Lecture Notes

Key Technologies (AI)

Master Wirtschftinformatik

Umberto Michelucci

Lucerne University of Applied Sciences and Arts (HSLU)

Course	Key Technologies (AI)
Term	Fall 2025
Version	3.1
Location	Lucerne, Switzerland
Contact	umberto.michelucci@hslu.ch

These notes provide a gentle, practice-oriented introduction to artificial intelligence, covering key milestones, core terminology, and typical project workflows. They are intended for students without a technical background, emphasizing clear concepts, intuition, and real-world examples.

Inhaltsverzeichnis

Warning

Hinweis zur Übersetzung. Der Originaltext wurde auf Englisch verfasst und automatisch mit *ChatGPT v5.0* ins Deutsche übersetzt. Es fand keine inhaltliche Prüfung statt. Etwaige Fehler oder Ungenauigkeiten sind möglich; im Zweifel gilt der englische Originaltext.

Kapitel 1

Grundlagen der Künstlichen Intelligenz

Chapter Abstract

Dieses Kapitel führt in zentrale Ideen und den Kontext des modernen maschinellen Lernens ein. Wir beginnen mit einer kurzen Geschichte von Turings Frage „Können Maschinen denken?“ über Perzeptrons, Backpropagation, Support Vector Machines und die jüngste Dominanz des Deep Learning und generativer Modelle. Wir klären die Beziehung zwischen *KI*, *Maschinellem Lernen*, *Deep Learning* und *Generativer KI* und heben Unterschiede in Umfang, Methoden und Anwendungen hervor. Wir vergleichen Grundlagen- und angewandte Forschung und unterscheiden Forschung, Technologieentwicklung und Produktarbeit mit ihren Zielen, Evidenzen, Fähigkeiten und Erfolgskriterien. Eine einsteigerfreundliche *ML-Pipeline* (Datenerhebung, Aufbereitung, Training, Evaluation, Deployment) bietet eine praktische Perspektive für Projekte. Wir untersuchen repräsentative Anwendungsfelder (Gesundheitswesen, Finanzen, Einzelhandel, Mobilität, öffentlicher Sektor, Kreativindustrien) und skizzieren Chancen und Risiken. Abschließend führen wir den Technologie-Lebenszyklus ein und bieten reflektierende Impulse dazu, was „intelligentes Verhalten“ ausmacht und wie man es misst. Der Schwerpunkt liegt auf Intuition, klaren Konzepten und entscheidungsorientierter Rahmung.

1.1 Kurze Geschichte des Maschinellen Lernens

Maschinelles Lernen erschien erstmals in der Mitte des 20. Jahrhunderts. Alan Turing, einer der Pioniere, stellte in seinem wegweisenden Aufsatz „Computing Machinery and Intelligence“ [1] grundlegende Fragen zu dem Potenzial von Maschinen, Aspekte menschlicher Intelligenz zu imitieren. In dieser Arbeit

bemühte sich Turing, die Frage „können Maschinen denken?“ zu beantworten und führt das Konzept ein, das heute als Turing-Test bekannt ist. Dieser ist dazu bestimmt, die Fähigkeit einer Maschine zu bewerten, intelligentes Verhalten zu zeigen, das von dem eines Menschen nicht zu unterscheiden ist. Er beinhaltet, dass ein menschlicher Befrager ein Gespräch mit sowohl einem Menschen als auch einer Maschine führt, beide außer Sicht. Die Maschine gilt als habe den Test bestanden, wenn der Befrager sie nicht konsistent vom Menschen unterscheiden kann. Im gesamten Aufsatz spricht Turing verschiedene Einwände gegen die Vorstellung maschineller Intelligenz an und entkräftet sie. Diese Einwände reichen von theologischen Argumenten über die Auffassung, Bewusstsein sei eine Voraussetzung für Intelligenz, bis hin zu der Idee, dass Maschinen unfähig zu Fehlern oder zum Lernen aus Erfahrungen seien. Turing diskutiert auch das Potenzial digitaler Computer, die sich zu dieser Zeit in einem frühen Entwicklungsstadium befanden. Er hebt ihre Fähigkeit hervor, jeden Prozess formalen Schließens zu simulieren, erkennt ihre bestehenden Einschränkungen an, sagt aber bedeutende Fortschritte ihrer Fähigkeiten voraus. Ein faszinierender Teil von Turings Aufsatz ist seine Spekulation über die Möglichkeit, dass Maschinen lernend sich im Laufe der Zeit weiterentwickeln. Er schlägt vor, dass es effektiver sein könnte, anstatt eine Maschine mit einem umfangreichen Verständnis der Welt auszustatten, eine einfachere Maschine zu entwickeln und sie aus ihren Interaktionen und Erfahrungen lernen zu lassen (das ist schließlich das, worum es beim maschinellen Lernen geht). Dieser Aufsatz ist wichtig wegen seines philosophischen Ansatzes, da Turing über die weitergehenden Implikationen und zukünftigen Möglichkeiten maschineller Intelligenz nachdenkt. Seine Einsichten und Hypothesen haben die Entwicklung der Künstlichen Intelligenz tiefgreifend beeinflusst und prägen bis heute Diskussionen über die Natur von Intelligenz und Bewusstsein.

Ende der 1950er Jahre wurde das Perzeptron, das erste neuronale Netz, von Rosenblatt eingeführt [2]. Diese Entwicklung legte den Grundstein für zukünftige Forschung in neuronalen Netzen. Das Perzeptronmodell wurde als vereinfachte mathematische Abstraktion eines biologischen Neurons konzipiert und dazu entworfen, bestimmte Arten von Klassifikation durchzuführen. Es war in der Lage, die optimalen Gewichtungskoeffizienten automatisch zu lernen, die dann mit Eingangsmerkmalen multipliziert werden, um zu bestimmen, ob ein Neuron feuert oder nicht, was im Wesentlichen eine Entscheidung auf der Grundlage einer linearen Kombination seiner Eingangssignale darstellt. Rosenblatts Arbeit war bahnbrechend, weil sie zeigte, wie eine Maschine so programmiert werden konnte, dass sie aus Daten lernt, ein Konzept, das zu dieser Zeit relativ neuartig war. Es wurde gezeigt, dass das Perzeptron durch einen Prozess des Anpassens der Gewichte basierend auf Fehlern, die in früheren Vorhersagen gemacht wurden, lernen konnte, eine frühe Form dessen, was später als überwachtes Lernen bekannt werden sollte (siehe nächsten Abschnitt für eine Erklärung des Begriffs *supervised*). Während das ursprüngliche Perzeptron Einschränkungen hatte, insbesondere seine Unfähigkeit, Daten zu verarbeiten, die nicht linear separierbar sind (was später durch die Erfindung von Mehrschicht-Perzeptrons adressiert wurde), legte Rosenblatts Aufsatz die erste Grundlage für weitere Forschung zu

neuronalen Netzen. Er löste eine Welle von Interesse und Optimismus hinsichtlich des Potenzials von Maschinen aus, zu lernen und Entscheidungen zu treffen, und ebnete so den Weg für das moderne Feld des maschinellen Lernens und der Künstlichen Intelligenz.

Das Buch „Perceptrons“ von Marvin Minsky und Seymour Papert [3], veröffentlicht 1969, steht als ein kritisches und einflussreiches Werk. Dieses Buch ist besonders bemerkenswert für seine Analyse und Kritik des Perzeptrons von Rosenblatt. Minskys und Paperts Arbeit lieferte eine gründliche mathematische Kritik des Perzeptrons, indem sie auf seine Fähigkeiten und, noch wichtiger, seine Grenzen einging. Die Autoren zeigten, dass Perzeptrons, so wie sie damals konzipiert waren, nicht in der Lage waren, einige relativ einfache, aber grundlegende Probleme zu lösen, wie das XOR-Problem, das die korrekte Klassifikation von Eingaben beinhaltet, die nicht linear separierbar sind. Diese Einschränkung begrenzte den Bereich der Probleme erheblich, auf die das Perzeptron angewendet werden konnte.

Die in „Perceptrons“ enthaltene Kritik war so überzeugend, dass sie zu einer signifikanten Reduktion des Interesses und der Finanzierung für die Forschung an neuronalen Netzen führte. Diese Periode, oft als der erste „KI-Winter“ bezeichnet, war durch Skepsis und gesenkte Erwartungen an das Feld der Künstlichen Intelligenz gekennzeichnet, insbesondere im Bereich der neuronalen Netze. Langfristig legten jedoch die Strenge und Tiefe von Minskys und Paperts Analyse auch die Grundlage für spätere Fortschritte. Ihre Arbeit hob die Notwendigkeit komplexerer und geschichteter Architekturen neuronaler Netze hervor, was schließlich zur Entwicklung von Mehrschicht-Perzeptrons und Deep-Learning-Techniken führte. Auf diese Weise spielte „Perceptrons“ eine paradoxe Rolle in der Geschichte des maschinellen Lernens: Während es die Begeisterung für die Forschung an neuronalen Netzen vorübergehend dämpfte, bereitete es auch die Bühne für einige der bedeutendsten Durchbrüche des Feldes in den folgenden Jahrzehnten. Die Wirkung von „Perceptrons“ ist ein Zeugnis für die Bedeutung kritischer Analyse und rigoroser Evaluation beim Fortschritt wissenschaftlicher Felder und dient sowohl als warnendes Beispiel für die Risiken von Hype und ungetesteten Annahmen als auch als Leuchtfeuer, das den Weg zu robusteren und leistungsfähigeren Modellen im maschinellen Lernen weist.

Die Renaissance der neuronalen Netze wurde in den 1980er Jahren mit vielen wichtigen neuen Ergebnissen ausgelöst. Das Jahrzehnt von 1980 bis 1990 war eine Periode signifikanter Fortschritte im Feld des maschinellen Lernens. Eine der großen Entwicklungen war die Popularisierung von Entscheidbaumalgorithmen (zuerst in den 1960ern entstanden) für Klassifikationsaufgaben. Die Einführung des ID3-Algorithmus durch Quinlan im Jahr 1986 [4] markierte einen Schlüsselmoment in diesem Bereich. ID3 war ein revolutionärer Schritt, der das Konzept der Informationsentropie, ein Maß aus der Informationstheorie, nutzte, um an jedem Ast des Baums das Attribut auszuwählen, das die Daten auf die informativste Weise partitioniert. Auf den Erfolg von ID3 folgten weitere Entwicklungen und Iterationen, die zu ausgefeilteren Algorithmen wie C4.5 und CART (Classification and Regression Trees) führten, welche die Entscheidungsfähigkeiten dieser Modelle erweiterten und verfeinerten. Diese Fortschritte

schlossen den Umgang mit kategorialen und numerischen Daten, den Umgang mit fehlenden Daten und die Verbesserung der Recheneffizienz ein. Die Entwicklung von Entscheidbäumen hatte einen erheblichen Einfluss auf das Feld des maschinellen Lernens, da sie eine Grundlage für komplexere Modelle und Algorithmen bereitstellte. Ihre Fähigkeit, einen komplexen Entscheidungsprozess in eine einfachere Form zu zerlegen, machte sie nicht nur zu leistungsfähigen Werkzeugen im prädiktiven Modellieren, sondern trug auch zum breiteren Verständnis datengetriebener Entscheidungsprozesse bei.

Einer der großen Fortschritte im Bereich der neuronalen Netze war die Entwicklung des Backpropagation-Algorithmus durch Rumelhart, Hinton und Williams im Jahr 1986 [5], der das Training neuronaler Netze revolutionierte. Backpropagation, formal bekannt als *Rückpropagation der Fehler*, ist eine Methode, die bei künstlichen neuronalen Netzen verwendet wird, um die Gewichtsaktualisierungen zu berechnen, die von jedem Neuron nach der Verarbeitung eines Datenbatches beigetragen werden. Sie ist ein Grundpfeiler des Trainings neuronaler Netze und ermöglicht die effiziente Berechnung von Gradienten. Die Wurzeln der Backpropagation lassen sich auf frühe Arbeiten in den 1960er und 1970er Jahren zurückverfolgen. Es war jedoch erst in den 1980ern, dass der Algorithmus substanzielle Anerkennung erlangte. Das Papier von Rumelhart *et al.* war entscheidend, weil es eine bedeutende Herausforderung der Zeit adressierte: wie die Gewichte von Neuronen in verborgenen Schichten eines neuronalen Netzes anzupassen sind. Zuvor wurden Ein-Schicht-neuronale Netze effektiv eingesetzt, aber das Hinzufügen verborgener Schichten, das die Modellierung komplexerer Funktionen erlaubte, stellte hinsichtlich der Gewichtsanpassung durch Lernen eine Herausforderung dar. Der Backpropagation-Algorithmus verwendet die Kettenregel der Analysis, um iterativ Gradienten für jede Schicht im Netz zu berechnen, beginnend bei der Ausgabeschicht und rückwärts arbeitend. Diese Methode erlaubte das effektive Training tiefer neuronaler Netze, indem die Gewichte so angepasst wurden, dass die Verlustfunktion des Netzes minimiert wird. Die Einführung von Backpropagation löste ein erneutes Interesse an neuronalen Netzen aus und führte zu dem, was oft als zweite Welle der Forschung zu neuronalen Netzen bezeichnet wird. Die Fähigkeit des Algorithmus, tiefe Netze zu trainieren, legte den Grundstein für die Entwicklung des Deep Learning, das seither viele Aspekte des maschinellen Lernens und der Künstlichen Intelligenz revolutioniert hat. Trotz seines Erfolgs ist Backpropagation nicht ohne Einschränkungen, wie etwa das Problem des verschwindenden Gradienten, bei dem Gradienten beim Rückwärtspropagieren zu frühen Schichten zunehmend klein werden und das Lernen des Netzes behindern. Nichtsdestoweniger stellt seine Entwicklung einen Wendepunkt in der Geschichte des maschinellen Lernens dar und liefert ein Schlüsselwerkzeug für das Training komplexer neuronaler Netze, wodurch Fortschritte in einem breiten Spektrum von Anwendungen ermöglicht wurden, von Bild- und Spracherkennung bis hin zur Verarbeitung natürlicher Sprache.

Diese Periode sah auch das Aufkommen der ersten praktischen Anwendungen neuronaler Netze, wie das NetTalk-System von Sejnowski und Rosenberg im Jahr 1987 [6], das das Potenzial neuronaler Netze in der Sprachsynthese

demonstrierte. NetTalk wurde entworfen, um geschriebenen englischen Text in phonetische Aussprachen zu konvertieren, im Wesentlichen einem Computer beizubringen, wie man spricht. Im Kern verwendete NetTalk ein sieben-schichtiges neuronales Netz, ein bedeutendes Design für seine Zeit, angesichts der rechnerischen Beschränkungen und des relativ neuen Stands der Forschung zu neuronalen Netzen. Dieses neuronale Netz wurde mit dem Backpropagation-Algorithmus trainiert, der es ihm ermöglicht, die passende phonetische Aussprache englischer Texte zu erlernen, indem es Beispielen von Text und den entsprechenden gesprochenen Wörtern ausgesetzt wird. Einer der bemerkenswerten Aspekte von NetTalk war seine Fähigkeit, aus den Trainingsdaten zu lernen und zu generalisieren. Während es mehr Beispielen ausgesetzt wurde, verbesserte das neuronale Netz seine Aussprachefähigkeiten, ähnlich wie ein menschliches Kind das Sprechen lernt, indem es Erwachsenen zuhört. Dieser Lernprozess bestand nicht nur im Memorieren spezifischer Aussprachen, sondern vielmehr im Verständnis der zugrunde liegenden Muster und Regeln der englischen Phonetik. Das NetTalk-System war maßgeblich bei der Demonstration praktischer Anwendungen neuronaler Netze, insbesondere im Bereich der Sprachsynthese. Es zeigte, dass neuronale Netze Aufgaben ohne explizite Programmierung von Regeln erlernen konnten, ein signifikanter Bruch mit den konventionellen Ansätzen, die zu dieser Zeit in der Informatik verwendet wurden. Darüber hinaus trug NetTalk zum wachsenden Interesse an neuronalen Netzen und ihren potenziellen Anwendungen bei. Es war ein frühes Beispiel, das den Weg für fortgeschrittenere Systeme der Spracherkennung und -synthese ebnete, die heute in Geräten und Anwendungen wie Smartphones, virtuellen Assistenten und Übersetzungsdiensten verbreitet sind. Die Entwicklung von NetTalk durch Sejnowski und Rosenberg ist ein bemerkenswertes Beispiel dafür, wie innovative Anwendungen neuronaler Netze zu Durchbrüchen in der Künstlichen Intelligenz führen können, indem sie die Leistungsfähigkeit dieser Systeme demonstrieren, in einer dem menschlichen Lernen ähnlichen Weise zu lernen und sich anzupassen.

Die 1990er Jahre waren ebenfalls eine entscheidende Ära im Feld des maschinellen Lernens, gekennzeichnet durch einen Übergang von theoretischen Grundlagen hin zu praktischeren Anwendungen. Diese Periode erlebte eine signifikante Evolution von Algorithmen, angetrieben durch zunehmende Rechenleistung und Datenverfügbarkeit. Einer der großen Meilensteine war der Fortschritt bei neuronalen Netzen. Die Arbeit von LeCun *et al.* [7] zur Handschrifterkennung mit konvolutionalen neuronalen Netzen legte die Grundlage für moderne Deep-Learning-Techniken. LeCun stellte einen Datensatz zusammen, der heute weithin als MNIST (Modified National Institute of Standards and Technology) bekannt ist. Dieser hat eine zentrale Rolle im Feld des maschinellen Lernens gespielt, insbesondere bei der Entwicklung und dem Benchmarking von Algorithmen zur Erkennung handgeschriebener Ziffern. Der Datensatz wurde als zugänglichere und vorverarbeitete Version des früheren NIST-Datensatzes erstellt. Er besteht aus 70 000 gelabelten Bildern handgeschriebener Ziffern (0 bis 9), aufgeteilt in einen Trainingssatz von 60 000 Beispielen und einen Testsatz von 10 000 Beispielen. Jedes Bild ist eine 28×28 Pixel große Graustufendarstellung einer Ziffer, was ihn ideal zum Testen von Bildverarbeitungssystemen

macht. Die Einfachheit und Größe des MNIST-Datensatzes haben ihn zu einem Standard-Benchmark für Algorithmen der Bilderkennung gemacht und dienen als Testfeld für eine breite Palette von Ansätzen vom klassischen maschinellen Lernen bis zum Deep Learning. Insbesondere wurden die Entwicklung und der Erfolg konvolutionaler neuronaler Netze durch Experimente auf dem MNIST-Datensatz signifikant gestützt, wie von LeCun *et al.* demonstriert [8]. Trotz seiner Einfachheit bleibt MNIST eine wertvolle Ressource für Bildungszwecke und erste Algorithmustests.

