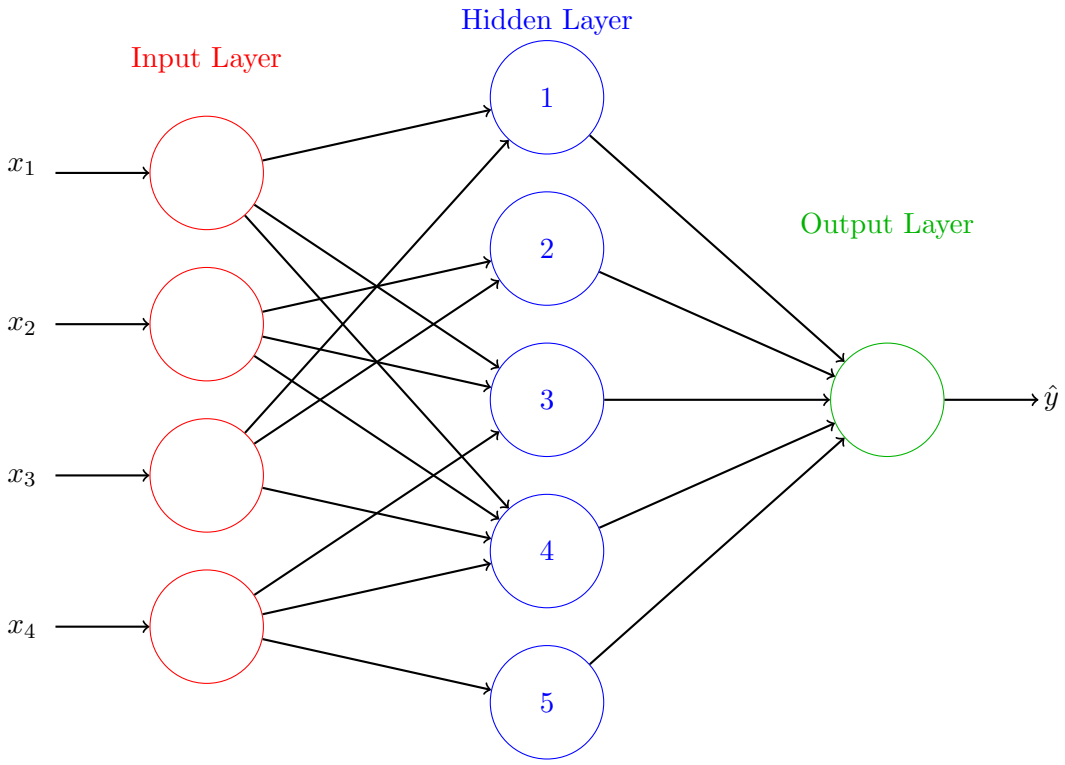


Abbildung 5.12: ANN - Intuitive Erklärung der Arbeitsweise

Stadtzentrum "herausfiltert". Neuron 3 im Hidden Layer verwendet nur die Eingangsfeatures x_1 (Fläche), x_2 (Schlafzimmer) und x_4 (Alter). Solche Immobilien könnten für große Familien, die Neubauwohnungen suchen, interessant sein. Schließlich verwendet Neuron 5 nur x_5 (Alter) und und spezialisiert sich vielleicht auf renovierungsbedürftige Immobilen.

5.6.2 Lernprozess

Der Lernprozess eines ANN wurde bereits in Abschnitt 5.5 kurz vorgestellt, soll in diesem Abschnitt aber noch einmal im Detail erklärt werden. Dabei wird eine Vorgangsweise gewählt, die auch für SchülerInnen der Sekundarstufe verständlich erscheint. Für eine Erklärung von einem höheren mathematischen Standpunkt aus wird auf (*2*), S 305ff bzw. (*26*) S 26ff verwiesen.
Der Lernprozess unterschiedet sich nicht wesentlich von jedem supervised Machine-Leraning Algorithmus und besteht aus zwei Grundschritten:

1. **Definieren einer Cost- oder Lossfunction**

   Die Cost-, Loss- oder auch Errorfunction vergleicht die vom Netzwerk errechneten Schätzwerte $\hat{y}$ mit den tatsächlichen Labeln und ist daher von den Gewichten $w_i, i = 1, \ldots, n$ abhängig, aus denen sich ja die Schätzwerte berechnen.

$$C(w_1, \ldots, w_n) = f\left(\hat{y}(w_1, \ldots, w_n), y\right) \tag{5.15}$$

Je nach Aufgabestellung werden verschieden Fehlerfunktionen verwendet:

a) Soll das Netzwerk einen reellen Wert voraussagen (Regression), so verwendet man als Fehlerfunktion in der Regel die Summe der Fehlerquadrate[6]:

$$C(w_1, \ldots, w_n) = \frac{1}{2} \sum_{j=1}^{m} (\hat{y}_j - y_j)^2 \tag{5.16}$$

b) Bei einer binären Klassifikation ist die Methode der kleinsten Quadrate nicht zielführend. Statt dessen verwendet man die Cross-Entropy-Funktion:

$$C(w_1, \ldots, w_n) = -\sum_{j=1}^{m} \ln \left(\hat{y}_j^{y_j} \cdot (1 - \hat{y}_j)^{1-y_i} \right) = -\sum_{j=1}^{m} \left(y_j \ln \hat{y}_j + (1 - y_j) \ln (1 - \hat{y}_i) \right) \tag{5.17}$$

Dabei bedeuten $\hat{y}_j$ die zum Eingabevektor $\mathbf{x_j}$ berechnete Wahrscheinlichkeit der Klassenzugehörigkeit und y_j die tatsächliche (gelabelte) Klassenzugehörigkeit. Im Rahmen dieser Arbeit wird wird hier nur auf (*20*), S 158 verwiesen.

c) Hat man mehr als zwei Klassen vorliegen, so ist noch über die Outputs aller k Klassen zu summieren:

$$C(w_1, \ldots, w_n) = -\sum_{j=1}^{m} \sum_{l=1}^{k} \left(y_j^{(l)} \ln \hat{y}_j^{(l)} + \left(1 - y_j^{(l)}\right) \ln \left(1 - \hat{y}_i^{(l)}\right) \right) \tag{5.18}$$

2. **Minimieren der Costfunction**

Nun versucht man das Minimum der Costfunction zu berechnen. Im Wesentlichen handelt es sich hier um eine Extremwertaufgabe, wobei die Gewichte w_i so zu verändern sind, dass der Wert von $C(w_1, \ldots, w_n)$ möglichst klein wird. Analytisch können lokale Minima gefunden werden, indem man alle partiellen Ableitungen gleichzeitig Null setzt:

$$\frac{\partial C}{\partial w_1} = 0, \quad \ldots, \quad \frac{\partial C}{\partial w_n} = 0 \tag{5.19}$$

Da dieses analytische Verfahren in der Praxis oft nicht durchführbar ist, wird versucht, ein Minimum der Costfunction iterativ zu bestimmen. Ein häufig eingesetztes Verfahren ist das Gradientenabstiegsverfahren, das in Abschnitt 5.6.3 zunächst AHS-lehrplankonform für nur ein Gewicht und anschließend für n Gewichte vorgestellt wird.

5.6.3 Gradientenabstiegsverfahren

Ist die Costfunction nur von einem Gewicht abhängig, so kann das (globale) Minimum mit Hilfe einer Extremwertaufgabe gefunden werden. In der Praxis wird hier aber das Gradientenabstiegs-

[6] Methode der kleinsten Quadrate, Mean squared error

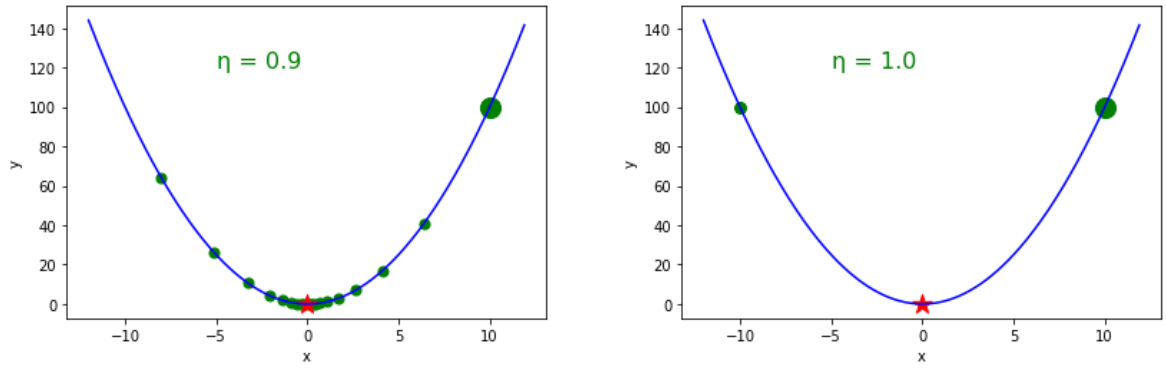
verfahren herangezogen, das nun an Hand der sehr einfachen Costfunction $C(w) = w^2$ vorgestellt wird.

Beim Gradientenabstiegsverfahren (vgl. (*18*), OnlineResource) beginnt man mit einem zufälligen Ausgangspunkt $(w, C(w))$ (zufällig zugewiesenes Gewicht) und versucht nun w so zu verändern, dass der Funktionswert kleiner wird. Zu diesem Zweck bestimmt man die 1.Ableitung $C'(w)$ und berechnet eine neues Gewicht mit Hilfe der Formel

$$w_{neu} = w_{alt} - \eta \cdot C'(w_{alt}) \quad \text{mit} \quad \eta > 0 \tag{5.20}$$

Ist hier $C'(w_{alt})$ positiv, so bewegt man sich nach links, ist $C'(w_{alt})$ negativ, so bewegt man sich nach rechts. Für $C'(w_{alt})$ gleich Null befindet man sich in einem lokalen Minimum und bewegt sich nicht von der Stelle.

Der Wert von η bestimmt die Größe der Schritte, die bei diesem Verfahren ausgeführt werden. η heißt Lernrate und sollte nicht zu groß sein, da das Verfahren dann gegebenenfalls nicht konvergiert. Die Lernrate wird in der Regel mit $\eta \in (0, 1]$ angenommen.



(a) (b)

Abbildung 5.13: Gradient descent bei verschiedenen Lernraten

Bei einer Lernrate von $\eta = 0.9$ und dem Ausgangsgewicht $w_{alt} = 10$ ergibt sich mit $C(w) = w^2$ und $C'(w) = 2w$ im ersten Iterationsschritt $w_{neu} = 10 - 0.9 \cdot 20 = -8$. Führt man das Verfahren fort so ergibt sich als Folge der Gewichte $\langle 10, -8, 6.4, -5.12, 4.096, \ldots \rangle$. Diese Folge konvergiert gegen Null und nach 100 Iterationen gilt bereits $w \approx 0.00085$ und $C(w) \approx 7.237 \cdot 10^{-7}$.

Ändert man die Lernrate nur geringfügig auf $\eta = 1.0$ so ergibt sich im ersten Iterationsschritt $w_{neu} = 10 - 1.9 \cdot 20 = -10$. Führt man das Verfahren fort so ergibt sich als Folge der Gewichte $\langle 10, -10, 10, -10, \ldots \rangle$. Man erhält also eine oszillierende Folge mit 2 Häufungspunkten, die keinen Grenzwert hat. Das Verfahren findet das lokale Minimum nicht.

Verallgemeinerung auf 2 bzw. n Gewichte

Da im Lehrplan von höheren technischen Lehranstalten auch Funktionen in mehreren unabhängigen Variablen vorkommen, soll das Gradientenabstiegsverfahren auch auf solche Funktionen verallgemeinert werden.
Für die Costfunction $C(w_1, w_2, \ldots, w_n)$ ist der Gradient $grad(C) = \nabla C$ definiert als Vektor der partiellen Ableitungen:

$$grad(C) = \nabla C = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \vdots \\ \frac{\partial C}{\partial w_n} \end{pmatrix} \tag{5.21}$$

Ersetzt man nun in Formel 5.20 die erste Ableitung $C'(w)$ durch den Gradienten, so hat man das Verfahren bereits auf n Gewichte verallgemeinert:

$$\mathbf{w_{neu}} = \mathbf{w_{alt}} - \eta \cdot \nabla C(\mathbf{w_{alt}}) \quad \text{mit} \quad \eta > 0 \tag{5.22}$$

Im Folgenden werden zwei Beispiele für eine Costfunktion $C(w_1, w_2)$ mit zwei Gewichten vorgestellt, die im HTL-Curriculum beim Lehrinhalt "Funktionen in mehreren Veränderlichen" verwendet werden können:

Beispiel 1

Als Costfunktion wird ein einfaches Drehparaboloid verwendet:

$$C : \mathbb{R}^2 \to \mathbb{R}, \quad C(w_1, w_2) = w_1^2 + w_2^2 + 10 \quad \Rightarrow \quad \nabla C = \begin{pmatrix} 2w_1 \\ 2w_2 \end{pmatrix} \tag{5.23}$$

Startet man mit den zufällig gewählten Gewichten $w_1 = 2$ und $w_2 = 4$ und der Lernrate $\eta = 0.1$ so ergibt sich im ersten Iterationsschritt:

$$\mathbf{w_{neu}} = \begin{pmatrix} 2 \\ 4 \end{pmatrix} - 0.1 \cdot \begin{pmatrix} 4 \\ 8 \end{pmatrix} = \begin{pmatrix} 1.6 \\ 3.2 \end{pmatrix} \tag{5.24}$$

Die folgende Grafik 5.14 visualisiert die ersten 20 Iterationen, wobei die Punkte sowohl am Drehparaboloid (schwarz) als auch ihre Spur in der w_1w_2-Ebene (rot) dargestellt sind.
Bereits nach 20 Iterationen erreicht man den Punkt $(0.029, 0.058)$ und nach 100 Iterationen $(5.09 \cdot 10^{-10}, 1.02 \cdot 10^{-9})$ und hat man sich damit dem tatsächlichen Minimum bei $(0, 0)$ mit ausgezeichneter Genauigkeit genähert.

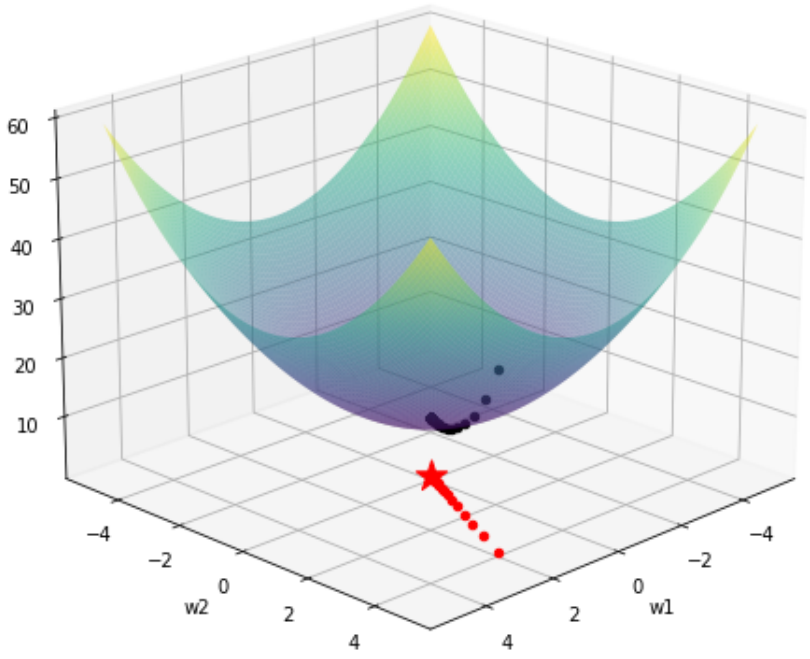


Abbildung 5.14: Gradient Descent - Drehparaboloid

Beispiel 2

Nun wird als Costfunktion die Rosenbrockfunktion (vgl. (*27*), Online Resource) verwendet:

$$C : \mathbb{R}^2 \to \mathbb{R}, \quad C(w_1, w_2) = 2w_1^4 - 4w_1^2 w_2 + w_1^2 + 2w_2^2 - 2w_1 + 4 \tag{5.25}$$

mit

$$\nabla C = \begin{pmatrix} 8w_1^3 - 8w_1 w_2 + 2w_1 - 2 \\ -4w_1^2 + 4w_2 \end{pmatrix} \tag{5.26}$$

Das Minimum dieser Funktion kann einfach berechnet werden und liegt bei $(1, 1)$. Es befindet sich in einem ausgesprochen schmalen und flachen Tal, was eine schlechte Konvergenz des Gradientenabstiegsverfahrens erwarten lässt.

