Machine Learning für das KMU

Martin Sterchi

2024-03-19

Contents

Über das Buch

Die Motivation für dieses Buch kam aus der Erkenntnis, dass viele kleine und mittelgrosse Unternehmen (KMU) in der Schweiz zwar über grosse Datenmengen verfügen, aber nicht das nötige Knowhow haben, um die Daten zu analysieren und für die Optimierung von Entscheidungsprozessen zu nutzen. Mit diesem Buch möchte ich einen kleinen Beitrag leisten, den Knowhow Transfer von Fachhochschulen in die Unternehmen zu katalysieren.

Das Buch versucht, sowohl die klassischen Machine Learning Methoden als auch neueste Entwicklungen im Deep Learning mit einem Fokus auf die Anwendung zu vermitteln. Deep Learning kann als eine Teilmenge des Machine Learnings gesehen werden. Das heisst, jede Deep Learning Methode ist automatisch auch eine Machine Learning Methode. Machine Learning entält jedoch weitere Methoden, welche nicht dem Deep Learning zugeordnet werden können. Das Gebiet Machine Learning ist wiederum eine Teilmenge der Methoden der Künstlichen Intelligenz. Letztere enthält weitere Methoden, welche nicht dem Machine Learning zuzuordnen sind. Abbildung 1 versucht diesen Sachverhalt schematisch darzustellen.

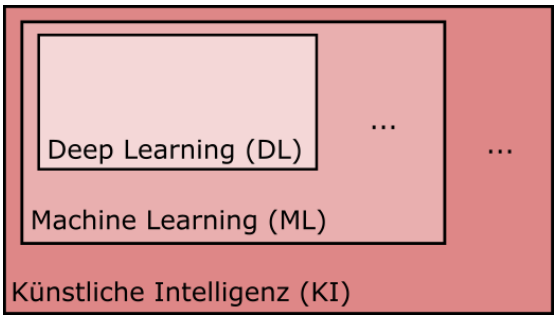


Figure 1: Unterscheidung zwischen KI, ML und DL.

Wir werden im ganzen Buch die folgenden (üblichen) Abkürzungen verwenden:

- Künstliche Intelligenz = KI (oft spricht man auch von AI, was die Abkürzung für den englischen Begriff *Artificial Intelligence* ist).

- Machine Learning = ML
- Deep Learning = DL

Obwohl das Buch einen anwendungsorientierten Ansatz verfolgt, soll die mathematisch-statistische Intuition hinter den beschriebenen Modellen und Methoden nicht zu kurz kommen. Diese Intuition ist aus meiner Sicht zwingend, um beurteilen zu können, ob sich ein Modell überhaupt für ein gegebenes Problem eignet. Am Schluss geht es nämlich darum, dass wir mit dem Einsatz von Machine Learning einen Mehrwert für ein Unternehmen oder für die Gesellschaft schaffen können. Das erfordert, dass wir uns eingehend und kritisch mit den Modellen und deren Eignung für ein gegebenes Problem auseinander setzen.

Zielgruppe

Das Buch richtet sich insbesondere an Fachhochschulstudierende in der deutschsprachigen Schweiz mit einem intrinsischen Interesse an quantitativen Methoden im Allgemeinen und Machine Learning im Besonderen. Vorausgesetzt werden Mathematikkenntnisse auf Stufe Mittelschule (Berufs- oder gymnasiale Matur), d.h. Sie sollten vertraut sein mit den Grundlagen bezüglich mathematischer Funktionen, der Integral- und Differentialrechnung sowie den wichtigsten Resultaten aus der Algebra. Ausserdem gehe ich davon aus, dass Sie bereits eine Einführung in das Thema Statistik besucht haben und Konzepte aus der deskriptiven Statistik (Mittelwert, Median, Varianz, Quantile, etc.) sowie aus der Inferenzstatistik (Verteilungen, statistisches Testen, etc.) bekannt sind.

Bevor Sie sich aber nun Sorgen machen: Kapitel 2 enthält eine Einführung in die wichtigsten Mathematik- und Statistikgrundlagen, die nötig sind für das Verständnis von Machine Learning Modellen.

Da ich mit diesem Buch einen anwendungsorientierten Ansatz verfolge, werden wir auch in das Programmieren einsteigen. Dazu verwenden wir in diesem Buch die Programmiersprache `R`. Es werden keine Vorkenntnisse vorausgesetzt. Kapitel 3 enthält eine kurze Einführung in die Programmiersprache `R` und verweist Sie auf weiterführende Ressourcen zum Thema Programmieren. Jedes Modell, das wir uns anschauen werden, ist mit R-Code dokumentiert, so dass Sie lernen, wie die Modelle in der Praxis angewendet werden können.

Aufbau des Buchs

Das Buch enthält folgende Kapitel:

- Kapitel 1: Einführung in das Thema Machine Learning mit **Definitionen** sowie Anwendungsbeispielen.
- Kapitel 2: Wichtigste **Mathematik- und Statistikgrundlagen**, die für das Verständnis der Modelle in den späteren Kapitel elementar sind.
- Kapitel 3: Einführung in das **Programmieren** mit `R` sowie Überblick über die wichtigsten `R`-Packages, die wir verwenden werden.
- Kapitel 4: Hier erlernen wir die Grundmodelle, um **Regressionsprobleme** zu lösen. Es sind lineare Modelle, was bedeutet, dass die funktionale Form der Modelle linear von den Parametern des Modells abhängen. Grafisch bedeutet dies, dass ein solches Modell im einfachsten Fall durch eine Gerade beschrieben werden kann.
- Kapitel 5: In diesem Kapitel lernen wir die Grundmodelle für das **Klassifikationsproblem** kennen. Diese Modelle führen typischerweise zu einer linearen Entscheidungsgrenze (engl. *Decision Boundary*) zwischen den verschiedenen Klassen, die wir unterscheiden oder klassifizieren wollen.
- Kapitel 6: Damit wir ML in der Praxis anwenden können, lernen wir hier die typische **ML-Pipeline** kennen. Sie werden die Techniken und Methoden kennen lernen, die es braucht, um überhaupt erst an den Punkt zu kommen, um ein ML-Modell rechnen zu können. Oft werden diese Techniken und Methoden unter dem Begriff Preprocessing der Daten zusammengefasst. Doch die Pipeline endet nicht mit dem Rechnen eines ML-Modells. Danach muss ein Modell evaluiert werden und wenn Sie als Analyst*in zufrieden sind, müssen Sie sich Gedanken machen, wie das Deployment des Modells aussehen soll. Das heisst, wie kann Ihr Modell Dritten zur Verfügung gestellt werden? Wir werden uns hier auch kurz mit den wichtigsten Techniken aus dem Unsupervised Learning befassen.
- Kapitel 7: Nach den ersten linearen Modellen für das Regressions- und Klassifikationsproblem lernen wir hier ein flexibleres Modell kennen, nämlich den **Entscheidungsbaum** (engl. *Decision Tree*). Entscheidungsbäume eignen sich sowohl für das Regressions- als auch für das Klassifikationsproblem. Obwohl sie in realen Projekten typischerweise anderen Modellen unterlegen sind, wenn es um die Vorhersagequalität geht, sind sie trotzdem attraktive Modelle, da sie gut visualisierbar sind.
- Kapitel 8: Aufbauend auf den Entscheidungsbäumen aus dem vorherigen Kapitel können sehr mächtige Modelle erstellt werden, die in der Praxis oft die besten Vorhersagen liefern. Weil es sich dabei üblicherweise um eine clevere Aggregierung der Resultate einer grossen Anzahl individueller Entscheidungsbäume handelt, werden diese Modelle **Ensembles** genannt. Wie die individuellen Entscheidungsbäume eignen sich Ensembles sowohl für das Regressions- als auch für das Klassifikationsproblem.
- Kapitel 9: Ein weiteres mächtiges Modell, das sich sowohl für das Regressions- als auch für das Klassifikationsproblem eignet, sind die **Support Vector Machines**. Ihre Popularität ist mit dem Aufstieg von Deep Learning etwas verblasst. Es lohnt sich aber immer noch allemal, diese Familie von Modellen kennen zu lernen, insbesondere auch weil sie nicht als Blackbox-Modelle gelten und theoretisch gut fundiert sind.

- Kapitel 10: Ab diesem Kapitel steigen wir in das Thema Deep Learning ein. Sie werden die Architektur von einfachen **Articial Neural Networks** (ANNs) kennen lernen. Ausserdem schauen wir uns in diesem Kapitel den genialen Backpropagation Algorithmus anhand eines einfachen linearen Regressionsproblems an. Dieser Algorithmus ist der Schlüssel für die viel diskutierten Fortschritte im Bereich der künstlichen Intelligenz, weil er das Trainieren von riesigen Modellen überhaupt erst möglich macht.
- Kapitel 11: Hier lernen wir sogenannte **Convolutional Neural Networks** (CNNs) kennen. Sie sind die Basis für die Fortschritte auf dem Gebiet Computer Vision und erlauben beispielsweise Anwendungen im Bereich automatische Gesichtserkennung in Bildern oder Videos.
- Kapitel 12: Nach ANNs und CNNs lernen wir hier **Recurrent Neural Networks** (RNNs) kennen. Diese Modelle bilden die Basis für Probleme, in denen die Daten als Sequenzen vorliegen. Das können einache Zeitreihen (z.B. Börsenkurse) sein, aber auch komplexere Sequenzdaten wie beispielsweise geschriebene oder gesprochene Sprache oder Tonaufnahmen.
- Kapitel 13: In diesem letzten Kapitel geht es schliesslich um **Generative KI**. Wir beschäftigen uns hier also mit Modellen, die nicht nur einfach ein Vorhersageprobleme lösen können, sondern auch neue Inhalte (z.B. Texte, Musik, Bilder) generieren können. Abbildung 2 enthält als Beispiel den Output einer generativen Software, die basierend auf einem Prompt ein Bild erstellt. Nach dem Lesen dieses Kapitels sollten Sie ein grundlegendes Verständnis für die Funktionsweise von Modellen wie Chat-GPT haben.

Weiterführende Literatur

Ein grosser Teil des vorliegenden Buchs baut auf bestehenden Büchern zum Thema Machine Learning auf. Ich werde im Buch immer wieder auf die Quellen verweisen. Die wichtigsten Referenzen für dieses Buch sind folgende:

- Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2021). An Introduction to Statistical Learning: with Applications in R. New York: Springer. 2nd Edition.
- Aurélien Géron. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. Sebastopol: O'Reilly Media Inc. 3rd Edition.
- Christopher M. Bishop. (2006). Pattern Recognition and Machine Learning. Berlin, Heidelberg: Springer.
- Kevin P. Murphy. (2012). Machine Learning A Probabilistic Perspective. The MIT Press.