Diese Periode sah auch den Aufstieg der Support Vector Machines (SVMs), in ihrer modernen Form eingeführt von Cores und Vapnik im Jahr 1995 [9], die eine robuste Methode für Klassifikations- und Regressionsaufgaben bereitstellten. SVMs sind eine Reihe überwachter Lernverfahren, die für Klassifikation, Regression und Ausreißererkennung eingesetzt werden. Die Ursprünge der SVMs lassen sich auf die Arbeit von Vapnik *et al.* aus dem Jahr 1964 [10] zurückverfolgen, in der das Konzept eines linearen Klassifikators mit maximalem Margin erstmals eingeführt wurde. Dieses frühe Modell legte die Grundlage für das, was schließlich moderne SVMs werden sollte. Die formale Entwicklung der SVMs begann in den 1990ern. Das Schlüsselpapier von Boser *et al.* [11] führte eine Möglichkeit ein, nichtlineare Klassifikatoren zu erzeugen, indem der Kernel-Trick auf Maximum-Margin-Hyperebenen angewendet wurde. Dies wurde von Cores und Vapnik [9] weiter verfeinert und popularisiert, die ein umfassendes Rahmenwerk für das Training von SVMs bereitstellten. Seitdem haben SVMs verschiedene Erweiterungen erfahren und sind in zahlreichen Feldern angewandt worden, von Bilderkennung bis Bioinformatik. Ihre Fähigkeit, große Merkmalsräume zu handhaben, und ihre Flexibilität bei der Modellierung unterschiedlicher Datenquellen machen sie zu einem leistungsfähigen Werkzeug im maschinellen Lernen. SVMs stellen eine bedeutende Entwicklung im Feld des statistischen Lernens dar und bieten einen robusten Ansatz sowohl für Klassifikations- als auch Regressionsprobleme. Ihre Entwicklung von einem theoretischen Konzept zu einem festen Bestandteil der Werkzeugkästen des maschinellen Lernens ist ein Zeugnis für ihre Vielseitigkeit und Effektivität.

Das 21. Jahrhundert war besonders gekennzeichnet durch den Aufstieg des Deep Learning, einer Untergruppe des maschinellen Lernens, die auf neuronalen Netzen mit einer großen Anzahl von Schichten und Parametern basiert. Wegweisende Arbeiten von Forschenden wie Yann LeCun, Geoffrey Hinton und Yoshua Bengio [12] haben zu Durchbrüchen in Bereichen wie Bild- und Spracherkennung sowie Verarbeitung natürlicher Sprache geführt. Doch die frühen 2000er markierten auch eine Periode signifikanter Fortschritte und Meilensteine in vielen Bereichen des maschinellen Lernens und bereiteten die Bühne für die rasche Entwicklung, die in den folgenden Jahren stattfinden sollte. Eine der wirkungsvollsten Veränderungen zu Beginn der 2000er war der dramatische Anstieg der Rechenleistung, vor allem aufgrund des Fortschritts bei GPUs (Graphic Processing Units). Dies, zusammen mit der wachsenden Verfügbarkeit großer Datensätze, erlaubte es Forschenden, komplexere Modelle zu trainieren, insbesondere tiefe neuronale Netze. Die frühen 2000er erlebten ebenfalls bedeutende Fortschritte in der Verarbeitung natürlicher Sprache (NLP). Die Einführung statistischer

Methoden gegenüber traditionellen regelbasierten Ansätzen markierte einen Paradigmenwechsel in der NLP. Wegweisende Modelle wie die Latent-Dirichlet-Allocation (LDA) von Blei *et al.* [13] begannen zu entstehen und veränderten die Art und Weise, wie Maschinen menschliche Sprache verarbeiteten und verstanden. Diese Periode sah auch den Aufstieg von Ensemblemethoden im maschinellen Lernen. Techniken wie Random Forests und Boosting-Verfahren wurden aufgrund ihrer Robustheit und Effektivität in verschiedenen Anwendungen populär. Boosting, eine leistungsstarke Ensembletechnik, beinhaltet die Kombination mehrerer schwacher Lerner zu einem starken Klassifikator. Eine entscheidende Entwicklung im Boosting war der AdaBoost-Algorithmus, eingeführt von Freund und Schapire [14]. AdaBoost, kurz für Adaptive Boosting, arbeitet, indem schwache Lerner, typischerweise Entscheidbäume, sequentiell hinzugefügt werden und dabei die Instanzen fokussiert werden, die in früheren Runden falsch vorhergesagt wurden. Der Algorithmus weist diesen herausfordernden Instanzen höhere Gewichte zu, wodurch sichergestellt wird, dass nachfolgende Lerner sich stärker auf sie konzentrieren. AdaBoost zeigte eine bemerkenswerte Effektivität bei der Verbesserung der Genauigkeit von Klassifikationsmodellen. Ein weiterer kritischer Fortschritt war die Formulierung von Gradient Boosting Machines (GBMs) durch Friedman [15]. GBMs erweitern den Boosting-Rahmen, indem sie beliebige Verlustfunktionen optimieren. Dieses Verfahren beinhaltet den stufenweisen Aufbau eines Modells und die Verallgemeinerung, indem die Optimierung einer beliebigen differenzierbaren Verlustfunktion erlaubt wird. GBMs waren in einem breiten Spektrum praktischer Anwendungen äußerst erfolgreich, von Standard-Regressionstasks bis hin zu komplexen Lernproblemen.

Parallel zu diesen Entwicklungen hat sich das Reinforcement Learning weiterentwickelt, mit bedeutenden Beiträgen von Sutton und Barto [16]. Dieses Gebiet des maschinellen Lernens, das sich darauf konzentriert, wie Agenten in einer Umgebung handeln sollten, um eine Art Belohnung zu maximieren, hat zu beeindruckenden Demonstrationen wie den Siegen von AlphaGo im Spiel Go [17] geführt. Ein grundlegendes Konzept im Reinforcement Learning ist das Temporal-Difference (TD)-Lernen, das zuvor von Sutton eingeführt wurde [18]. In den frühen 2000ern demonstrierten die Verfeinerung und Anwendung von TD-Lernverfahren, insbesondere TD-Gammon von Tesauro [19], das Potenzial von RL in komplexen Domänen. TD-Gammon, ein Computer-Backgammon-Programm, verwendete ein neuronales Netz, das durch TD-Lernen trainiert wurde, und war in der Lage, ein Leistungsniveau zu erreichen, das dem menschlicher Expertinnen und Experten vergleichbar war. Der Beginn der 2000er Jahre erlebte auch bedeutende Fortschritte bei Policy-Gradient-Methoden, einem Ansatz im RL, bei dem die Policy, die zentrale Entscheidungsfunktion eines Agenten, direkt optimiert wird. Die Arbeit von Sutton *et al.* [20] lieferte ein umfassendes Rahmenwerk für diese Methoden, die für die Lösung von Aufgaben entscheidend wurden, die komplexe Aktionssequenzen erfordern. Das zwanzigste Jahrhundert sah ebenfalls wichtige theoretische Fortschritte im RL. Die Entwicklung von Algorithmen mit garantierter Konvergenz und die Erforschung des Gleichgewichts zwischen Exploration und Ausbeutung waren zentrale Fokusbereiche. Die Formalisierung des Exploration-Exploitation-Trade-offs, wie von Auer *et al.* [21] dis-

kutiert, lieferte ein tieferes Verständnis des Entscheidungsprozesses im RL. Die Entwicklung von Algorithmen des maschinellen Lernens, die in der Lage sind, das antike Brettspiel Go zu spielen, markiert einen bedeutenden Meilenstein in der Geschichte der Künstlichen Intelligenz (KI) und des maschinellen Lernens. Go, bekannt für seine tiefe strategische Komplexität, war lange eine gewaltige Herausforderung für KI-Forschende. Traditionelle KI-Ansätze, die im Schach erfolgreich gewesen waren, wie Brute-Force-Suche, waren für Go aufgrund der enormen Anzahl möglicher Stellungen unwirksam. Der Durchbruch kam 2016 mit AlphaGo, entwickelt von Google DeepMind. Im maßgeblichen Papier von Silver *et al.* [17] wurde der AlphaGo-Algorithmus vorgestellt. Er kombinierte tiefe neuronale Netze mit Monte-Carlo-Baumsuche (MCTS), um Brettstellungen zu bewerten und Züge auszuwählen. Dieser Ansatz ermöglichte es AlphaGo, 2015 einen professionellen menschlichen Go-Spieler zu besiegen, eine Leistung, die zuvor für mindestens ein Jahrzehnt in der Zukunft gehalten wurde. Darauf folgend stellte AlphaGo Zero, wie von Silver *et al.* [22] detailliert, einen weiteren Fortschritt dar. Es lernte, Go ausschließlich durch Spiele gegen sich selbst zu spielen, ohne jegliche menschliche Daten, und erreichte Übermenschenleistung. Dies war ein bedeutender Schritt im unüberwachten Lernen innerhalb des maschinellen Lernens. Der Erfolg von AlphaGo und seinen Nachfolgern hat einen tiefgreifenden Einfluss auf das Feld des maschinellen Lernens gehabt. Er demonstrierte nicht nur das Potenzial von Deep Learning und Reinforcement Learning bei der Lösung komplexer Probleme, sondern inspirierte auch zahlreiche Anwendungen ähnlicher Techniken in verschiedenen Domänen. Die Entwicklung von AlphaGo und seinen Nachfolgern markiert einen Wendepunkt in der Geschichte des maschinellen Lernens und zeigt die Leistungsfähigkeit der Integration tiefer neuronaler Netze mit fortgeschrittenen Suchstrategien. Dieser Meilenstein der KI-Forschung hat den Horizont für die Anwendung des maschinellen Lernens in komplexen Entscheidungsfindungsszenarien erweitert.

Die Dekade 2010 bis 2020 war gekennzeichnet von bemerkenswerten Durchbrüchen im Feld des maschinellen Lernens. Einer der bedeutendsten Meilensteine war die Dominanz des Deep Learning. Der entscheidende Moment trat 2012 ein, als [23] die Leistungsfähigkeit tiefer konvolutionaler neuronaler Netze (CNNs) demonstrierten, indem sie den ImageNet-Wettbewerb gewannen. Dieser Durchbruch verbesserte die Leistung bei Bildklassifikationsaufgaben drastisch und löste eine Welle von Forschung und Anwendungen im Deep Learning über verschiedene Domänen hinweg aus. In der Verarbeitung natürlicher Sprache (NLP) markierte die Entwicklung von Transformer-Modellen, insbesondere die Einführung von BERT (Bidirectional Encoder Representations from Transformers) durch [24], einen bedeutenden Fortschritt. Dieses Modell und seine Architektur revolutionierten den Ansatz für Aufgaben wie Textklassifikation, Sentimentanalyse und Frage-Antwort und setzten neue Leistungsstandards in der NLP. Eine weitere bemerkenswerte Entwicklung gab es bei generativen Modellen, insbesondere mit der Einführung der Generative Pretrained Transformer (GPT)-Modelle durch OpenAI. Wie von [25] beschrieben, zeigten diese Modelle eine beispiellose Fähigkeit, kohärenten und kontextuell relevanten Text zu erzeugen.

Das frühe 21. Jahrhundert sah eine Verschiebung hin zu komplexeren Modellen mit der Einführung neuronaler Netze bei Sprachaufgaben. Ein Meilenstein war die Entwicklung von Sequence-to-Sequence-Lernmodellen [26], die die Leistung von maschinellen Übersetzungssystemen verbesserten. Ein entscheidender Moment in der Geschichte großer Sprachmodelle (LLMs) war die Einführung des Transformer-Modells durch Vaswani et al. im Jahr 2017 [27], die zu einem Paradigmenwechsel in der Gestaltung von Sprachmodellen führte. Dieses Modell, basierend auf Selbstaufmerksamkeitsmechanismen, verbesserte die Effizienz und Effektivität des Sprachverständnisses und der Sprachgenerierung signifikant. Transformer haben das Feld der Verarbeitung natürlicher Sprache (NLP) und darüber hinaus revolutioniert, indem sie ein neues Paradigma für die Handhabung sequentieller Daten bieten. Anders als frühere Modelle, die Sequenzen Schritt für Schritt verarbeiteten, verarbeiten Transformer ganze Sequenzen gleichzeitig, was zu erheblichen Gewinnen in Effizienz und Effektivität führt. Die Kerninnovation der Transformer ist der Aufmerksamkeitsmechanismus. Dieser Mechanismus erlaubt es dem Modell, sich bei der Vorhersage jedes Teils der Ausgabesequenz auf verschiedene Teile der Eingabesequenz zu konzentrieren. Einfach ausgedrückt ermöglicht der Aufmerksamkeitsmechanismus dem Modell, dynamisch den relevantesten Teilen der Eingabe Aufmerksamkeit zu schenken, um Vorhersagen zu treffen, ähnlich wie Menschen sich auf bestimmte Wörter oder Phrasen konzentrieren, wenn sie einen Satz verstehen. Ein Transformer-Modell besteht aus einem Encoder und einem Decoder. Der Encoder liest die Eingabesequenz und erzeugt eine hochdimensionale Repräsentation, die der Decoder dann verwendet, um die Ausgabesequenz zu generieren. Sowohl der Encoder als auch der Decoder bestehen aus Schichten, die Aufmerksamkeitsmechanismen und Feedforward-Neuronale Netze enthalten. Die Aufgabe des Encoders ist es, die Eingabedaten (wie einen Satz in NLP-Aufgaben) zu verarbeiten und eine kontextreiche Repräsentation zu erzeugen. Jede Schicht im Encoder besteht aus zwei Hauptteilen: einem Selbstaufmerksamkeitsmechanismus und einem Feedforward-Neuronalen Netz. Der Selbstaufmerksamkeitsmechanismus erlaubt es jeder Position im Encoder, auf alle Positionen in der vorherigen Schicht des Encoders zu achten. Der Decoder, in seiner Struktur dem Encoder ähnlich, generiert die Ausgabesequenz. Er enthält ebenfalls einen Selbstaufmerksamkeitsmechanismus, umfasst jedoch eine zusätzliche Aufmerksamkeitslage, die sich auf die Ausgabe des Encoders fokussiert. Dieses Design ermöglicht es dem Decoder, die gesamte Eingabesequenz zu berücksichtigen, wenn er jedes Element der Ausgabe produziert. Transformer bieten parallele Verarbeitung von Sequenzen, was die Trainings- und Inferenzzeiten erheblich beschleunigt. Sie haben in verschiedenen NLP-Aufgaben, einschließlich Übersetzung, Textzusammenfassung und Frage-Antwort, Ergebnisse auf dem Stand der Technik erzielt. Ihre Architektur hat auch Anpassungen in anderen Domänen inspiriert, wie der Computer Vision. Aufbauend auf der Transformer-Architektur führte OpenAI GPT (Generative Pretrained Transformer) und seine nachfolgenden Iterationen ein, die bemerkenswerte Fähigkeiten in der Sprachgenerierung zeigten [25]. Ebenso brachte Googles BERT (Bidirectional Encoder Representations from Transformers) [24] Fortschritte im Sprachverständnis, insbesondere bei Auf-

gaben wie Frage-Antwort und Sentimentanalyse. Die aktuelle Landschaft der LLMs ist gekennzeichnet durch ihre zunehmende Größe und Raffinesse, wobei Modelle wie GPT-3 und GPT-4 die Grenzen dessen verschieben, was in der Verarbeitung und Generierung natürlicher Sprache möglich ist [28].

Eines der auffälligsten Merkmale moderner Modelle des maschinellen Lernens, insbesondere im Bereich der Verarbeitung natürlicher Sprache, ist ihre beispiellose Größe. Große Sprachmodelle (LLMs) sind auf kolossale Skalen mit einer erstaunlichen Anzahl von Parametern angewachsen, die konventionelle Erwartungen sprengen. Diese Parameter repräsentieren das erlernte Wissen und die Muster innerhalb des Modells. Der Umfang von LLMs wird oft in Milliarden oder sogar Billionen von Parametern gemessen. Modelle wie GPT-3 von OpenAI verfügen beispielsweise über 175 Milliarden Parameter und übertreffen ihre Vorgänger um Größenordnungen. Das Wachstum von LLMs hat in den letzten Jahren einer exponentiellen Trajektorie folgt. Von Modellen mit Millionen von Parametern zu solchen mit Milliarden und darüber hinaus hat diese rasche Expansion die Grenzen dessen verschoben, was zuvor für möglich gehalten wurde. Jede neue Iteration von LLMs verschiebt die Grenze weiter und bricht Rekorde sowohl in Bezug auf Leistung als auch auf Parameteranzahl. Das Training und die Feinabstimmung dieser gigantischen Modelle erfordern eine immense Menge an Rechenressourcen. Das Training eines Modells mit Billionen von Parametern erfordert nicht nur leistungsfähige GPUs oder TPUs (Tensor Processing Units), sondern auch umfangreiche Speicherkapazität und Hochgeschwindigkeitsnetzwerkverbindungen. Diese rechnerische Anforderung hat zur Entwicklung spezialisierter Hardware und verteilter Trainings-Setups geführt. Trotz ihrer enormen Größe haben LLMs Anwendungen in einem breiten Spektrum von Domänen gefunden. Sie glänzen in Aufgaben des Sprachverständnisses und der Sprachgenerierung, einschließlich Übersetzung, Zusammenfassung, Frage-Antwort und Inhaltserzeugung. Ihre immense Wissensbasis ermöglicht es ihnen, kohärenten und kontextuell relevanten Text zu einer Vielzahl von Themen zu generieren.