Die Grafik 5.15 visualisiert die ersten 100 Iterationen für die beiden Ausgangspunkte $P(1.5, -1)$ (schwarz-orange) und $Q(-1.5, -1.5)$ (blau-grün):

Hier ergibt sich für P der Näherungswert $(0.64, 0.31)$ und für Q der Näherungswert $(0.57, 0.21)$. Erst nach 1000 Iterationen erreicht man mit $(0.99, 0.98)$ (gerundet) sowohl für P als auch für Q ein brauchbares Ergebnis.

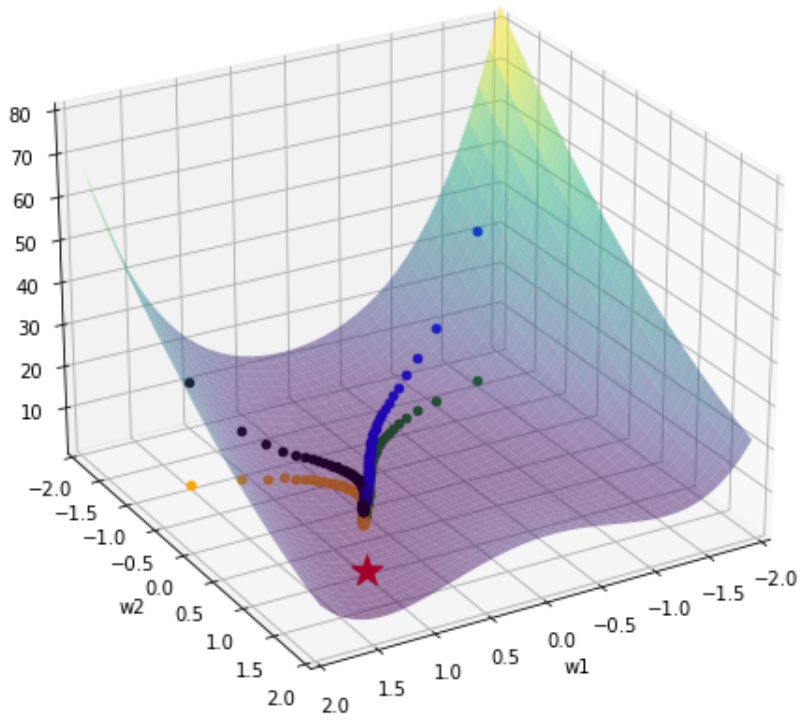


Abbildung 5.15: Gradient Descent - Rosenbrockfunktion

5.6.4 Forward- und Backpropagation

Beim eigentlichen Lernprozess unterscheidet man zwischen Forward- und Backpropagation.

Bei der Foreward-Propagation wird auf Grund der aktuell gesetzten Gewichte für eine bestimmte Anzahl an Trainingsdaten (Batchsize) der Output des Netzwerkes berechnet und der Wert der Fehlerfunktion bestimmt. Dabei wird das Netzwerk vom Inputlayer über die Hidden Layer zum Outputlayer hin abgearbeitet, daher die Bezeichnung Forward-Propagation.

Nun wird der Gradient der Fehlerfunktion bestimmt. Dies setzt voraus, dass alle zur Bestimmung der Fehlerfunktion verwendeten Rechenoperationen und Funktionen differenzierbar sind. Dabei wird das Netzwerk in umgekehrter Reihenfolge von der Outputschicht hin zum Inputlayer abgearbeitet. Dies entspricht der Kettenregel, wo zunächst die äußerste Funktion abgeleitet wird. Dabei werden Schrittweise die Gradienten der einzelnen Netzwerkparameter (Gewichte und Biase) bestimmt. Dieser Vorgang heißt Back-Propagation. (vgl (*2*) S. 313ff)

In wissenschaftlicher Strenge kann der Vorgang der Back-Propagation unter (*20*), S159ff nachgelesen werden.

In dieser Arbeit wird der Lernprozess an einem konkreten Beispiel auch für Schüler nachvollziehbar erläutert. Dabei werden für das in Abbildung 5.16 dargestellte einfache Netzwerk für den

Testdatensatz $\mathbf{x} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$ und dem erwarteten Output $y = 3$ Forward- und Back-Propagation konkret durchgerechnet. Der Hiddenlayer besteht der Einfachheit halber nur aus einem Neuron und verwendet die Aktivierungsfunktion RELU, die Costfunction ist Mean Squared Error. Die aktuellen Gewichte und Biase können der Grafik entnommen werden.

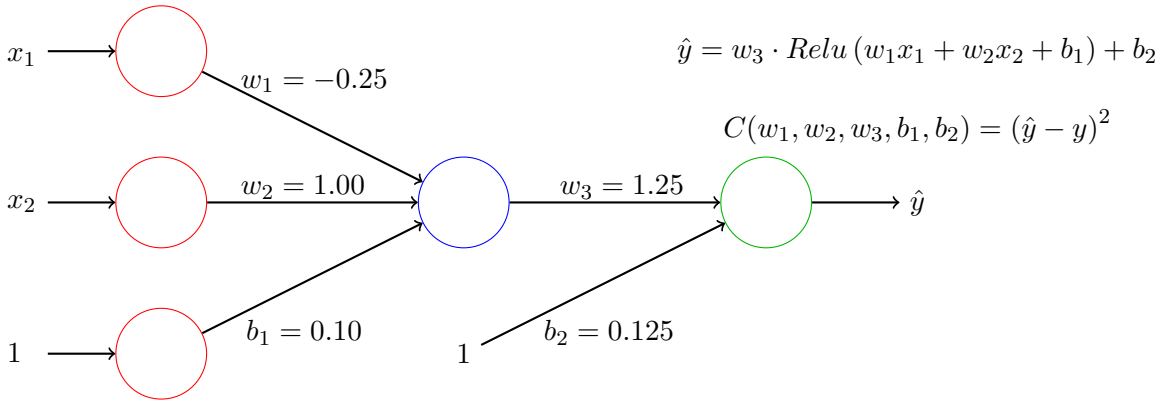


Abbildung 5.16: Foreward- und Backpropagation Ausgangssituation

Damit ergibt die Forward-Propagation für $x_1 = 2$, $x_2 = 2$ und $y = 3$ und die aktuellen Gewichte

$$\hat{y} = 2.125 \quad \text{und} \quad C(-0.25, 1.00, 1.25, 0.10, 0.125) = 0.765625$$

Für die Back-Propagation hat man nun den Gradienten der Fehlerfunktion

$$C(w_1, w_2, w_3, b_1, b_2) = (w_3 \cdot Relu\,(w_1x_1 + w_2x_2 + b_1) + b_2 - y)^2 \tag{5.27}$$

zu bestimmen:

$$\nabla C = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \frac{\partial C}{\partial w_3} \\ \frac{\partial C}{\partial b_1} \\ \frac{\partial C}{\partial b_2} \end{pmatrix} = \begin{pmatrix} 2 \cdot (w_3 \cdot Relu\,(w_1x_1 + w_2x_2 + b_1) + b_2 - y) \cdot w_3 \cdot Relu'(w_1x_1 + w_2x_2 + b_1) \cdot x_1 \\ 2 \cdot (w_3 \cdot Relu\,(w_1x_1 + w_2x_2 + b_1) + b_2 - y) \cdot w_3 \cdot Relu'(w_1x_1 + w_2x_2 + b_1) \cdot x_2 \\ 2 \cdot (w_3 \cdot Relu\,(w_1x_1 + w_2x_2 + b_1) + b_2 - y) \cdot Relu'(w_1x_1 + w_2x_2 + b_1) \\ 2 \cdot (w_3 \cdot Relu\,(w_1x_1 + w_2x_2 + b_1) + b2 - y) \cdot w_3 \cdot Relu'(w_1x_1 + w_2x_2 + b_1) \\ 2 \cdot (w_3 \cdot Relu\,(w_1x_1 + w_2x_2 + b_1) + b_2 - y) \end{pmatrix} \tag{5.28}$$

Da die RELU-Funktion (vgl. Abschnitt 5.3.2) an der Stelle $x_0 = 0$ nicht differenzierbar ist, hat man hier eine Zusatzdefinition zu treffen:

$$Relu' : \mathbb{R} \to \mathbb{R}, \quad Relu'(x) = \begin{cases} 1 & \text{wenn } x \geq 0 \\ 0 & \text{wenn } x < 0 \end{cases} \tag{5.29}$$

Damit ergeben sich für das aktuelle Netzwerk der folgende Gradientenvektor und bei einer Lernrate von $\eta = 0.1$ die folgenden neue Gewichte:

$$\nabla C \mid_{w_1=-0.25,w_2=1.00,w_3=1.25,b_1=0.10,b2=0.125} = \begin{pmatrix} -4.375 \\ -4.375 \\ -2.8 \\ -2.1875 \\ -1.75 \end{pmatrix} \tag{5.30}$$

$$\begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} -0.25 \\ 1.00 \\ 1.25 \\ 0.10 \\ 0.125 \end{pmatrix} - 0.1 \cdot \begin{pmatrix} -4.375 \\ -4.375 \\ -2.8 \\ -2.1875 \\ -1.75 \end{pmatrix} = \begin{pmatrix} -0.20625 \\ 1.04375 \\ 1.278 \\ 0.121875 \\ 0.1425 \end{pmatrix} \tag{5.31}$$

Führt man für diese neuen Gewichte wieder die Forward-Propagation durch, so erhält man die verbesserten Werte $\hat{y} = 2.4389$ und $C(w_1, w_2, w_3, b_1, b_2) = 0.3148$

In der Praxis werden die neuen Gewichte aber Schritt für Schritt bestimmt, indem man zuerst die neuen Gewichte der letzten Schicht (hier w_3 und b_2) berechnet und sich so schichtweise bis zur Inputschicht "vorarbeitet" (Back-Propagation). Zu diesem Zweck wird $C()$ als verkettete Funktion geschrieben. Im Folgenden werden die Teilfunktionen und ihre Differentialquotienten angegeben:

$$C() = f^2, \quad f = g - y, \quad g = w_3 \cdot h + b_2, \quad h = Relu(i), \quad i = w_1 x_1 + w_2 x_2 + b_1 \tag{5.32}$$

$$\frac{dC}{df} = 2f, \quad \frac{df}{dg} = 1, \quad \frac{df}{dy} = -1 \quad \frac{dg}{dw_3} = h \quad \frac{dg}{dh} = w_3 \quad \frac{dg}{db_2} = 1 \tag{5.33}$$

$$\frac{dh}{di} = \begin{cases} 1 & \text{wenn } i \geq 0 \\ 0 & \text{wenn } i < 0 \end{cases}, \quad \frac{di}{dw_1} = x_1, \quad \frac{di}{dx_1} = w_1, \quad \frac{di}{dw_2} = x_2, \quad \frac{di}{dx_1} = w_2, , \quad \frac{di}{db_1} = 1 \tag{5.34}$$

Die aktuellen Funktionswerte sind dem Netz bekannt. Im Folgenden sind die Werte für den Zustand des Netzes aus Abbildung 5.16 angegeben:

$$i = 1.6, \quad h = 1.6, \quad g = 1.215, \quad f = -0.875$$

Die einzelnen Gradienten können nun sehr effizient mit Hilfe der Kettenregel berechnet werden:

$$\frac{dC}{dw_3} = \frac{dC}{df} \cdot \frac{df}{dg} \cdot \frac{dg}{dw_3} = 2f \cdot 1 \cdot h = 2 \cdot (-0.875) \cdot 1.6 = -2.8$$

$$\frac{dC}{db_2} = \frac{dC}{df} \cdot \frac{df}{dg} \cdot \frac{dg}{db_2} = 2f \cdot 1 \cdot 1 = 2 \cdot (-0.875) = -1.75$$

$$\frac{dC}{dw_1} = \frac{dC}{df} \cdot \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{di} \cdot \frac{di}{dw_1} = 2f \cdot 1 \cdot w_3 \cdot 1 \cdot x_1 = 2 \cdot (-0.875) \cdot 2 = -4.375$$

$$\frac{dC}{dw_2} = \frac{dC}{df} \cdot \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{di} \cdot \frac{di}{dw_2} = 2f \cdot 1 \cdot w_3 \cdot 1 \cdot x_2 = 2 \cdot (-0.875) \cdot 2 = -4.375$$

$$\frac{dC}{db_1} = \frac{dC}{df} \cdot \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{di} \cdot \frac{di}{db_1} = 2f \cdot 1 \cdot w_3 \cdot 1 \cdot 1 = 2 \cdot (-0.875) \cdot 1.25 = -2.1875$$

5.6.5 Ablauf des Trainings

Das Training eines neuronalen Netzes läuft in der Regel in mehreren Epochen ab. In jeder Epoche werden alle Datensätze des Trainingdatensatzes verwendet um die Gewichte zu optimieren. Die Gewichte werden in der Regel nicht nach jedem Trainingsadatensatz optimiert, sondern nach der sogenannten Batchsize.
Die Anzahl der Epochen und die Batchsize sind Inputparameter für das Training.

Zusammenfassend kann der Trainingsablauf also wie folgt beschrieben werden:

1. Die Gewichte und Biase des Netzwerk werden mit zufälligen Werten ungleich Null initialisiert.
2. Für die erste (nächste) Batchsize wird Forward-Propagation durchgeführt und der mittlere Fehler wird berechnet.
3. Die Gewichte werden mit Hilfe von Back-Propagation angepasst.
4. Die Schritte 2 und 3 werden solange wiederholt, bis alle Epochen abgearbeitet sind.

5.7 Funktionsapproximation durch ein neuronales Netz

Neuronale Netze erzielen in vielen Bereichen wie Klassifikation, Bild- und Spracherkennung ausgezeichnete Ergebnisse. Trotzdem ist das Verständnis für den theoretischen Hintergrund und die Arbeitsweise solcher Netze vor allem in der Sekundarstufe schwer erfassbar. Hilfreich zur Förderung dieses Verständnisses kann die synthetische Erarbeitung ihrer Fähigkeit sein, beliebige stetige Funktionen zu approximieren. Abhandlungen zu diesem universellen Approximationstheorem findet man z.B. in (*20*) S. 156ff. Die Arbeit (*26*) beschäftigt sich vom Thema her ausführlich mit dieser Fähigkeit.

In dieser Arbeit wird versucht einen für Schüler*innen verständlichen Ansatz zu wählen. Vorausgesetzt werden dabei lediglich Kenntnisse über stückweise definierte lineare Funktionen sowie elementare Funktionstransformationen. Die Idee zu diesem Ansatz entstammt (*28*), Online Resource. Dort wird aber lediglich ein konkretes Beispiel ohne begleitende Rechenschritte vorgestellt, die im Folgenden beschriebene Vorgangsweise wurde im Rahmen dieser Arbeit selbst entwickelt.