Die ersten beiden Referenzen sind einführende Texte und können parallel zum vorliegenden Buch gelesen werden. Die letzten zwei Referenzen sind fortgeschrit-

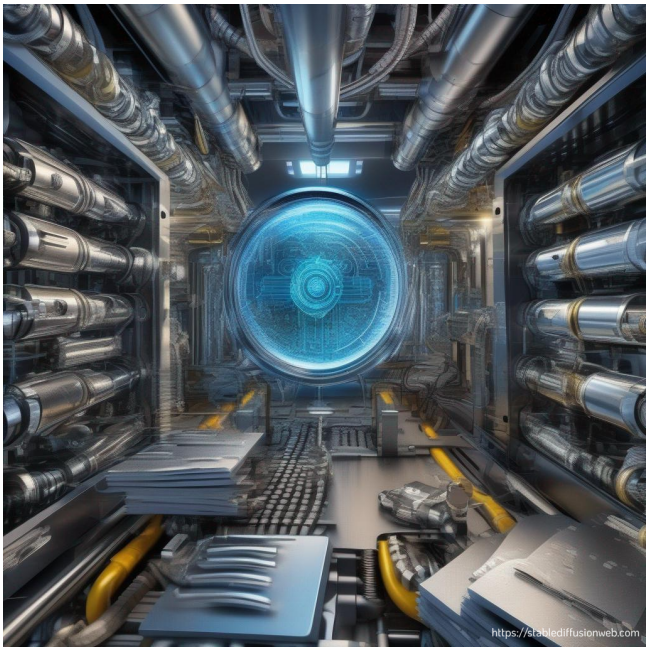


Figure 2: Beispielsoutput einer generativen Bildgenerierungssoftware (https://stablediffusionweb.com/) basierend auf dem Prompt "A title image for a textbook about Machine Learning targeting small and medium companies."

tene Texte und ich empfehle, sie erst nach dem vollständigen Verständnis des vorliegenden Buchs oder der ersten beiden Referenzen zu lesen.

Lizenz

Kontakt

Für Fragen und Anregungen zum Buch stehe ich gerne zur Verfügung:

Martin Sterchi
Riggenbachstrasse 16
4600 Olten
martin.sterchi@fhnw.ch

Chapter 1

Einführung

In diesem Kapitel geht es darum zu verstehen, was ML überhaupt ist, warum es nützlich sein kann und was typische Anwendungsfälle von ML sind. Wir werden ausserdem verschiedene Unterkategorien von ML kennen lernen.

1.1 Was ist Machine Learning?

Im Prinzip geht die Geschichte des MLs weit zurück, nämlich zu den Anfängen der Statistik. Viele Modelle, die heutzutage im ML angewendet werden sind nämlich eigentlich von Statistiker*innen erfundene Modelle. Die Geschichte des MLs und der Statistik sind darum eng verknüpft. Einen eigentlichen Startpunkt des MLs könnte man vielleicht in den 1960er Jahren ausmachen, mit den Arbeiten von Frank Rosenblatt[1], welcher das sogenannte **Perceptron** und einen dazugehörigen Lernalgorithmus prägte (dazu später mehr). Danach blieb es aber rund 20 Jahre relativ ruhig bis die Forschung im Bereich Machine Learning so richtig Fahrt aufnahm. Ein grosser Schub für die Entwicklung von ML ging vom Aufkommen von extrem grossen Datenmengen (**Big Data**) und dem Internet aus. Das führte nämlich dazu, dass sich immer mehr Leute aus den Fachbereichen Informatik und Computer Science mit dem Thema ML befassten und effiziente Hard- und Software sowie algorithmische Kniffs und Tricks beisteuerten. Ausserdem ermöglichte das Internet den Zugang zu gewaltigen Datenmengen an Bildern, Videos, Klicks, etc. - denken Sie beispielsweise nur schon an die Informationen, die jede*r von uns tagtäglich im Internet hinterlässt. Ein weiterer Schub für das Machine Learning war (und ist) zudem die immer besser werdende Rechenleistung von Computern. Diese Entwicklungen haben sich im November 2022 kulminiert in der erstmaligen breiten öffentlichen Wahrnehmung von sogenannten **Large Language Models** wie ChatGPT.

[1] https://en.wikipedia.org/wiki/Frank_Rosenblatt

Wie der Name sagt, geht es im ML darum, dass eine Maschine (oder präziser, ein Computer) aus einem gegebenen Datensatz automatisch Muster lernt, ohne dass ein Mensch dem Computer (explizit) sagen muss, was er lernen soll. Der Mensch gibt jedoch dem Computer die Rahmenbedingungen für das selbständige Lernen vor. Die erlernten Muster sind selbstverständlich nur nützlich, wenn sie **genereller Natur** sind und auch für neue bzw. zukünftige Beobachtungen gelten. Beispiel: ein Spital hat während der Corona Pandemie ein Modell trainiert, um den täglichen Pflegebedarf je nach Wochentag, Saison, und weiteren Indikatoren vorherzusagen. Das Modell funktioniert nun nach der Pandemie aber nicht wunschgemäss und prognostiziert in der Tendenz einen zu hohen Pflegebedarf. Das Problem ist, dass die erlernten Muster nicht gut auf eine Zeit nach der Pandemie generalisierbar sind. Mit anderen Worten: die Trainingsdaten waren nicht repräsentativ genug.

Bevor wir etwas konkreter anschauen, wie genau ein Computer selbständig aus Daten lernen kann, schauen wir uns die Definitionen von zwei Experten im Gebiet ML an:

"[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed." Arthur Samuel, 1959

"Machine Learning is the science (and art) of programming computers so they can learn from data." Aurélien Géron[2]

Zusammenfassend lässt sich sagen, dass wir mit ML dem Computer die Möglichkeit geben, automatisch und selbständig aus Daten zu lernen. Nichtsdestotrotz braucht es Sie als ML-Expert*in, und zwar wie folgt:

1. Sie entscheiden sich für ein spezifisches ML Modell. Typischerweise kann ein ML Modell durch eine mathematische Funktion (siehe Kapitel 2) charakterisiert werden. ML Modelle können unterschiedlich flexibel sein und es liegt im Ermessen von Ihnen, wie flexibel das Modell sein soll. Sie müssen bei der Wahl des Modells die Komplexität des Problems berücksichtigen. Grundsätzliche gilt bei der Wahl des Modells, dass flexiblere Modelle komplexere Sachverhalte abbilden können. Ein zu flexibles Modell kann aber zu Overfitting führen, aber dazu später mehr. Dieser Schritt wird im Fachjargon typischerweise **Model Selection** (Modelauswahl) genannt.
2. Sobald Sie das Modell ausgewählt haben, übergeben Sie dem Computer (etwas vereinfacht gesagt) das Modell, einen Datensatz sowie einen Lernalgorithmus. Nun hat der Computer alle Zutaten, um automatisch zu lernen. Doch was lernt er eigentlich? Der Computer lernt die Parameter Ihres gewählten Modells, so dass das Modell sich optimal an die Daten anpasst. Dieser Schritt wird im Fachjargon **Model Training** (Trainieren des Modells) genannt.

[2] Aurélien Géron. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. Sebastopol: O'Reilly Media Inc. 3rd Edition.

3. Falls Sie mit dem erlernten Modell zufrieden sind, können Sie es nun entweder dazu verwenden Vorhersagen zu machen oder um Zusammenhänge in den Daten zu interpretieren und daraus wertvolle Einsichten gewinnen. Dieser Schritt wird im Fachjargon als **Model Inference** (Modellinferenz) zusammengefasst. Typischerweise sind Sie in der Realität mit dem ersten erlernten Modell allerdings noch nicht zufrieden und gehen zurück zu Schritt 1 und wählen ein anderes Modell.

Es handelt sich bei dieser Vorgehensweise um eine sehr allgemeine Beschreibung des Machine Learning Prozesses. Wie diese drei Schritte konkret funktionieren, werden Sie in den nachfolgenden Kapiteln dieses Buchs erfahren.

1.2 Wann macht es Sinn ML einzusetzen?

Ein ML Modell zu trainieren kann viel Zeit und Geld kosten. Zum Beispiel müssen Sie unter Umständen überhaupt erst die Daten sammeln (oder von einem Datendienstleister kaufen), um ein Modell zu trainieren. Oder das Projekt ist so komplex, dass Sie als Analyst*in unzählige Stunden benötigen, um die Daten überhaupt erst in eine Form zu bringen, die es erlaubt ein Modell zu trainieren. Für neuartige DL Modelle oder Generative KI kann das Trainieren bzw. Lernen eines Modells durch den reinen Stromverbrauch bzw. die vom Cloud-Betreiber in Rechnung gestellten Kosten so hoch sein, dass sich Ihr ursprüngliches Vorhaben nicht mehr lohnt. Es ist also ungemein wichtig, dass Sie sich vor Projektbeginn gut überlegen, ob ML für Ihr vorliegendes Problem überhaupt Sinn macht und einen Mehrwert generieren kann.

Folgende Daumenregeln[3] können Ihnen dabei helfen, zu entscheiden, ob ML für Ihr Projekt Sinn macht:

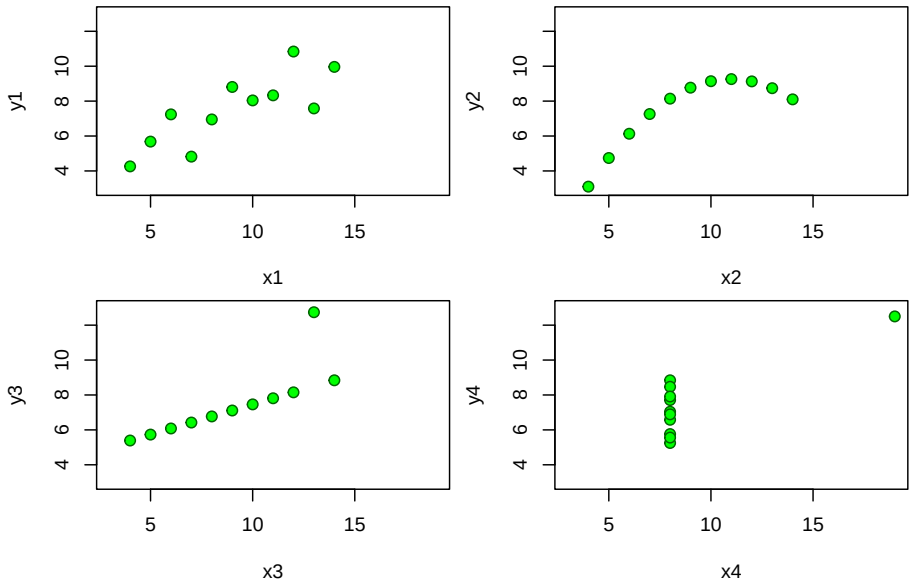
- Ihr Problem entspricht einem Standard ML-Problem, das bereits mehrfach gelöst wurde und für das es sogenannte “off-the-shelf” Lösungen gibt. Beispiel: Sie wollen das Sentiment (positive vs. negative Grundhaltung) von Social Media Posts über Ihr Unternehmen automatisch klassifizieren. Dazu gibt es viele vortrainierte Modelle, die teilweise gratis verwendet werden können.
- Der manuelle Arbeitsaufwand ist sehr gross, wenn das Problem durch Menschen gelöst werden soll. Das Problem ist aber ansonsten klar strukturiert und benötigt keinen grossen kognitiven Einsatz eines Menschen. Beispiel: In den Post-Verteilzentren werden die von Hand geschriebenen Postleitzahlen (PLZ) problemlos mittels Computer bzw. ML Modellen erkannt und “gelesen” und die Briefe und Pakete entsprechend sortiert.

[3] siehe auch Seiten 6 - 7 in Aurélien Géron. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. Sebastopol: O’Reilly Media Inc. 3rd Edition.