Ich würde wichtiger Teile der Geschichte des maschinellen Lernens schuldig werden, wenn ich nicht zumindest etwas über Generative KI erwähnen würde. Wahrscheinlich der wichtigste Beitrag zur Generativen KI war die Einführung Generativer Gegenspielnetzwerke (GANs) durch Goodfellow *et al.* [29], die einen bedeutenden Sprung in diesem Feld markierte [29]. GANs revolutionierten die Art und Weise, wie Maschinen realistische Bilder erzeugen konnten, und führten zu einem Anstieg der Forschung und Anwendungen generativer Modelle. Die Entwicklung Variationaler Autoencoder (VAE) durch Kingma *et al.* [30], veröffentlicht 2013, erforschte eine statistische Methode und bot einen neuen Ansatz für generatives Modellieren [30]. Diese Meilensteine repräsentieren gemeinsam die schnelle und fortlaufende Entwicklung der Generativen KI und zeichnen den Weg von einfacher Mustererkennung zur Erzeugung reicher, komplexer und vielfältiger Ausgaben nach, die menschliche Kreativität nachahmen können.

Generative KI ist zu einem wichtigen Werkzeug in der synthetischen Datenerzeugung innerhalb der wissenschaftlichen Forschung geworden. Dieser Ansatz ermöglicht es Wissenschaftlerinnen und Wissenschaftlern, große, realistische Datensätze zu erstellen, die zum Training von Modellen des maschinellen

Lernens verwendet werden können, insbesondere in Bereichen, in denen reale Daten knapp oder schwer zu erhalten sind. Durch die Erzeugung hochwertiger synthetischer Daten, die reale Bedingungen eng widerspiegeln, kann Generative KI robustere und umfangreichere Experimente erleichtern und die Forschung in Bereichen wie Gesundheitswesen, Umweltwissenschaften und Materialtechnik erheblich voranbringen.

1.2 Maschinelles Lernen in der Wissenschaft

Es ist notwendig, einige der wichtigsten Errungenschaften des maschinellen Lernens in verschiedenen wissenschaftlichen Disziplinen kurz hervorzuheben.

In der Chemie hat ML mehrere Bereiche revolutioniert, insbesondere Wirkstoffentdeckung und Materialwissenschaft. Der traditionelle Prozess der Wirkstoffentdeckung ist oft langwierig und teuer. ML-Algorithmen haben diesen Prozess durch die Vorhersage von Eigenschaften und Aktivitäten von Molekülen erheblich beschleunigt. ML-Modelle können beispielsweise große chemische Räume schnell nach potenziellen Wirkstoffkandidaten durchsuchen, ein Prozess, der in der Arbeit von Gomez *et al.* [31] hervorgehoben wird. Darüber hinaus war ML in der Materialwissenschaft maßgeblich, um die Eigenschaften neuer Materialien vorherzusagen und so experimentelle Bemühungen gezielter und effizienter zu leiten. Die Physik, insbesondere Hochenergiephysik und Astrophysik, hat immens von ML profitiert. In der Hochenergiephysik waren ML-Verfahren entscheidend bei der Analyse von Daten aus Teilchenbeschleunigern. Der Nachweis des Higgs-Bosons, wie in [32] erläutert, ist ein herausragendes Beispiel, in dem Deep Learning die Trennung von Signalen vom Hintergrundrauschen signifikant verbesserte. In der Astrophysik hilft ML bei der Interpretation großer Datenmengen aus Teleskopen und Weltraummissionen und unterstützt die Entdeckung neuer Himmelsobjekte und Phänomene. In der Biologie liegt der vielleicht bemerkenswerteste Beitrag von ML im Bereich der Genomik und Proteomik. Die Vorhersage von Proteinstrukturen, eine langjährige Herausforderung in der Biologie, wurde durch das von Google DeepMind entwickelte AlphaFold-System [33] revolutioniert. Die Fähigkeit dieses Systems, Proteinstrukturen genau und schnell vorherzusagen, hat tiefgreifende Implikationen für das Verständnis biologischer Prozesse und die Entwicklung neuer Therapeutika. Darüber hinaus wird ML in der Genomik zunehmend eingesetzt, um genetische Variationen und ihre Verbindungen zu Krankheiten zu verstehen. Die Integration des maschinellen Lernens in die wissenschaftliche Forschung hat nicht nur Entdeckungen beschleunigt, sondern auch neue Wege für Erkundung und Innovation eröffnet. Da sich die ML-Technologie weiterentwickelt, wird erwartet, dass ihre Rolle beim Fortschritt wissenschaftlicher Erkenntnis weiter wächst.

1.3 Arten des Maschinellen Lernens

Nun müssen wir uns auf einige Terminologie konzentrieren, die in der Community des maschinellen Lernens weit verbreitet ist. Das Feld des maschinellen Lernens ist groß, und es gibt viele *Arten* von Algorithmen. Es ist wichtig, etwas Licht auf die verschiedenen Arten zu werfen, um zu klären, was Sie hier finden werden. Unten wird eine sehr breite und grobe Klassifikation von ML-Ansätzen gegeben.

- **Überwachtes Lernen:** Diese Methode, grundlegend für maschinelles Lernen, beinhaltet das Trainieren eines Algorithmus auf einem gelabelten Datensatz, bei dem jedes Beispiel mit einer erwarteten Ausgabe gepaart ist. Das Modell lernt, auf der Grundlage dieser Daten Ergebnisse vorherzusagen, indem es mithilfe von Beispielen Muster ableitet.
- **Unüberwachtes Lernen:** Anders als beim überwachten Lernen beinhaltet unüberwachtes Lernen das Trainieren von Algorithmen auf Daten ohne vordefinierte Labels. Es fokussiert auf die Identifikation verborgener Strukturen und Muster. Historisch bedeutsam für seine Rolle bei Clustering- und Assoziationsproblemen ist unüberwachtes Lernen ein Schlüssel in der explorativen Datenanalyse und bei Techniken der Dimensionsreduktion wie der Hauptkomponentenanalyse (PCA).
- **Semisupervised Learning:** Dieser Ansatz kombiniert Elemente des überwachten und des unüberwachten Lernens. Er ist besonders nützlich, wenn gelabelte Daten begrenzt oder kostspielig zu beschaffen sind – eine häufige Herausforderung im maschinellen Lernen. Durch die Nutzung einer Mischung aus einer kleinen Menge gelabelter Daten und einem größeren Pool ungelabelter Daten können Modelle im semisupervisierten Lernen eine höhere Genauigkeit erreichen als rein unüberwachte Verfahren.
- **Bestärkendes Lernen (Reinforcement Learning):** Diese Art des Lernens zeichnet sich durch ihren Fokus auf das Treffen von Entscheidungssequenzen aus. Der Algorithmus, oft Agent genannt, lernt, in einer unsicheren und potenziell komplexen Umgebung ein Ziel zu erreichen. Historisch hat Reinforcement Learning seine Wurzeln in der Spieltheorie und der optimalen Regelungstheorie.

1.4 KI, Maschinelles Lernen und Deep Learning

Wenn Menschen über Künstliche Intelligenz (KI), Maschinelles Lernen (ML) und Deep Learning (DL) sprechen, verwenden sie die Begriffe oft austauschbar. Sie beschreiben jedoch unterschiedliche Konzepte mit variierendem Umfang und Abstraktionsebenen. Das Verständnis dieser Unterschiede ist wesentlich, um ein Projekt korrekt zu positionieren, Erwartungen zu managen und sowohl mit technischen als auch nichttechnischen Stakeholdern zu kommunizieren.

Künstliche Intelligenz (KI)

Scope	Umfasst Schlussfolgern, Planung, Entscheidungsfindung, Wahrnehmung, Sprachverständnis, Robotik und Lernen.
Techniques	Umfasst symbolische KI (regelbasierte Systeme, Expertensysteme), Suchalgorithmen, Optimierung und modernere Ansätze wie maschinelles Lernen und Reinforcement Learning.
Examples	Schachprogramme wie Deep Blue, Systeme für autonomes Fahren und Konversationsagenten.
Key Question	"Können wir Maschinen bauen, die sich intelligent verhalten?"

Tabelle 1.1: Überblick über Künstliche Intelligenz: Umfang, Techniken, Beispiele und Leitfrage.

Künstliche Intelligenz ist der weiteste Begriff. Er bezieht sich auf das Gebiet der Informatik, das darauf abzielt, Maschinen oder Systeme zu schaffen, die Aufgaben ausführen können, die normalerweise menschliche Intelligenz erfordern. In Tabelle 1.1 sind die Hauptmerkmale zu sehen.

Maschinelles Lernen (ML)

Scope	Befasst sich mit Algorithmen, die aus Daten generalisieren können, einschließlich überwachtem, unüberwachtem und bestärkendem Lernen.
Techniques	Lineare Regression, Entscheidbäume, Support Vector Machines, Clustering-Verfahren, Ensemblemethoden usw.
Examples	Spam- E-Mail-Klassifikation, Erkennung von Kreditkartenbetrug, personalisierte Empfehlungssysteme.
Key Question	"Kann das System aus Daten lernen, seine Leistung bei einer Aufgabe zu verbessern?"

Tabelle 1.2: Überblick über Maschinelles Lernen: Umfang, Techniken, Beispiele und Leitfrage.

Maschinelles Lernen ist eine Teilmenge der KI, die sich speziell auf Algorithmen konzentriert, die sich automatisch durch Erfahrung verbessern, üblicherweise durch Lernen aus Daten. Anstatt Regeln explizit zu programmieren, entdecken ML-Systeme Muster und treffen Vorhersagen. In Tabelle 1.2 sind die Hauptmerkmale zu sehen.

Deep Learning (DL)

Scope	Fokussiert auf künstliche neuronale Netze mit vielen Schichten (tiefe Architekturen), die hierarchische Repräsentationen lernen können.
Techniques	Konvolutionale neuronale Netze (CNNs) für Bilder, rekurrente und Transformer-Architekturen für Sequenzen, Generative Adversarial Networks (GANs) sowie Große Sprachmodelle (LLMs).
Examples	Bildklassifikation in der Computer Vision, Spracherkennung, maschinelle Übersetzung, ChatGPT und andere großskalige generative KI-Systeme.
Key Question	"Können wir große neuronale Netze auf massiven Datenmengen trainieren, um zuvor unlösbare Aufgaben zu bewältigen?"

Tabelle 1.3: Überblick über Deep Learning: Umfang, Techniken, Beispiele und Leitfrage.

Deep Learning ist ein spezialisierter Zweig des maschinellen Lernens, der mehrschichtige neuronale Netze (sogenannte „tiefe" Architekturen) verwendet, um Merkmale und Repräsentationen automatisch aus Daten zu extrahieren. In Tabelle 1.3 sind die Hauptmerkmale zu sehen.

Generative KI (GenAI)

Generative Künstliche Intelligenz (GenAI) ist ein Zweig der Künstlichen Intelligenz, der Daten nicht nur analysiert, sondern auch *neue Inhalte erzeugt*, wie Text, Bilder, Audio oder sogar Computercode. In Tabelle 1.4 sind die Hauptmerkmale zu sehen.

Beziehung zwischen den drei Konzepten

Künstliche Intelligenz (KI) ist der breite Oberbegriff, der alle Techniken und Ansätze umfasst, die darauf ausgelegt sind, es Computern zu ermöglichen, Aspekte menschlicher Intelligenz zu simulieren. Dies umfasst Schlussfolgern, Problemlösen, Wahrnehmung und Entscheidungsfindung. KI ist nicht an eine einzelne Methode gebunden, sondern bezieht sich auf das übergeordnete Ziel, Systeme zu schaffen, die Aufgaben ausführen können, die traditionell mit menschlicher Kognition verbunden sind. Beachten Sie, dass mitunter sogar regelbasierte Systeme als „KI" bezeichnet werden. Es ist also ein eher generischer Begriff, und seine Bedeutung variiert stark von Person zu Person.

Innerhalb dieses breiten Feldes stellt **Maschinelles Lernen (ML)** eine wichtige Teilmenge dar. Intuitiv ist ML eine Menge von Algorithmen, die aus

Scope	Systeme, die die zugrunde liegenden Muster und Strukturen von Daten erlernen, um neue, realistische Ausgaben zu generieren.
Techniques	Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), Diffusionsmodelle für Bildsynthese sowie Große Sprachmodelle (LLMs) für Textgenerierung.
Examples	Textgeneratoren wie ChatGPT, Bildgeneratoren wie DALL-E und Stable Diffusion, Systeme zur Musik- und Sprachsynthese sowie Anwendungen im Moleküldesign und in der Wirkstoffentdeckung.
Key Question	"Können Maschinen kreativen Inhalt in so hoher Qualität erzeugen, dass er von menschlich Erstelltem ergänzt oder sogar ersetzt?"

Tabelle 1.4: Überblick über Generative KI: Umfang, Techniken, Beispiele und Leitfrage.

Daten lernen, anstatt sich ausschließlich auf explizit programmierte Regeln zu stützen. Anstatt jedes mögliche Szenario manuell zu kodieren, leiten Systeme des maschinellen Lernens Muster ab und treffen Vorhersagen oder Entscheidungen basierend auf Erfahrung, die aus Beispielen gelernt wurde (denken Sie an das überwachtes-Lernen-Setting, das wir in Abschnitt 1.3 diskutiert haben). Klassische Beispiele umfassen Spamfilterung, Betrugserkennung oder Empfehlungssysteme, bei denen große Mengen vergangener Daten verwendet werden, um auf neue, ungesehene Fälle zu generalisieren.

Eine weitere Teilmenge ist **Deep Learning (DL)**, das sich auf Verfahren stützt, die auf tiefen neuronalen Netzen mit vielen Schichten basieren (man kann sich ein neuronales Netz mit vielen Schichten als Algorithmen vorstellen, die extrem flexibel sind und daher viele feine Unterschiede in Daten verstehen können). Diese Modelle zeichnen sich durch das Lernen hierarchischer Repräsentationen von Daten aus und sind besonders effektiv für komplexe Aufgaben wie Bilderkennung, Sprachverständnis und Sprachverarbeitung. Deep Learning ist die Basis für jüngste Fortschritte in der KI und treibt Anwendungen wie selbstfahrende Autos, virtuelle Assistenten und große Sprachmodelle wie chatGPT an. Obwohl DL nicht der einzige Ansatz im ML ist, ist es derzeit die leistungsfähigste und am weitesten verbreitete Technik beim Umgang mit hochdimensionalen, großskaligen Daten.

Eine Teilmenge des Deep Learning ist die **Generative Künstliche Intelligenz** (Generative AI oder GenAI), die sich auf Verfahren bezieht, die nicht nur Daten analysieren, sondern auch neue Inhalte wie Text, Bilder, Musik oder sogar Computercode erzeugen. Diese Modelle lernen die zugrunde liegenden Muster und Strukturen von Trainingsdaten und generieren dann Ausgaben, die oft von menschlich erstellten Inhalten nicht zu unterscheiden sind. Generative KI glänzt

bei Aufgaben, bei denen neues Material erzeugt werden muss, und ist daher besonders effektiv für Anwendungen wie automatische Texterzeugung, realistische Bildsynthese oder die Erstellung neuer molekularer Strukturen in der Wirkstoffentdeckung. Die derzeit prominentesten Beispiele für GenAI umfassen große Sprachmodelle (LLMs) wie ChatGPT, Bildgeneratoren wie DALL-E oder Stable Diffusion sowie multimodale Systeme, die Text, Vision und Audio kombinieren.

Veranschaulichende Analogie

- Denken Sie an KI als die gesamte Disziplin des Baus „intelligenter Maschinen“.
- Maschinelles Lernen ist eine Menge von Algorithmen innerhalb der KI.
- Deep Learning ist eine sehr fortgeschrittene und derzeit dominante Technik innerhalb des Maschinellen Lernens, basierend auf großen und komplexen neuronalen Netzen.
- Generative KI ist ein neuer und schnell wachsender Zweig der KI, der sich darauf fokussiert, neue Inhalte zu erzeugen (Text, Bilder, Audio, Code), statt nur vorhandene Daten zu analysieren.

KI, ML, DL und GenAI unterscheiden sich in Umfang, Methodik und Anwendungen. Obwohl Deep Learning derzeit aufgrund von Durchbrüchen in der Verarbeitung natürlicher Sprache, der Computer Vision und generativen Modellen die öffentliche Wahrnehmung von KI dominiert, ist KI als Ganzes wesentlich breiter. Generative KI hat insbesondere die Grenzen dessen erweitert, was KI leisten kann, indem sie kreative und synthetische Aufgaben ermöglicht, die einst als eindeutig menschlich galten. Um in der Praxis erfolgreich zu sein, müssen Organisationen erkennen, wo ihr Projekt auf diesem Spektrum liegt, und entsprechend die richtigen Methoden wählen.