5.7.1 Darstellung einer stückweise linear definierten stetigen Funktion durch ein neuronales Netz

Zunächst wird an Hand eines konkreten Beispiels demonstriert, wie eine stetige stückweise linear definierte Funktion durch ein neuronales Netz mit einem Hidden Layer dargestellt werden kann. Als konkretes Beispiel wird die folgende Funktion $f(x)$ verwendet:

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) = \begin{cases} -x + 2 & \text{wenn} \quad x < 1 \\ \frac{1}{2}x - \frac{1}{2} & \text{wenn} \quad 1 \leq x \leq 5 \\ \frac{3}{2}x - \frac{9}{2} & \text{wenn} \quad x > 5 \end{cases} \tag{5.35}$$

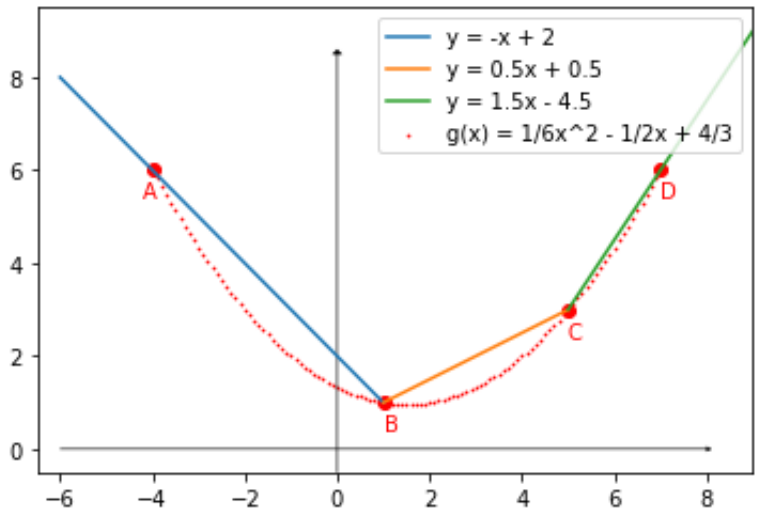


Abbildung 5.17: Stückweise linear definierte Funktion

Diese Funktion kann auch als Näherung für die quadratische Funktion $g(x) = \frac{1}{6}x^2 - \frac{1}{2}x + \frac{4}{3}$ im Intervall $[-4; 7]$ verwendet werden, da der Streckenzug $ABCD$ mit $A(-4, 6)$, $B(1, 1)$, $C(5, 3)$, $D(7, 6)$ aus Sekanten dieser Funktion besteht. Die Abbildung 5.17 visualisiert den Sachverhalt. Das neuronale Netz mit einem Hidden Layer und der zugehörigen Aktivierungsfunktion RELU wird schrittweise entwickelt. Zunächst wird die Funktion

$$f_1 : \mathbb{R} \to \mathbb{R}, \quad f_1(x) = \begin{cases} -x + 2 & \text{wenn} \quad x < 1 \\ \frac{1}{2}x - \frac{1}{2} & \text{wenn} \quad 1 \leq x \end{cases} \tag{5.36}$$

dargestellt. Im Anschluss wird $f_1(x)$ auf $f(x)$ erweitert. Da dieser Vorgang beliebig fortgesetzt werden kann ist intuitiv klar, dass jede stetige stückweise linear definierte Funktion durch ein solches neuronales Netz dargestellt werden kann.

Weiter kann jede stetige Funktion beliebig genau durch einen geeigneten Sekantenzug angenähert werden. Damit lässt sich jede stetige Funktion beliebig genau durch ein neuronales Netzwerk mit einem (beliebig großen) Hiden Layer approximieren.

Die folgende Abbildung 5.18 demonstriert[7], wie $f_1(x)$ als Summe von RELU-Funktionen dargestellt werden kann:

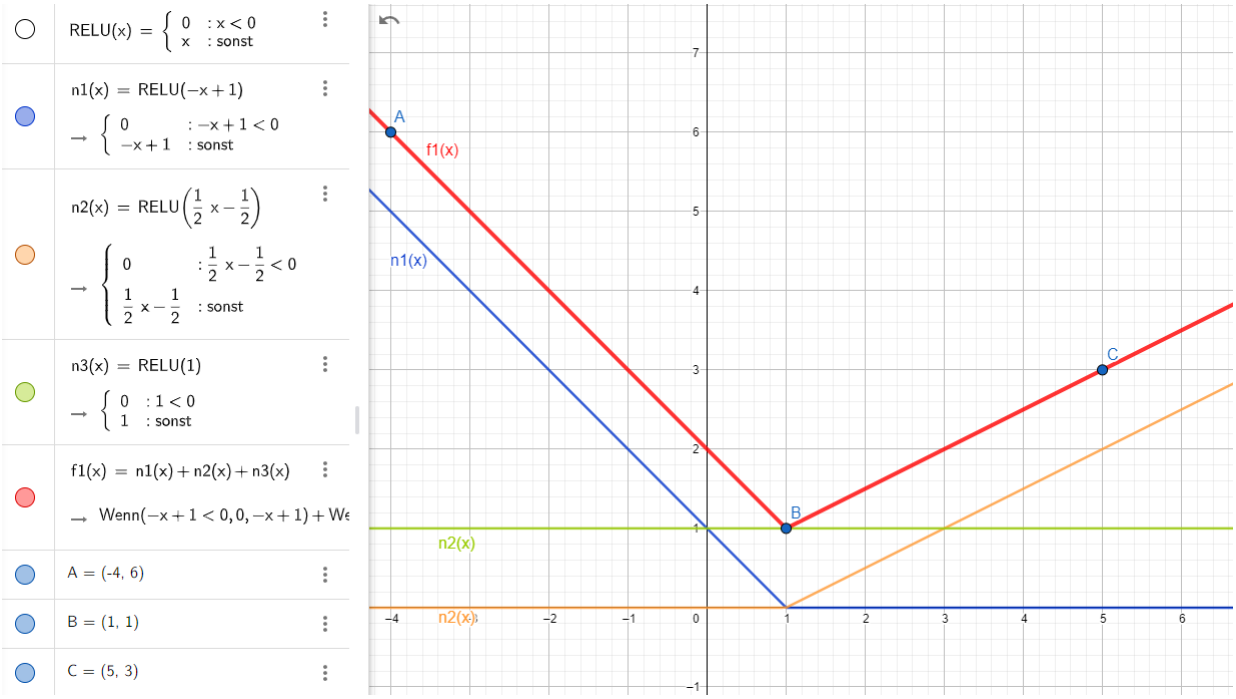


Abbildung 5.18: Darstellung von $f_1(x)$

Zu diesem Zweck wird zunächst die RELU-Funktion definiert:

```
RELU(x) = If(x < 0, 0, x)
```

Nun können die beiden Funktionen mit $n_1(x)$ und $n_2(x)$ mit Hilfe von elementaren Funktionstransformationen aus der RELU-Funktion so erzeugt werden, dass ihre Summe $n_1(x) + n_2(x)$ bereits die Gestalt von $f_1(x)$ aufweist, der Knickpunkt aber noch auf der x-Achse liegt:

```
n1(x) = RELU(-(x - 1)) = RELU(-x + 1)
n2(x) = RELU(0.5(x - 1)) = RELU(0.5x - 0.5)
```

Verschiebt man nun die Summe $n_1(x)+n_2(x)$ um eine Einheit nach oben erhält man die gesuchte Funktion $f_1(x)$. Diese Verschiebung kann wieder mit Hilfe von RELU formuliert werden:

[7] Unter Verwendung von Geogebra

```
n3(x) = 1 = RELU(1)
f1(x) = n1(x) + n2(x) + n3(x) = RELU(-x + 1) + RELU(0.5x - 0.5) + RELU(1)
```

Damit kann $f_1(x)$ durch das in folgender Grafik visualisierte neuronale Netz dargestellt werden:

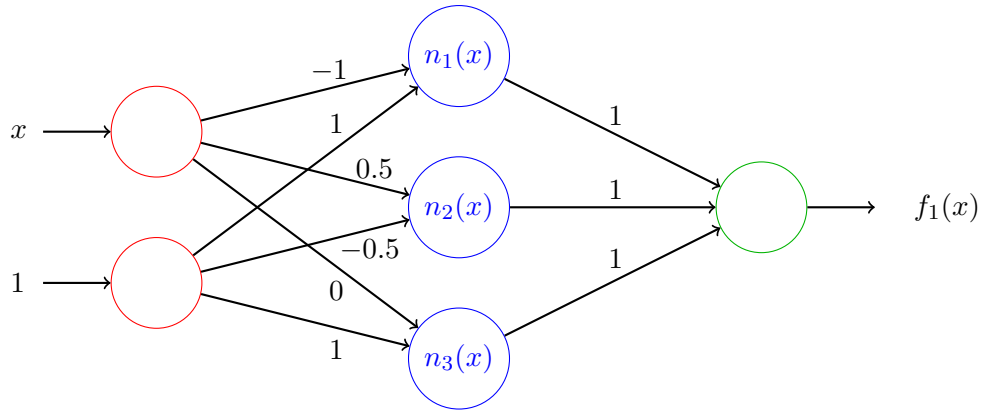


Abbildung 5.19: Darstellung von $f_1(x)$ durch ein neuronales Netz

Nun wird $f_1(x)$ auf $f(x)$ (vgl. Formel 5.35) erweitert. Zu diesem Zweck wird zu $f_1(x)$ eine Funktion $n_4(x)$ mit

$$n_4 : \mathbb{R} \to \mathbb{R}, \quad n_4(x) = \begin{cases} 0 & \text{wenn} \quad x \leq 5 \\ \left(\frac{3}{2}x - \frac{9}{2}\right) - f_1(x) & \text{wenn} \quad x > 5 \end{cases} \tag{5.37}$$

addiert.

Wegen $\left(\frac{3}{2}x - \frac{9}{2}\right) - \left(\frac{1}{2}x - \frac{1}{2}\right) = x - 5$ ergibt sich also:

```
n4(x) = RELU(x - 5)
f(x) = n1(x) + n2(x) + n3(x) + n4(x) =
     = RELU(-x + 1) + RELU(0.5x - 0.5) + RELU(1) + RELU(x - 5)
```

Die folgende Grafik 5.20 wurde mit Geogebra erzeugt. Anhang zeigt den Prozess im Detail.

$f(x)$ kann also nun durch das in Abbildung 5.21 visualisierte neuronale Netz erzeugt werden.

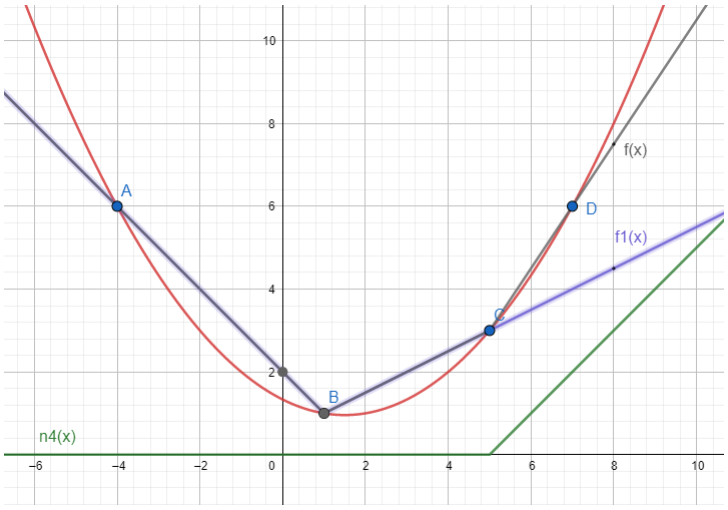


Abbildung 5.20: Erweiterung von $f_1(x)$ auf $f(x)$

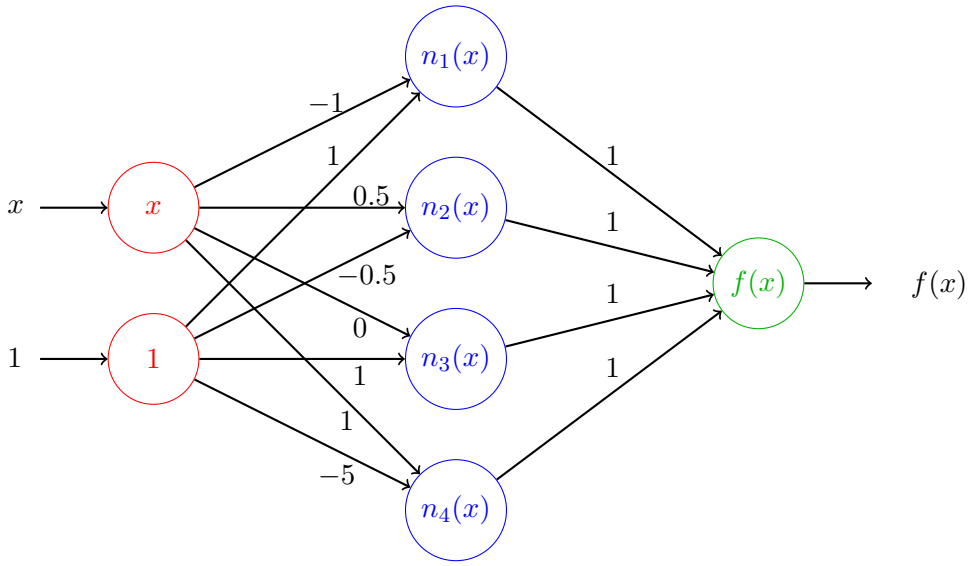


Abbildung 5.21: Darstellung von $f(x)$ durch ein neuronales Netz

Umsetzung mit Tensorflow - Keras

Trainiert man ein neuronales Netz der gegebenen Architektur mit Hilfe von Tensorflow - Keras, so erzielt man im Allgemeinen sehr gute Ergebnisse[8], die Gewichte werden allerdings völlig anders berechnet als im obigen mathemtischen Ansatz. Das im Anhang B.4 angegebene Pythonskript trainiert ein neuronales Netz mit einem Hidden Layer mit 4 Neuronen und liefert ein ausgezeichnetes Ergebnis. Abbildung 5.22 zeigt die visualisierten Ergebnisse. Die blaue breitere Linie markiert die dem Netz übergebenen Trainingsdaten und die schmale orange Linie die vom Netz gemachten Predictions. Die beiden Linien sind optisch deckungsgleich.

[8]Wenn auch nicht bei jedem Training, die Güte der Vorhersagen ist stark von den zufällig zugewiesenen Ausgangsgewichten abhängig

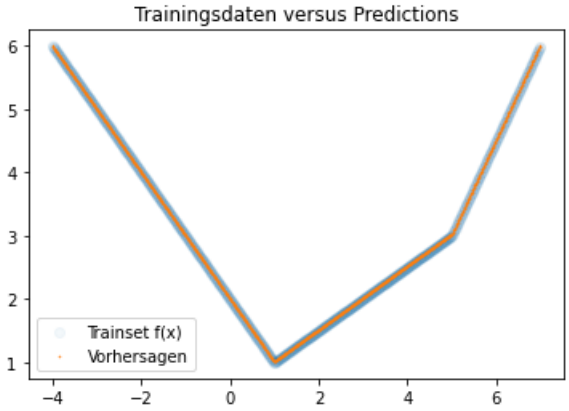


Abbildung 5.22: Visualisierung der Vorhersagen bei der Approximation von f(x)

Aus didaktischen Gründen wurde die Grundidee an Hand eines konkreten Zahlenbeispiels entwickelt. Im folgenden Abschnitt wird ein allgemeiner Ansatz vorgestellt.

5.7.2 Allgemeiner Ansatz für die Funktionsapproximation

In diesem Abschnitt wird ohne einen strengen Beweis zu führen synthetisch erläutert, dass jede stetige stückweise aus n linearen Teilbereichen definierte Funktion $f(x)$ durch ein neuronales Netz mit einem Hidden Layer und der Aktivierungsfunktion RELU dargestellt werden kann.

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) = \begin{cases} k_1 x + d_1 & \text{wenn} \quad x \in (-\infty, a_1) \\ & \vdots \\ k_i x + d_i & \text{wenn} \quad x \in [a_{i-1}, a_i)\,, \quad i = 2, \ldots, n-1 \\ & \vdots \\ k_n x + d_n & \text{wenn} \quad x \in [a_{n-1}, \infty) \end{cases} \tag{5.38}$$

Die Forderung der Stetigkeit kann dabei über die Bedingung

$$k_i a_i + d_i = k_{i+1} a_i + d_{i+1} = b_i, \quad i = 1, \ldots, n-1 \tag{5.39}$$

erfüllt werden.