- Komplexe Probleme, in denen ein Mensch keinen Überblick hat, weil so grosse und komplexe Datenmengen vorhanden sind. Wir Menschen haben grosse Mühe damit, in Rohdaten (reinen Datentabellen) irgendwelche Muster zu erkennen. In diesem Fall können wir entweder versuchen, die Daten zu visualisieren oder mithilfe von ML Zusammenhänge zu lernen, die wir sonst nicht erkennen könnten. Ein illustratives Beispiel ist das Anscombe Quartett[4], das vier kleine Stichproben mit jeweils elf Datenpunkten enthält. Jeder Datenpunkt wird durch eine x und eine y Variable beschrieben. Die vier x- sowie die vier y-Variablen haben identische Mittelwerte. Erst eine einfache Visualisierung der vier Stichproben mithilfe eines Streudiagramms zeigt die Muster sowie die Unterschiede zwischen den vier Stichproben deutlich auf.

```
#>    x1 x2 x3 x4    y1   y2    y3    y4
#> 1  10 10 10  8  8.04 9.14  7.46  6.58
#> 2   8  8  8  8  6.95 8.14  6.77  5.76
#> 3  13 13 13  8  7.58 8.74 12.74  7.71
#> 4   9  9  9  8  8.81 8.77  7.11  8.84
#> 5  11 11 11  8  8.33 9.26  7.81  8.47
#> 6  14 14 14  8  9.96 8.10  8.84  7.04
#> 7   6  6  6  8  7.24 6.13  6.08  5.25
#> 8   4  4  4 19  4.26 3.10  5.39 12.50
#> 9  12 12 12  8 10.84 9.13  8.15  5.56
#> 10  7  7  7  8  4.82 7.26  6.42  7.91
#> 11  5  5  5  8  5.68 4.74  5.73  6.89
```

[4] https://de.wikipedia.org/wiki/Anscombe-Quartett



1.3 Anwendungsfälle von ML

In diesem Abschnitt stelle ich erfolgreiche Anwendungsfälle von ML vor. Einige davon treffen Sie womöglich tagtäglich in Ihrem Alltag an:

- **Spam Filter** sind ein frühes Beispiel einer erfolgreichen Anwendung von ML. Ein Klassifikationsmodell entscheidet dabei automatisch aufgrund der Inhalte einer Email, des Betreffs sowie des Absenders, ob es sich um eine Spam oder eine sogenannte Ham Email (unproblematische Email) handelt. Falls Sie gängige Email Software verwenden, dann arbeitet im Hintergrund ein Spam Filter daran, Sie vor lästigen Emails zu schützen.
- Ein grosser Teil des wirtschaftlichen Erfolgs von **Google** basiert auf der Idee, dass aufgrund der Suchhistorie hervorgesagt werden kann, welche Nutzerin oder welcher Nutzer mit welcher Wahrscheinlichkeit eine bestimmte Werbung anklickt. Dies erlaubt Google für jede Nutzer*in die Werbung mit den höchsten "Erfolgschancen" zu schalten. Da jeder Klick Einnahmen generiert, ist es für das Geschäftsmodell von Google entscheidend, dass möglichst viele Klicks stattfinden.
- Ein grosser Bereich des MLs und speziell des DLs befasst sich mit **Computer Vision**. Dabei geht es darum, das Hauptmotiv von Bildern zu klassifizieren (z.B. Zeigt ein Bild ein Tier oder einen Menschen?), Objekte in Bildern zu entdecken (z.B. Enthält das Bild eine Person?) und das entdeckte Objekt dann auch zu klassifizieren (z.B. Handelt es sich bei

der Person um XY?). Als konkreteres Beispiel können Sie sich einen Industriebetrieb vorstellen, welcher ein Computer Vision Modell einsetzen möchte, um den Abnützungsgrad der von ihnen produzierten Werkzeuge automatisch zu erkennen und den Kundinnen und Kunden den optimalen Ersatzzeitpunkt für das Werkzeug vorhersagen zu können.

- Ähnlich wie im vorherigen Beispiel gibt es bereits viele Anwendungen im öffentlichen Verkehr, in denen es um **Predictive Maintenance** geht. Z.B. kann der optimale Wartungszeitpunkt für eine Weiche oder einen Gleisabschnitt aufgrund einer Vielzahl an Indikatoren und Messungen vorhergesagt werden.
- Ein grosses Einsatzgebiet für ML ergibt sich im Finanzsektor durch das automatische Erkennen von potentiell **betrügerischen Transaktionen**. Falls Sie auch schon mal eine Kreditkartentransaktion direkt am Telefon einer Kundenberaterin oder einem Kundenberater bestätigen mussten, dann ist es wahrscheinlich, dass Ihre Transaktion von einem ML System zur manuellen Überprüfung geflaggt wurde. In diesem Zusammenhang spricht man manchmal auch vom Erkennen von Anomalien (engl. *Anomaly Detection*).
- Sogenannte **Recommender Systems** sind insbesondere in Online Verkaufspunkten von grossem Nutzen. Betreiben Sie beispielsweise einen grossen Onlinehandel, dann wollen Sie Ihren Kundinnen und Kunden Produkte zum Kauf vorschlagen. Dazu verwenden Sie ein Modell, das basierend auf der Ähnlichkeit zwischen Kundinnen und Kunden potentiell interessante Produkte vorschlägt.
- Die rasanten Entwicklungen im Bereich **Natural Language Processing** (NLP) in den letzten 10 Jahren haben viele neue und interessante Anwendungsgebiete zutage gefördert. Zum Beispiel eignen sich *Large Language Models* (LLMs) als erste Anlaufstelle für Kundinnen und Kunden (automatisierter Kundenservice). LLMs werden vermutlich aber auch immer mehr in internen Prozessen in Unternehmen eingesetzt, z.B. um komplexe Dokumente zusammenzufassen oder Sitzungsprotokolle zu erstellen.

Die obige Liste ist bei weitem nicht komplett und die Entwicklungen im Bereich ML sind aktuell so rasant, dass jeden Tag eine grosse Zahl von neuen ML-basierten Produkten und Dienstleistungen auf den Markt kommen.

1.4 Supervised vs. Unsupervised Learning

Den Unterschied zwischen dem Supervised Learning und dem Unsupervised Learning können wir am besten erklären, indem wir uns mit ein paar mathematischen Grundlagen des Machine Learnings befassen. Keine Sorge, diese Grundlagen sind sehr einfach, aber versuchen Sie, diese bereits gut zu verstehen, denn wir bauen später darauf auf.

Im **Supervised Learning** haben wir einerseits sogenannte Input-Daten und andererseits einen Output, den wir vorhersagen wollen. Für die Input-Daten gibt es ganz viele verschiedene Begriffe, die synonym verwendet werden: z.B. Features, unabhängige Variablen, Attribute, Prädiktoren. Dasselbe gilt für den Output, hier gibt es folgende Synonyme: Zielvariable, abhängige Variable, Label, oder auch einfach y. Unsere Konvention hier ist aber folgende: es gibt Input-Daten (oder Input-Variablen) und einen Output (oder Output-Variable).

Die Input-Daten für eine Beobachtung i schreiben wir mathematisch wie folgt:

$$\mathbf{x}_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix},$$

Diese Notation bedarf ein paar Erklärungen:

- Den Index i brauchen wir, um die verschiedenen Beobachtungen zu kennzeichnen. i kann eine Ganzzahl zwischen 1 und n annehmen, wobei n die Anzahl Beobachtungen im Datensatz bezeichnet. Wenn wir zum Beispiel etwas über die Input-Daten der dritten Beobachtung sagen wollen, dann können wir die Notation $\mathbf{x}_3$ verwenden.
- Für jede Beobachtung i haben wir insgesamt p Variablen, welche die verschiedenen Attribute einer Beobachtung enthalten. x_{i1} bezeichnet also die erste Variable der i-ten Beobachtung, x_{i2} die zweite Variable der i-ten Beobachtung und x_{ip} die p-te (letzte) Variable der i-ten Beobachtung.
- Was Sie oben sehen, ist aus mathematischer Sicht ein Spaltenvektor. Im Moment reicht es, wenn Sie wissen, dass wir mit diesem Spaltenvektor die Input-Daten einer Beobachtung *kompakt* darstellen können.

Neben den Input-Daten haben wir im Supervised Learning aber wie erwähnt auch einen Output und den bezeichnen wir üblicherweise mit y_i. Auch hier hilft uns der Index i dabei, die Beobachtungen eindeutig zu kennzeichnen. Schauen wir uns am besten kurz ein konkretes Beispiel an:

Aufgabe

Wichtig: Beim Supervised Learning geht es um ML Probleme, in denen sowohl Input-Daten als auch ein Output vorhanden ist. Ziel beim Supervised Learning ist es, ein Modell zu trainieren, das basierend auf den Input-Daten möglichst gute Vorhersagen für den Output macht. Es geht also hier um Vorhersageprobleme. In einem gewissen Sinn ist der Output die überwachende Instanz (engl. Supervisor), welche den Lernprozess des Modells kontrolliert.

Im Gegensatz zum Supervised Learning haben wir im **Unsupervised Learning** nur Input-Daten und *keinen Output*. Im Unsupervised Learning geht es darum, aus den Input-Daten interessante Muster zu lernen, welche für bessere unternehmerische Entscheidungen verwendet werden können. Ein einfaches

Beispiel ist das Clustering von Kundinnen und Kunden eines Unternehmens in ähnliche Kundengruppen, so dass die verschiedenen Kundengruppen gezielter mit Marketingaktionen angesprochen werden können. Techniken, um komplexe Datensätze zu visualisieren, werden typischerweise auch zum Unsupervised Learning gezählt.

Neben dem Supervised und dem Unsupervised Learning gibt es noch eine dritte Kategorie von Machine Learning, nämlich das **Reinforcement Learning** (RL). Dieser Kategorie gehören Modelle an, die (virtuelle) Agenten so trainieren, dass sie langfristig möglichst optimal handeln. Das bekannteste Beispiel aus dem RL ist Googles AlphaGo Agent, welcher den menschlichen Go Weltmeister im Jahr 2017 schlug.[5]. Reinforcement Learning ist aber auch eine wichtige Komponente in der Optimierung von grossen Sprachmodellen wie ChatGPT. In einer ersten Fassung dieses Buchs werden wir uns nicht (oder nur am Rande) mit RL befassen.

Die Unterscheidung zwischen den drei Arten von Machine Learning ist im oberen Teil der Abbildung 1.1 visualisiert:

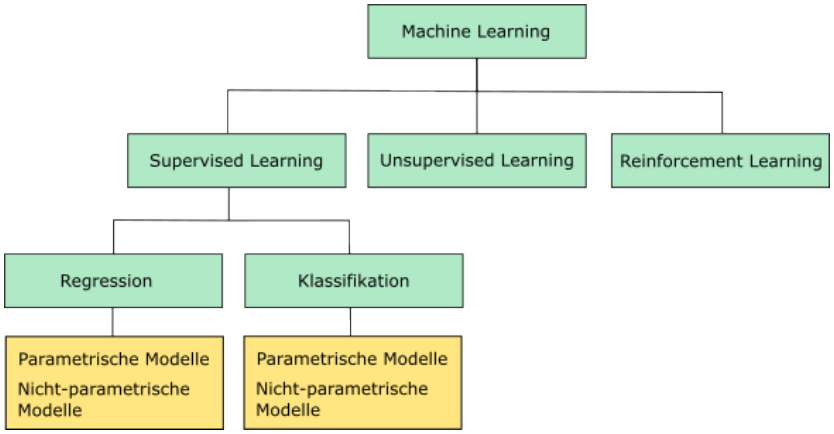


Figure 1.1: Die verschiedenen Kategorien des Machine Learnings und deren Hierarchie.

1.5 Regression vs. Klassifikation

In der Kategorie des Supervised Learnings unterscheiden wir weiter zwischen Regressions- und Klassifikationsproblemen (siehe auch Abbildung 1.1).

Im Regressionsproblem ist der Output eine **stetige** Variable (Intervall- oder Verhältnisskalierung), d.h. die Variable enthält reelle (numerische) Werte. Math-

[5] https://deepmind.google/technologies/alphago/

ematisch schreibt man dies als $y_i \in \mathbb{R}$, wobei $\mathbb{R}$ die Menge der reellen Zahlen beschreibt.