1.5 Grundlagen- und Angewandte Forschung

Tabelle 1.5 vergleicht **Grundlagenforschung** und **angewandte Forschung** und betont, dass, obwohl beide einer systematischen und rigorosen Untersuchung folgen, ihre Ziele und Ergebnisse sehr unterschiedlich sind. Die Grundlagenforschung ist in erster Linie wahrheitssuchend: Sie zielt darauf ab, präzise wissenschaftliche Fragen zu beantworten, Hypothesen zu testen und verallgemeinerbare Theorien oder Methoden zu entwickeln. Ihr Erfolg wird an Reproduzierbarkeit, statistischer Validität und Beitrag zum größeren Wissenskorpus gemessen. Typische Ergebnisse sind wissenschaftliche Publikationen, Datensätze oder theoretische Einsichten statt unmittelbarer Lösungen für praktische Anspruchsgruppen.

Die angewandte Forschung hingegen ist problemgetrieben. Sie ist darauf ausgelegt, handlungsrelevante Evidenz für Entscheidungen in einem spezifischen, realweltlichen Kontext zu erzeugen. Statt breiter Generalisierung konzentriert

sich die angewandte Forschung auf Prototypen, Pilotierungen oder validierte Interventionen, die den Wert für Stakeholder demonstrieren. Evidenz wird in Bezug auf Kosten–Nutzen, Risiken und Auswirkungen auf konkrete Ergebnisse bewertet, statt auf universelle Gültigkeit. Während die Grundlagenforschung das *Warum* verstehen priorisiert, betont die angewandte Forschung den Nachweis dessen, *was in der Praxis funktioniert.*

Die zentrale Botschaft der Tabelle ist, dass beide Ansätze essenziell sind, aber unterschiedlichen Zwecken dienen. Das Verwechseln der beiden führt oft zu Frustration: allgemeine Theorie von angewandten Projekten zu erwarten oder von Grundlagenstudien sofortigen Geschäftsnutzen zu verlangen. Erfolgreiche Organisationen und Teams erkennen die Unterscheidung und balancieren beide Forschungsarten strategisch: Grundlagenarbeit, um die Wissensgrenzen zu erweitern, und angewandte Arbeit, um diese Einsichten in greifbare Wirkung zu übersetzen.

1.6 Forschung, Technologieentwicklung und Produktentwicklung

KI-Projekte können sehr unterschiedliche Arbeitsarten beinhalten, die klar unterschieden werden müssen, um Verwirrung bei Erwartungen, Planung und Ergebnissen zu vermeiden. Im Allgemeinen lassen sich drei Bereiche grob trennen: **Forschung**, **Technologieentwicklung** und **Produktentwicklung**, wobei die beiden letzteren unter das Dach der angewandten Forschung fallen.

Forschung

Forschung zielt auf neues Wissen und die Beantwortung grundlegender Fragen wie: *„Ist das machbar?“* und *„Warum funktioniert es?“*.

- Typisches Beispiel: eine neue Art neuronaler Netzwerkarchitektur erfinden oder einen neuen Satz in der Mathematik beweisen.
- Üblicherweise in der Wissenschaft oder in industriellen Forschungslaboren durchgeführt.
- Output: Papers, Algorithmen, theoretische Rahmenwerke.

Technologieentwicklung

Technologieentwicklung nimmt Forschungsideen auf und verwandelt sie in nutzbare Prototypen oder praktische Implementierungen. Die Leitfrage lautet: *„Wie bekommen wir das in der Praxis zum Laufen?“*.

- Typisches Beispiel: einen Prototyp des neuen neuronalen Netzes implementieren und ihn effizient, skalierbar und für einen spezifischen Anwendungsfall einsetzbar machen.

Tabelle 1.5: Vergleich von Grundlagenforschung und Angewandter Forschung

Aspekt	**Grundlagenforschung**	**Angewandte Forschung**
Definition	Systematische Untersuchung zur Beantwortung einer präzisen Frage und zur Erzeugung verallgemeinerbaren Wissens.	Systematische Untersuchung, die handlungsrelevante Evidenz erzeugt; zur Ermöglichung spezifischer realweltlicher Entscheidungen.
Zweck	Erklären/Verstehen (das *Warum*), nicht nur Bauen; Wahrheitssuche vor Nützlichkeit.	Eine konkrete reale Entscheidung für einen spezifischen Stakeholder in einem definierten Kontext informieren.
Kennzeichen	Klare Forschungsfrage, testbare Hypothese, definierte Methode, Kontrollen/Baselines, Transparenz, Reproduzierbarkeit, Peer-Review.	Glaubwürdiger, kontextrelevanter kausaler Effekt auf bedeutsame Outcomes, gezeigt durch Prototyp oder Proof-of-Concept.
Evidenz	Statistische Validität (Effektgrößen, Konfidenzintervalle) plus externe Validität/Generalisierung.	Kurzer Pilot mit Learnings, Kosten–Nutzen-Analyse und dokumentierten Risiken.
Typische Outputs	Papers/Reports, Code & Daten, theoretische oder empirische Ergebnisse.	Feldevaluation, Entscheidungsvorlage, validierte Intervention/Prototyp (oft ein Pilot, kein ausgereiftes Produkt).
Nicht	Nachweisen, dass ein Prototyp eine Spezifikation erfüllt (Technologieentwicklung) oder ein Feature shippen, um eine KPI zu bewegen (Produkt).	Breite allgemeine Theorie oder eine fast produktionsreife Lösung; Ergebnisse garantieren keine Übertragbarkeit über den getesteten Kontext hinaus.

- Oft in industriellen Forschungs- und Entwicklungsteams durchgeführt.
- Output: Prototypen, Proof-of-Concept-Systeme, optimierte Algorithmen.

Produktentwicklung

Produktentwicklung konzentriert sich darauf, Endnutzern und Unternehmen Wert zu liefern. Die zentrale Leitfrage ist: *„Löst das ein reales Problem für unsere Nutzer?“*.

- Typisches Beispiel: das neue neuronale Netz als konkretes Feature in eine mobile App oder eine SaaS-Plattform integrieren.
- Orientiert an Geschäftszielen, Nutzererlebnis und Skalierbarkeit.
- Output: ausgerollte Produkte, kundenseitige Services, messbare Geschäftswirkung.

Diese drei Bereiche erfordern unterschiedliche Fähigkeiten, Erfolgskriterien und Lebenszyklen. Das Verwechseln der Bereiche führt zu Frustration und gescheiterten Projekten. Erfolgreiche Einführung von KI erfordert Klarheit: Entscheiden Sie bewusst, ob Ihr Projekt Forschung, Technologieentwicklung oder Produktentwicklung ist.

In Tabelle 1.6 sehen Sie einen Vergleich zwischen Forschung, Technologie- und Produktentwicklung mit den wichtigsten Aspekten. Bemerkenswert und relevant sind die folgenden Punkte.

- **Zeithorizont**: reicht von 3–5 Jahren in der Forschung bis zu 6 Monaten–1 Jahr in der Produktentwicklung. Wichtig ist, dass selbst für die Entwicklung eines Produkts in weniger als 6–12 Monaten nicht zu rechnen ist. Das ist eine optimistische Schätzung.
- **Validierung**: die Art, wie Sie Ihr Projekt validieren (also entscheiden, ob es erfolgreich war), unterscheidet sich stark zwischen den drei Projekttypen. Für Unternehmen sind *Nutzertests* und *Product-Market-Fit* am wichtigsten (wenn auch nicht die einzigen Wege).
- **Benötigte Fähigkeiten**: während in der Forschung eher ein akademisches Skillset (Schwerpunkt Statistik, Informatik, Mathematik) erforderlich ist, ist in der Produktentwicklung ein stärker *engineering*-geprägtes Skillset (Fokus auf Business und IT-Infrastruktur) wesentlich wichtiger, und ein akademisches Skillset ist weitgehend nachrangig (wenn auch in vielen Fällen nützlich).

Exercise

Übung 1.6.1 (Classifying AI Work)
Time: 15 minutes **Group size:** 2–3 students
Task: Lesen Sie die kurzen Projektbeschreibungen unten. Entscheiden Sie für jedes, ob es zu **Forschung**, **Technologieentwicklung** oder **Produktentwicklung** gehört. Notieren Sie Ihre Wahl und eine Begründung in einem Satz.

1) Ein Universitätsteam entwirft einen neuen Optimierungsalgorithmus und beweist, dass er unter bestimmten Bedingungen konvergiert.

2) Eine F&E-Gruppe implementiert den Algorithmus effizient in TensorFlow und benchmarkt ihn gegen bestehende Optimierer.

3) Ein Startup integriert den Algorithmus in seine SaaS-Plattform, damit Kundinnen und Kunden Modelle schneller trainieren können.

4) Eine Doktorandin/ein Doktorand testet mittels Experimenten, ob große Sprachmodelle eine „Theory of Mind“ zeigen.

5) Ein Unternehmen passt ein Open-Source-Sprach-zu-Text-Modell so an, dass es offline auf Mobilgeräten läuft.

6) Eine Produktmanagerin/ein Produktmanager leitet ein Team zum Bau eines sprachgesteuerten Meeting-Assistenten, der Speech-to-Text mit Kalendern integriert.

Deliverable (ca. 5 Min): Vergleichen Sie die Antworten zwischen den Gruppen und diskutieren Sie, wo es Uneinigkeit gab.

1.7 Einführung in den Technologie-Lebenszyklus

Wenn wir vom *Technologie-Lebenszyklus* sprechen, meinen wir die typische Reise, die jede neue Technologie von ihrem ersten Auftreten bis zu ihrem schlussendlichen Niedergang durchläuft. Das Verständnis dieses Zyklus hilft uns zu erkennen, warum Technologien aufkommen, wie sie sich verbreiten und warum sie schließlich ersetzt werden.

In der **Einführungsphase** taucht eine Technologie erstmals auf. Das kann ein früher Prototyp, ein Pilotprojekt oder ein begrenzter Release sein. Der Fokus liegt hier darauf zu zeigen, dass die Idee machbar ist und ein reales Problem löst – meist in einem engen, aber wertvollen Anwendungsfall. Die Nutzerbasis ist klein, die Kosten sind hoch, und die Technologie ändert sich schnell, während Entwicklerinnen und Entwickler lernen und verbessern. Das Hauptrisiko in dieser Phase ist, zu viel zu versprechen oder es nicht zu schaffen, einen klaren Wert nachzuweisen.

Wenn die Technologie erfolgreich ist, tritt sie in die **Wachstumsphase** ein. Die Adoption beginnt sich zu beschleunigen, Wettbewerber treten in den Markt ein, und die Nachfrage steigt. Der Fokus verlagert sich auf Zuverlässigkeit und wiederholbare Prozesse, um sicherzustellen, dass Nutzerinnen und Nutzer der Technologie vertrauen können und dass sie konsistent geliefert werden kann. Zentrale Indikatoren in dieser Phase sind bessere Leistungskennzahlen, strukturiertere Preisgestaltung sowie verbesserter Support und Dokumentation. Wachs-

tum kann jedoch auch schmerzhaft sein: Skalierung schafft neue technische und organisatorische Herausforderungen, und es entsteht Druck, zu schnell zu viele Funktionen hinzuzufügen.

Schließlich erreicht die Technologie die **Reife**. Zu diesem Zeitpunkt ist sie Mainstream geworden, und der Markt beginnt sich zu sättigen. Der Fokus liegt nun auf Effizienz, Integration in tägliche Arbeitsabläufe und Einhaltung von Standards. Rund um die Technologie entstehen Ökosysteme mit Partnerschaften und etablierten Best Practices. Differenzierung erfolgt nicht mehr hauptsächlich über Kernfunktionen, sondern über Nutzererlebnis und Servicequalität. Die Gefahr in dieser Phase ist Selbstzufriedenheit; Unternehmen riskieren, zurückzufallen, wenn sie aufhören zu innovieren.

Am Ende steht jede Technologie vor dem **Niedergang**. Neuere und effektivere Lösungen erscheinen, und die Nachfrage nach alter Technologie sinkt. An diesem Punkt müssen Organisationen entscheiden, ob sie die Technologie auslaufen lassen (Sunset) oder in etwas Neues transformieren (Erneuerung). Signale des Niedergangs sind sinkende Marktanteile, reduzierte Investitionen und ein Übergang in einen reinen Wartungsmodus. Wird das Lebensende schlecht gemanagt, drohen technische Schulden, zurückgelassene Nutzerinnen und Nutzer und Vertrauensverlust.

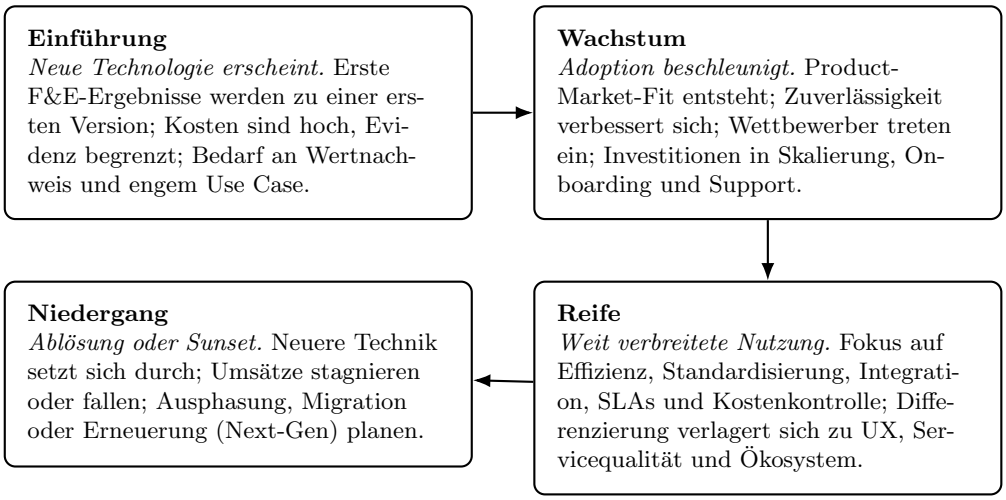

Abbildung 1.1: Technologie-Lebenszyklus (Einführung → Wachstum → Reife → Niedergang).

Exercise

Übung 1.7.2 (Abgrenzung eines KI-Projekts)
Time: 20 minutes **Group size:** 3–4 students
Task: Wählen Sie *einen* realistischen Use Case (z. B. Radiologie-Triage,

Betrugserkennung, Bedarfsprognose, Tutor-Chatbot).
In 15 Minuten, diskutieren Sie die folgenden Punkte:

1) **Phase & Typ:** Lebenszyklusphase (Einführung/Wachstum/Reife/Niedergang) und Projektkategorie (Forschung/Tech Dev/Produkt). Begründung in einem Satz.

2) **Methodenfit:** Wählen Sie die ML-Familie (supervised/unsupervised/RL/GenAI) und begründen Sie *warum* sie passt (Daten, Labels usw.).

3) **Erfolg & Evidenz:** Eine primäre Erfolgsmetrik (Task/KPI) *und* ein Generalisierungs-/Robustheitscheck, den Sie vor dem Ausliefern eines hypothetischen Produkts verlangen.

4) **Risiken & Leitplanken:** Ein ethisches/Datenschutzrisiko und eine Maßnahme (Privacy-by-Design, Fairness-Audit, menschliche Aufsicht).

5) **Kosten/Nutzen-Skizze:** Nennen Sie einen Kostentreiber (Compute, Labeling, Integration) und einen Nutzenpfad (Qualität, Sicherheit, Zeit, Umsatz).

Deliverable (ca. 5 min): Eine kurze Diskussion im Kurs.

1.8 Die grundlegenden Komponenten einer Machine-Learning-Pipeline

Maschinelles Lernen mag komplex klingen, folgt im Kern aber einer einfachen Logik: wir geben dem Computer *Beispiele*, der Computer *lernt daraus Muster*, und kann dann *Vorhersagen* für neue Fälle treffen. Um diesen Prozess zu organisieren, ist es nützlich, in Form einer **Pipeline** zu denken. Eine Pipeline ist einfach eine Abfolge von Schritten, die uns von Rohdaten zu nützlichen Erkenntnissen oder Vorhersagen führen.

1.8.1 Datensammlung

Jedes ML-Projekt beginnt mit **Daten**. Das können medizinische Bilder, Kundentransaktionen, Sensormessungen oder Textdokumente sein. Qualität und Quantität der Daten sind entscheidend, denn der Computer kann nur aus dem lernen, was wir bereitstellen.

2. Datenaufbereitung

Rohdaten sind oft unordentlich: Sie können fehlende Werte, inkonsistente Formate oder irrelevante Teile enthalten. In diesem Schritt *bereinigen* wir die Daten und organisieren sie so, dass der Computer damit arbeiten kann. Manchmal transformieren wir sie auch in ein geeignetes Format (z. B. das Umwandeln von Datumsangaben in Zahlen oder das Skalieren von Bildern).

1.8.2 Modelltraining

Das **Modell** ist das Computerprogramm, das versucht, die Muster in den Daten zu lernen. Ein Modell trainieren bedeutet, ihm viele Beispiele zu zeigen und seine internen Regeln so anzupassen, dass es sich schrittweise verbessert. Wenn die Aufgabe z. B. darin besteht, Katzen von Hunden auf Fotos zu unterscheiden, passt sich das Modell so lange an, bis es besser darin wird, sie auseinanderzuhalten.

1.8.3 Evaluation

Nachdem das Modell trainiert wurde, müssen wir prüfen, ob es zuverlässig ist. Wir testen es auf Daten, die es noch nie gesehen hat, und messen, wie gut es performt. Ist die Leistung schwach, brauchen wir ggf. mehr oder bessere Daten oder wir müssen die Art und Weise des Trainings anpassen.

1.8.4 Bereitstellung und Nutzung

Wenn das Modell gut genug funktioniert, kann es in die Nutzung gehen: eingebettet in eine Handy-App, integriert in Krankenhaus-Workflows oder von Analystinnen und Analysten zur Entscheidungsunterstützung verwendet. Diese Phase nennt man **Deployment**. Von da an kann das Modell neue Eingaben verarbeiten und automatisch Vorhersagen liefern.