Die Behauptung wird über die Idee der vollständigen Induktion verifiziert.

1. Zunächst wird der linke Teilbereich dargestellt, der für $x \geq a_1$ mit der konstanten Funktion $y = k_1 a_1 + d_1 = b_1$ fortgesetzt wird. Dies kann mit einem neuronalen Netz mit einem Neuron im Hidden Layer und Bias realisiert werden.

2. Nun wird iterativ der jeweils nächste Teilbereich im Intervall $[a_{i-1}, \infty)\,, i = 2, \ldots, n$ ergänzt, wobei sich $f(x)$ für $x \in (-\infty, a_{i-1})$ nicht verändert. Dies kann durch Addition

einer geeigneten RELU-Funktion $n_i(x)$ erfolgen. Im neuronalen Netz erfolgt dieser Vorgang durch das Hinzufügen eines Neurons im Hidden Layer. Eine Sonderbehandlung für das letzte Teilintervall $[a_{n-1}, \infty)$ ist nicht erforderlich.

Schritt 1

Hier wird also folgende Funktion $f(x)$ realisiert:

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) = \begin{cases} kx + d & \text{wenn} \quad x < a \\ b = k \cdot a + d = const. & \text{wenn} \quad x \geq a \end{cases} \tag{5.40}$$

Dabei sind je nach Vorzeichen von k und b vier Fälle[9] zu unterscheiden:

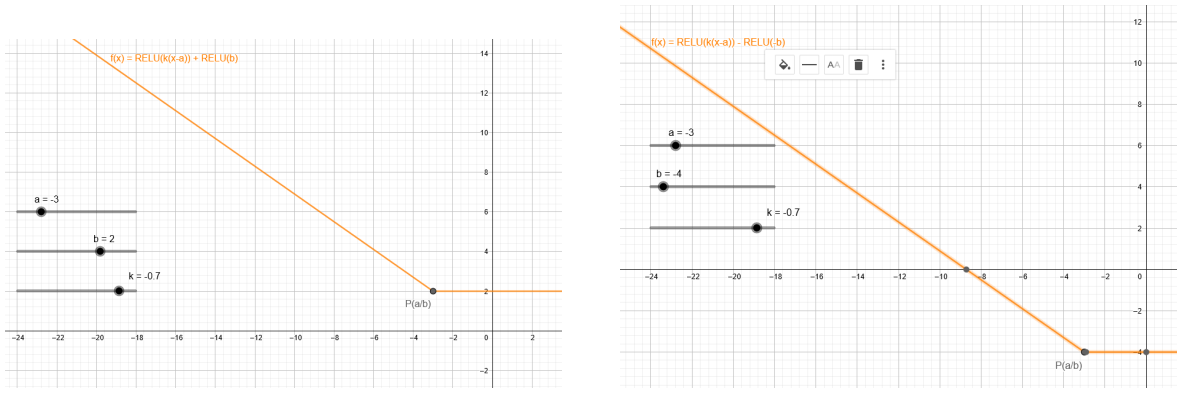
Fall 1: $k < 0$ und $b \geq 0$: $f(x) = RELU(k(x-a)) + RELU(b)$

Fall 2: $k < 0$ und $b < 0$: $f(x) = RELU(k(x-a)) - RELU(-b)$

Fall 3: $k \geq 0$ und $b \geq 0$: $f(x) = -RELU(-k(x-a)) + RELU(b)$

Fall 4: $k \geq 0$ und $b < 0$: $f(x) = -RELU(-k(x-a)) - RELU(-b)$

Die folgenden Abbildungen 5.23 und 5.24 visualisieren die vier Fälle mit Hilfe von Geogebra:

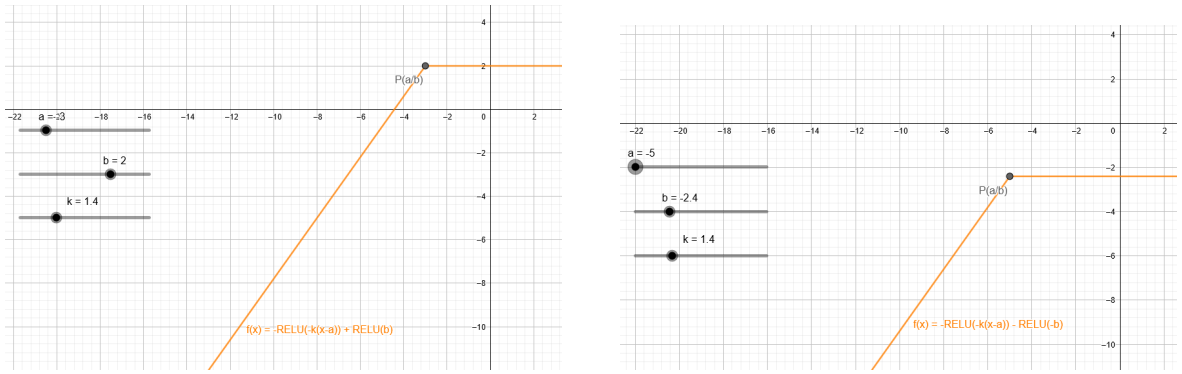


(a) (b)

Abbildung 5.23: (a) Fall 1 (b) Fall 2

Die Geogebradatei zur Visualisierung dieser vier Fälle wird im Anhang angegeben.

[9] Mit Hilfe der Vorzeichenfunktion $sgn(x)$ könnte hier auch eine kompakte Formel angegeben werden



(a) (b)

Abbildung 5.24: (a) Fall 3 (b) Fall 4

Schritt 2

Nun wird davon ausgegangen, dass die ersten l Teilintervalle ($x \in (-\infty, a_l)$)von der in Gleichung 5.38 definierten Funktion $f(x)$ bereits richtig dargestellt werden, nach Schritt 1 gilt das jedenfalls für das erste Teilintervall. In diesem Schritt soll das Netz iterativ durch Hinzunahme eines weiteren Neurons so erweitert werden, dass die Funktionswerte auch für das Intervall $[a_l, a_{l+1})$ richtig berechnet werden.

$$f_{l+1} : \mathbb{R} \to \mathbb{R}, \quad f_{l+1}(x) = \begin{cases} f(x) & \text{wenn} \quad x \in (-\infty, a_l) \\ k_{l+1}\, x + d_{l+1} & \text{wenn} \quad x \in [a_l, \infty) \end{cases} \tag{5.41}$$

Für $x \geq a_l$ ergibt sich f_{l+1} aus der Differenz der zu erzeugenden Funktion $f_{l+1}(x) = k_{l+1}\, x + d_{l+1}$ und der in diesem Bereich bereits vorhandenen Funktion $f(x) = k_l\, x + d_l$. Daher reicht es, zu $f(x)$ eine Funktion $r(x)$ der folgenden Bauart zu addieren:

$$r : \mathbb{R} \to \mathbb{R}, \quad r(x) = \begin{cases} 0 & \text{wenn} \quad x \in (-\infty, a_l) \\ (k_{l+1} - k_l)\, x + d_{l+1} - d_l & \text{wenn} \quad x \in [a_l, \infty) \end{cases} \tag{5.42}$$

Berechnet man durch Lösen der Gleichung $k_{l+1}\, x + d_{l+1} = k_l\, x + d_l$ die x-Koordinate a_l des Schnittpunktes von $f_l(x)$ und $f_{l+1}(x)$, so ergibt sich

$$x = \frac{d_l - d_{l+1}}{k_{l+1} - k_l} = a_l \tag{5.43}$$

und wegen

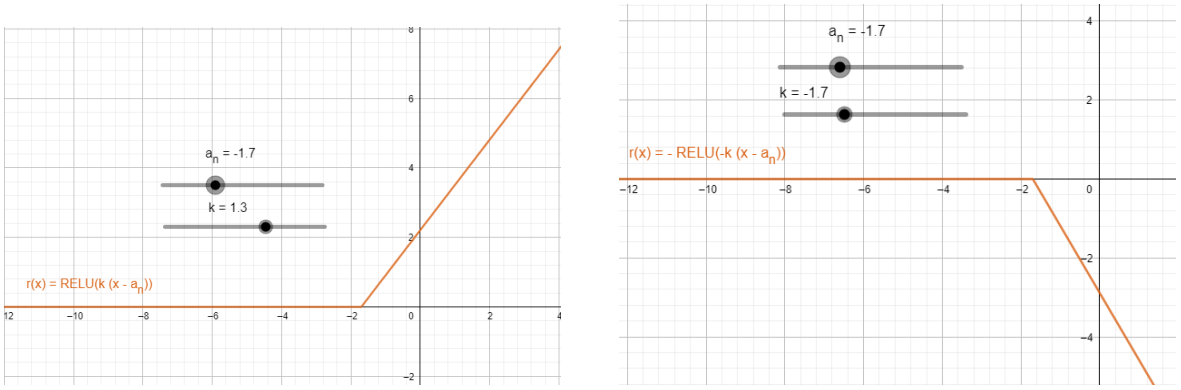
$$(k_{l+1} - k_l)\, x + d_{l+1} - d_l = (k_{l+1} - k_l) \left(x + \frac{d_{l+1} - d_l}{k_{l+1} - k_l} \right) = (k_{l+1} - k_l)\, (x - a_l) \tag{5.44}$$

dass $r(x)$ als eine RELU-Funktion mit Knickpunkt bei a_l und der Steigung $k_{l+1} - k_l$ geschrieben werden kann:

$$r(x) = RELU\left((k_{n+1} - k_n)\, (x - a_n) \right) \tag{5.45}$$

Da diese Funktion links vom Knickpunkt konstant gleich Null sein muss, führen obige Überlegungen allerdings nur für $k_{n+1} - k_n \geq 0$ zum Ziel. Für $k_{n+1} - k_n < 0$ ist $r(x)$ wie folgt zu adaptieren:

$$r(x) = -RELU\left(\left(k_n - k_{n+1}\right)\left(x - a_n\right)\right) \tag{5.46}$$



(a) (b)

Abbildung 5.25: (a) $k_{n+1} - k_n \geq 0$ (b) $k_{n+1} - k_n < 0$

5.7.3 Abschließendes Beispiel

Im abschließenden Beispiel soll die Polynomfunktion 3.Grades

$$g : \mathbb{R} \to \mathbb{R}, \quad g(x) = \frac{1}{27}\, x^3 + \frac{1}{9}\, x^2 - \frac{1}{3}\, x - 1 \tag{5.47}$$

durch ein neuronales Netz mit einem Hidden Layer mit 6 Neuronen approximiert werden. Dabei wird gemäß der Grafik 5.26 der Streckenzug $[A, B, C, D, E, F]$ mit $A(-6, g(-3))$, $B(-4, g(-4))$, $C(-3, g(-3))$, $D(0, g(0))$, $E(2, -1)$ und $E(4, g(4))$ verwendet.
Es ist also die folgende Funktion $f(x)$ zu erzeugen[10]:

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) = \begin{cases} 1.37x + 5.22 & \text{wenn} \quad x \in (-\infty, -4) \\ 0.26x + 0.78 & \text{wenn} \quad x \in [-4, -3) \\ -0.33x - 1 & \text{wenn} \quad x \in [-3, 0) \\ -1 & \text{wenn} \quad x \in [0, 2) \\ 1.41x + -3.81 & \text{wenn} \quad x \in [2, \infty) \end{cases} \tag{5.48}$$

[10] Alle numerischen Werte sind wenn notwendig auf 2 Dezimalstellen gerundet

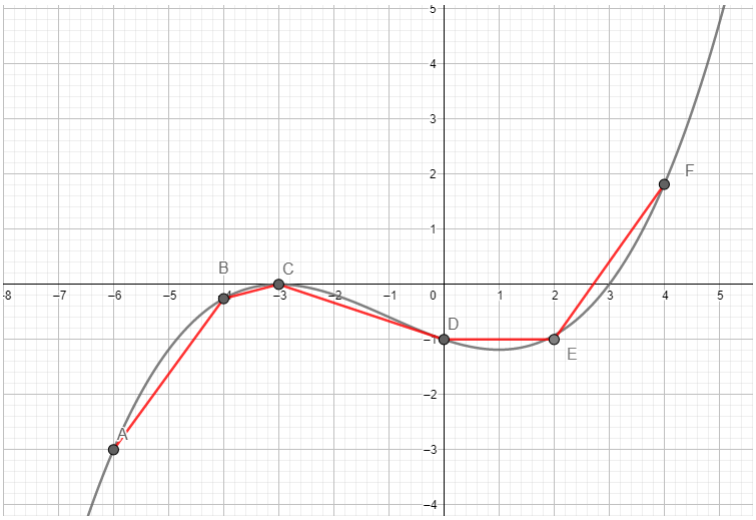


Abbildung 5.26: Ausgangssituation abschließendes Beispiel

Schritt 1

Es ist der Fall 4 ($k = 1.37 > 0$ und $b = 1.37 \cdot (-4) + 5.22 = -0.26 < 0$) zu berücksichtigen. Damit ergibt sich nach Durchführung von Schritt 1 das folgende Ergebnis:

```
f(x) = - RELU(-1.37(x+4)) - RELU(0.26) = - RELU(-1.37x-5.48) - RELU(0.26)
```

Die folgende Abbildung visualisiert das Ergebnis nach Schritt 1:

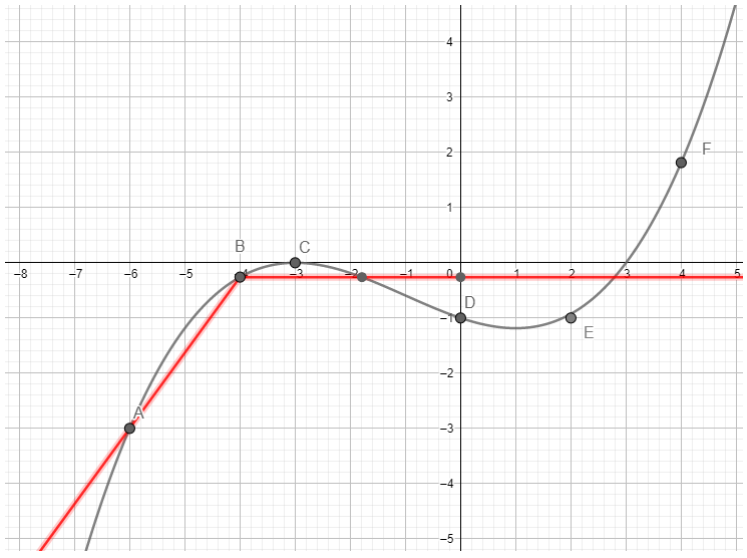


Abbildung 5.27: Abschließendes Beispiel - Schritt 1

Schritt 2 - Iteration 1

Wegen $k_2 - k_1 = 0.26 - 0 = 0.26 > 0$ ist gemäß 5.45 folgende Funktion zu addieren:

```
r(x) = RELU(0.26(x+4))
f(x) = f(x) + r(x) = - RELU(-1.37x-5.48) - RELU(0.26) + RELU(0.26x + 1.04)
```