Im Klassifikationsproblem ist der Output bzw. die Zielvariable eine **kategorische** Variable (Nominal- oder Ordinalskalierung). Mathematisch schreibt man dies als $y_i \in \{1, ..., C\}$, wobei C die Anzahl Kategorien beschreibt. Wenn wir nur $C = 2$ Kategorien haben wie im Beispiel oben mit $y_i \in \{\text{Betrug}, \text{kein Betrug}\}$ sprechen wir von einem binären Klassifikationsproblem. Falls $C > 2$ sprechen wir vom mehrklassigen (engl. *multiclass*) Klassifikationsproblem.

Aufgabe

1.6 Parametrische vs. nicht-parametrische Modelle

Ein ML Modell gehört entweder der Familie **parametrischer** Modelle oder der Familie **nicht-parametrischer** Modelle an. Dabei spielt es keine Rolle, ob wir mit dem Modell ein Regressions- oder ein Klassifikationsproblem lösen wollen.

Womöglich sind Sie in Ihrer Ausbildung bereits **parametrischen Modellen** begegnet, denn das einfache lineare Regressionsmodell ist ein typisches Beispiel für ein parametrisches ML Modell. Das Modell ist vollkommen charakterisiert durch die beiden lernbaren (optimierbaren) Parameter w_0 und w_1[6] und kann wie folgt (mathematisch) aufgeschrieben werden:

$$\hat{y_i} = f(x_i) = w_0 + w_1 \cdot x_i$$

Wenn Ihnen der obige Ausdruck noch fremd vorkommt, dann ist das nicht schlimm. Wir werden im Kapitel 4 ausführlich auf lineare Regressionsmodelle eingehen. Im Moment müssen Sie nur wissen, dass ein parametrisches Modell wie oben mit einer mathematischen Funktion beschrieben werden kann und dass diese Funktion durch lernbare **Parameter** (hier w_0 und w_1) charakterisiert wird.

Nicht-parametrische Modelle wiederum sind Modelle, welche nicht (oder zumindest nicht explizit) durch Parameter charakterisiert sind. Am besten schauen wir uns gleich ein einfaches nicht-parametrisches Modell an, nämlich das **K-Nearest-Neighbors** (KNN) Modell. Stellen Sie sich vor, Sie haben einen Datensatz mit 55 Produkten aus Ihrem Sortiment. Sie haben jedes dieser 55 Produkte auf Instagram und auf Tiktok durch Influencer*innen bewerben lassen. Für jedes der 55 Produkte hatten Sie ein Werbebudget für Instagram (x_{i1}) und ein Werbebudget für Tiktok (x_{i2}). Am Ende des Geschäftsjahrs haben

[6] In Statistikvorlesungen werden die beiden Parameter oft eher mit b_0 und b_1 oder mit β_0 und β_1 bezeichnet. Im Machine Learning nennt man Parameter oft Gewichte (engl. Weights), weshalb die Parameter typischerweise mit w bezeichnet werden.

Sie für jedes der 55 Produkte bestimmt, ob die Absatzziele erreicht wurden oder nicht (Output y_i). Die erfolgreichen Produkte (= Absatzziel erreicht) sind in untenstehender App als blaue Punkte eingezeichnet. Die roten Dreiecke repräsentieren die nicht-erfolgreichen Produkte. Sie sehen, dass erfolgreiche Produkte tendenziell höhere Instagram und Tiktok Werbebudgets aufwiesen als nicht-erfolgreiche Produkte. Sie möchten nun ein Modell schätzen, dass die Produkte automatisch klassifizieren kann. Dazu verwenden Sie das KNN Modell, das die K nächsten Nachbarn unter den 55 gegebenen Produkten sucht und dann die häufigste Beobachtung unter den K nächsten Nachbarn vorhersagt. In anderen Worten: wir suchen die K **ähnlichsten** Beobachtungen und nutzen diese, um eine Vorhersage zu machen.

Selbstverständlich spielt der konkrete Wert von K hier eine grosse Rolle - sollen wir nur $K = 1$ Nachbarn berücksichtigen? Oder $K = 10$ Nachbarn? Die erste Abbildung in der App zeigt nicht nur die 55 Datenpunkte, sondern auch die **Entscheidungsgrenze** (in schwarz). Untersuchen Sie kurz, wie sich diese Entscheidungsgrenze verändert, wenn Sie K erhöhen oder reduzieren.

Ausserdem können Sie in der ersten Abbildung auch den schwarzen Punkt mit der Maus setzen, wodurch Ihnen die K nächsten Punkte des schwarzen Punkts angezeigt werden.

Die zweite Abbildung zeigt die Entscheidungsregionen mit unterschiedlicher Intensität je nachdem wie sicher sich das Modell ist. In einer Region, in der alle K Nachbarn nicht-erfolgreiche Produkte sind, sind wir uns eher sicher bezüglich der Vorhersage als in einer Region, in der die Anteile zwischen erfolgreichen und nicht-erfolgreichen Produkten ausgeglichen sind.

Um die K nächsten Nachbarn zu finden, müssen wir die Distanzen zwischen Punkten rechnen können. Dazu verwenden wir die Euklidische Distanz, welche wir in Kapitel 2 kennen lernen werden.

Das KNN Modell ist ein sehr einfaches ML Modell, welches in der Praxis allerdings nicht allzu häufig angewendet wird. Warum nicht? Weil es am sogenannten **Fluch der Dimensionalität** (engl. Curse of Dimensionality) leidet. Doch was bedeutet das? Je mehr Input-Variablen wir haben, desto weiter entfernt sind Datenpunkte voneinander (das ist etwas, das man sich nur schwer vorstellen kann, aber Sie können es mir für den Moment einfach mal glauben). Das KNN beruht auf der Grundidee, dass wir K nahe, ähnliche Beobachtungen für die Vorhersage verwenden. Wenn diese K nahen Beobachtungen im hochdimensionalen Raum (= viele Input-Variablen) nicht mehr nahe sind, dann funktioniert auch das Modell nicht mehr gut.

Aufgaben

1.7 Machine Learning Pipeline

Abbildung 1.2 zeigt, wie eine typische ML-Pipeline aussieht.[7]

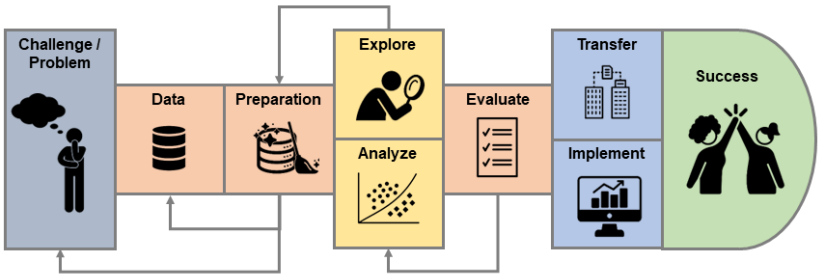


Figure 1.2: Eine typische ML-Pipeline.

Sie starten typischerweise mit einem **Problem** oder einer Herausforderung. Ihr ganzes Projekt sollte darauf ausgelegt sein, dieses Problem zu lösen. Es ist grundsätzlich nicht ratsam, auf Biegen und Brechen eine ML Lösung zu implementieren, wenn kein klar definiertes Problem vorliegt. Nehmen Sie sich also zu Beginn eines Projekts Zeit, das Problem grundlegend zu definieren. Sprechen Sie auch mit den entsprechenden Fachexpert*innen im Unternehmen, um genau zu verstehen, was verbessert oder effizienter gemacht werden soll und was die technischen oder ökonomischen Einschränkungen sind.

Sobald das Problemverständnis vorhanden ist, beginnen Sie, sich mit den **verfügbaren Daten** zu befassen. Auch hier müssen Sie sich wahrscheinlich mit den entsprechenden Expert*innen im Unternehmen (z.B. Datenbankadministrator*innen) austauschen. Es geht hier unter anderem darum abzuklären, welche Daten verfügbar sind, in welchem Format die Daten vorhanden sind wie die Datenqualität ist.

Danach beginnen Sie mit den Datenarbeiten. Häufig wird dieser Schritt **Preprocessing** oder **Data Cleaning** genannt. Oft verschlingt dieser Arbeitsschritt sehr viel Zeit und es ist nicht unüblich, dass 80% der Projektzeit hier aufgewendet werden. Es ist auch völlig normal, wenn Sie von diesem Schritt zurück zur Problemdefinition gehen und sie verfeinern oder anpassen müssen oder zum Beispiel nochmals Fragen mit den Datenbankexpert*innen klären müssen, weil Ihr Datenverständnis noch nicht vollständig ist.

Nachdem die Daten vorbereitet wurden, gehen Sie typischerweise zu einer **explorativen Analyse** der Daten über. Das heisst, Sie visualisieren die vorhandenen Variablen univariat (d.h. jede Variable einzeln) oder multivariat (d.h. zwei oder mehr Variablen zusammen). Ein Beispiel einer univariaten Visualisierung ist ein Histogramm einer quantitativen Variable (z.B. Quartalsumsätze). Ein

[7] Icons stammen von https://thenounproject.com/.

Beispiel einer multivariaten Visualisierung ist ein Streudiagramm zweier quantitativer Variablen (z.B. Quartalsumsätze und Wechselkurse). Auch hier ist es üblich, dass Sie einen Schritt zurück gehen und weitere Datenbereinigungen vornehmen müssen.

Nach der explorativen Analyse der Daten sollten Sie eine erste Idee von den wichtigsten Zusammenhängen in den Daten haben. Basierend darauf können Sie Ihr erstes Modell wählen und trainieren und mit der eigentlichen **Analyse** bzw. der Lösung des Problems beginnen.

Einer der wichtigsten Schritte ist die saubere und gründliche **Evaluation** Ihrer Modelle. Dieser Schritt dient einerseits dazu das beste Modell auszuwählen und andererseits dazu die Qualität Ihrer Lösung bzw. Ihres Modells abzuschätzen. Mit diesem zweiten Schritt wollen Sie nämlich bereits während der Projektphase einschätzen können, wie gut Ihr Modell das gegebene Problem löst oder einen bestehenden Betriebsprozess verbessert oder effizienter macht. Die beiden Schritte Analyse und Evaluation werden typischerweise ein paar Mal iteriert, bis Sie das beste Modell gefunden haben.

Am Schluss geht es darum, dass Sie Ihr Wissen und Ihre Erkenntnisse an die relevanten Fachexpert*innen weitergeben (**Wissenstransfer**) und Ihr finales Modell in einer produktiven Umgebung implementieren (oft **Deployment** genannt). Zum Beispiel können Sie Ihr Modell in einer mobilen App einbetten oder als REST API Service zur Verfügung stellen.

Chapter 2

Mathematik- und Statistik-Grundlagen

In diesem Kapitel repetieren wir die wichtigsten Grundlagen aus der Mathematik und Statistik, die es braucht, um Machine Learning Modelle zu verstehen. Das Thema *Lineare Algebra* wird für die meisten von Ihnen wahrscheinlich Neuland sein.

2.1 Funktionen

Eine Funktion, die wir in der Mathematik typischerweise mit f bezeichnen, ordnet jedem **Argument** x aus dem Definitionsbereich D (engl. *Domain*) **genau einen Wert** y aus dem Wertebereich W (engl. *Codomain*) zu. Oft sind D und W die Menge der reellen Zahlen, also $\mathbb{R}$. Die Menge der reellen Zahlen enthält alle möglichen Zahlen, die Sie sich vorstellen können.[1] Zum Beispiel die Zahlen 3, -4.247, $\sqrt{14}$, $5/8$, etc.