Zusammengefasst folgt eine Machine-Learning-Pipeline typischerweise dem Pfad (siehe Abbildung 1.2):

Datensammlung → Aufbereitung → Training → Evaluation → Bereitstellung.

Dieses Ablaufverständnis hilft uns zu sehen, dass ML keine „Magie“ ist, sondern ein strukturierter Prozess mit klaren Schritten und Verantwortlichkeiten.

1.9 Typische Anwendungsfelder von KI in Wirtschaft und Gesellschaft

Künstliche Intelligenz findet in Wirtschaft und Gesellschaft breite Anwendung. Nachfolgend einige repräsentative Use Cases, die Wirkung und Vielfalt illustrieren.

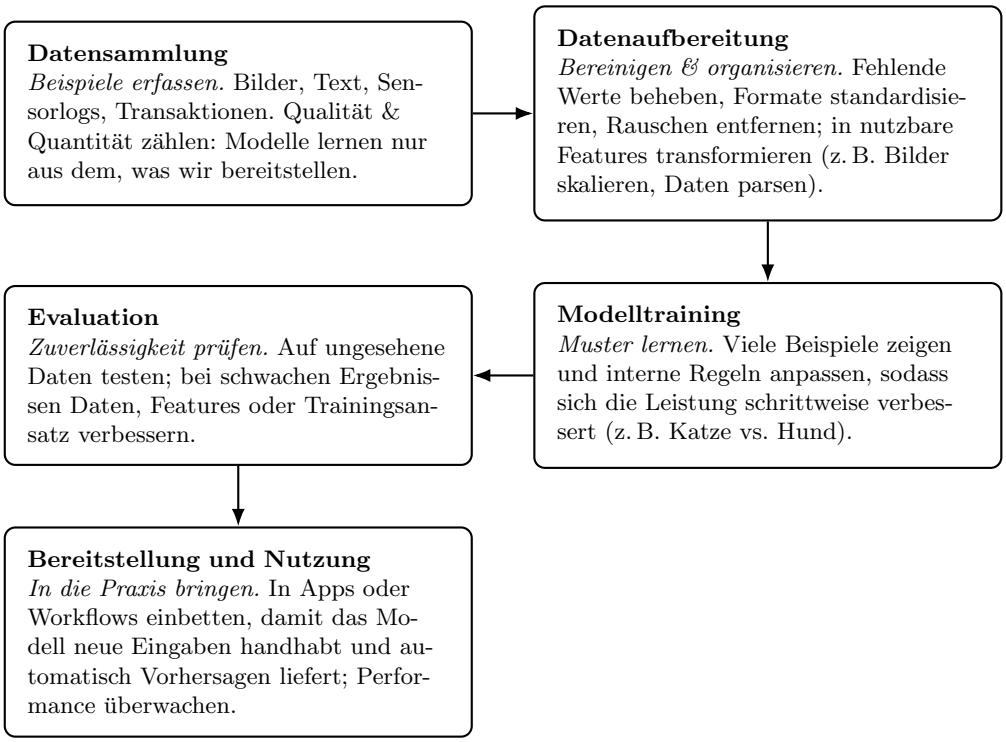

Abbildung 1.2: Eine einfache Machine-Learning-Pipeline: Datensammlung → Datenaufbereitung → Modelltraining → Evaluation → Bereitstellung und Nutzung.

Gesundheitswesen und Medizin

KI transformiert den Gesundheitssektor in der Diagnostik, der personalisierten Medizin und im Krankenhausbetrieb.

- **Bildgebende Diagnostik:** Deep-Learning-Modelle können Tumoren, Frakturen oder Anomalien in radiologischen Scans mit einer Genauigkeit erkennen, die mit der von menschlichen Spezialistinnen und Spezialisten vergleichbar ist oder sie sogar übertrifft.
- **Wirkstoffentdeckung:** KI beschleunigt die Suche nach neuen Arzneimitteln, indem sie Molekülinteraktionen vorhersagt – drastische Reduktion von Zeit und Kosten.
- **Betriebliche Effizienz:** Prädiktive Analytik hilft Krankenhäusern, Patientenströme zu steuern, Ressourcen zu optimieren und Wartezeiten zu reduzieren.

Finanzen und Banking

Der Finanzsektor gehört zu den frühesten Anwendern von KI, vor allem aufgrund seiner Ausrichtung auf datengetriebene Entscheidungen.

- **Betrugserkennung:** ML-Systeme überwachen in Echtzeit Millionen Transaktionen und erkennen verdächtige Muster, die auf Betrug hindeuten können.
- **Algorithmischer Handel:** KI-getriebene Handelsalgorithmen passen sich schnell an Marktbedingungen an und führen Trades mit hoher Frequenz aus.
- **Kreditscoring:** Über traditionelle Metriken hinaus bewerten KI-Modelle Kreditwürdigkeit mithilfe alternativer Daten und ermöglichen so finanzielle Inklusion.

Einzelhandel und E-Commerce

Im Handel verbessert KI sowohl das Kundenerlebnis als auch Backend-Operationen.

- **Empfehlungssysteme:** Plattformen wie Amazon oder Netflix nutzen Collaborative Filtering und Deep Learning, um Produktempfehlungen oder Inhalte zu personalisieren.
- **Bedarfsprognosen:** Prädiktive Modelle antizipieren Kundennachfrage und erlauben die Optimierung von Lagerhaltung und Lieferketten.
- **Chatbots und Virtuelle Assistenten:** NLP ermöglicht Kundensupport rund um die Uhr mit schnellen und relevanten Antworten.

Transport und Mobilität

KI steht im Zentrum der Zukunft intelligenter Mobilität.

- **Autonome Fahrzeuge:** Selbstfahrende Autos nutzen KI für Wahrnehmung, Planung und Steuerung mit dem Ziel, Sicherheit und Effizienz zu erhöhen.
- **Verkehrsmanagement:** Prädiktive Modelle optimieren Ampelschaltungen und urbane Mobilitätsmuster, um Staus zu reduzieren.
- **Logistikoptimierung:** Unternehmen wie UPS oder DHL nutzen KI, um Lieferrouten dynamisch zu optimieren, Kraftstoff zu sparen und den Service zu verbessern.

Öffentlicher Sektor und Gesellschaft

Regierungen und öffentliche Organisationen setzen KI ein, um Entscheidungsfindung und öffentliche Dienste zu verbessern.

- **Smart Cities:** KI ermöglicht prädiktive Wartung von Infrastruktur, intelligentes Energiemanagement und sicherere öffentliche Räume durch Überwachungssysteme.
- **Bildung:** Intelligente Tutoringsysteme personalisieren Lernerfahrungen und identifizieren Stärken und Schwächen von Lernenden.
- **Krisenreaktion:** KI-Modelle werden im Katastrophenmanagement eingesetzt, z. B. zur Vorhersage der Ausbreitung von Waldbränden oder Überschwemmungen.

Kreativindustrien

Über traditionelle Sektoren hinaus prägt KI zunehmend kreative Bereiche.

- **Generative Kunst und Musik:** Systeme wie Generative Adversarial Networks (GANs) und große Sprachmodelle (LLMs) erzeugen Gemälde, Musik oder Literatur.
- **Medienproduktion:** KI unterstützt beim Videoschnitt, beim Dubbing und beim Erstellen realistischer digitaler Schauspieler.
- **Marketing:** Tools zur natürlichen Sprachgenerierung produzieren maßgeschneiderte Werbetexte und Social-Media-Content.

Diese Beispiele verdeutlichen die Bandbreite von KI-Anwendungen. Während sich die technischen Methoden oft überschneiden, unterscheiden sich gesellschaftliche Implikationen, regulatorische Anforderungen und Geschäftsmodelle zwischen den Domänen deutlich.

Exercise

Übung 1.9.3
Goal: Denken Sie über 2–3 Situationen in Ihrem Alltag nach, in denen KI präsent ist.

Task (ca. 15 min): Arbeiten Sie in kleinen Gruppen von 3–4 Studierenden. Diskutieren Sie die folgenden Fragen und notieren Sie kurze Stichpunkte:

- Wo begegnen Ihnen Technologien, die scheinbar Entscheidungen oder Vorhersagen für Sie treffen?
- Woran können Sie erkennen, dass eine Form von KI beteiligt ist (und nicht nur eine einfache Regel oder ein Programm)?
- Was ändert sich in Ihrem Alltag durch dieses KI-System?

1.10 Fünf Hauptkonzepte und wann man sie verwendet

Um Ihre Arbeit in Projekten zu strukturieren, benötigen Sie fünf Hauptkonzepte: Forschungsfragen, Hypothesen, Ziele, Deliverables und Metriken. Je nach Art der Arbeit (Grundlagenforschung, Technologie- oder Produktentwicklung) sind manche wichtiger als andere. Der Vergleich in Tabelle 1.7 stellt die fünf Konzepte heraus und gibt einen Überblick, wann man sie verwendet. Die mit **Primary** markierten sind für einen bestimmten Projekttyp am grundlegendsten, während die mit **Secondary** markierten nützlich, aber nicht zwingend sind.

In der Forschung liegt der primäre Fokus auf der Formulierung präziser Fragen und testbarer Hypothesen, um Wissen voranzubringen. In der Technologieentwicklung verschiebt sich der Schwerpunkt darauf, Machbarkeit durch das Erreichen technischer Spezifikationen mittels Prototypen und Benchmarks nachzuweisen. In der Produktentwicklung steht im Zentrum, geschäftlichen oder Nutzerwert zu schaffen, wobei Ziele, Deliverables und Metriken an konkrete Key Performance Indicators und Kundenergebnisse gekoppelt sind.

Die zentrale Botschaft ist daher, dass, obwohl der Wortschatz ähnlich aussehen mag, seine Bedeutung und Priorität je nach Kontext grundlegend wechseln. Das Erkennen dieser Unterschiede hilft Teams, realistische Erwartungen zu setzen, Erfolgskriterien auszurichten und Verwirrung beim Übergang von Forschung zu Engineering zu Produkten zu vermeiden.

Exercise

Übung 1.10.4

Goal: Verstehen, wie die fünf Konzepte (Forschungsfragen, Hypothesen, Ziele, Deliverables und Metriken) je nach Projekttyp unterschiedliche Rollen spielen.

Task: Arbeiten Sie in kleinen Gruppen von 3–4 Studierenden. Diskutieren Sie Folgendes und notieren Sie kurze Stichpunkte:

1. Wählen Sie je ein Beispielprojekt für *Grundlagenforschung*, *Technologieentwicklung* und *Produktentwicklung*.
2. Entscheiden Sie für jedes Beispiel, welche der fünf Konzepte **Primary** und welche **Secondary** sind.
3. Vergleichen Sie über die drei Projekttypen hinweg: Was ändert sich? Was bleibt gleich?
4. Reflektieren Sie: Warum ist es wichtig, Bedeutung und Priorität dieser Konzepte an den Projekttyp anzupassen?

1.11 Additional Exercises

Exercise

Übung 1.5 (Create a Blueprint for an AI Project)
Time: 20 minutes **Group size:** 3–4 students
Task: Erstellen Sie einen knappen, entscheidungsreifen Projektentwurf für ein KI-Projekt, das in Ihrer Institution/Firma durchgeführt werden könnte. Der Entwurf soll *Scope, Value, Feasibility und Risks* explizit machen.
Choose one use case (z. B. Radiologie-Triage, Bedarfsprognose, Tutoring Assistant, Predictive Maintenance, Dokumentzusammenfassung, Betrugserkennung).

Deliverable: Füllen Sie eine Blaupause mit den unten stehenden Punkten. Seien Sie konkret und knapp (Bullet Points; max. 3 Zeilen pro Feld). Sie können ein Blatt Papier, das Whiteboard etc. nutzen.

1) **Problem & Stakeholder**
   Wer ist Nutzer/Kunde? Welche Entscheidung/Aktion soll dieses System informieren oder automatisieren?

2) **Success Metric & Business Value**
   Eine *primäre* Metrik (Task/KPI) und *warum sie wichtig ist.*

3) **Method Choice**
   Wählen Sie eine Familie (supervised / unsupervised / RL / GenAI) und begründen Sie in einem Satz.

4) **Risks & Guardrails**
   Ein Ethik/Fairness-Risiko, ein Privacy-Risiko und ein operatives Risiko.

5) **System Sketch**
   Einfache Box-und-Pfeil-Skizze: Data Ingestion → Training → Eval → Deployment → Monitoring.

6) **Cost & Timeline**
   Wichtigste Kostentreiber (Labeling, Compute, Integration) und eine grobe Timeline (z. B. 4–6–10 Wochen) über die Phasen.

7) **Lifecycle Stage & Type**
   Wo stehen wir im Technologie-Lebenszyklus (Einführung/Wachstum/Reife/Niedergang)? Ist es Research / Tech Dev / Product? Begründung in einer Zeile.

Tip: Eng abgrenzen. Bevorzugen Sie eine Nutzergruppe, eine Entscheidung, eine Primärmetrik, eine initiale Datenquelle.

Exercise

Übung 1.6 (What is intelligence?)
Goal: Erkennen, dass die Definition von Intelligenz nicht trivial ist und von der Perspektive abhängt.

Task: Arbeiten Sie in kleinen Gruppen von 3–4 Studierenden. Diskutieren Sie die folgenden Fragen und notieren Sie kurze Stichpunkte:

1. Jedes Gruppenmitglied nennt ein Beispiel für Verhalten (Mensch, Tier oder Maschine), das es als „intelligent“ betrachtet.
2. Vergleichen Sie Ihre Beispiele: Was haben sie gemeinsam? Worin unterscheiden sie sich?
3. Versuchen Sie, sich auf eine kurze Definition „intelligenten Verhaltens“ zu einigen, die alle Beispiele abdeckt.
4. Reflexion: War es leicht oder schwierig, Einigkeit zu erzielen? Warum ist die Definition von Intelligenz herausfordernd?

1.12 Tipps und Hinweise für Übungen

Lösung 1.5
Beispiel-Blaupause (Radiologie-Triage):

- **Problem & Stakeholder:** Krankenhaus-Radiologinnen und -Radiologen müssen dringende CT-Scans (z. B. Verdacht auf Schlaganfall) schnell identifizieren, damit Patientinnen und Patienten schneller behandelt werden.
- **Success Metric & Value:** Das System soll nahezu alle Schlaganfallfälle erfassen (sehr hohe Sensitivität), damit Ärztinnen/Ärzte rechtzeitig handeln und Outcomes sich verbessern.
- **Method Choice:** Vergangene CT-Beispiele nutzen, um dem System „beizubringen“, wie Schlaganfall aussieht (Lernen aus Beispielen).
- **Risks & Guardrails:**
  - *Fairness:* Es muss über verschiedene Krankenhäuser und Geräte hinweg gut funktionieren.
  - *Privacy:* Patientendaten müssen anonymisiert werden.
  - *Safety:* Bei Systemausfall muss die ärztliche Arbeit wie üblich möglich sein.
- **System Sketch:** Scan → Computer prüft und gibt Score → Ärztin/Arzt sieht Score in der gewohnten Software.

- **Cost & Timeline:** Benötigt ärztliches Labeling (mehrere Monate), spezielle Rechner fürs Training und anschließend Integration in Krankenhaussysteme. Erste funktionsfähige Version nach ca. 6 Monaten.
- **Lifecycle Stage & Type:** Frühes Pilotprojekt, in kleinen Settings getestet, noch kein Standard.

Lösung 1.6

Hier sind einige mögliche Ergebnisse, zu denen Gruppen gelangen könnten:

- **Beispiele für intelligentes Verhalten:** Eine Matheaufgabe lösen (Mensch), ein Hund lernt einen Trick, ein GPS findet die kürzeste Route, ChatGPT beantwortet eine Frage.
- **Gemeinsame Themen:** Sich an Situationen anpassen, aus Erfahrung lernen, Entscheidungen zielgerichtet treffen.
- **Unterschiede:** Manche betonen Kreativität (ein Gedicht schreiben), andere Effizienz (Rätsel schnell lösen), wieder andere soziale Aspekte (Emotionen verstehen).
- **Gruppendefinition (Beispiel):** „Intelligentes Verhalten ist die Fähigkeit, aus Erfahrung zu lernen und Wissen flexibel anzuwenden, um Ziele in neuen Situationen zu erreichen.“
- **Reflexion:** Gruppen finden es oft schwer, sich zu einigen, weil Intelligenz je nach Kontext Logik, Kreativität, Überleben oder Empathie bedeuten kann. Das zeigt, warum Intelligenz nicht leicht zu definieren ist.

Tabelle 1.6: Forschung, Technologieentwicklung und Produktentwicklung

Aspekt	**Forschung**	**Technologieentwicklung**	**Produktentwicklung**
Ziel	Neues Wissen oder Verständnis schaffen.	Forschungsergebnisse in nutzbare Technologie überführen.	Ein lieferbares, nutzerreifes Produkt bauen.
Fokus	Ideen erkunden, Prinzipien entdecken.	Etwas Funktionales, Skalierbares und Testbares schaffen.	Ein ausgereiftes Produkt erstellen, das Nutzer-bedürfnisse erfüllt.
Output	Theorien, Publikationen, Prototypen, Proofs-of-Concept.	Tools, Libraries, Methoden, frühe Plattformen.	Anwendungen, APIs, physische Güter, Services.
Unsicherheit	Sehr hoch (Ausgang ungewiss).	Moderat (technische Machbarkeit und Leistung getestet).	Niedriger (Nutzerbedürfnisse bekannt, Fokus auf Umsetzung).
Zeithorizont	Langfristig (3–5 Jahre).	Mittelfristig (≈ 2 Jahre).	Kurz- bis mittelfristig (0,5–1 Jahre).
Validierung	Peer-Review, Experimente, theoretische Fundierung.	Benchmarks, Prototypen, Simulationen.	Nutzertests, Product-Market-Fit, Deployment.
Benötigte Fähigkeiten	Domänenwissen, Statistik, theoretisches Denken.	Engineering, Systemdesign, Soft-/Hardware-Integration.	UX, Produkt-management, agile Entwicklung, QA.
Erfolgskriterien	Neuartigkeit, Strenge, Wissensbeitrag.	Machbarkeit, Skalierbarkeit, Performance.	Nutzerakzeptanz, Geschäftswert, Zuverlässigkeit.
Beispiele	Entwicklung eines neuen ML-Algorithmus.	Effiziente Implementierung auf GPUs.	Integration in eine Mobile App für Vorhersagen in Echtzeit.