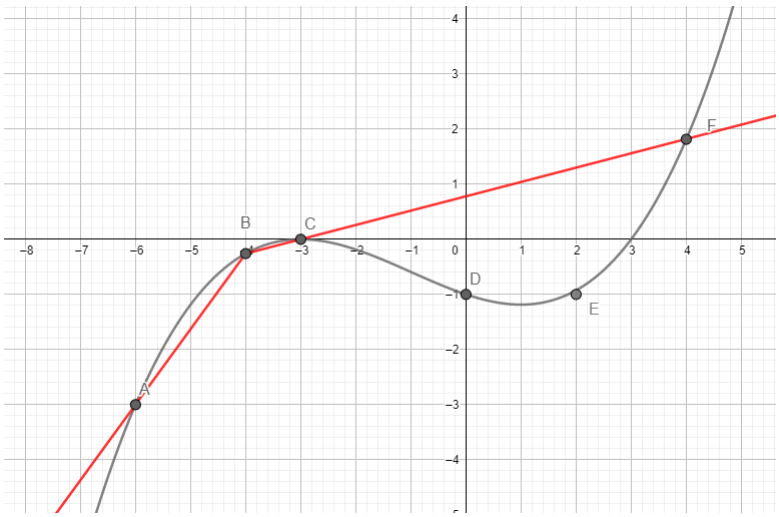


Abbildung 5.28: Abschließendes Beispiel - nach Schritt 2 1.Iteration

Schritt 2 - Iterationen 2, 3 und 4

Analog werden gemäß den Formeln 5.45 und 5.46 die nächsten 3 Iterationen durchgeführt:

- **Iteration 2** $k3 - k2 = -0.33 - 0.26 = -0.59 < 0$

```
r(x) = -RELU(0.59(x+3)) = -RELU(0.59x + 1.78)
f(x) = f(x) + r(x) =
     = - RELU(-1.37x-5.48) - RELU(0.26) + RELU(0.26x + 1.04) -RELU(0.59x + 1.77)
```

- **Iteration 3** $k4 - k3 = 0.00 - (-0.33) = 0.33 > 0$

```
r(x) = RELU(0.33(x-0)) = RELU(0.33x)
f(x) = f(x) + r(x) =
     = - RELU(-1.37x-5.48) - RELU(0.26) + RELU(0.26x + 1.04)
       - RELU(0.59x + 1.77) + RELU(0.33x)
```

- **Iteration 4** $k5 - k4 = 1.41 - 0.00 = 1.41 > 0$

```
r(x) = RELU(1.41(x-2)) = RELU(1.41x - 2.82)
f(x) = f(x) + r(x) =
     = - RELU(-1.37x-5.48) - RELU(0.26) + RELU(0.26x + 1.04)
       - RELU(0.59x + 1.77) + RELU(0.33x) + RELU(1.41x - 2.82)
```

Das Endergebnis visualisiert die folgende Grafik:

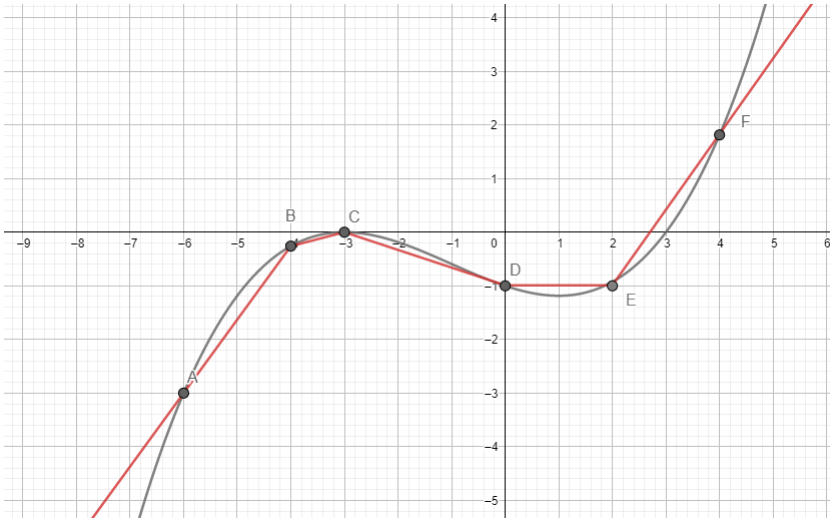
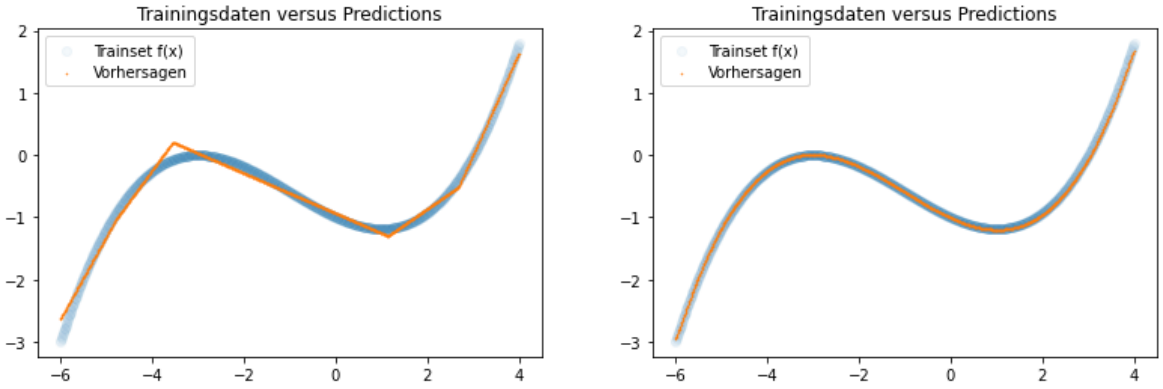


Abbildung 5.29: Abschließendes Beispiel - Endergebnis

Ergebnisse der Approximation mit Tensorflow / Keras

Approximiert man die Funktion $f(x)$ im Intervall $[-6, 4]$ mit Hife von Tensorflow/Keras gemäß Listing B.6 (es wurden nur die Definition von $f(x)$ sowie die Intervallgrenzen und die Anzahl der Neuronen im Hidden Layer angepasst), so ergeben sich mit 6 bzw. 100 Neuronen die in Abbildung 5.30 dargestellten Ergebnisse:



(a) (b)

Abbildung 5.30: (a) 6 Neuronen im Hidden Layer (b) 100 Neuronen im Hidden Layer

5.8 Praktische Anwendung neuronaler Netze

5.8.1 Lineare Regression durch ein ANN - Gebrauchtwagenpreise

Die im Abschnitt 4.1.3 durchgeführte lineare Regression zur Schätzung von Gebrauchtwagenpereisen wird in diesem Abschnitt mit Hilfe eines neuronalen Netzes durchgeführt. Das Listing im Anhang B.5 zeigt die Implementierung unter Keras-Tensorflow.

Im Folgenden werden für die ersten 10 Datensätze im Testdatensatz die tatsächlichen Verkaufspreise und die vom Netz prognostizierten Werte gegenübergestellt.

```
y_test:      [ 0.35 10.11  4.95  0.15   6.95  7.45  1.1   0.5   0.45  6.00 ]
predictions: [ 0.12 10.58  4.16  0.00  11.10  6.23  0.0   0.0   0.34  6.17 ]
Korrelation:  0.9439527679948129
```

Die folgende Abbildung visualisiert den Zusammenhang zwischen diesen Daten für alle Datensätze im Testset:

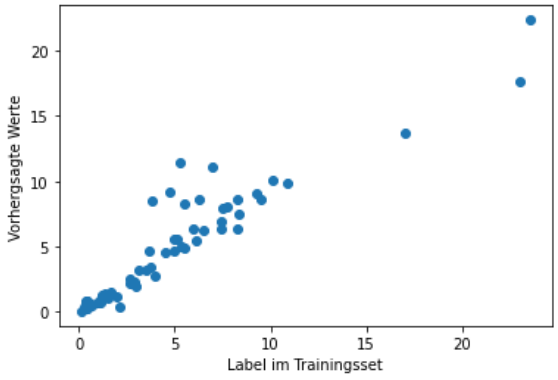


Abbildung 5.31: ANN: Lineare Regression - Tatsächlichen und vorhergesagte Werten

Die obigen Grafik und der Korrelationskoeffizent von 0.94 zwischen tatsächlichen und vorhergesagten Werten im Trainingsset zeigen eine hohe Güte des Modells. Diese erreicht man aber nur durch die Skalierung der Features (Zeilen 18-22 in Listing B.7). Damit sind die zu bestimmenden Gewichte annähernd im gleichen Wertebereich und können im Training ausreichend gut optimiert werden. Führt man das exakt gleiche Training ohne Featurescaling durch erhält man ein völlig unbefriedigendes Ergebnis. Der Korrelationskoeffizient beträgt bei einem konkreten Training nur mehr 0.028 und die folgende Grafik visualisiert wieder die Zusammenhänge zwischen Label und Vorhersagen:

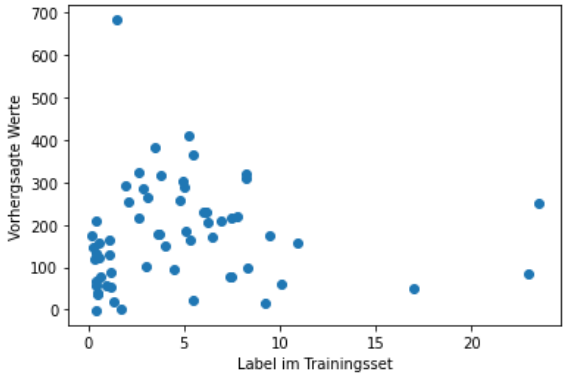


Abbildung 5.32: ANN: Lineare Regression - Güte ohne Featurescaling

5.8.2 Binäre Klassifikation durch ein ANN - Titanic Disaster Dataset

In diesem Abschnitt wird mit Keras-Tensorflow (Quellode im Anhang B.6)ein neuronales Netz entwickelt, das an Hand des Titanic Disaster Datasets eine binäre Klassifikation durchführt. Der Inputlayer nimmt pro Datensatz 8 Features entgegen (*Pclass*, *Age*, *SibSp*, *Parch*, *Fare*, *Q*, *S*). Das Netz besteht aus 2 Hidden Layer mit je 10 Neuronen und der Aktivierungsfunktion RELU, der Outputlayer aus einem Neuron und der Aktivierungsfunktion Sigmoid.Als Lossfunction wird Binary_Crossentropy (vgl. Formel 5.17) verwendet und als Sover GradientDecent.

Der Programmlauf liefert folgende Ergebnisse:

```
Score auf den Trainingsdaten:  0.8045006990432739
Score auf den Testdaten:       0.8146067261695862
[ [0.106] [0.854] [0.587] [0.206] [0.854] [0.102] [0.235] [0.131] [0.854] [0.854] ]
Vorhergesagte Klassenzugehörigkeit: [0 1 1 0 1 0 0 0 1 1]
Tatsächliche Klassenzugehörigkeit:  [0 1 1 0 1 1 0 0 1 1]
```

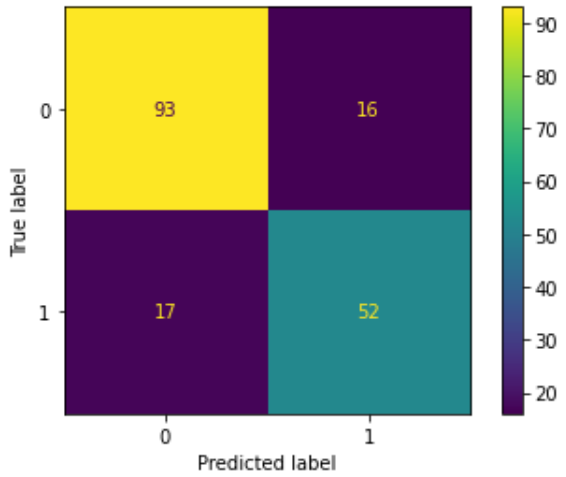


Abbildung 5.33: Logistische Regression - Titanic Disaster Dataset - Confusionmatrix

Kapitel 6

ML und KI im Unterricht der Sekundarstufe

6.1 Fazit

Gerade die Themen Machine Learning und Künstliche Intelligenz sind vergleichsweise junge und aktuelle Themen, die aber schon Einzug in sehr viele Bereiche unseres Alltags gefunden haben. Allein mit dem Smartphone benutzt jede*r Schüler*in täglich eine Technik, die mit Künstlicher Intelligenz arbeitet. KI wird momentan ständig weiterentwickelt, könnte eine Revolution auslösen und wird in der Gesellschaft sehr kontrovers diskutiert. Beispielsweise war der vor einigen Jahren verstorbene britische Astrophysiker Stephen Hawking der Meinung, dass Künstliche Intelligenz eine Bedrohung und sogar das Ende der Menschheit bedeuten könnte, vor allem, weil sie sich rascher entwickle als die biologische Evolution voranschreitet. Er selbst profitierte von einer Künstlichen Intelligenz, seinem Sprachcomputer, den er aufgrund seiner schweren Erkrankung benötigte, um sich anderen mitzuteilen. (vgl. (*29*), Online Ressource) Die Themen ML und KI haben daher auf jeden Fall einen Bezug zur Lebenswelt der Schüler*innen. Ein Ziel des Mathematikunterrichts ist die Erkenntnis, dass die Mathematik in vielen Bereichen des Lebens eine wichtige Rolle spielt. (vgl. (*10*), Online Ressource)

Eine Behandlung der KI im Unterricht könnte dazu beitragen, die Hintergründe dieses kontrovers diskutierten und zukunftsträchtigen Themas zu verstehen. Sie würde dabei helfen, die Schüler*innen mit dem nötigen technischen Wissen auszustatten, sodass sie einschätzen können, inwiefern Künstliche Intelligenz in Zukunft eine Gefahr oder eine Chance darstellen kann. Dies könnte die Schüler*innen mit der Fähigkeit ausstatten, dieses Thema für sich selbst kritisch zu reflektieren und sie damit unabhängig von den Ansichten etwaiger Meinungsmacher machen, was den demokratischen Prozess fördert.

Mit einem Thema wie Machine Learning und Künstlicher Intelligenz kann die Nützlichkeit der Mathematik in einem anwendungsorientierten Kontext verdeutlicht werden. Gerade dieses Thema ist prädestiniert für einen fächerübergreifenden Unterricht mit dem Fach Informatik. Besonders hier liegt ein Lernen mit technologischer Unterstützung auf der Hand, konnte ML und KI doch erst durch die voranschreitende Leistungsfähigkeit der Computer so erfolgreich werden. Für viele Inhalte kann Geogebra verwendet werden, darüber hinaus könnte in einem fächerübergreifenden Unterricht in Informatik eine Durchführung der verschiedenen Algorithmen mit gängigen Programmen, die auch tatsächlich in der Praxis verwendet werden, wie Python oder R, stattfinden. Eine häufig verwendete Python-Bibliothek für Machine Learning ist Scikit-Learn und für neuronale Netze Keras/Tensorflow von Google [1].

6.2 Beantwortung der Fragestellung

Das Ziel der Arbeit war die Beantwortung der Frage, inwieweit Inhalte aus dem Gebiet des Machine Learning und der Künstlichen Intelligenz im Rahmen des aktuell gültigen Mathematiklehrplans Anwendung im Unterricht der Sekundarstufe finden können. Dabei wurden die Schultypen AHS, HAK und HTL näher betrachtet.

Viele der mathematischen Grundlagen stimmen mit den Unterrichtsinhalten aus der Sekundarstufe im Fach Mathematik überein. So arbeiten gängige ML-Algorithmen zum Beispiel mit euklidischen Abständen (KNN), dem Satz von Bayes (Naive Bayes), oder linearer und logistischer Regression. Neuronale Netze arbeiten mit Vektoren, Funktionen und iterativen Verfahren. Die Datenvorverarbeitung deckt sich in vielen Teilen mit Lehrplaninhalten aus dem Bereich der beschreibenden Statistik, neben der Aufbereitung von Daten liegt der Fokus auf Darstellung und Interpretation derselben, beispielsweise in Form von Histogrammen, Boxplots oder Punktwolken. Aufgrund der Komplexität des Themas passt es eher in die Oberstufe als in die Unterstufe, obwohl auch in der Sekundarstufe I ein Schwerpunkt auf der beschreibenden Statistik liegt. Eine Voraussetzung für das gute Verständnis von Elementen aus dem Themenfeld des ML und der KI ist bestimmt ein Grundwissen über die Funktionsweise von Computern und ein grundlegendes Interesse an Künstlicher Intelligenz. Diese beiden Anforderungen sind vermutlich eher erst in der Oberstufe gegeben.