Wie eine Funktion grafisch aussieht, ist aus Panel (a) der Abbildung 2.1 ersichtlich. Hier zeigen wir die Form einer Funktion in einem kartesischen Koordinatensystem. Die Funktionskurve weist jedem Wert x auf der x-Achse genau einen Wert y auf der y-Achse zu. Der wichtigste Teil der oben aufgeführten Definition ist der Teil "genau einen Wert", denn eine Funktion kann einem Element x nicht zwei oder mehr Werte zuweisen, sondern nur genau einen. Genau aus diesem Grund handelt es sich bei Panel (b) in Abbildung 2.1 *nicht* um eine Funktion, da gewissen x-Werten mehrere Werte y zugeordnet werden. *Wichtig*: das heisst aber nicht, dass zwei verschiedenen x-Werten, nennen wir sie x' und x'', derselbe y-Wert zugeordnet werden kann (vgl. Panel (a)).

[1] Einzige Ausnahme sind die komplexen Zahlen.

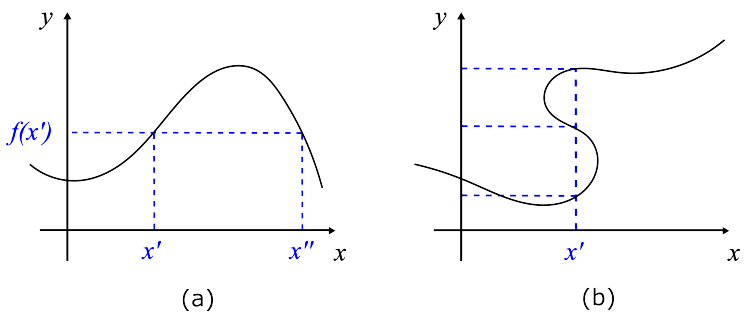


Figure 2.1: (a) Eine Funktion, die jedem x-Wert genau einen y-Wert zuweist. (b) Keine Funktion.

Mathematisch wird diese allgemeine Definition einer Funktion häufig wie folgt beschrieben:

$$f : x \mapsto y$$

Wir haben also eine Funktion f, die jedem Element x genau einen Wert y zuweist. Der Pfeil in obiger mathematischer Schreibweise beschreibt genau dieses Mapping. Wie genau dieses Mapping einem Argument x den entsprechenden y-Wert zuordnet, wird durch die Funktion $f(x)$ beschrieben. In den folgenden Abschnitten schauen wir uns typische Beispiele von Funktionen an, angefangen mit linearen Funktionen. Doch vorher wollen wir uns kurz überlegen, warum Funktionen für das Machine Learning überhaupt wichtig sind. Ein grosser Teil des Machine Learnings, der **Supervised Learning** genannt wird, befasst sich mit dem Problem, wie eine Zielvariable y mithilfe von einem oder mehreren Prädiktoren x vorhergesagt werden kann. Ein Machine Learning Modell ist darum nichts anderes als eine Funktion $y = f(x)$, die basierend auf den Prädiktoren x die Zielvariable y möglichst gut beschreiben kann.[2]

2.1.1 Lineare Funktionen

Nun schauen wir uns an, wie eine **lineare** Funktion aussieht. Eine lineare Funktion kann allgemein wie folgt geschrieben werden:

$$y = f(x) = a \cdot x + b$$

Obige Funktionsgleichung besagt, dass wir den entsprechenden y-Wert kriegen, indem wir den Wert des Arguments x mit a multiplizieren und danach eine Konstante b addieren. a und b sind die **Parameter** dieser Funktion. Die konkreten Zahlenwerte dieser beiden Parameter definieren, wie die Funktion am Schluss genau aussieht.

[2] Zumindest aus einer nicht-probabilistischen Perspektive.

Eine lineare Funktion hat auch eine geometrische Interpretation und zwar entspricht eine lineare Funktion einer Gerade. Das ist auch der Grund, warum wir diese Funktionen **linear** nennen, sie können graphisch durch eine "Linie" dargestellt werden. Der Parameter a ist die Steigung dieser Geraden und der Parameter b entspricht dem Ort, wo die Gerade die y-Achse schneidet (sogenannter y-Achsenabschnitt).

Am besten schauen wir uns ein paar konkrete Beispiele an (Abb. 2.2).

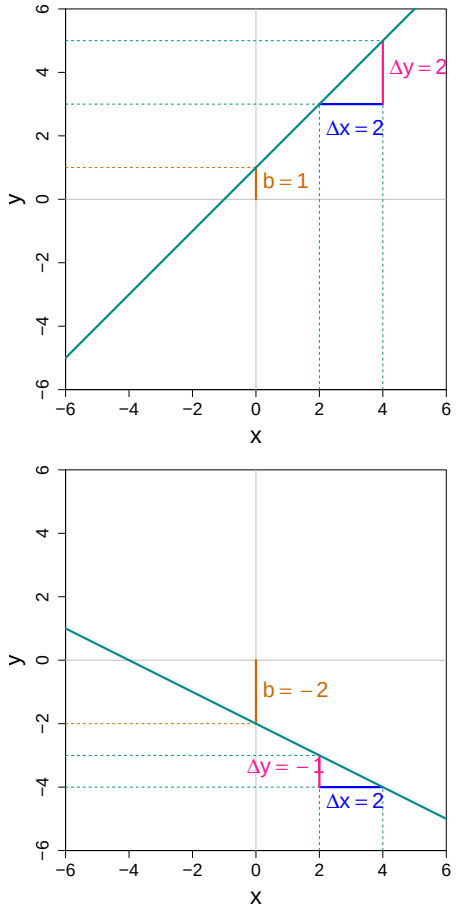


Figure 2.2: Beispiele linearer Funktionen.

Aus der linken Abbildung können wir ablesen, dass die Steigung dieser Geraden $\frac{\Delta y}{\Delta x} = \frac{2}{2} = 1$ ist und dass die Gerade die y-Achse am Ort 1 schneidet. Die entsprechende lineare Funktion kann dementsprechend als $y = x + 1$ geschrieben werden.[3]

[3]Wir müssen hier die Steigung 1 nicht explizit schreiben, aber selbstverständlich ist es nicht

Aus der rechten Abbildung können wir ablesen, dass die Steigung $\frac{\Delta y}{\Delta x} = \frac{-1}{2} = -0.5$ ist und dass die Gerade die y-Achse am Ort -2 schneidet. Die entsprechende lineare Funktion kann dementsprechend als $y = -0.5 \cdot x - 2$ geschrieben werden.

Es ist wichtig zu sehen, dass der Effekt einer Veränderung von x (also Δx) auf y überall derselbe ist. Es spielt also keine Rolle, ob wir von $x = -2$ zu $x = -1$ gehen oder von $x = 100$ zu $x = 101$, die entsprechende Veränderung in y (also Δy) wird dieselbe sein. Das muss so sein, denn die Gerade steigt (oder sinkt) mit konstanter Steigung.

Aufgaben

1. Zeichnen Sie die Funktion $y = 2 \cdot x$ in ein Koordinatensystem ein. Warum fehlt der Parameter b?
2. Zeichnen Sie die Funktion $y = -3$ in ein Koordinatensystem ein. Ist das überhaupt eine Funktion nach obiger Definition?

2.1.2 Quadratische Funktionen

Nun wollen wir uns eine etwas interessantere (und flexiblere) Familie von Funktionen anschauen, nämlich **quadratische** Funktionen. Auch hier wollen wir die Funktion erstmal allgemein aufschreiben:

$$y = f(x) = a \cdot x^2 + b \cdot x + c$$

Eine quadratische Funktion hat drei **Parameter**, nämlich a, b und c. Grafisch entspricht die quadratische Funktion einer **Parabel** (vgl. Abb. 2.3). Die Parameter sind hier nicht mehr so einfach grafisch zu interpretieren, aber die vier Beispiele in unten stehender Abbildung geben Anhaltspunkte, was passiert, wenn die Parameterwerte sich ändern.

Aufgaben

1. Sie haben folgende quadratische Gleichung: $y = 2 \cdot x^2 + x - 2$. Berechnen Sie mit der bekannten Lösungsformel $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ die Orte auf der x-Achse, wo die Parabel die Achse schneidet (oder einfacher gesagt die Nullstellen).
2. Verwenden Sie folgenden R-Code, um beliebige quadratische Funktionen grafisch darzustellen, indem Sie die Parameterwerte auf der ersten Code-Zeile verändern.

falsch die lineare Funktion als $y = 1 \cdot x + 1$ zu schreiben.

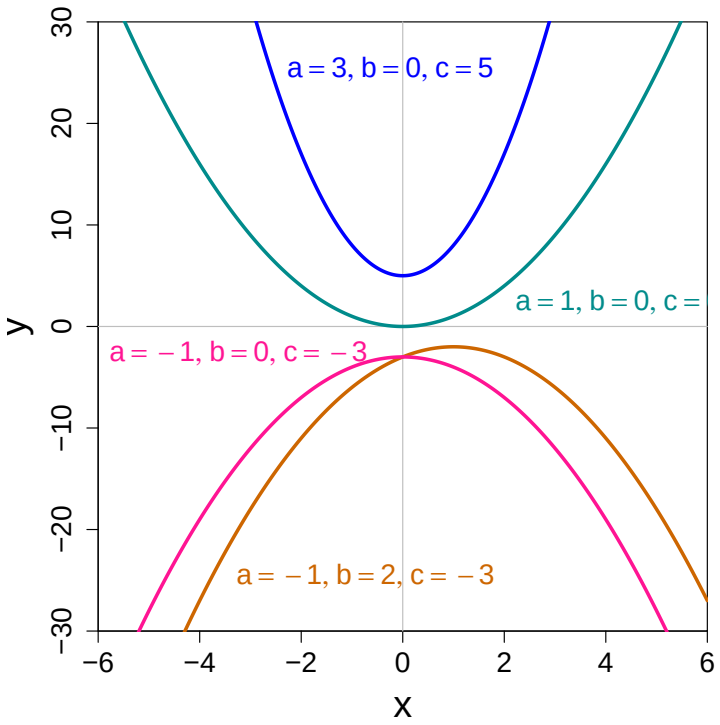


Figure 2.3: Beispiele quadratischer Funktionen.

```r
# Parameter setzen
a <- 2; b <- 0; c <- 1
# Quadratische Funktion
quad <- function(x, a, b, c) {a * x^2 + b * x + c}
# x-Werte
x <- seq(-6, 6, 0.01)
# y-Werte
y <- quad(x, a, b, c)
# Plot
plot(x, y, type = "l", lwd = 2, col = "darkcyan")
```

Sie wundern sich nun vielleicht, könnte man nicht auch eine Funktion antreffen, in der x^3, x^4, etc. vorkommen? Das ist selbstverständlich möglich. In diesem Fall spricht man dann von einem sogenannten **Polynom**. Die höchste Potenz des Arguments x definiert den Grad des Polynoms.

Schauen wir uns doch am besten gleich wieder ein Beispiel an:

$$y = f(x) = 1 \cdot x^4 - 2 \cdot x^3 - 5 \cdot x^2 + 8 \cdot x - 2$$

Die Visualisierung dieser Funktion ist in Abb. 2.4 gegeben. Diese Funktion ist nun bereits enorm flexibel und kann je nach Parameterwerten ganz unterschiedliche Zusammenhänge abbilden.

Aufgaben

1. Eine quadratische Funktion ist ein Polynom welchen Grades?
2. Handelt es sich bei der Funktion $y = 2x^5 + x + 1$ immer noch um ein Polynom? Falls ja, ein Polynom welchen Grades?
3. Handelt es sich bei der Funktion $y = x^{0.5} + 2$ um ein Polynom?

2.1.3 Funktionen mehrerer Argumente

Bisher haben wir nur Funktionen mit **einem Argument** x angeschaut, doch die meisten für das Machine Learning interessanten Funktionen sind Funktionen **mehrerer Argumente**.