Tabelle 1.7: When to use what.

Area → / Context ↓	**Research**	**Technology Development (prototype/ feasibility)**	**Product Development (user/ business value)**
Research Question (RQ)	**Primary.** Frames the unknown/why.	*Occasional.* Use when a fundamental unknown blocks engineering.	*Rare.* User discovery may use "questions," but scientific RQs not needed.
Hypothesis (H)	**Primary.** Testable claim with effect size & error rate.	**Useful.** Engineering hypothesis about meeting a technical threshold.	**Useful.** Product/behavioral hypothesis for A/B tests (conversion, retention, NPS).
Goal (G)	*Secondary.* Usually meta-goals (publish, share dataset).	**Primary.** Technical SMART goals (meet spec).	**Primary.** Business/user SMART goals (move KPI).
Deliverables (D)	*Secondary.* Paper, code, dataset card, repo.	**Primary.** Prototype/API, benchmark report, deployment plan.	**Primary.** Shipped feature/service, dashboard, docs, acceptance tests.
Metrics	Significance, CIs, effect sizes, generalization, etc.	Spec thresholds (e.g., $\leq 50\,\text{ms}$ p95, $\geq 92\%$ F1, $\leq \$X/1\text{k}$ calls).	Business/UX KPIs (AHT, CTR, revenue), guardrails (fairness, latency, compliance).

Here's the wörtlich übersetzte LaTeX-Datei (ohne Korrekturen an Inhalt, Rechtschreibung oder Struktur):

“‘latex

Kapitel 2

Daten als Grundlage für KI

Contents

Chapter Abstract

Hochleistungsfähige KI-Systeme bauen auf der Qualität, Eignung und Governance ihrer Daten auf. Dieses Kapitel gibt einen Überblick über die Landschaft der Daten für KI: Typen und Formate (strukturiert, semi-strukturiert, unstrukturiert), übliche Quellen (interne und externe) und zentrale Qualitätsdimensionen (Vollständigkeit, Genauigkeit, Konsistenz, Validität, Einzigartigkeit, Rechtzeitigkeit). Wir diskutieren Labels und Metadaten, typische Fallstricke (Stichproben- und Label-Bias, Messfehler, Leakage und Shift) sowie praktische Guardrails für verantwortungsvolle Nutzung. Transparenzpraktiken wie Datasheets und Data Cards werden neben leichtgewichtigen Dokumentationsgewohnheiten (Lineage, Aufbewahrung, Zugriffskontrolle, Minimierung und Auditierbarkeit) eingeführt. Wir verbinden Prozess und Plattform mit einem Überblick über den Datenlebenszyklus und die Medaillon-Architektur (Bronze - Silver - Gold) als einfaches Muster für progressive Datenveredelung.

2.1 Datentypen

Modellkapazität hilft, aber Daten sind das, was den Unterschied zwischen einem nützlichen und einem nutzlosen Modell ausmacht. Die meisten Fehlschläge stammen von schlechten Daten statt von der Wahl des Algorithmus. Daten kommen in vielen Formen, aber wir können sie in einige Schlüssel-Kategorien gruppieren. Das Verständnis dieser Kategorien ist wesentlich, weil der Datentyp bestimmt, welche Art von Machine-Learning-Ansatz Sie verwenden können oder was Sie erwarten können (oder nicht). In Tabelle 2.1 sehen Sie einen Überblick über die wichtigsten Datentypen, die wir besprechen werden. In Tabelle 2.2 sehen Sie einige Beispiele für jede Kategorie.

2.1.1 Strukturierte Daten: eine Tabellenkalkulation

Dies sind Daten, die in einem festen Format hochorganisiert sind, etwa eine Tabelle mit Zeilen und Spalten. Jede Spalte hat einen spezifischen Datentyp (wie Zahl, Datum oder Kategorie), und jede Zeile ist ein einzelner Datensatz. Es ist die traditionellste Form von Daten und für Computer am einfachsten zu verarbeiten. Beispiele sind ein Bibliothekskartenkatalog. Alles hat einen zugewiesenen Platz, und Sie wissen genau, wo Sie die benötigten Informationen finden. Weitere Beispiele sind Banktransaktionen mit Spalten für Datum, Betrag, Händler und Kategorie, eine Patientendatenbank mit Feldern für Patienten-ID, Alter, Blutgruppe und Diagnosecode sowie eine Bestellhistorie-Tabelle mit Benutzer-ID, gekauftem Produkt, Preis und Kaufdatum.

Datentyp / Format	Beschreibung
Strukturiert	Daten, die in festen Feldern und Datensätzen organisiert sind; Zeilen und Spalten folgen einem konsistenten Schema.
Tabellarisch	Ein Untertyp strukturierter Daten, der in Tabellen angeordnet ist, wie Tabellenkalkulationen oder SQL-Tabellen.
Relational	Strukturierte Daten mit definierten Beziehungen zwischen Tabellen (z. B. Primär-/Fremdschlüssel).
Semi-strukturiert	Daten mit einigen Organisationselementen, aber flexiblem Schema (z. B. verschachtelte oder hierarchische Formate).
Hierarchisch	Baumartige Datenorganisation, bei der Elemente Unterelemente enthalten (häufig in XML, JSON).
Key–Value	Einfache Paarstruktur, bei der jedem Schlüssel ein Wert zugeordnet ist; oft in Logs oder Konfigurationen verwendet.
Unstrukturiert	Daten ohne vordefiniertes Schema oder Format; oft freiform und heterogen.
Textuell	Sequenzielle Zeichendaten ohne strikte Struktur, wie Klartext oder Dokumente.
Multimedia	Nicht-textuelle Daten wie Bilder, Audio oder Video.
Räumlich / Geospatial	Daten mit Position oder Geometrie im Raum (Karten, Koordinaten, 3D-Modelle).
Temporal / Zeitreihen	Daten entlang einer Zeitachse organisiert, die Messungen oder Ereignisse über die Zeit darstellen.

Tabelle 2.1: Gängige Datentypen und -formate im Machine Learning.

2.1.2 Semi-strukturierte Daten

Diese Daten sind nicht auf eine starre Tabelle beschränkt, enthalten aber Tags, Schlüsselwörter oder andere Markierungen, um verschiedene Elemente zu trennen und eine Hierarchie zu erzeugen. Sie haben einige organisatorische Eigenschaften, aber auch einen Grad an Flexibilität. Denken Sie z. B. an ein Rezept. Es hat strukturierte Elemente wie eine Zutatenliste (Posten, Menge) und Kochzeit, aber auch unstrukturierten, frei fließenden Text für die Anweisungen. Eine E-Mail ist ein weiteres Beispiel; sie hat einen strukturierten Header (‚From‘, ‚To‘, ‚Subject‘, ‚Date‘), aber einen unstrukturierten Textkörper, der Bilder oder Dokumente enthalten kann.

2.1.3 Unstrukturierte Daten

Dies sind Informationen, die kein vordefiniertes Datenmodell oder keine Organisation aufweisen. Es ist die häufigste Datenart in der Welt (über 80%) und oft die wertvollste, aber auch die am schwersten zu analysierende. Denken Sie z. B. an eine Schuhschachtel, gefüllt mit einer zufälligen Sammlung handge-

Datentyp / Format	Typische Beispiele
Strukturiert	Datenbanken, Tabellenkalkulationen, CSV-Dateien.
Tabellarisch	Excel-Tabellen, SQL-Tabellen, relationale Datensätze.
Relational	Kunden–Bestell-Datenbanken, Krankenhausinformationssysteme, ERP-Daten.
Semi-strukturiert	JSON-Dateien, XML-Dokumente, HL7/FHIR-Nachrichten.
Hierarchisch	Dateisystembäume, verschachteltes JSON, XML-Konfigurationen.
Key–Value	Konfigurationsdateien, Logs, NoSQL-Datenbanken (z. B. Redis).
Unstrukturiert	Freitext, Bilder, Audio, Video, PDFs.
Textuell	E-Mails, klinische Notizen, Berichte, Transkripte.
Multimedia	Bilder, CT-Scans, aufgezeichnete Sprache, Überwachungsvideos.
Räumlich / Geospatial	Karten, GPS-Koordinaten, Satellitenbilder, 3D-Scans.
Temporal / Zeitreihen	Sensorströme, EKG-Spuren, Aktienkurse, Server-Logs.

Tabelle 2.2: Gängige Datentypen und -formate mit typischen Beispielen.

schriebener Briefe, Fotos, Audiokassetten und Zeitungsausschnitte. Die Information ist reichhaltig, aber in keiner Weise organisiert. Weitere Beispiele sind Kundenrezensionen, Social-Media-Beiträge, Rechtsverträge, medizinische Notizen eines Arztes. Betrachten Sie zusätzlich Satellitenbilder, medizinische Scans (Röntgenaufnahmen, MRTs), Fotos von der Kamera eines selbstfahrenden Autos.

2.1.4 Labels und Ziele

Wir haben besprochen, wie Lernen (in einem überwachten Setting) Labels erfordert (Sie müssen wissen, was Sie vorhersagen wollen, um zu beurteilen, wie gut oder schlecht Ihre Modelle sind). Labels (manchmal Ziele genannt) sind nicht die Rohdaten selbst, sondern vielmehr die „Antwort“ oder das „Ergebnis“, das unser Machine-Learning-Modell vorhersagen soll. Sie werden oft von Menschen erstellt, die die Rohdaten manuell annotieren. Dies ist die Grundlage des *überwachten Lernens* (siehe Abschnitt 1.3 für eine Übersicht).

Zum Beispiel könnten Labels der Lösungsschlüssel am Ende eines Mathebuchs sein. Der Schüler (das Modell) übt an den Aufgaben (den Daten) und prüft seine Arbeit anhand der Antworten (der Labels), um zu lernen. Für eine E-Mail (unstrukturierte Daten) könnte das Label “Spam” oder “Nicht-Spam” sein. Für medizinische Bildgebung (unstrukturierte Daten) könnte das Label eine Diagnose wie “bösartig” oder “gutartig” sein.

Gute Labels sind grundlegend für gute Modelle. Wenn die Labels falsch, in-

konsistent oder fehlend sind, wird das Modell falsche Assoziationen lernen. Zuverlässige Labels zu erstellen erfordert oft menschliche Expertise und ist häufig zeitaufwändig und kostspielig. Labels können allgemein verschiedene Formen annehmen:

- **Binär:** ja/nein, positiv/negativ.
- **Kategorisch:** mehrere mögliche Klassen wie *Katze*, *Hund*, *Pferd*.
- **Numerisch:** kontinuierliche Werte wie Temperatur, Preis oder Tumorgröße.

In vielen realen Anwendungen sind Labels nicht offensichtlich oder direkt messbar. Zum Beispiel könnte das Label „Churn“ im Geschäftskontext bedeuten, dass ein Kunde einen Dienst nach sechs Monaten nicht mehr nutzt, während es im Gesundheitswesen ein Krankheitsergebnis nach Behandlung darstellen könnte. Im Allgemeinen **vertrauen Sie Labels niemals** ohne zu prüfen, wie sie generiert wurden, welcher Prozess verwendet wurde und wie erfahren die Personen waren, die das getan haben. Zum Beispiel ist im medizinischen Bereich bekannt, dass junge Ärzte mehr Fehler machen als erfahrene, daher werden mehr Labels falsch sein, wenn sie von unerfahrenen Ärzten vergeben wurden.

Labels sind nie völlig *neutral*. Sie spiegeln menschliche Entscheidungen darüber wider, was relevant, normal oder wünschenswert ist. Daher wirft der Prozess der Definition und Sammlung von Labels wichtige ethische Fragen auf.

- **Wer definiert die Labels?** Zum Beispiel hängt davon, was als „Erfolg“, „Risiko“ oder „verdächtiges Verhalten“ gilt, vom menschlichen Urteil ab.
- **Sind die Labels objektiv?** Einige Labels können soziale oder kognitive Verzerrungen kodieren, die zu unfairen oder diskriminierenden Modellen führen.
- **Wie zuverlässig sind die Labels?** Menschliche Annotatoren können uneinig sein oder Fehler machen, und automatisierte Labeling-Verfahren können bestehende Verzerrungen fortpflanzen.
- **Wie sehr können wir der Datenquelle vertrauen?** Aus historischen Entscheidungen abgeleitete Labels können vergangene Ungerechtigkeiten reproduzieren.

Diese Punkte sind die Gründe, warum die meisten heute verwendeten Modelle Bias oder unfaires Verhalten zeigen.

2.1.5 Metadaten

Es ist erwähnenswert *Daten über Daten*, sogenannte Metadaten. Sie liefern gewöhnlich Kontext und Informationen über Ursprung, Qualität und Eigenschaften der Primärdaten. Sie werden verwendet, um die Lineage und mögliche

Einschränkungen der Daten zu verstehen. Denken Sie zum Beispiel an die Informationen auf der Rückseite eines Fotos. Sie beschreiben nicht die Personen im Bild, aber sie teilen mit, wo und wann es aufgenommen wurde und mit welcher Art von Kamera. Für Bilder könnten Metadaten den GPS-Standort, Zeitstempel und Kameraeinstellungen umfassen. Für einen Patientendatensatz könnten Metadaten das Krankenhaus, das die Daten erhoben hat, die ID des Arztes, der die Notiz eingegeben hat, und das Datum der letzten Aktualisierung enthalten. Für ein Textdokument könnten Metadaten den Autor, das Erstellungsdatum, die Dateigröße und die Sprache enthalten. In einigen Fällen können Metadaten als Labels verwendet werden, aber das hängt vom spezifischen Anwendungsfall ab.

Übung

Übung 2.7 (Klassifikation von Datentypen und Labels)
Zeit: 15 Minuten **Gruppengröße:** 2–3 Studierende
Aufgabe: Diskutieren Sie die kurzen Projektsituationen unten. Entscheiden Sie für jede:

(a) Was ist der **hauptsächliche Datentyp oder das Format** (wählen Sie aus: Strukturiert, Semi-strukturiert, Unstrukturiert, Tabellarisch, Relational, Hierarchisch, Key–Value, Textuell, Multimedia, Räumlich/Geospatial, Temporal/Zeitreihen).

(b) Ob das Projekt **Labels** verwendet oder braucht (falls ja, beschreiben Sie, was das Label repräsentiert).

Schreiben Sie Ihre Wahl und eine kurze Begründung (ein oder zwei Sätze) auf.

1) Ein Krankenhaus sammelt Vitalzeichen von Patienten jede Sekunde von Monitoren und möchte frühe Anzeichen von Herzversagen erkennen.

2) Eine Nachrichtenorganisation trainiert ein Modell, um eingehende Artikel als „Politik“, „Sport“ oder „Kultur“ zu klassifizieren.

3) Ein Logistikunternehmen analysiert GPS-Daten von Lieferfahrzeugen, um Routen zu optimieren.

4) Ein Forschungsteam baut ein Modell, um lange Rechtsverträge automatisch zu summarisieren.

5) Ein Online-Händler untersucht Nutzerbewertungen und Sternebewertungen, um die Kundenzufriedenheit zu verstehen.

6) Ein Telekommunikationsunternehmen speichert Verbindungsdatensätze (Datum, Dauer, Anrufer, Angerufener) und sagt Kundenabwanderung voraus.

7) Eine Stadt möchte Satellitenbilder verwenden, um die Abdeckung von Grünflächen zu schätzen.

8) Ein Entwickler pflegt Konfigurationsdateien (YAML/JSON), die die Einstellungen und Abhängigkeiten des Microservice beschreiben.

Liefergegenstand (ca. 5 Min): Vergleichen Sie die Antworten zwischen den Gruppen und diskutieren Sie, wo es Uneinigkeit gab, insbesondere dort, wo ein Datensatz in mehr als eine Kategorie passen könnte.

2.2 Quellen, Zugriff und Governance

Machine-Learning-Projekte beginnen selten bei Null. Daten existieren oft irgendwo, gewöhnlich in den internen Systemen einer Organisation, in öffentlichen Repositorien oder können von externen Anbietern bezogen werden. Ursprung und Governance der Daten bestimmen ihre Qualität, Zuverlässigkeit und ob sie ethisch und rechtlich genutzt werden können.

Interne Daten werden innerhalb einer Organisation im Zuge ihrer normalen Abläufe erzeugt. Typische Beispiele umfassen operative Datenbanken, Enterprise-Resource-Planning-(ERP)-Systeme, Data Warehouses oder Data Lakes, Nutzer-Logs und interne Umfragen. Zum Beispiel speichert das elektronische Gesundheitsaktensystem eines Krankenhauses strukturierte medizinische Daten; die Transaktionsdatenbank einer Bank enthält detaillierte Aufzeichnungen der Kundenaktivitäten; die Website eines Unternehmens erzeugt Clickstream-Logs, die das Nutzerverhalten beschreiben. Interne Daten bringen mehrere Herausforderungen mit sich:

- Daten sind gewöhnlich **über mehrere Systeme verteilt** mit unterschiedlichen Formaten und Ownern (daher könnte der Zugriff schwierig sein, da eine komplexe Zugriffsrecht-Hierarchie nötig sein könnte).
- Der Zugriff kann aus Datenschutz- oder Sicherheitsgründen eingeschränkt sein, besonders in regulierten Sektoren wie Gesundheitswesen oder Finanzen.
- Datenqualität kann variieren: fehlende Felder, inkonsistente Identifikatoren oder veraltete Einträge sind wirklich häufig.