In der AHS-Oberstufe könnte ML und KI allerdings als großes und immer wiederkehrendes Thema interessierte Schüler*innen und ihre Lehrperson alle Schulstufen hindurch im Rahmen des Erweiterungsstoffes beziehungsweise als Anwendungsbeispiele im Kernstoff begleiten, weil es zu einer großen Anzahl der Lehrplanthemen passt. Natürlich können Inhalte aus dem Gebiet

[1] diese beiden Bibliotheken wurden auch bei der Erstellung der Quellcodes dieser Arbeit verwendet

mögliche Themen für vorwissenschaftliche Arbeiten, ein Wahlfach oder ein fächerübergreifendes Projekt darstellen.

In facheinschlägige HTLs mit Informatik-Schwerpunkt passt dieses Thema besonders gut, da hier das Interesse der Schüler*innen und der Lehrkräfte auf jeden Fall gegeben ist. Darüber hinaus ist hier sicherlich eine intensivere Auseinandersetzung mit dem Thema im Rahmen eines fächerübergreifenden Unterrichts möglich.

In die Handelsakademien passen ML und KI eher weniger und wenn dann ausschließlich punktuell, weil in dieser Schulform die Schwerpunkte anders gesetzt sind. Hier fanden sich auch die geringsten Übereinstimmungen der Inhalte mit dem Lehrplan. Jedoch arbeiten heute bereits viele Unternehmen mit Künstlicher Intelligenz (zum Beispiel in Form eines Chatbots für die Kommunikation mit den Kunden), was wiederum ein Argument dafür wäre, das Thema mindestens zu streifen.

Im Rahmen dieser Arbeit wurde an verschiedenen Stellen immer wieder versucht, Anreize und Ideen für den Einsatz dieses Themas im Unterricht bereitzustellen. Konkrete Umsetzungsvorschläge sprengen allerdings den Rahmen dieser Arbeit, können je nach Schulform und Klasse sehr unterschiedlich sein und liegen somit im Ermessen jeder Lehrperson.

Literatur

1. J. Kaplan, *Künstliche Intelligenz. Eine Einführung*, ger (Oxford University Press 2016, 1. Auflage 2017, 2017), 204 S., ISBN: 978-3-95845-633-4.
2. W. Ertel, *Grundkurs Künstliche Intelligenz*, ger (Springer Fachmedien Wiesbaden GmbH, Wiesbaden, 5. Auflage, 2020), 422 S., ISBN: 978-3-658-32075-1.
3. R. Otte, *Künstliche Intelligenz für dummies*, ger (2019 WILEY-VCH Verlag GmbH und Co. KGaA, Weinheim, 1. Auflage 2019, 2019), 456 S., ISBN: 978-3-527-71494-0.
4. G. Görz, U. Schmid, T. Braun, *Handbuch der Künstlichen Intelligenz*, ger (Walter de Gruyter GmbH, Berlin/Boston, 6. Auflage, 2021), 976 S., ISBN: 978-3-11-065984-9.
5. H. Helder, *Woebot will mit künstlicher Intelligenz gegen Depressionen helfen*, Stuttgarter Zeitung (2022; https://www.stuttgarter-zeitung.de/inhalt.chatbot-als-therapiestunde-woebot-will-mit-kuenstlicher-intelligenz-gegen-depressionen-helfen.70991e8f-d111-4a1e-8d20-12f3274ccc4f.html).
6. M. Bhattacharyya, *3 things you need to know before you train test split*, Towards Data Science (2022; https://towardsdatascience.com/3-things-you-need-to-know-before-you-train-test-split-869dfabb7e50).
7. S. K. Chinnamgari, *R Machine Learning Projects. Implement supervised, unsupervised and reinforcement learning techniques using R 3.5*, eng (Packt Publishing Ltd., Birmingham, 2019), 313 S., ISBN: 978-1-78980-794-3.
8. TechTarget, *garbage in, garbage out (GIGO)*, TechTarget (2021; https://www.techtarget.com/searchsoftwarequality/definition/garbage-in-garbage-out).
9. J. Cleve, U. Lämmel, *Data Mining*, ger (Oldenbourg, 2014), 318 S., ISBN: 978-3-486-71391-6.
10. *Verordnung des Bundesministers für Unterricht und Kunst vom 14. November 1984 über die Lehrpläne der allgemeinbildenden höheren Schulen; Bekanntmachung der Lehrpläne für den Religionsunterricht an diesen Schulen*, Bundesministerium für Unterricht und Kunst

(2022; https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10008568).

11. *Verordnung des Bundesministers für Unterricht und Kunst über die Lehrpläne für die Handelsakademie und die Handelsschule; Bekanntmachung der Lehrpläne für den Religionsunterricht*, Bundesministerium für Unterricht und Kunst (2022; https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10008944).

12. *Verordnung der Bundesministerin für Bildung und Frauen über die Lehrpläne der Höheren technischen und gewerblichen Lehranstalten 2015; Bekanntmachung der Lehrpläne für den Religionsunterricht*, Bundesministerium für Bildung und Frauen (2022; https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20009288).

13. Kaggle, *Titanic disaster dataset*, Kaggle (2021; https://www.kaggle.com/c/titanic/data).

14. Pandas, *Pandas*, Pandas (2021; https://pandas.pydata.org/).

15. M. Waskom, *seaborn: statistical data visualization*, Seaborn Reference (2021; https://seaborn.pydata.org/).

16. DeepAI, *One hot encoding defintion*, DeepAI (2021; https://deepai.org/machine-learning-glossary-and-terms/one-hot-encoding).

17. G. James, D. Witten, T. Hastie, R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, eng (Springer, New York, 2nd Edition 2021, 2021), 607 S., ISBN: 978-1-0716-1417-4.

18. R. Kwiatkowski, *Gradient Descent Algorithm a deep dive*, Towards Data Science (2021; https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21).

19. V. Pfleger, *Bestimmtheitsmaß*, INWTStatistics (2021; https://www.inwt-statistics.de/blog-artikel-lesen/Bestimmtheitsmass_R2-Teil2.html).

20. P. Willmot, *Machine Learning An applied Mathematics Introduction*, eng (Panda Ohana Publishing, First Edition, 2017), 226 S., ISBN: 978-1-9160816-0-4.

21. C. Yen-Chi, *Density Estimation: Histogram and Kernel Density Estimator*, University of Washington (2021; http://faculty.washington.edu/yenchic/18W_425/Lec6_hist_KDE.pdf).

22. *Scikit Learn Naive Bayes*, Scikit Learn Developers (2022; https://scikit-learn.org/stable/modules/naive_bayes.html).

23. J. Hale, *How Many Neurons Are In The Human Brain?*, Center for INQUIRY (2021; https://centerforinquiry.org/blog/how-many-neurons-are-in-the-human-brain-a-journey-to-find-the-answer/).

24. L. Grünschloß, *Perzeptron*, Universität Ulm (2022; http://www.informatik.uni-ulm.de/ni/Lehre/WS04/ProSemNN/pdf/perzeptron.pdf).

25. H.-J. Böhme, *Perzeptron*, Hochschule für Technik und Wirtschaft Dresden (2022; https://www2.htw-dresden.de/~boehme/Neuroinformatik/GNI_Prakt_TUI/ni_grundlagen_prak/deltalearning/deltalearning.html).

26. E. Hüll, *Das universelle Approximationstheorem für neuronale Netze*, Universität Wien, 2021.

27. S. Suranovic, D. Bingham, *Rosenbrock Function*, Simon Fraser University (2022; https://www.sfu.ca/~ssurjano/rosen.html).

28. K. Agrawal, A. Kanodia, *How do ReLU Neural Networks approximate any continuous function?*, Towards Data Science (2022; https://towardsdatascience.com/how-do-relu-neural-networks-approximate-any-continuous-function-f59ca3cf2c39).

29. R. Nestler, *Stephen Hawking sieht die Menschheit bedroht*, Der Tagesspiegel (2022; https://www.tagesspiegel.de/wissen/kuenstliche-intelligenz-stephen-hawking-sieht-die-menschheit-bedroht/11069712.html).

Anhang A

Zusammenfassung und Abstract

A.1 Zusammenfassung

Mit Künstlicher Intelligenz sind wir in unserem Alltag immer mehr und mehr konfrontiert. Sie ist eine junge wissenschaftliche Disziplin, die sich in einer rasanten Geschwindigkeit entwickelt und auf dem Weg ist, unser Leben grundlegend zu verändern. Gleichzeitig wird das Thema in der Gesellschaft kontrovers diskutiert. Im Zuge dieser Arbeit wurde untersucht, inwieweit Inhalte aus dem Gebiet des Machine Learning (ML) und der Künstlichen Intelligenz (KI), im Rahmen des aktuell gültigen österreichischen Mathematiklehrplans (AHS, HTL, HAK) Anwendung im Unterricht der Sekundarstufe finden können. Diverse Anwendungen wurden auf Basis von Literaturrecherche analog sowie computerunterstützt vor allem mit Geogebra und Python vorgestellt und mit den Inhalten der Lehrpläne verglichen. Darüber hinaus wurde an manchen Stellen versucht, konkrete Anwendungsbeispiele für den Einsatz in der Schule zu entwickeln. Das Thema ist stark mit dem Einsatz von Computern verknüpft, weshalb ein Grundwissen über die Funktionsweise derselben von Seiten der Schüler*innen vorhanden sein sollte.

Übereinstimmungen wurden bei den Lehrplanthemen Statistik und Wahrscheinlichkeit (Histogramme, Boxplots, Nominal- und Ordinalskalen, Lineare Abhängigkeit, Mittelwert, Standardabweichung, Lineare Regression, Interpretation von Wahrscheinlichkeiten, Bedingte Wahrscheinlichkeit), Funktionen und Differentialrechnung (Geradengleichungen, funktionale Zusammenhänge, stückweise definierte Funktionen, Polynomfunktionen, Funktionstransformationen, Ableitungen, verkettete Funktionen, partielle Ableitungen, Extremwertaufgaben, Funktionen in mehreren Veränderlichen, Korrelationskoeffizienten, Logistisches Wachstum), Vektorrechnung (Euklidische Abstände, Skalares Produkt, Normalvektorform einer Geraden) gefunden. Darüber hinaus treten iterative Prozesse und Fallunterscheidungen auf.

Aufgrund der Inhalte und der Komplexität eignet sich die Behandlung von ML und KI beson-

ders für den Unterricht in der Sekundarstufe II. Sie können sowohl in AHS als auch in BHS als Anwendungsgebiet der Mathematik vorgestellt werden. Besonders gut eignen sie sich für den Unterricht an Schulen mit einem Informatikschwerpunkt.

A.2 Abstract

In everyday life we nowadays are more and more confronted with Artificial Intelligence. It is a young scientific field, developing at enourmous speed on its way to change our life irrevocably. At the same time this topic is discussed as a controversy in our society. In the course of these thesis the author has researched, how contents from the field of Machine Learning (ML) and Artificial Intelligence (AI) can be used in context of the currently valid Austrian mathematics curriculum (AHS, HTL, HAK) for teaching in secondary schools. The author presents applications on the basis of analogue as well as computer assisted literature research (primarily with Geogebra and Python) and compares them to the content of the curriculum. Furthermore the author tries in several chapters to develop concrete tasks for practice in schools. In order to be able to understand the topic properly the pupils should already have basic knowledge of how computer science.

In the following topics of the curriculum there have been found matches to ML and AI by the author: statistics and probability (histograms, boxplots, nominal and ordinal scales, linear dependence, average, standard deviation, linear regression, interpretation of probabilities, conditional probability), functions and differentation (equation of a line, functional relations, piecewise defined functions, polynomial functions, function transformations, partial derivative, exercises with extreme values, functions with several variables, correlation coefficients, logistic growth), vector calculation (euclidean distances, dot product, normal form of a straight line) and also iterative processes and case distinctions.

Because of its contents and complexity the discussion of ML and AI is especially suited for teaching at advanced level of secondary schools. They can be presented as applications of the different mathematical topics in AHS as well as in BHS. They are specially well suited to be presented at schools specialising on computer science.

Anhang B

Python Quellcodes

B.1 Quellcode zum Erstellen der Grafiken in Abschnitt 3.3

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Titanic Dataset einlesen und die ersten beiden Datensaetze ausgeben
dataset = pd.read_csv('titanic_train_cleaned.csv')
print(dataset.head(2))

# Historamme
# Erzeugen von Abbildung 3.3
plt.figure(figsize = (16,5))
plt.subplot(121)
sns.histplot(data=dataset, x="Age", bins=[0,10,20,30,40,50,60,70,80])
plt.subplot(122)
sns.histplot(data=dataset, x="Age", hue="male", multiple="dodge", bins
    =[0,10,20,30,40,50,60,70,80])
plt.show()

# Erzeugen von Grafik 3.4
sns.histplot(data=dataset, x="Age", hue="Survived", multiple="dodge",
    bins=[0,10,20,30,40,50,60,70,80])
plt.show()

# Erzeugen von Grafik 3.5
plt.figure(figsize = (16,5))
plt.subplot(121)
sns.countplot(x="Survived",hue="Pclass",data=dataset)
plt.subplot(122)
sns.countplot(x="Survived",hue="male",data=dataset)
plt.show()

# Boxplot
# Erzeugen von Grafik 3.6
```

```
plt.figure(figsize = (16,5))
plt.subplot(121)
sns.boxplot(x=dataset["Age"], showfliers = False, whis=3)
plt.subplot(122)
sns.boxplot(x="Pclass", y="Age", data=dataset, showfliers=False, whis
    =2.2)
plt.show()

# Cars Dataset einlesen und die ersten beiden Datensaetze ausgeben
dataset = pd.read_csv('car_data.csv', delimiter=',')
print(dataset.head())

# Scatter und Regplot
# Erzeugen von Grafik 3.7

plt.figure(figsize = (16,5))
plt.subplot(121)
sns.scatterplot(x="Present_Price", y="Selling_Price", data=dataset)
plt.subplot(122)
sns.regplot(x="Present_Price", y="Selling_Price", data=dataset, ci=None)
plt.show()
# Korrelationkoeffizient berechnen
print(np.corrcoef(dataset['Present_Price'], dataset['Selling_Price'])
    [0,1])

# Neues Dataset erzeugen
cars = dataset.drop(['Car_Name', 'Fuel_Type', 'Seller_Type', '
    Transmission', 'Owner'], axis=1)
# Paiplt - Erzeugen von Grafik 3.8
sns.pairplot(cars)
plt.show()
# Matrix der Korrelationskoeffizienten erzeugen
print(cars.corr())
```

Listing B.1: Datenvisualisierung

B.2 Code zum Abschnitt Lineare Regression 4.1.3

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Dataset einlesen
dataset = pd.read_csv('car_data.csv', delimiter=',')

# One Hot Encoding fuer Seller_Type durchfuehren
seller = pd.get_dummies(dataset['Seller_Type'], drop_first=True)

# Matrix der Features erzeugen und anzeigen
X = dataset[['Year', 'Present_Price', 'Kms_Driven', 'Owner']]
```

```
X = pd.concat([X, seller], axis=1)
print(X.head())
print(X.info())
print(X.describe())

# Vektor der Label erzeugen
y = dataset['Selling_Price']

# Zusammenhaenge zwischen Year und Selling_Price bzw. Owner und
    Selling_Price
# visualisieren (Abbildung 4.2)
plt.figure(figsize = (16,5))
plt.subplot(121)
sns.regplot(x = 'Year', y = 'Selling_Price', data = dataset)
plt.subplot(122)
sns.regplot(x = 'Owner', y = 'Selling_Price', data = dataset)
plt.show()
# Entsprechende Korrelationskoeffizienten berechnen und ausgeben
print('Korrelation Year-Selling_Price         : ', np.corrcoef(X['Year'
    ], y )[0,1])
print('Korrelation Owner-Selling_Price        : ', np.corrcoef(X['Owner'
    ], y )[0,1])
print('Korrelation Present_Price-Selling_Price: ', np.corrcoef(X['
    Present_Price'], y )[0,1])

# Train-Test-Split (80% Trainingsdaten, 20% Testdaten)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
     random_state=42)

# Lineare Regression durchfuehren
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X, y)

# Ergebnisse ausgeben
print("Koeffizienten : ", model.coef_)
print("Intercept     : ", model.intercept_)
print('Trainingsscore: ', model.score(X_train, y_train))
print('Testscore     : ', model.score(X_test, y_test))

# Vorhesegen fuer das gesamte Dataset (Trainings- und Testdaten)
predictions = model.predict(X)
print('Korrelation zwischen Label und vorhergesagten Werten: ', np.
    corrcoef(y, predictions)[0,1])

# Residuen (Fehler) berechenen und analysieren
res = y - predictions

sns.distplot(res, bins=20)
plt.xlabel('Fehler bei der Vorhersage')
plt.show()
```

```
print("Mittewert der Fehler          : ", res.mean())
print("Standardabweichung der Fehler: ", res.std())

# Erzeugen von Abbildung 4.3
plt.figure(figsize = (16,5))
plt.subplot(121)
plt.scatter(y, predictions)
plt.xlabel('Tatsaechliche y-Werte')
plt.ylabel('Vorhergsagte y-Werte')
plt.subplot(122)
sns.distplot(res, bins=20)
plt.xlabel('Fehler bei der Vorhersage')
plt.show()
```