Der Einfachheit halber schauen wir uns hier nur mal eine **lineare** Funktion zweier Argumente, nennen wir sie x_1 und x_2, an, denn diese können wir in 3D immer noch visualisieren. Wir betrachten folgende Funktion: $y = f(x_1, x_2) = 1 \cdot x_1 + 0.5 \cdot x_2 + 5$.

Aha! Während eine lineare Funktion eines Arguments grafisch einer Gerade entspricht, sehen wir nun, dass eine lineare Funktion zweier Argumente nichts anderes als eine Ebene darstellt. Wir sehen, dass die Ebene die y-Achse am Punkt 5 schneidet. Etwas schwieriger zu sehen ist die Steigung der Ebene in

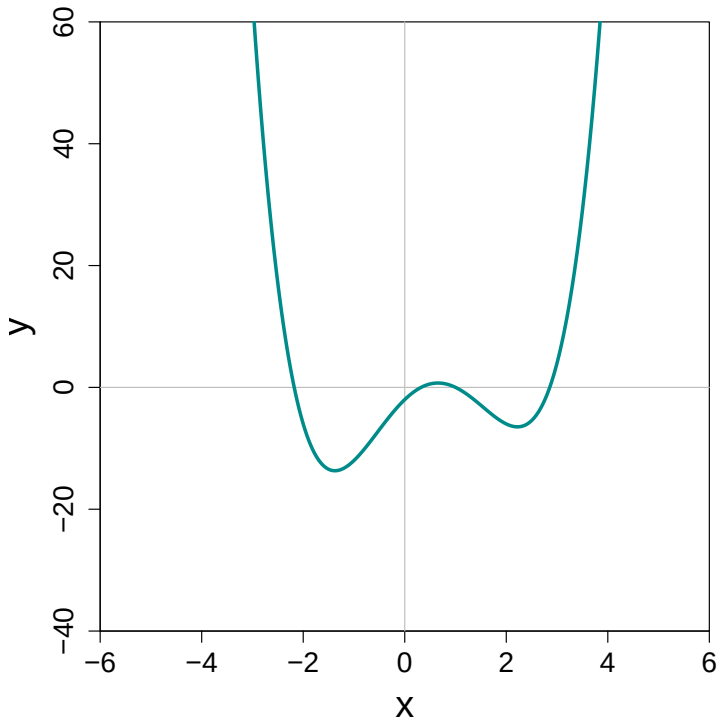


Figure 2.4: Beispiel einer polynomischen Funktion vierten Grades.

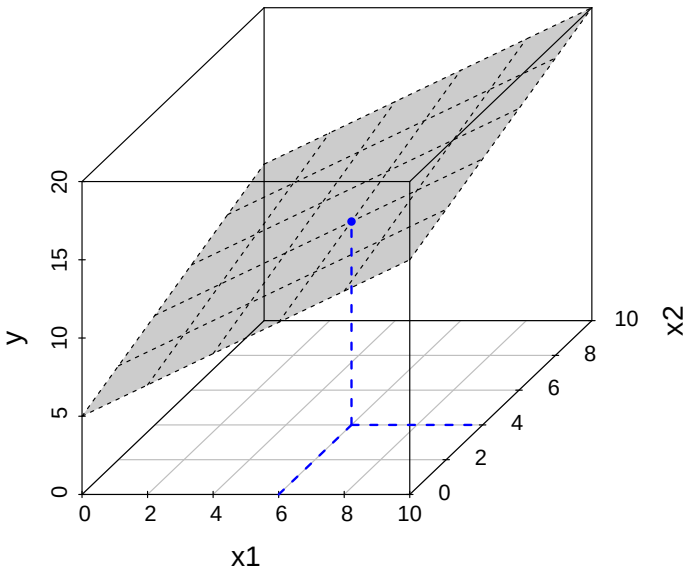


Figure 2.5: Lineare Funktion zweier Argumente (Ebene).

die Richtung der x_1-Achse und in die Richtung der x_2-Achse. Sie können aber vielleicht bereits erraten, dass die (partiellen) Steigungen 1 und 0.5 betragen.

Die Funktion ordnet jeden möglichen Punkt (x_1, x_2) einem Punkt auf der Ebene zu. Wir können zum Beispiel für den in Abb. 2.5 eingezeichneten Punkt $(6, 4)$ den entsprechenden Punkt auf der Ebene ausrechnen:

$$\begin{aligned} y &= 1 \cdot x_1 + 0.5 \cdot x_2 + 5 \\ &= 1 \cdot 6 + 0.5 \cdot 4 + 5 \\ &= 13 \end{aligned}$$

Selbstverständlich könnten wir uns nun auch quadratische Funktionen oder Polynome mehrerer Argumente anschauen, aber darauf verzichten wir vorerst.

2.1.4 Potenzen und Logarithmen

Blabla…

2.2 Integral- und Differentialrechnung

Olteanu materials: Local vs. global minima From a maximization to a minimization problem Basic definition of derivative Differentiation rules local min., max. and saddle point Second derivative test Partial derivatives What is a gradient? What is Hessian? What is Jacobian? Chain rules Lagrange optimization

2.3 Lineare Algebra

Olteanu materials: What is a scalar? What is a vector? What is a matrix? Vector norms Inner products Symmetric, diagonal, square and identity matrix Associative, commutative laws for matrices Matrix addition and multiplication Matrix inversion Eigenvectors and eigenvalues Quadratic form and positive (semi-) definiteness Differentiation rules for matrices

2.4 Wahrscheinlichkeitsrechnung

Olteanu materials: Sample space and axioms of probability Conditional probability definition Discrete vs. continuous random variables Joint probability distributions Expectation and variance, covariance (always for discrete and continuous) Bernoulli, Binomial, Normal, Multivariate Normal, Laplace

2.4.1 Diskrete Zufallsvariablen

Wir werden später sehen, dass im Machine Learning oftmals Dinge als **Zufallsvariablen** modelliert werden. Eine Zufallsvariable X ist eine Variable, für die der konkrete Wert nicht von vornherein klar ist. Wir können mit X zum Beispiel das Resultat eines Münzwurfs modellieren. Die zwei möglichen Resultate sind Kopf und Zahl. Vor dem Münzwurf ist nicht klar, ob Kopf oder Zahl erscheinen wird. Genau darum modellieren wir das Resultat des Münzwurfs als Zufallsvariable.

Es gibt in diesem einfachen Beispiel nur zwei mögliche Resultate (Kopf und Zahl), d.h. die Anzahl möglicher Resultate ist endlich (= nicht unendlich). Darum handelt es sich in diesem Fall um eine **diskrete** Zufallsvariable.

2.5 Verteilungen

Chapter 3

Einführung in das Programmieren mit R

leaRn Materialen

tidymodels

Referenzen auf andere Ressourcen (Hadley et al.)

Chapter 4

Lineare Regression

In diesem Kapitel werden wir uns eingehend mit dem einfachsten Modell für das Regressionsproblem auseinander setzen, nämlich dem linearen Regressionsmodell. Liegt ein Regressionsproblem vor, dann macht es in der Praxis fast immer Sinn mit diesem Modell zu starten und dann die Komplexität nach Bedarf zu erhöhen.

4.1 ML-Modelle im Allgemeinen

Wie bereits in Kapitel 1 gesehen, geht es beim Regressionsproblem darum, eine stetige Variable $y_i \in \mathbb{R}$ möglichst optimal vorherzusagen. Dazu verwenden wir eine oder mehrere Input-Variablen, welche wir kompakt als Vektor $\mathbf{x}_i$ schreiben.

Das Problem ist nur lösbar, falls es tatsächlich einen Zusammenhang zwischen den Input-Variablen $\mathbf{x}_i$ und dem Output y_i gibt. Wir nehmen ganz allgemein an, dass der Zusammenhang zwischen dem Output y_i und den Input-Variablen $\mathbf{x}_i$ mathematisch wie folgt ausgedrückt werden kann:

$$y_i = f(\mathbf{x}_i) + \epsilon$$

- Die Funktion $f(\mathbf{x}_i)$ bezeichnet die **systematische Information**, die wir aus $\mathbf{x}_i$ im Hinblick auf y_i lernen können.
- ϵ ist ein Fehlerterm, der die Differenz zwischen y_i und $f(\mathbf{x}_i)$ abbildet,[1] also den **nicht-lernbaren** (unsystematischen) **Teil**. Der Fehlerterm beinhaltet einerseits den Effekt von Variablen, die uns nicht zur Verfügung stehen, aber einen Einfluss auf den Output y_i haben und andererseits nicht-messbare Variation, oft auch einfach "Noise" genannt. Grob gesagt: alles nicht-messbare.

[1] $\epsilon = y_i - f(\mathbf{x}_i)$

Der Output y_i ergibt sich also aus der Addition eines systematischen Teils $f(\mathbf{x}_i)$ sowie eines Fehlerterms ϵ.

Wichtig: Ziel des Machine Learnings ist es, eine Funktion $\hat{f}(\mathbf{x}_i)$ zu trainieren (schätzen), die der wahren aber unbekannten Funktion $f(\mathbf{x}_i)$ so nahe wie möglich kommt. Im (unrealistischen) Idealfall ist unser trainiertes Modell gleich der wahren Funktion, also $\hat{f}(\mathbf{x}_i) = f(\mathbf{x}_i)$ und wir haben die systematische Information perfekt gelernt. Jedes ML-Modell, das wir uns in diesem Buch anschauen werden, kann als eine mathematische Funktion $\hat{f}(\mathbf{x}_i)$ der Input-Variablen $\mathbf{x}_i$ aufgeschrieben werden. Sobald wir $\hat{f}(\mathbf{x}_i)$ trainiert haben, können wir damit Vorhersagen machen, denn die Vorhersage für einen gegebenen Input-Vektor $\mathbf{x}_0$ ist nichts anderes als der Wert der trainierten Funktion an diesem Punkt, also $\hat{y}_0 = \hat{f}(\mathbf{x}_0)$.

4.2 Das Modell (ausgeschrieben)

Nun wollen wir uns konkret mit dem linearen Regressionsmodell befassen. Das bedeutet nun nichts anderes, als dass wir die allgemein geschriebene Funktion $f(\mathbf{x}_i)$ durch eine konkrete mathematische Funktion ersetzen. Das Modell kann wie folgt geschrieben werden:

$$f(\mathbf{x}_i) = w_0 + w_1 \cdot x_{i1} + w_2 \cdot x_{i2} + ... + w_p \cdot x_{ip}$$

Wir verzichten hier bewusst darauf, den Hut für f zu schreiben, da es sich lediglich um eine allgemein gültige Funktion handelt und noch nichts geschätzt bzw. trainiert wurde. Dieses Modell bzw. diese Funktion hat sogenannte **Parameter**, die es zu schätzen gilt. Hier sind dies die Parameter w_0, w_1, ..., w_p. Wegen der Konstante w_0 haben wir immer einen Parameter mehr als es Input-Variablen hat, also $p + 1$ Parameter.

Diese Parameter sind die Schlüsselzutat in einem ML-Modell. Wir wollen sie optimieren, so dass die trainierte Funktion $\hat{f}(\mathbf{x}_i)$ der wahren Funktion $f(\mathbf{x}_i)$ möglichst nahe kommt.

4.3 Das Modell (kompakt)

Sie sehen oben, dass es ziemlich umständlich sein kann, das lineare Regressionsmodell aufzuschreiben, insbesondere wenn wir viele Input-Variablen haben. Mithilfe von **Vektoren und Matrizen** können wir das Modell viel kompakter aufschreiben.