Gewöhnlich ist es sehr nützlich, Datenowner und IT-Administratoren früh im Projekt einzubinden. Dokumentieren Sie, woher die Daten kommen, wie sie gesammelt wurden und welche Transformationen angewendet wurden. Schon einfache Schritte wie das Prüfen auf Duplikate oder fehlende Werte können die Nutzbarkeit interner Daten stark verbessern.

Oft sind externe Daten notwendig, um die internen Daten zu ergänzen. Das Problem ist, dass sie außerhalb der Organisation entstehen und Sie daher

möglicherweise keine vollständige Kontrolle darüber oder über deren Qualität haben. Beispiele umfassen Open-Data-Portale (z. B. `data.gov`, `opendata.swiss`), lizenzierte Datensätze, die über Verträge bezogen werden, oder Daten, die von Anbietern und Forschungspartnern geteilt werden. Beispiele reichen von Wetter- und demografischen Daten bis hin zu Satellitenbildern oder medizinischen Benchmarks. Die Arbeit mit externen Daten bringt eigene Herausforderungen mit sich:

- **Zugriff und Lizenzierung:** Überprüfen Sie immer Nutzungsrechte und ob Weiterverteilung oder kommerzielle Nutzung erlaubt ist.
- **Kompatibilität:** Externe Datensätze verwenden oft unterschiedliche Formate, Einheiten oder Definitionen und erfordern sorgfältige Harmonisierung mit internen Daten.
- **Aktualität und Pflege:** Externe Daten werden möglicherweise nicht regelmäßig aktualisiert oder können verschwinden, wenn der Anbieter Bedingungen ändert oder den Zugriff einstellt.

Halten Sie stets das Download-Datum, die Datenquellen-URL, den Lizenztyp und alle angewendeten Vorverarbeitungen fest. Externe Daten sollten als lebende Ressource behandelt werden, prüfen Sie immer auf neuere Versionen oder Korrekturen.

Governance-Grundlagen

Data Governance definiert, wie Daten innerhalb ethischer und rechtlicher Grenzen verwaltet, geschützt und genutzt werden. Sie stellt sicher, dass die Datennutzung mit dem Zweck übereinstimmt, für den sie erhoben wurden, und dass Datenschutz- und Sicherheitsverpflichtungen respektiert werden. Gute Governance ist nicht nur Compliance; sie baut auch Vertrauen auf und verbessert Reproduzierbarkeit. Zentrale Governance-Prinzipien umfassen Folgendes.

- **Zweckbindung:** Verwenden Sie Daten nur für den spezifischen Grund, für den sie erhoben wurden. Beispielsweise sollten Umfragedaten, die für interne Zufriedenheit erhoben wurden, nicht ohne Einwilligung für Marketing wiederverwendet werden.
- **Einwilligung und Verträge:** Stellen Sie sicher, dass Einzelpersonen oder Partner der beabsichtigten Datennutzung zugestimmt haben, insbesondere wenn personenbezogene Daten involviert sind. Bei der Arbeit mit personenbezogenen oder partnerbereitgestellten Daten ist ausdrückliche und informierte **Einwilligung** die Grundlage rechtmäßiger und ethischer Datennutzung. Das bedeutet, dass Personen verstehen müssen, welche Daten erhoben werden, zu welchem Zweck, wie lange sie gespeichert werden und ob sie geteilt oder zur Schulung von KI-Systemen verwendet werden. Einwilligung muss *freiwillig, spezifisch, informiert und widerrufbar*

sein; Menschen sollten sie jederzeit widerrufen können. In organisatorischen oder kollaborativen Kontexten wird Einwilligung oft durch **Verträge oder Datenteilungsvereinbarungen** formalisiert. Diese Dokumente definieren, wem die Daten gehören, wer darauf zugreifen darf und für welche Zwecke sie verwendet werden können. Sie klären auch Verantwortlichkeiten bezüglich Anonymisierung, Sicherheit und Löschung nach Projektende. Eine häufige Herausforderung ist, dass Daten, die für einen Zweck erhoben wurden (zum Beispiel medizinische Behandlung oder Kundenservice), später für einen anderen (wie Forschung oder Produktentwicklung) umgewidmet werden. Unter Datenschutzregelungen wie der EU-**Datenschutz-Grundverordnung (DSGVO)** oder dem schweizerischen **Datenschutzgesetz (DSG)** ist diese „Zweitnutzung" nur erlaubt, wenn sie mit dem ursprünglichen Zweck kompatibel ist oder wenn neue Einwilligung eingeholt wird.

- **Lineage:** Führen Sie einen klaren Nachweis darüber, woher die Daten kamen, wie sie verarbeitet wurden und welche Versionen in welchem Modell verwendet wurden. Üblicherweise versucht *Data Lineage* die Fragen zu beantworten: (i) Wo sind die Daten entstanden (Quellsystem, Datei oder externer Anbieter)? (ii) Welche Transformationen, Filter oder Aggregationen wurden angewendet? (iii) Welche Datenversion oder welcher Snapshot wurde für Training und Evaluierung verwendet? (iv) Wer hat die Daten wann verändert? Ohne ordentliche Lineage wird es beinahe unmöglich zu erklären, warum ein Modell sich auf eine bestimmte Weise verhält oder vergangene Ergebnisse zu reproduzieren. Zum Beispiel: Wenn ein Gesundheitsmodell, das auf einem 2022er-Snapshot von Patientendaten trainiert wurde, sich auf 2024er-Daten anders verhält, helfen klare Lineage-Aufzeichnungen zu identifizieren, ob das Problem auf Veränderungen in der Population, Datenerhebung oder Vorverarbeitungspipelines beruht. Lineage ist auch eine zentrale Anforderung in regulierten Domänen wie Gesundheitswesen, Finanzen und öffentlichem Sektor, wo Transparenz und Verantwortlichkeit verpflichtend sind. Moderne Datenmanagement-Tools (zum Beispiel *Data Catalogs* oder *Model Registries*) beinhalten Lineage-Tracking oft als Teil ihrer Funktionalität und zeichnen automatisch Versionen, Transformationen und Abhängigkeiten auf. Open-Source-Data-Lineage-Tools, die Sie prüfen könnten, sind *OpenLineage*, *Marquez*, *DataHub*, *OpenMetaData* und *Apache Atlas* (nur um einige zu nennen).

- **Aufbewahrung und Löschung:** Speichern Sie Daten nur so lange wie notwendig und haben Sie Verfahren, um sie zu löschen oder zu anonymisieren, wenn sie nicht mehr gebraucht werden. Jeder Datensatz hat einen Lebenszyklus. Daten sollten nur so lange gespeichert werden, wie sie einem legitimen und notwendigen Zweck dienen, danach sollten sie entweder gelöscht oder irreversibel anonymisiert werden. Daten auf unbestimmte Zeit zu behalten, erhöht das Risiko von Verstößen, Missbrauch

und regulatorischen Verletzungen, und es kann auch zu unnötigen Speicherkosten und Verwirrung darüber führen, welche Version maßgeblich ist. Dieses Prinzip der **Datenaufbewahrung und -löschung** ist in den meisten Datenschutzgesetzen verankert, einschließlich der EU-DSGVO und des Schweizer Bundesgesetzes über den Datenschutz (DSG). Beide verlangen, dass Organisationen festlegen, wie lange Daten aufbewahrt werden, und diese Dauer begründen. Zum Beispiel dürfen medizinische Daten aufgrund gesetzlicher Verpflichtungen für eine feste Anzahl von Jahren aufbewahrt werden, während Aktivitätslogs von Nutzern für eine Online-Plattform möglicherweise nur für einige Monate benötigt werden.

- **Zugriffskontrollen:** Beschränken Sie den Datenzugriff auf autorisiertes Personal und verwenden Sie sichere Speicherumgebungen. Zugriffskontrolle bedeutet, dass nur Personen, die die Daten wirklich für ihre Arbeit oder Forschung benötigen, sie einsehen oder ändern können. Dieses Prinzip wird oft als **Least Privilege** zusammengefasst: Geben Sie jedem Nutzer das Minimum an Zugriff, das für seine Rolle notwendig ist. Beispielsweise könnte ein Data Scientist Zugriff auf anonymisierte Datensätze für das Modelltraining benötigen, während ein Administrator Rohdaten unter strengeren Vertraulichkeitsvereinbarungen handhabt. Sicherer Speicher umfasst sowohl **technische Maßnahmen** (wie Verschlüsselung, Authentifizierung und Netzwerksicherheit) als auch **organisatorische Maßnahmen** (wie rollenbasierte Zugriffsrichtlinien und Vertraulichkeitsschulungen). Daten sollten auf institutionellen Servern oder vertrauenswürdigen Cloud-Umgebungen gespeichert werden, die anerkannte Standards erfüllen (zum Beispiel ISO 27001, HIPAA oder DSGVO-Anforderungen), nicht auf persönlichen Laptops oder unverschlüsselten Laufwerken. Verwenden Sie nach Möglichkeit verschlüsselte Dateisysteme, passwortgeschützte Repositorien und sichere Übertragungsmethoden wie `sftp` oder HTTPS statt E-Mail-Anhängen oder USB-Sticks. Zugriffskontrolle ist auch dynamisch: Berechtigungen sollten regelmäßig überprüft und entzogen werden, wenn sie nicht mehr benötigt werden, etwa wenn ein Teammitglied ein Projekt verlässt. Audit-Logs und Überwachungssysteme können aufzeichnen, wer wann auf Daten zugegriffen hat und welche Aktionen durchgeführt wurden; dies verbessert Transparenz und unterstützt die Vorfalluntersuchung bei Missbrauch oder Verstoß.

- **Datenminimierung:** Das Prinzip der **Datenminimierung** bedeutet, dass Sie nur die Daten erheben und verarbeiten sollten, die für den spezifischen Zweck Ihres Projekts zwingend notwendig sind. Mit anderen Worten: „gerade genug Daten, um die Aufgabe zu erledigen, und nicht mehr.“ Das Sammeln großer Mengen irrelevanter oder übermäßig detaillierter Informationen erhöht Risiken, Datenschutzverletzungen, Sicherheitsbrüche und Model-Bias, ohne die Leistung notwendigerweise zu verbessern. Bevor Sie Daten erfassen oder verwenden, fragen Sie: Welche Frage versuche ich zu beantworten, und welche Daten brauche ich wirklich, um sie zu beantworten? Zum Beispiel ist es zur Vorhersage von Krankenhauseinweisun-

gen möglicherweise nicht erforderlich, Namen, Adressen oder vollständige Krankenakten der Patienten einzubeziehen. Ähnlich könnten beim Analysieren von Kundenabwanderung aggregierte Statistiken wie Altersgruppe oder Region ausreichen, statt personenbezogene Details wie Telefonnummern oder exakte Geburtsdaten zu speichern. Datenminimierung ist nicht nur ein ethisches Ideal, sondern auch eine rechtliche Anforderung gemäß Regelwerken wie der EU-DSGVO und dem schweizerischen Datenschutzgesetz (DSG). Diese Rahmenwerke verlangen, dass die Datenerhebung dem angegebenen Zweck angemessen ist und unnötige Informationen vermieden oder anonymisiert werden. Kleinere, gut zielgerichtete Datensätze sind außerdem leichter zu verwalten, zu sichern und zu dokumentieren, was Kosten und Komplexität reduziert.

- **Auditierbarkeit** bedeutet, dass die gesamte Datennutzung und -transformation später überprüft, nachverfolgt und reproduziert werden kann. Sie stellt sicher, dass jeder, ob Kollege, externer Prüfer oder Regulator, verstehen kann, wie die Daten erhalten, verarbeitet und zur Schulung oder Bewertung eines Modells verwendet wurden. In der Praxis erfordert Auditierbarkeit das Führen detaillierter Aufzeichnungen über Datenverarbeitungsaktivitäten. Dazu gehört die Dokumentation von Datenquellen, Vorverarbeitungsskripten, Bereinigungsregeln, Filterkriterien und allen manuellen Eingriffen oder Labeling-Prozessen. Jede Hauptversion eines Datensatzes oder Modells sollte identifizierbar sein, idealerweise über Versionskontrollsysteme oder Metadaten-Tracking-Tools. Automatisierte Logging-Systeme können aufzeichnen, wer Daten wann aufgerufen oder verändert hat und bieten so eine überprüfbare Historie der Operationen. Mangelnde Auditierbarkeit ist eine häufige Ursache für nicht reproduzierbare Forschung und unerklärliches Modellverhalten. Wenn die Datenpipeline intransparent ist, können selbst kleine Änderungen, zum Beispiel ein fehlender Vorverarbeitungsschritt oder eine erneut exportierte CSV-Datei, es unmöglich machen, vergangene Ergebnisse zu replizieren. In regulierten Umgebungen wie Gesundheitswesen oder Finanzen kann unzureichende Dokumentation der Datennutzung außerdem zu Compliance-Verstößen führen.

Denken Sie daran, Daten-Governance als fortlaufenden Prozess zu behandeln, nicht als einmalige Aufgabe. Dokumentieren Sie alles: woher die Daten kamen, wie sie verändert wurden und wer Zugriff darauf hat.

2.3 Datenqualität: Dimensionen und schnelle Checks

Die in Tabelle 2.3 aufgeführten Dimensionen fassen die wichtigsten Aspekte der Datenqualität zusammen, die bestimmen, ob ein Datensatz für Machine Learning geeignet ist. Selbst das ausgefeilteste Modell wird scheitern, wenn die zugrunde liegenden Daten unvollständig, ungenau oder inkonsistent sind. Das Verstehen und Prüfen dieser Dimensionen hilft, häufige Datenprobleme früh in einem Projekt zu identifizieren, bevor sie in irreführende Ergebnisse münden.

Vollständigkeit

Vollständigkeit beschreibt, ob alle erforderlichen Datenfelder und Datensätze vorhanden sind. Fehlende Werte können durch Systemfehler, optionale Felder oder Unterschiede in der Dateneingabepraxis entstehen. Wenn zum Beispiel Patientengeburtstage in 20% der Krankenhausakten fehlen, kann das Modell verzerrte Assoziationen lernen. Schnelle Checks umfassen das Berechnen von Null- oder Fehlwert-Raten pro Spalte oder pro Gruppe und das Überprüfen, ob bestimmte Entitäten (z. B. Patienten, Kunden, Sensoren) systematisch unterrepräsentiert sind.

Genauigkeit

Genauigkeit misst, wie nah die erfassten Werte an den Fakten der realen Welt liegen, die sie repräsentieren. Fehler können durch fehlerhafte Sensoren, manuelle Eingabefehler oder veraltete Quellen entstehen. Wenn GPS-Koordinaten zum Beispiel auf den nächsten Kilometer gerundet werden, können Standortvorhersagen unzuverlässig werden. Genauigkeit kann durch Bereichsregeln überprüft werden (z. B. muss die menschliche Körpertemperatur zwischen 35–42 °C liegen) oder durch Abgleich von Werten mit Referenzsystemen oder unabhängigen Datensätzen.

Konsistenz

Konsistenz bezieht sich darauf, ob Daten die gleiche Bedeutung, Einheiten und Interpretation über Quellen hinweg haben. Verschiedene Systeme können dasselbe Konzept leicht unterschiedlich darstellen, beispielsweise „M“ versus „Male“, oder Gewicht in Kilogramm versus Pfund. Inkonistenzen können die Modellqualität stark beeinflussen.

Validität

Validität prüft, ob Daten erwarteten Formaten, Typen oder Geschäftsregeln entsprechen. Zum Beispiel sollten Daten Datumsangaben in einem Standardformat enthalten, Postleitzahlen müssen bekannten Mustern entsprechen, und kategoriale Felder sollten nur erlaubte Werte enthalten. Automatisierte Tools können ungültige Einträge mit regulären Ausdrücken, Aufzählungen oder Parsing-Regeln markieren. Hohe Raten ungültiger Werte deuten häufig auf vorgelagerte Probleme in der Datenerhebung hin.

Einzigartigkeit

Einzigartigkeit stellt sicher, dass Entitäten nur einmal repräsentiert werden, ohne Duplikate oder Leakage. Doppelte Datensätze blähen Stichprobengrößen auf und können Modelle verzerren, während doppelte Identifikatoren Verwechslungen zwischen Individuen oder Fällen verursachen können. Typische Checks beinhalten das Erkennen wiederholter Primärschlüssel, den Vergleich von Hashes für

nahezu doppelte Zeilen oder die Überprüfung, dass Trainings- und Testsets keine überlappenden Entitäten teilen.

Rechtzeitigkeit

Rechtzeitigkeit erfasst, wie aktuell die Daten im Verhältnis zum modellierten Phänomen sind. Für dynamische Prozesse wie Finanztransaktionen oder Patientenmonitoring können veraltete Daten zu überholten Vorhersagen führen. Das Messen von Altersverteilungen der Daten oder das Berechnen der Verzögerung zwischen Ereigniseintritt und Datenverfügbarkeit liefert einen schnellen Hinweis auf Frische. Wenn Datenpipelines automatisiert sind, kann kontinuierliches Monitoring Teams alarmieren, falls Aktualisierungen stoppen oder Verzögerungen zunehmen.

Dimension	Typische Fragen	Schnelle Checks
Vollständigkeit	Fehlen Felder?	Nullraten, gruppenweise Missingness.
Genauigkeit	Spiegeln Werte die Realität wider?	Bereiche (ist Alter realistisch?), konsistente Messungen, etc..
Konsistenz	Gleiche Bedeutung über Quellen?	Schema diffrences, Unit-of-Measurements-Checks (etwas in cm und etwas in m gemessen).
Validität	Entsprechen erlaubten Formaten?	Datums-Parsing-Fehler.
Einzigartigkeit	Duplikate? Leakage?	Primärschlüssel-Duplikate, wenn nicht erzwungen, etc.
Rechtzeitigkeit	Genug frisch?	Alters-Histogramme, fehlende Zeiträume, nur alte Daten.