Listing B.2: Lineare Regression - Gebrauchtwagenmarkt

Ausgabe des Programms (ohne Grafiken):

```
   Year  Present_Price  Kms_Driven  Owner  Individual
0  2014           5.59       27000      0           0
1  2013           9.54       43000      0           0
2  2017           9.85        6900      0           0
3  2011           4.15        5200      0           0
4  2014           6.87       42450      0           0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 5 columns):
#   Column         Non-Null Count  Dtype
---  ------         --------------  -----
0   Year           301 non-null    int64
1   Present_Price  301 non-null    float64
2   Kms_Driven     301 non-null    int64
3   Owner          301 non-null    int64
4   Individual     301 non-null    uint8
dtypes: float64(1), int64(3), uint8(1)
memory usage: 9.8 KB
None
Year  Present_Price     Kms_Driven       Owner  Individual
count   301.000000     301.000000     301.000000  301.000000  301.000000
mean   2013.627907       7.628472   36947.205980    0.043189    0.352159
std       2.891554       8.644115   38886.883882    0.247915    0.478439
```

```
min     2003.000000       0.320000     500.000000     0.000000     0.000000
25%     2012.000000       1.200000   15000.000000     0.000000     0.000000
50%     2014.000000       6.400000   32000.000000     0.000000     0.000000
75%     2016.000000       9.900000   48767.000000     0.000000     1.000000
max     2018.000000      92.600000  500000.000000     3.000000     1.000000
Korrelation Year-Selling_Price         :  0.23614098016042734
Korrelation Owner-Selling_Price        :  -0.0883440990872023
Korrelation Present_Price-Selling_Price:  0.8789825451614951
Koeffizienten :  [ 4.53828641e-01  4.93404940e-01 -1.86971786e-06 -6.84430542e-01
-1.14815946e+00]
Intercept     :  -912.4416701573135
Trainingsscore:  0.8653938775004216
Testscore     :  0.8388146354270003
Korrelation zwischen Label und vorhergesagten Werten:  0.9276758414485455
```

B.2.1 Code zum Abschnitt Logistische Regression 4.2.3

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Dataset einlesen und die ersten beiden Datensaetze ausgeben
dataset = pd.read_csv('titanic.csv')
print(dataset.head(2))

# Matrix X der Features und Vektor y der Label erzeugen
X = dataset.drop(['Survived', 'PassengerId'], axis=1)
y = dataset['Survived']

# Train-Test-Split (80% Trainingsdaten, 20% Testdaten)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

# Logistische Regession durchfuehren
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='newton-cg')
model.fit(X_train, y_train)

# Score fuer Trainings- und Testdaten ausgeben
print('Score auf den Trainingsdaten: ', model.score(X_train, y_train))
print('Score auf den Testdaten:      ', model.score(X_test, y_test))

# Vorhergesagte Wahrscheinlichkeiten und Klassenzugehoerigkeiten fuer
    die Testdaten
```

```
27  # berechnen und die ersten 10 Datensaetze die Ergebnisse ausgeben
28  predictions_proba = model.predict_proba(X_test)
29  predictions = model.predict(X_test)
30  print(predictions_proba[:10])
31  print('Vorhergesagte Klassenzugeoerigkeit: ', predictions[:10])
32  print('Tatsaechliche Klassenzugehoerigkeit: ', y_test.values[:10])
33
34  # Confusionmatrix berechnen und visualisieren
35  # Abbildung 4.6. erzeugen
36  cm = confusion_matrix(y_test, predictions, labels=model.classes_)
37  disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=
        model.classes_)
38  disp.plot()
39  plt.show()
```

Listing B.3: Logistische Regression - Titanic Disaster Dataset

Ausgabe des Programms (ohne Grafiken):

```
   PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare  male  Q  S
0            1         0       3  22.0      1      0   7.2500     1  0  1
1            2         1       1  38.0      1      0  71.2833     0  0  0
Score auf den Trainingsdaten:  0.8073136427566807
Score auf den Testdaten     :  0.7921348314606742
[[0.90199939 0.09800061]
[0.03959998 0.96040002]
[0.24101603 0.75898397]
[0.74842691 0.25157309]
[0.03445491 0.96554509]
[0.89778239 0.10221761]
[0.70045147 0.29954853]
[0.86776022 0.13223978]
[0.07748579 0.92251421]
[0.03368887 0.96631113]]
Vorhergesagte Klassenzugeoerigkeit:  [0 1 1 0 1 0 0 0 1 1]
Tatsaechliche Klassenzugehoerigkeit:  [0 1 1 0 1 1 0 0 1 1]
```

B.3 Code zum Abschnitt KNN 4.3.3

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3  from sklearn.model_selection import train_test_split
4  from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
5
6   # Dataset einlesen und die ersten beiden Datensaetze ausgeben
7   dataset = pd.read_csv('titanic.csv')
8   print(dataset.head(2))
9
10  # Matrix X der Features und Vektor y der Label erzeugen
11  X = dataset.drop(['Survived', 'PassengerId'], axis=1)
12  y = dataset['Survived']
13
14  # Train-Test-Split (80% Trainingsdaten, 20% Testdaten)
15  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
        =0.2, random_state=42)
16
17  # Logistische Regession durchfuehren
18  from sklearn.linear_model import LogisticRegression
19  model = LogisticRegression(solver='newton-cg')
20  model.fit(X_train, y_train)
21
22  # Score fuer Trainings- und Testdaten ausgeben
23  print('Score auf den Trainingsdaten: ', model.score(X_train, y_train))
24  print('Score auf den Testdaten:      ', model.score(X_test, y_test))
25
26  # Vorhergesagte Wahrscheinlichkeiten und Klassenzugehoerigkeiten fuer
        die Testdaten
27  # berechnen und die ersten 10 Datensaetze die Ergebnisse ausgeben
28  predictions_proba = model.predict_proba(X_test)
29  predictions = model.predict(X_test)
30  print(predictions_proba[:10])
31  print('Vorhergesagte Klassenzugeoerigkeit: ', predictions[:10])
32  print('Tatsaechliche Klassenzugehoerigkeit: ', y_test.values[:10])
33
34  # Confusionmatrix berechnen und visualisieren
35  # Abbildung 4.6. erzeugen
36  cm = confusion_matrix(y_test, predictions, labels=model.classes_)
37  disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=
        model.classes_)
38  disp.plot()
39  plt.show()
```

Listing B.4: Logistische Regression - Titanic Disaster Dataset

Ausgabe des Programms (ohne Grafiken):

```
   PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare  male  Q  S
0            1         0       3  22.0      1      0   7.2500     1  0  1
1            2         1       1  38.0      1      0  71.2833     0  0  0
Score auf den Trainingsdaten:  0.8720112517580872
Score auf den Testdaten:       0.7696629213483146
[[1.  0. ]
[0.  1. ]
```

```
[0.4 0.6]
[0.6 0.4]
[0.  1. ]
[0.8 0.2]
[1.  0. ]
[1.  0. ]
[0.  1. ]
[0.  1. ]]
Vorhergesagte Klassenzugeörigkeit:  [0 1 1 0 1 0 0 0 1 1]
Tatsächliche Klassenzugehörigkeit:  [0 1 1 0 1 1 0 0 1 1]
```

B.3.1 Code zum Abschnitt Naive Bayes 4.4.4

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Dataset einlesen und die ersten 2 Datensaetze anzeigen
dataset = pd.read_csv('titanic_train_cleaned.csv')
print(dataset.head(2))

# Matrix X der Features und Vektor y der Label erzeugen und
# Train-Test-Split durchfuehren
X = dataset.drop(['Survived', 'PassengerId'], axis=1)
y = dataset['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)


# Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, y_train)

# Score fuer Trainings- und Testdaten ausgeben
print('Score auf den Trainingsdaten: ', model.score(X_train, y_train))
print('Score auf den Testdaten:      ', model.score(X_test, y_test))

# Vorhergesagte Wahrscheinlichkeiten und Klassenzugehoerigkeiten fuer
    die Testdaten
# berechnen und die ersten 10 Datensaetze die Ergebnisse ausgeben
predictions_proba = model.predict_proba(X_test)
predictions = model.predict(X_test)
print(predictions_proba[:10])
print('Vorhergesagte Klassenzugeoerigkeit: ', predictions[:10])
print('Tatsaechliche Klassenzugehoerigkeit: ', y_test.values[:10])
```

```
33
34  # Confusionmatrix berechnen und visualisieren
35  # Abbildung 4.8. erzeugen
36  cm = confusion_matrix(y_test, predictions, labels=model.classes_)
37  disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=
        model.classes_)
38  disp.plot()
39  plt.show()
```

Listing B.5: Logistische Regression - Titanic Disaster Dataset

Ausgabe des Programms (ohne Grafiken):

```
PassengerId  Survived  Pclass   Age  SibSp  Parch     Fare  male  Q  S
0            1         0       3  22.0      1      0   7.2500     1  0  1
1            2         1       1  38.0      1      0  71.2833     0  0  0
Score auf den Trainingsdaten:  0.8720112517580872
Score auf den Testdaten:       0.7696629213483146
[[1.  0. ]
[0.  1. ]
[0.4 0.6]
[0.6 0.4]
[0.  1. ]
[0.8 0.2]
[1.  0. ]
[1.  0. ]
[0.  1. ]
[0.  1. ]]
Vorhergesagte Klassenzugeoerigkeit:  [0 1 1 0 1 0 0 0 1 1]
Tatsächliche Klassenzugehoerigkeit:  [0 1 1 0 1 1 0 0 1 1]
```

B.4 Pythonskript zur Funktionsapproximation

```
1   import numpy as np
2   import matplotlib.pyplot as plt
3   from keras.models import Sequential
4   from keras.layers import Dense
5
6   # f(x) definieren
7   def f(x):
8   if x < 1:
9       return -x + 2
10  elif x <= 5:
```

```
    return 0.5 * x + 0.5
else:
    return 1.5 * x - 4.5

# Trainingsdaten erzeugen
X = np.arange(-4, 7, 0.01).reshape(-1, 1)
y = np.array([f(e) for e in X]).reshape(-1, 1)

# Das neuronale Netz definieren
model = Sequential()
# Hidden Layer, 4 Neuronen, Aktivierungsfunktion RELU
model.add(Dense(4, input_dim=1, activation='relu'))
# Outputlayer, 1 Neuron, keine Aktivierungsfunktion
model.add(Dense(1))
# Lossfunktion und Optimizer definieren
model.compile(loss='mse', optimizer='rmsprop')
# Modell trainieren
model.fit(X, y, epochs=500, batch_size=10, verbose=1)
# Berechnete Gewichte ausgeben
print(model.get_weights())
# Vorhersagen berechnen
y_hat = model.predict(X)

# Ergebnisse visualieren (Abbildung 5.22)
plt.scatter(X, y, alpha=0.05, label="Trainset f(x)")
plt.scatter(X, y_hat, label="Vorhersagen", s=0.3)
plt.title('Trainingsdaten versus Predictions')
plt.legend()
plt.show()
```

Listing B.6: Neuronales Netz zur Approximation von f(x)

B.5 Pythonscript: Lineare Regression durch ein ANN

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv('car_data.csv', delimiter=',')
dataset.drop(['Car_Name'], axis=1, inplace=True)
dataset.dropna(inplace=True)
seller = pd.get_dummies(dataset['Seller_Type'], drop_first=True)

X = dataset[['Year', 'Present_Price', 'Kms_Driven', 'Owner']]
X = pd.concat([X, seller], axis=1)
y = dataset['Selling_Price']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)
```

```
17
18  from sklearn.preprocessing import MinMaxScaler
19  scaler = MinMaxScaler()
20  scaler.fit(X_train)
21  X_train_scaled = scaler.transform(X_train)
22  X_test_scaled = scaler.transform(X_test)
23
24  from keras.models import Sequential
25  from keras.layers import Dense
26  model = Sequential()
27  # Hidden Layer, 10 Neuronen, Aktivierungsfunktion RELU, Inputdimesion
        = 5
28  model.add(Dense(10, input_dim=5, activation='relu'))
29  # Outputlayer, 1 Neuron, keine Aktivierungsfunktion
30  model.add(Dense(1))
31  # Lossfunktion und Optimizer definieren
32  model.compile(loss='mse', optimizer='rmsprop', metrics=['accuracy'])
33  # Modell trainieren
34  model.fit(X_train_scaled, y_train, epochs=500, batch_size=10, verbose
        =1)
35
36  # Vorhersagen berechnen und die ersten 10 Werte im Testset vergleichen
37  predictions = model.predict(X_test_scaled)
38  print('y_test:      ', y_test[:10])
39  print('predictions: ',predictions[:10])
40
41  # Korrelationskoeffizient zwischen Label und Vorhersagewerten
        berechnen
42  print('Korrelation: ', np.corrcoef(y_test, predictions.reshape(-1))
        [0,1])
43
44  #Zusammenhang plotten
45  plt.scatter(y_test, predictions)
46  plt.xlabel('Label im Trainingsset')
47  plt.ylabel('Vorhergsagte Werte')
48  plt.show()
```

Listing B.7: ANN Lineare Regression - Gebrauchtwagenpreise

B.6 Pythonscript: Binäre Klassifikation durch ein ANN

```
1   import pandas as pd
2   import numpy as np
3   import matplotlib.pyplot as plt
4   from sklearn.model_selection import train_test_split
5
6   from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
7   from keras.models import Sequential
8   from keras.layers import Dense
9
10  # Daten einlesen
```

```
dataset = pd.read_csv('titanic_train_cleaned.csv')
print(dataset.head())

# Features und Label definieren, Train-Test-Split
X = dataset.drop(['Survived', 'PassengerId'], axis=1)
y = dataset['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

# Features skalieren
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)  #Trainieren des Scalers nur mit X_train
X_train_scaled =scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Das neuronale Netz definieren
model = Sequential()
# Hidden Layer, 4 Neuronen, Aktivierungsfunktion RELU
model.add(Dense(10, input_dim=8, activation='relu'))
# Outputlayer, 1 Neurin, keine Aktivierungsfunktion
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Lossfunktion und Optimizer definieren
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['
    accuracy'])
# Modell trainieren
model.fit(X_train, y_train, epochs=500, batch_size=10, verbose=1)

# Score berechnene und ausgeben
print('Score auf den Trainingsdaten: ', model.evaluate(X_train,
    y_train)[1])
print('Score auf den Testdaten:      ', model.evaluate(X_test, y_test)
    [1])

# Predictions berechnen und die ersten 10 Ergebnisse ausgeben
y_test_predict = model.predict(X_test)
predictions = np.round(y_test_predict.reshape(-1)).astype(int)
print('Vorhergesagte Klassenzugehoerigkeit: ',predictions[:10])
print('Tatsaechliche Klassenzugehoerigkeit:  ', y_test.values[:10])

# Confusionmatrix visualisieren
cm = confusion_matrix(y_test.values, predictions, labels=[0,1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels
    =[0,1])
disp.plot()
plt.show()
```

Listing B.8: ANN Binäre Klassifikation - Titanic Disaster Dataset

Anhang C

Geogebra

C.1 Lineare Regression mit Geogebra

Im Folgenden wird erklärt, wie man mit Geogebra eine lineare Regression durchführen kann. Dazu wird die Punktwolke $P_1(1,1)$, $P_2(2,4)$, $P_3(2.5,3)$, $P_4(4,3.5)$, $P_5(5,5)$, $P_6(5.5,4)$, $P_7(6,5)$, $P_8(7,6.5)$ herangezogen.