Wir haben in Kapitel 1 bereits gesehen, dass die Input-Variablen für eine Beobachtung i als Spaltenvektor geschrieben werden können. Wir modifizieren

diesen Spaltenvektor in einem ersten Schritt, indem wir an erster Stelle eine 1 einfügen, also:[2]

$$\mathbf{x}_i = \begin{pmatrix} 1 \\ x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}$$

Nun stecken wir die Parameter des Modells ebenfalls in einen Spaltenvektor:

$$\mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{pmatrix}$$

Wir können nun das lineare Regressionsmodell (für die Beobachtung i) als **Skalarprodukt** dieser beiden Vektoren aufschreiben:

$$f(\mathbf{x}_i) = \mathbf{w}'\mathbf{x_i} \tag{4.1}$$

$$= \begin{pmatrix} w_0 & w_1 & w_2 & \dots & w_p \end{pmatrix} \begin{pmatrix} 1 \\ x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix} \tag{4.2}$$

$$= w_0 \cdot 1 + w_1 \cdot x_{i1} + w_2 \cdot x_{i2} + \cdots + w_p \cdot x_{ip} \tag{4.3}$$

Die Form $\mathbf{w}'\mathbf{x_i}$ ist schon ziemlich kompakt, aber es geht noch besser. Wir können nämlich das Modell gleich für alle n Beobachtungen (und nicht nur für die i-te Beobachtung) aufschreiben. Dazu müssen wir die Input-Variablen für jede Beobachtung i in einer Matrix anordnen:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \cdots & \cdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

Die Matrix $\mathbf{X}$ wird typischerweise **Design Matrix** genannt. Die erste Zeile enthält die Input-Variablen für die erste Beobachtung, die zweite Zeile die Input-Variablen für die zweite Beobachtung, usw. Nun können wir das Modell mithilfe

[2] So müssen wir die Konstante w_0 nicht separat aufschreiben.

einer Multiplikation zwischen der Design Matrix $\mathbf{X}$ und dem Spaltenvektor $\mathbf{w}$ in einem Schritt für alle Beobachtungen aufschreiben:

$$f(\mathbf{X}) = \mathbf{X}\mathbf{w} \tag{4.4}$$

$$= \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \cdots & \cdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \cdots \\ w_p \end{pmatrix} \tag{4.5}$$

$$= \begin{pmatrix} w_0 \cdot 1 + w_1 \cdot x_{11} + w_2 \cdot x_{12} + \cdots + w_p \cdot x_{1p} \\ w_0 \cdot 1 + w_1 \cdot x_{21} + w_2 \cdot x_{22} + \cdots + w_p \cdot x_{2p} \\ \cdots \\ w_0 \cdot 1 + w_1 \cdot x_{n1} + w_2 \cdot x_{n2} + \cdots + w_p \cdot x_{np} \end{pmatrix} \tag{4.6}$$

Überprüfen wir doch noch kurz die Dimensionen von obigem Matrix-Vektor Produkt. Die Matrix $\mathbf{X}$ hat n Zeilen und $p+1$ Spalten und darum eine Dimensionalität von $n \times (p+1)$. Der Spaltenvektor $\mathbf{w}$ hat Dimensionalität $(p+1) \times 1$. Das Matrix-Vektor Produkt hat dementsprechend eine Dimensionalität von $n \times 1$, genau was wir erwarten würden, nämlich einen Vektor mit den Vorhersagen für alle n Beobachtungen.

Warum wir all das tun, werden wir weiter unten sehen. Es wird unser Leben viel einfacher machen! Versuchen Sie diesen Abschnitt hier gut zu verstehen, so dass Sie sobald wie möglich mit der Matrixschreibweise von Modellen vertraut sind.

4.4 Modell Training

Grob gesagt rechnen wir ein ML-Modell in zwei Schritten. In einem **ersten Schritt** entscheiden wir uns für die funktionale Form unseres Modells $\hat{f}(\mathbf{x}_i)$. Man nennt dies in der Fachsprache **Model Selection**. Wir betrachten hier nur mal den vereinfachten Fall, in dem wir nur eine x_i-Variable pro Beobachtung als Input haben. Folgende Funktionen bzw. Modelle sind mögliche Kandidaten:

- $f(x_i) = b_0 + b_1 \cdot x_i$ (einfache lineare Regression)
- $f(x_i) = b_0 + b_1 \cdot x_i + b_2 \cdot x_i^2$ (polynomische Regression)
- $f(x_i) = \begin{cases} \bar{y}_1, & \text{falls } x_i > x^* \\ \bar{y}_2, & \text{sonst} \end{cases}$

Wir werden mit unserer Wahl der Funktion nie genau die wahre aber unbekannte Funktion $f(\mathbf{x}_i)$ treffen, aber wir versuchen möglichst nahe daran zu kommen.

"No Free Lunch" Theorem

Das *No Free Lunch* Theorem besagt, dass es kein universal bestes Modell gibt. Das heisst, dass es je nach Problem und Datensatz andere Modelle bzw. Funktionen braucht, um gute Vorhersagen zu machen. Das ist der Hauptgrund, warum wir Ihnen möglichst viele verschiedene Tools mit auf den Weg geben wollen.

In einem **zweiten Schritt** geht es darum, die Parameter des im ersten Schritt gewählten Modells zu schätzen. Man nennt dies in der Fachsprache **Model Fitting / Training**, weil wir versuchen, das Modell so gut wie möglich an die Daten zu fitten. Sie haben bereits eine Methode kennen gelernt, wie die Parameter eines Modells geschätzt werden können:

- Kleinstquadratemethode oder auf English "Least squares" (Modul EMBA)

Die Grundidee beim **Model Fitting** ist, eine **Kostenfunktion** (engl. *Loss Function*) aufzustellen, die von den Parameterwerten der Funktion $\hat{f}(\mathbf{x}_i)$ abhängt. Und nun der Schlüsselschritt:

Während des Trainings verändern wir die Parameterwerte so lange, bis die Kostenfunktion ein **Minimum** erreicht.

Nun haben wir die optimalen Parameterwerte und darum auch das optimale Modell $\hat{f}(\mathbf{x}_i)$ gefunden. Für das Regressionsproblem haben Sie bereits eine mögliche Kostenfunktion (Stichwort *Summe der quadrierten Residuen*) kennen gelernt:

$$J(\text{Modellparameter}) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \hat{f}(\mathbf{x}_i) \right)^2$$

Im Vergleich zur Summer der quadrierten Residuen haben wir hier noch den Faktor $\frac{1}{2n}$ drin. Dieser Faktor macht daraus eine Art Mittelwert und darum wird diese Kostenfunktion typischerweise **Mean Squared Error** (MSE) genannt.

Sie wundern sich nun vielleicht, wie diese Kostenfunktion von den Modellparameter abhängt, da diese in obiger Formel ja gar nicht ersichtlich sind. Schreiben wir die Kostenfunktion doch mal etwas um (unter der Annahme, dass es nur eine Input-Variable x_i gibt und wir ein einfaches lineares Regressionsmodell anwenden wollen):

$$J(\hat{b}_0, \hat{b}_1) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \hat{f}(x_i)\right)^2 \tag{4.7}$$

$$= \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - (\hat{b}_0 + \hat{b}_1 \cdot x_i)\right)^2 \tag{4.8}$$

$$= \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \hat{b}_0 - \hat{b}_1 \cdot x_i\right)^2 \tag{4.9}$$

$$\tag{4.10}$$

Nun ist offensichtlich, wie die Kostenfunktion J von den Modellparameter $\hat{b}_0$ und $\hat{b}_1$ abhängt. Im ML gibt es nun viele verschiedene Arten, wie man für die beiden Modellparameter die optimalen Werte findet. Hier ist die Lösung zum Glück einfach, denn es gibt eine sogenannte **analytische Lösung**, die man mit der oben erwähnten Kleinstquadratemethode findet. Wie Sie diese analytische Lösung finden, haben Sie bei Thomas Heimsch in Aufgabe 5 der Übungsserie 2 gelernt (zwar mit anderer Notation, aber vom Prinzip her gleich).

Für andere Modelle gibt es oft leider keine analytische Lösung für die optimalen Parameterwerte. In diesem Fall werden wir **iterative Optimierungsverfahren** anwenden, mit denen wir uns langsam den optimalen Parameterwerten annähern (dazu später mehr).

Grundsätzlich gilt: je besser die Modellparameter geschätzt werden können, desto kleiner sind die quadrierten Differenzen, $\left(y_i - \hat{f}(x_i)\right)^2$, und desto kleiner der Wert der Kostenfunktion.

Optional: Zerlegung des Vorhersagefehlers

Wir wollen hier kurz anschauen, wie der **Erwartungswert** des quadrierten Fehlers, $\left(y_i - \hat{f}(\mathbf{x}_i)\right)^2$, in zwei Komponenten zerlegt werden kann.

Dazu gilt folgendes:

- Von oben wissen wir, dass $y_i = f(\mathbf{x}_i) + \epsilon$ gilt.
- Wir nehmen an, dass der Erwartungswert des unsystematischen Teils ϵ Null ist, also $\mathrm{E}(\epsilon) = 0$.
- Allgemeine Regel zur Varianz einer Zufallsvariable: $\mathrm{Var}(\epsilon) = \mathrm{E}(\epsilon^2) - \mathrm{E}(\epsilon)^2 = \mathrm{E}(\epsilon^2) - 0^2 = \mathrm{E}(\epsilon^2)$.
- $\hat{f}$ und $\mathbf{x}_i$ sind fix und gegeben (keine Zufallsvariablen) und darum gilt $\mathrm{E}\left(\hat{f}(\mathbf{x}_i)\right) = \hat{f}(\mathbf{x}_i)$.

Nun können wir den **Erwartungswert** des quadrierten Fehlers rechnen:

$$\mathrm{E}\left[\left(y_i - \hat{f}(\mathbf{x}_i)\right)^2\right] = \mathrm{E}\left[\left(f(\mathbf{x}_i) + \epsilon - \hat{f}(\mathbf{x}_i)\right)^2\right] \tag{4.11}$$

$$= \mathrm{E}\left[f(\mathbf{x}_i)^2 - 2 \cdot f(\mathbf{x}_i) \cdot \hat{f}(\mathbf{x}_i) + \hat{f}(\mathbf{x}_i)^2 + 2 \cdot \epsilon \cdot f(\mathbf{x}_i) - 2 \cdot \epsilon \cdot \hat{f}(\mathbf{x}_i) + \epsilon^2\right] \tag{4.12}$$

$$= f(\mathbf{x}_i)^2 - 2 \cdot f(\mathbf{x}_i) \cdot \hat{f}(\mathbf{x}_i) + \hat{f}(\mathbf{x}_i)^2 + 2 \cdot \mathrm{E}(\epsilon) \cdot f(\mathbf{x}_i) - 2 \cdot \mathrm{E}(\epsilon) \cdot \hat{f}(\mathbf{x}_i) + \mathrm{E}(\epsilon^2) \tag{4.13}$$

$$= f(\mathbf{x}_i)^2 - 2 \cdot f(\mathbf{x}_i) \cdot \hat{f}(\mathbf{x}_i) + \hat{f}(\mathbf{x}_i)^2 + 2 \cdot 0 \cdot f(\mathbf{x}_i) - 2 \cdot 0 \cdot \hat{f}(\mathbf{x}_i) + \mathrm{Var}(\epsilon) \tag{4.14}$$

$$= f(\mathbf{x}_i)^2 - 2 \cdot f(\mathbf{x}_i) \cdot \hat{f}(\mathbf{x}_i) + \hat{f}(\mathbf{x}_i)^2 + \mathrm{Var}(\epsilon) \tag{4.15}$$

$$= \left(f(\mathbf{x}_i) - \hat{f}(\mathbf{x}_i)\right)^2 + \mathrm{Var}(\epsilon) \tag{4.16}$$

Der erste Teil auf der rechten Seite der Formel beschreibt den **reduzierbaren Fehler** und der zweite Teil den **nicht-reduzierbaren Fehler**. Wir sehen also auch hier: es ist sehr wichtig, dass wir eine Funktion $\hat{f}(\mathbf{x}_i)$ schätzen, welche dem wahren funktionalen Zusammenhang $f(\mathbf{x}_i)$ möglichst nahe kommt.

Komplexität des Modelltrainings erwähnen.

4.5 Interpretierbarkeit

Wir werden in diesem Modul sehr einfache, aber auch sehr komplexe Funktionen kennen lernen. Je komplexer die Funktionen sind, desto mehr haben wir es mit einer **Blackbox** zu tun und desto schwieriger wird es, das Modell zu interpretieren. Relativ unflexible Modelle wie z.B. die lineare Regression sind einfach interpretierbar. Die geschätzten Koeffizienten $\hat{b}_1$, $\hat{b}_2$, ... erlauben uns direkt zu quantifizieren, was der Effekt der verschiedenen Input-Variablen ist. Im Gegensatz dazu führen komplexe Modelle (mit vielen zu optimierenden Parameter) oft zu einer sehr guten Vorhersagegüte, weil komplexe Modelle flexibel sind und darum komplexe Zusammenhänge zwischen $\mathbf{x}_i$ und y_i modellieren können.

Kurz Variable Importance erwähnen

4.6 Regularisierte Regression

Kurz auf Variable Selection (siehe ISLR) eingehen.

Hier ist nun der richtige Moment, um mal kurz auf die Theorie zum linearen Regressionsmodell **mit Regularisierung** einzugehen.

Wir haben in der Einführung gelernt, dass wir für das lineare Regressionsproblem folgende Kostenfunktion minimieren (den Mean Squared Error oder kurz MSE):

$$J(\text{Modellparameter}) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \hat{f}(\mathbf{x}_i) \right)^2$$

Normalerweise gibt es hier sogar eine analytische Lösung, d.h. es gibt Formeln wie die optimalen Parameter des Modells aus den Trainingsdaten berechnet werden können. Das ist mathematisch aber nur dann möglich, wenn die Anzahl Beobachtungen im Trainingsdatensatz grösser ist als die Anzahl Input-Variablen. Oder mathematisch ausgedrückt, nur wenn $n > p$.

Wenn $n < p$ gibt es keine analytische Lösung für das Problem. Wir haben zu wenig Datenpunkte, um die vielen Parameter des Modells schätzen zu können. Selbst wenn $n > p$, aber p (Anzahl Input-Variablen) sehr gross ist, sind die Schätzungen oft nicht sehr gut, weil es dann zu Overfitting kommt. Die Lösung für dieses Problem ist **Regularisierung**.

Regularisierung bedeutet eigentlich nichts anderes, als dass wir die obige Kostenfunktion modifizieren. Dabei gibt es zwei mögliche Varianten, **Ridge** Regularisierung oder **LASSO**:

- Kostenfunktion für Ridge Regularisierung: $J(b_0, b_1, \dots, b_p) = \text{MSE} + \lambda \cdot \sum_{j=1}^{p} b_j^2$
- Kostenfunktion für LASSO Regularisierung: $J(b_0, b_1, \dots, b_p) = \text{MSE} + \lambda \cdot \sum_{j=1}^{p} |b_j|$

Diese beiden modifizierten Kostenfunktionen haben ein bisschen Erklärungsbedarf:

- In beiden Varianten wollen wir **gleichzeitig** den MSE so klein wie möglich und eine spezielle Summe über die Modellparameter (d.h. den Regularisierungsterm) so klein wie möglich machen. Das sind zwei konkurrenzierende Ziele und während des Trainings muss der beste Tradeoff gefunden werden.
- Bei Ridge ist der Regularisierungsterm eine Summe über die quadrierten Modellparameter. Das Quadrieren stellt sicher, dass sich positive und negative Parameterwerte nicht gegenseitig kompensieren.
- Bei LASSO ist der Regularisierungsterm eine Summe über die absoluten Werte der Modellparameter.
- Der Regularisierungsterm enthält die Konstante b_0 **nicht** (Summe startet bei $j = 1$ und nicht bei $j = 0$).
- Der Hyperparameter λ legt fest, wie viel Gewicht der Regularisierungsterm bekommt. Je grösser λ, desto stärker "bestrafen" wir komplexe Modelle.

- Die Optimierung der LASSO Kostenfunktion hat einen gewichtigen Vorteil gegenüber Ridge: unwichtige Parameter werden bei LASSO automatisch auf 0 gesetzt. Das Modell nimmt also selbständig eine Selektion der wichtigen Variablen vor.

Fragen:

- Was passiert wenn $\lambda = 0$?
- Was passiert wenn $\lambda \to \infty$?

Die grosse Frage ist nun, wie wir den Wert für den Hyperparameter λ wählen. Das schauen wir uns im nächsten Abschnitt an.

4.7 Bias-Variance Tradeoff

Was gibt es bei der Wahl des optimalen Modells zu berücksichtigen?

Das zentrale Thema bei der Wahl des Modells ist der **Bias-Variance Tradeoff**. Wir wollen ein Modell wählen, das weder zu viel Bias noch zu viel Varianz hat. In der Einführung zum Machine Learning haben wir bereits gelernt, dass der erwartete (quadrierte) Fehler in einen reduzierbaren und in einen nicht-reduzierbaren Fehler aufgeteilt werden kann. In der Einführung haben wir angenommen, dass $\hat{f}$ fix ist. Nun lockern wir diese Annahme und nehmen an, dass $\hat{f}$ eine Zufallsvariable ist, welche je nach Trainingsdatensatz eine unterschiedliche Form annimmt. Nach einer relativ komplizierten Herleitung kann man zeigen, dass der **erwartete quadrierte Fehler** für eine gegebene Testbeobachtung $(y_i, \mathbf{x}_i)$ wie folgt zerlegt werden kann:

$$\mathrm{E}\left[\left(y_i - \hat{f}(\mathbf{x}_i)\right)^2\right] = \mathrm{Var}\left(\hat{f}(\mathbf{x}_i)\right) + \left[\mathrm{Bias}\left(\hat{f}(\mathbf{x}_i)\right)\right]^2 + \mathrm{Var}(\epsilon)$$

Diese Formulierung beschreibt den erwarteten (quadrierten) Fehler, den wir erhalten würden, wenn wir mit einer grossen Anzahl Trainings-Datensätzen jeweils einzeln f schätzen würden und dann mit Testbeobachtung $(y_i, \mathbf{x}_i)$ evaluieren würden. D.h., es ist eine ziemlich theoretische Angelegenheit, denn in der Praxis haben wir ja immer nur einen Trainingsdatensatz zur Verfügung. Aber diese Überlegungen helfen uns, das richtige Modell zu wählen.

Schauen wir uns kurz die einzelnen Komponenten auf der rechten Seite etwas genauer an:

- $\mathrm{Var}\left(\hat{f}(\mathbf{x}_i)\right)$ misst, wie stark sich $\hat{f}$ ändert, wenn wir einen anderen Trainings-Datensatz verwenden. Ein Modell mit hoher Varianz passt sich jeweils sehr stark an die Trainingsdaten an. Je kleiner diese Varianz, desto tiefer der erwartete quadrierte Fehler.

- $\left[\text{Bias}\left(\hat{f}(\mathbf{x}_i)\right)\right]^2$ ist der quadrierte Bias und misst die systematische Abweichung vom wahren unbekannten f. Wir wollen natürlich auch den Bias möglichst klein halten.
- Die dritte Komponente, $\text{Var}(\epsilon)$, kennen Sie bereits. Es ist der nicht-redzierbare Fehler.

Ein Modell mit **viel Bias** führt zu einer schlechten Vorhersagequalität (auf Trainings- und Testdaten), weil das Modell zu rigide ist, um den wahren Zusammenhang zwischen der Output-Variable und den Features zu modellieren. Beispiel: wir verwenden ein einfaches lineares Regressionsmodell, um einen stark nicht-linearen Zusammenhang zwischen x und y zu modellieren. Im Fall von Modellen mit viel Bias spricht man auch von **Underfitting**.

Ein Modell mit **viel Varianz** führt zu einer hervorragenden Vorhersagequalität auf den Trainingsdaten, aber zu einer sehr schlechten Vorhersagequalität auf den Testdaten. Das Problem hier ist, dass das Model zu flexibel ist gemessen an der Grösse des Trainingsdatensatzes. Das Modell passt sich so zu stark an die Trainingsdaten an und modelliert auch sogenanntes **Noise** (und nicht nur das **Signal** in den Daten). Beispiel: wir modellieren ein neuronales Netzwerk, haben aber nur einen Trainingsdatensatz von einigen hundert Beobachtungen. Im Fall von Modellen mit viel Varianz spricht man auch von **Overfitting**.

Warum spricht man von einem **Tradeoff**? Flexiblere Modelle haben oft kleinen Bias, aber hohe Varianz, während unflexible Modelle oft eine kleine Varianz, aber einen hohen Bias haben. Es existiert also ein Tradeoff zwischen Bias und Varianz und wir wollen beim Modellieren und vor allem beim Hyperparameter Tuning den optimalen Tradeoff finden. (Die Intuition des Bias-Varianz Tradeoffs ist übrigens auch auf das Klassifikationsproblem übertragbar.)

Bias-Variance Tradeoff bei Regularisierter Regression

In unserem Beispiel wenden wir ein regularisiertes Regressionsmodell an. Hier spielt der Hyperparameter λ (in R: `penalty`) eine zentrale Rolle für den Tradeoff zwischen Bias und Variance. Ein zu tiefer Wert für λ kann zu einem zu flexiblen Modell mit viel Varianz führen. Ein zu hoher Wert für λ führt zu einem zu rigiden Modell mit viel Bias.

Wichtig:

- Indem wir den Hyperparameter via Resampling optimieren, wählen wir automatisch ein Modell mit einem guten Tradeoff zwischen Bias und Varianz!
- Mit grossen Datensätzen ist das Problem des Overfittings weniger dramatisch. Wir haben genügend Trainingsdaten, dass selbst flexible Modelle nicht zu stark overfitten. Das ist bei unserem Beispiel der Fall (wir haben einen ziemlich grossen Trainingsdatensatz). Darum ist der optimale Hyperparameter in unserem Beispiel `penalty = 0.001`, also eher einer der kleineren Werte.

Eine letzte Überlegung bezüglich Modellselektion geht folgendermassen: wenn wir mehrere Modelle haben, die ähnlich gut performen, dann wählen wir das einfachste (kleinste) oder **am wenigsten komplexe Modell**. Man nennt dies **Occam's Razor**. William of Occam war ein Englischer Mönch und hat dieses Prinzip in einem anderen Kontext erstmals formuliert.

4.8 Polynomische Regression

Wir machen hier nun einen kurzen Abstecher in die **polynomische Regression**, denn diese eignet sich sehr gut, um den Bias-Variance Tradeoff zu illustrieren.

Ein **ganz wichtiger Punkt**: das polynomische Regressionsmodell ist immer noch **linear in den Parametern**, es handelt sich also immer noch um ein lineares Modell. Sie sehen aber an obigen Modellkurven, dass dieses "lineare" Modell sehr wohl in der Lage ist, nicht-lineare Zusammenhänge zwischen x und y zu fitten!

4.9 Lineare Regression in R

Base R vs. `tidymodels`

Chapter 5

Lineare Klassifikation

Chapter 6

Machine Learning Pipeline

Chapter 7

Decision Trees

Chapter 8

Ensembles

Chapter 9

Support Vector Machines

Chapter 10

Artificial Neural Networks

Chapter 11

Convolutional Neural Networks

Chapter 12

Recurrent Neural Networks

Chapter 13

Generative AI