1. Man öffnet die Tabellenansicht und gibt die Ausgangsdaten ein. Anschließend markiert man den Bereich in der Tabelle und wählt wie die Grafik C.1 zeigt den Punkt "Analyse zweier Variablen."

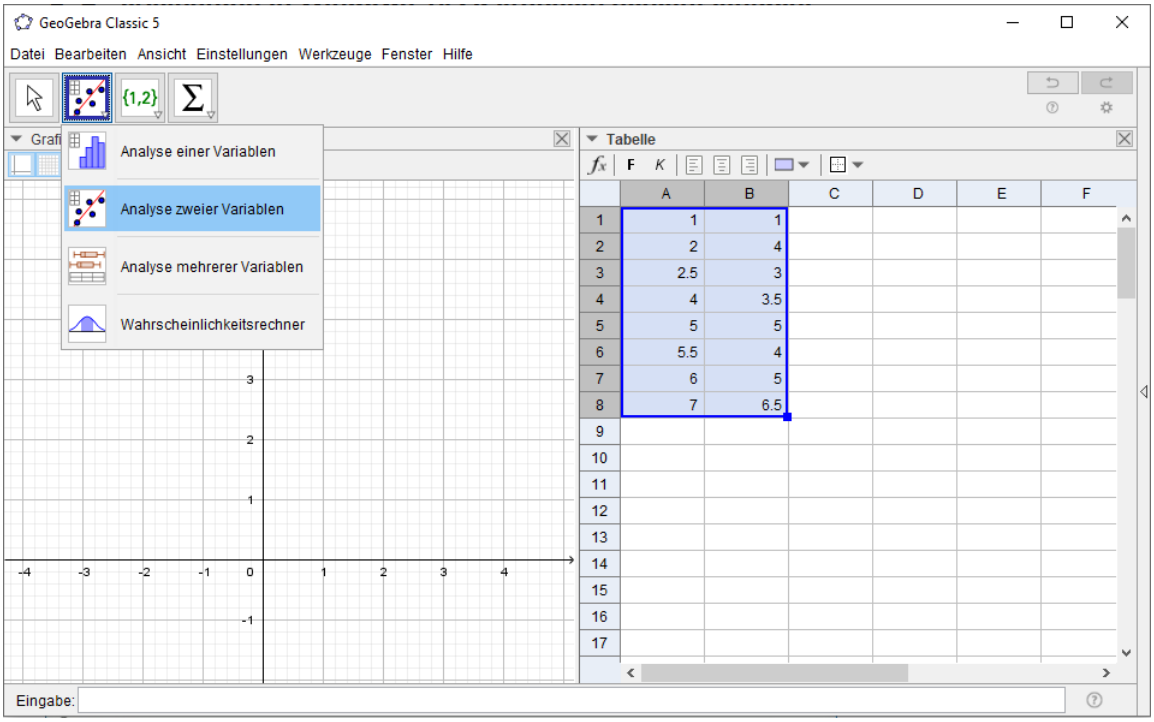


Abbildung C.1: Einfache lineare Regression mit Geogebra 1

2. In dem nun sich öffnenden Fenster "Datenanalyse" kann man nun das gewünschte Regressionsmodell (im aktuellen Beispiel linear) wählen und erhält sowohl eine grafische Darstellung als auch die Gleichung der Regressionsgeraden. Außerdem lässt sich über den Punkt "Statistik anzeigen" eine Übersicht der numerischen Parameter (z.B. Korrelationskoeffizient, Bestimmtheitsmaß) anzeigen.

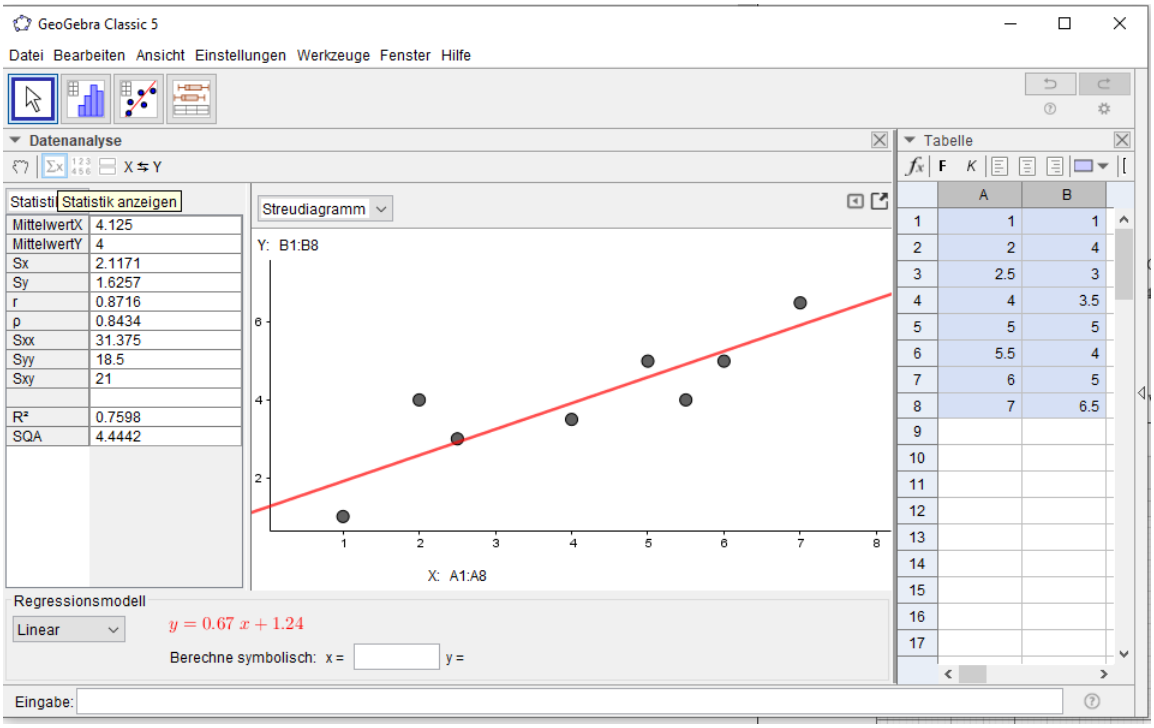


Abbildung C.2: Einfache lineare Regression mit Geogebra 2

C.2 Visualisierung der 4 Fälle zum Schritt 1 der Funktionsapproximation gemäß Abschnitt 5.7.2 mit Geogebra

Der Schritt 1 bei der Approximation einer Funktion durch eine neuronales Netz erfordert die Unterscheidung von 4 Fällen. Diese können für verschiedene Werte von `k`, `a` und `b` mit Hilfe von Schiebereglern in einfacher Form mit Geogebra veranschaulicht werden. Hier kann auch schön gezeigt werden, was passiert, wenn man einen konkreten Fall mit falschen Werten versorgt.
Da die in der Eingabezeile zu erfassenden Befehle im Algebrafenster benutzerfreundlich visualisiert werden, sind sie im Folgenden in der Originalsyntax angegeben:

```
RELU(x) = Wenn(x < 0, 0, x)
F1(x) = RELU(k (x - a)) + RELU(b)
F2(x) = RELU(k (x - a)) - RELU(-b)
F3(x) = -RELU(-(k (x - a))) + RELU(b)
```

```
F4(x) = -RELU(-(k (x - a))) - RELU(-b)
```

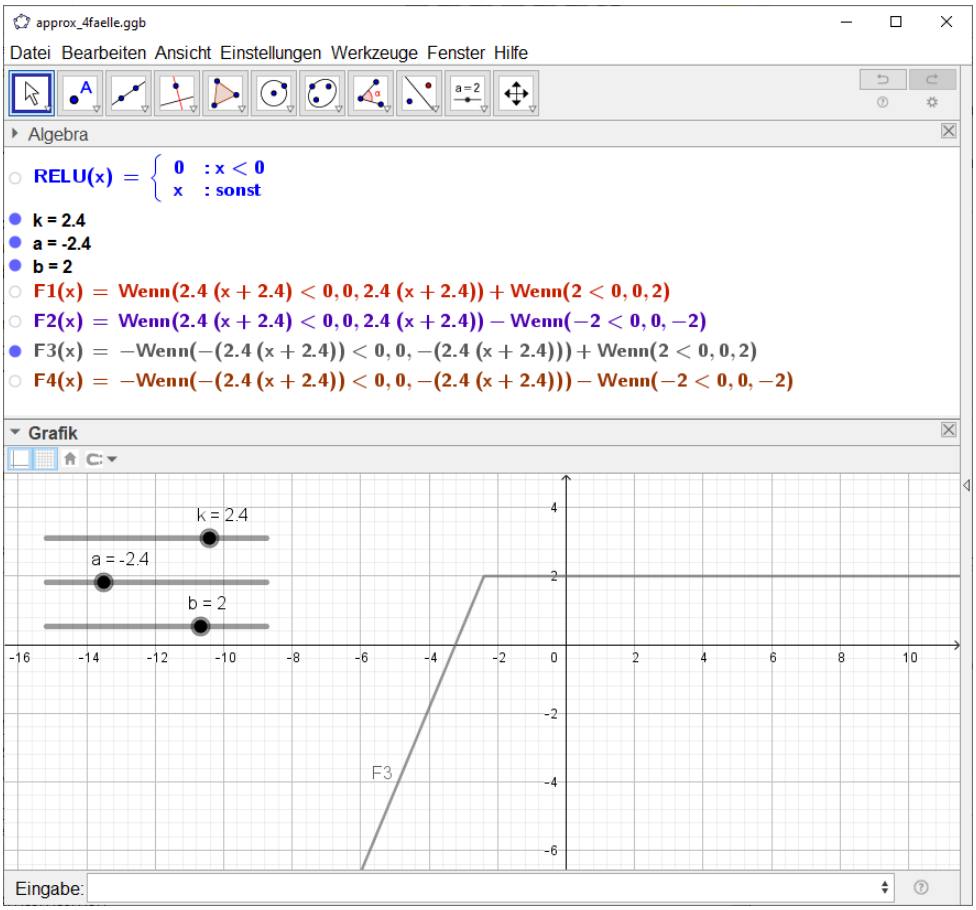


Abbildung C.3: 4 Fälle bei Schritt 1 der Funktionsapproximation mit Geogebra

C.3 Erstellen von Grafik 5.20 mit Geogebra

Die folgende Abbildung zeigt die Erstellung von Abbildung 5.20 mit Geogebra.
Wieder werden die Befehle auch in der Originalsyntax angegeben, da diese aus dem Algebra-finster nicht ersichtlich sind.

```
RELU(x) = Wenn(x < 0, 0, x)
n1(x) = RELU(-x + 1)
n2(x) = RELU(0.5*x - 0.5)
n3(x) = RELU(1)
n4(x) = RELU(x - 5)
A = (-4,6)
B = (1,1)
C = (5,3)
```

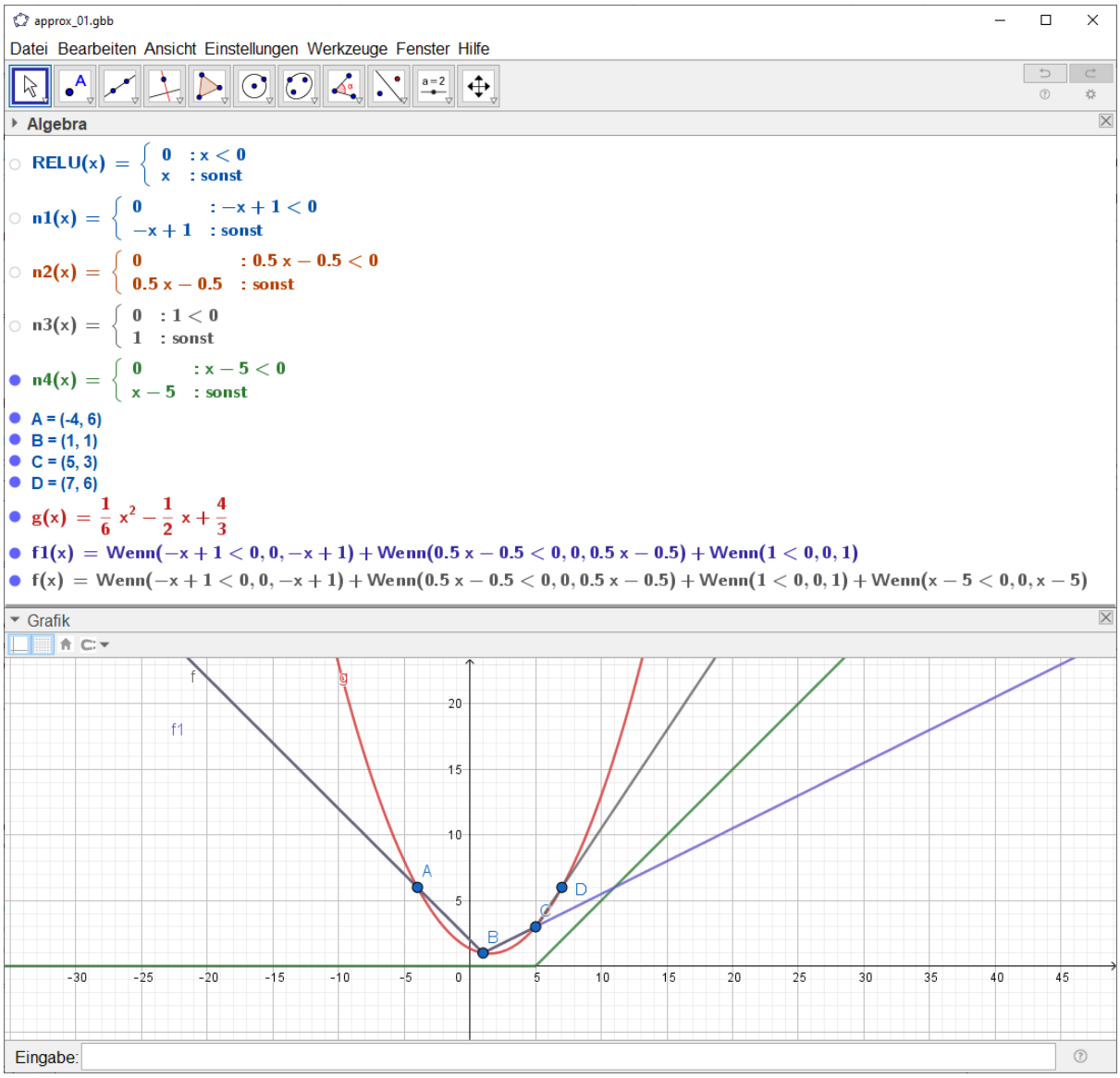


Abbildung C.4: Erstellen von Grafik 5.20

```
D = (7,6)

g(x) = 1 / 6 x² - 1 / 2 x + 4 / 3

f1(x) = n1(x) + n2(x) + n3(x)

f2(x) = n1(x) + n2(x) + n3(x) + n4(x)
```