Tabelle 2.3: Datenqualitäts-Spickzettel.

Schnellübung (10 Min)

Übung 2.8
Denken Sie an Daten, mit denen Sie in Ihrem Unternehmen oder Projekten arbeiten, oder imag(in)ieren Sie einen Datensatz. Listen Sie *zwei* Felder auf, bei denen fehlende oder ungültige Werte Entscheidungen kritisch schädigen würden. Notieren Sie für jedes einen automatisierten Check, den Sie implementieren würden, und einen manuellen Check.

2.4 Transparenz: Datasheets und Data Cards

Transparenz ist ein Eckpfeiler vertrauenswürdiger KI. So wie wissenschaftliche Experimente eine detaillierte Dokumentation von Materialien und Methoden benötigen, sollten auch Datensätze, die für Machine Learning verwendet werden, klare und standardisierte Dokumentation mitbringen. Dies stellt sicher, dass andere verstehen können, was die Daten darstellen, wie sie erhoben wurden, welche Einschränkungen sie haben und unter welchen Bedingungen sie verwendet werden dürfen. Schlecht dokumentierte Datensätze sind eine der Hauptursachen für versteckten Bias, Missbrauch und nicht reproduzierbare Ergebnisse. Ein praktischer Ansatz für Datentransparenz ist die Nutzung kurzer, strukturierter Zusammenfassungen, bekannt als **Datasheets** oder **Data Cards**. Ein Datasheet ist ein umfassender Bericht, der Motivation, Zusammensetzung, Erhebungsprozess und empfohlene Verwendungen eines Datensatzes beschreibt, während eine Data Card eine leichtere, zugänglichere Version ist, die auf eine oder zwei Seiten passt. Beide dienen demselben Zweck: wesentliche Informationen über den Datensatz an Forschende, Entwickler und Endnutzer zu kommunizieren.

Jeder Datensatz, der für Machine Learning verwendet wird, sollte zumindest eine minimale *Data Card* enthalten, die Ursprung, beabsichtigte Nutzung, Abdeckung, bekannte Lücken sowie rechtliche oder ethische Einschränkungen dokumentiert. Tabelle 2.4 zeigt ein Beispiel für eine kompakte Struktur. Da-

Feld	Inhalt
Name/Owner	Titel des Datensatzes, Maintainer, Kontaktperson oder Organisation.
Provenienz	Quellsysteme, Erhebungsdaten, Jurisdiktionen und verantwortliche Parteien.
Beabsichtigte Nutzung	Unterstützte Entscheidungen und Modelltypen; nennen Sie ausdrücklich nicht abgedeckte oder unangebrachte Nutzungen.
Population	Wer oder was repräsentiert wird (Patienten, Kunden, Geräte, Regionen) und bekannte Ausschlüsse oder Ungleichgewichte.
Labels	Wie Labels gewonnen wurden, Qualitätskontrollverfahren und Hinweise zu Inter-Rater-Variabilität.
Qualitätsnotizen	Bekannte Einschränkungen wie fehlende Daten, Messfehler, Drift oder potenzielle Bias-Quellen.
Recht/Ethik	Einwilligungsstatus, Lizenzbedingungen, Datenschutzmaßnahmen und Verweis auf Datenschutz-Folgenabschätzung (DPIA), falls verfügbar.
Updates	Aktualisierungsfrequenz, Versionierungsstrategie und Ausmusterungsplan für veraltete Daten.

Tabelle 2.4: Minimale Data-Card-Vorlage.

ta Cards machen das Unsichtbare sichtbar. Sie liefern wesentlichen Kontext, der verhindert, dass Datensätze als neutral oder universell behandelt werden,

wenn sie in Wahrheit spezifische Entscheidungen und Einschränkungen widerspiegeln. Eine gut vorbereitete Data Card hilft neuen Nutzern, schnell zu verstehen, was der Datensatz leisten kann und was nicht, und reduziert so das Risiko unangebrachter Nutzung oder Überverallgemeinerung. Zum Beispiel ermöglicht ein Gesichtserkennungsdatensatz, der seine demografische Verteilung klar dokumentiert, den Nutzern, Fairness-Risiken vor dem Deployment zu bewerten. Eine Data Card zu erstellen erfordert keine juristische Ausbildung oder einen langen Bericht; sie kann als einfache Textdatei beginnen, die zusammen mit dem Datensatz gespeichert wird. Der Schlüssel ist, Informationen so früh wie möglich zu erfassen: wer die Daten gesammelt hat, warum, wie sie aktualisiert werden und welche Probleme bereits bekannt sind. Wann immer Daten intern oder extern geteilt werden, sollte die Data Card sie begleiten. Sie sollte auch aktualisiert werden, wenn sich der Datensatz ändert, so wie versionierter Code oder Modelldokumentation.

Übung (20 Min)

Übung 2.9 (Schreiben Sie eine einseitige Data Card)
Erstellen Sie in Gruppen eine Data Card für einen Datensatz Ihrer Wahl. Wählen Sie eine der folgenden realen Datensammlungen oder Open-Data-Quellen. Ihr Ziel ist es nicht, die Daten herunterzuladen oder zu verarbeiten, sondern ihre Dokumentation online zu untersuchen und eine kurze Data Card zu entwerfen, die Ursprung, Zweck, Inhalt und Einschränkungen beschreibt. Verwenden Sie die Leitfragen aus Tabelle 2.4.

1) **OpenStreetMap (OSM):** Eine globale, von Freiwilligen erstellte Karte der Welt. Berücksichtigen Sie, wer beiträgt, welche geografischen Regionen gut oder schlecht repräsentiert sind und wie die Lizenz (Open Database License) die Wiederverwendung beeinflusst (`https://www.openstreetmap.org/`).

2) **MIMIC-III Clinical Database:** Ein frei verfügbarer Datensatz von Intensivstationsdaten (ICU) aus dem Beth Israel Deaconess Medical Center. Reflektieren Sie Einwilligungsverfahren, De-Identifizierung und ethische Nutzung in medizinischer KI (`https://physionet.org/content/mimiciii/1.4/`).

3) **COCO (Common Objects in Context):** Ein großer Bilddatensatz für Objekterkennung. Untersuchen Sie, wie Bilder gesammelt und annotiert wurden, welche Kategorien existieren und welche Populationen oder Kontexte unterrepräsentiert sein könnten (`https://cocodataset.org/`).

4) **IMDb Movie Dataset:** Weit verbreitet für Forschung zu Empfehlungen und Textanalyse. Berücksichtigen Sie, welche Entitäten repräsentiert sind, potenzielle kommerzielle Verzerrungen und die

Herausforderungen, einen solchen Datensatz aktuell zu halten (`https://ai.stanford.edu/~amaas/data/sentiment/`).

5) **Humanitarian Data Exchange (HDX):** Vom UN-Management, das globale Krisen- und Entwicklungsdaten hostet. Wählen Sie einen Datensatz (z. B. Flüchtlingsbewegungen, Ernährungssicherheit) und beschreiben Sie Ursprung, Sensitivität und Governance-Mechanismen (`https://data.humdata.org/`).

6) **Kaggle COVID-19 Datensätze:** Community-geteilte Pandemiedaten inklusive Fallzahlen, Mobilität und politische Maßnahmen. Berücksichtigen Sie Aktualität, Zuverlässigkeit der Quellen und Unterschiede zwischen den Berichtsstandards der Länder (`https://www.kaggle.com/datasets/imdevskp/corona-virus-report`).

2.5 Der Datenlebenszyklus

Daten sind nicht statisch; sie durchlaufen eine Reihe von Phasen von der anfänglichen Erhebung bis zur eventualen Archivierung oder Löschung. Das Verständnis des **Datenlebenszyklus** hilft sicherzustellen, dass Daten über ihre gesamte Existenz hinweg verantwortungsvoll und effizient verwaltet werden. Es klärt auch, wer in welcher Phase wofür verantwortlich ist und wo Governance-, Qualitäts- und Transparenzmaßnahmen angewendet werden sollten. Der Lebenszyklus umfasst typischerweise sechs bis acht Phasen, abhängig von der Organisation. Tabelle 2.5 fasst die wichtigsten zusammen, während Abbildung 2.1 ein Diagramm davon zeigt.

Denken Sie immer daran, dass der Datenlebenszyklus iterativ, nicht linear ist; Feedback aus Deployment und Monitoring führt oft zurück zu neuer Datenerhebung oder verbesserten Bereinigungsschritten. Jede Phase sollte Dokumentation, Versionierung und Checks für Qualität, Datenschutz und Sicherheit enthalten.

2.6 Die Medaillon-Architektur

Moderne Datenplattformen organisieren ihre Datenverarbeitungspipelines zunehmend mit der **Medaillon-Architektur**, einem geschichteten Ansatz, der Datenqualität und Governance verbessert. Ursprünglich von Databricks popularisiert, definiert sie eine Abfolge von Schichten: Bronze, Silver und Gold, jede mit zunehmendem Grad an Datenveredelung und Zuverlässigkeit.

Das Ziel der Medaillon-Architektur ist es, rohe, bereinigte und aggregierte Daten in getrennte, nachvollziehbare Stufen zu trennen. Jede Schicht baut auf der vorherigen auf und stellt sicher, dass Datentransformationen transparent,

reproduzierbar und reversibel sind. Tabelle 2.6 fasst die drei Schichten zusammen.
Jede Schicht spielt eine eigene Rolle:

- Die **Bronze**-Schicht erfasst Daten wie sie sind, stellt sicher, dass nichts verloren geht und dass Rohquellen jederzeit erneut betrachtet werden können.
- Die **Silver**-Schicht standardisiert und bereinigt Daten und schafft eine „Single Source of Truth“, die für die meisten analytischen und Modellierungsaufgaben geeignet ist.
- Die **Gold**-Schicht liefert vertrauenswürdige, kuratierte Daten, die Business Intelligence, Dashboards oder bereitgestellte Machine-Learning-Pipelines unterstützen.

Glossar

Künstliche Intelligenz (KI)
: Das breite Feld, das darauf abzielt, Systeme zu bauen, die Aufgaben ausführen, die menschliche Kognition erfordern (Schlussfolgern, Wahrnehmung, Planung, Sprache).

Maschinelles Lernen (ML)
: Ein Teilgebiet der KI, bei dem Systeme eine Aufgabe verbessern, indem sie Muster aus Daten lernen statt handgefertigte Regeln zu verwenden.

Tiefes Lernen (DL)
: Methoden des maschinellen Lernens basierend auf mehrschichtigen neuronalen Netzen, die hierarchische Repräsentationen lernen (z. B. CNNs, Transformer).

Generative KI (GenAI)
: Modelle, die Datenverteilungen lernen und neue Inhalte erzeugen (Text, Bilder, Audio, Code), die den Trainingsdaten ähneln.

Überwachtes Lernen
: Lernen aus gelabelten Paaren (x, y), um das korrekte y für neue Eingaben x vorherzusagen.

Unüberwachtes Lernen
: Struktur in ungelabelten Daten entdecken, wie Cluster oder niederdimensionale Repräsentationen.

Semi-überwachtes Lernen
: Training mit einem kleinen gelabelten Satz plus einem großen ungelabelten Satz, um Leistung und Dateneffizienz zu steigern.

Reinforcement Learning (RL)
: Agenten lernen Aktionssequenzen, indem sie kumulative Belohnung durch Interaktion mit einer Umgebung maximieren.

ML-Pipeline
: Der End-to-End-Prozess: Datenerhebung $\rightarrow$ Vorbereitung $\rightarrow$ Training $\rightarrow$ Evaluierung $\rightarrow$ Deployment.

Datenerhebung
: Rohes, relevantes Material (Bilder, Text, Logs, Signale) für die jeweilige Aufgabe beschaffen.

Datenvorbereitung
: Daten bereinigen, standardisieren und transformieren (Missingness behandeln, formatieren, Feature-Extraktion), damit Modelle sie verwenden können.

Modelltraining
: Modellparameter auf Trainingsdaten optimieren, um eine Verlustfunktion zu minimieren und aufgabenrelevante Muster zu lernen.

Evaluierung
: Auf ungesehenen Daten mit geeigneten Metriken testen, um Zuverlässigkeit, Generalisierung und Trade-offs zu überprüfen.

Deployment
: Ein validiertes Modell in Produkte oder Workflows integrieren; beinhaltet Monitoring für Leistung, Drift und Ausfälle.

Grundlagenforschung
: Wahrheitssuchende Arbeit, die präzise Fragen beantwortet und generalisierbares Wissen erzeugt (Theorie, Methoden, Datensätze).

Angewandte Forschung
: Problemgetriebene Untersuchung, die verwertbare Evidenz in einem spezifischen Kontext generiert (Prototypen, Piloten, Feldtests).

Technologieentwicklung
: Forschungsideen in nutzbare, skalierbare Prototypen und Methoden überführen; Machbarkeit gegen technische Spezifikationen belegen.

Produktentwicklung
: Benutzerorientierte Funktionen/Dienste liefern, die in Produktionsumgebungen geschäftlichen oder Nutzerwert schaffen.

Erfolgsmetrik / KPI
: Die primäre quantitative Maßzahl für Fortschritt oder Wert (Task-Metriken oder Business-KPIs), plus etwaige Guardrail-Metriken.

Technologie-Lebenszyklus
: Der Verlauf einer Technologie: *Einführung* (Wert belegen) → *Wachstum* (Skalierung, Zuverlässigkeit) → *Reife* (Effizienz, Standards) → *Niedergang* (Sunset oder Erneuerung).

‟‟

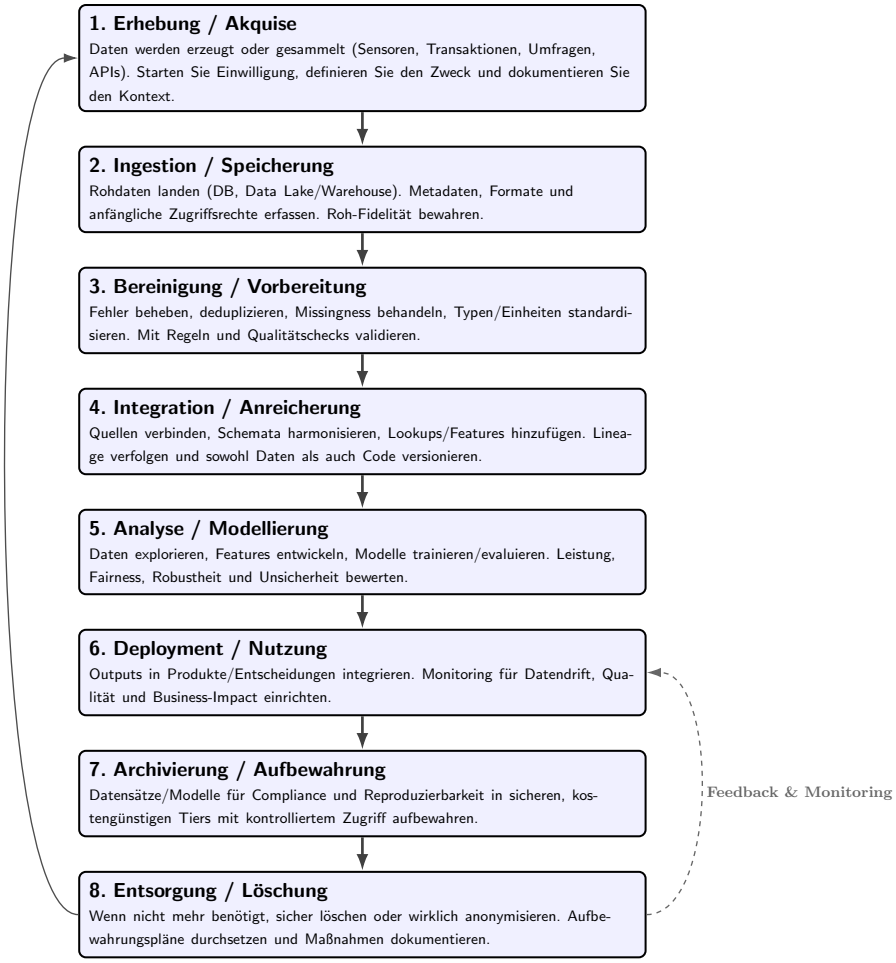

Abbildung 2.1: Datenlebenszyklus. Die Schleife links betont den kontinuierlichen Charakter von Datenprozessen.

Phase	Beschreibung und typische Aktivitäten
1. Erhebung / Akquise	Daten werden von Sensoren, Transaktionen, Umfragen oder externen APIs generiert oder gesammelt. Governance beginnt hier, mit Einwilligung, Stichprobe und Dokumentation.
2. Ingestion / Speicherung	Rohdaten werden in Speichersysteme wie Datenbanken, Data Lakes oder Warehouses importiert. Metadaten werden erfasst, Formate standardisiert und Zugriffsrechte definiert.
3. Bereinigung / Vorbereitung	Fehler, Duplikate und fehlende Werte werden korrigiert; Daten werden transformiert, verbunden und validiert. Diese Phase beansprucht oft 60–80% des Projektaufwands.
4. Integration / Anreicherung	Daten aus mehreren Quellen werden kombiniert und ergänzt (z. B. Hinzufügen von Geolokation oder demografischen Informationen). Lineage und Versionskontrolle sind kritisch.
5. Analyse / Modellierung	Machine-Learning-Modelle, Dashboards oder Berichte werden entwickelt und getestet. Data Scientists und Analysten arbeiten auf kuratierten, hochwertigen Datensätzen.
6. Deployment / Nutzung	Analytische Ergebnisse oder Modelle werden in Produkte, Entscheidungen oder automatisierte Systeme integriert. Monitoring stellt laufende Leistung und Fairness sicher.
7. Archivierung / Aufbewahrung	Daten werden langfristig für Compliance, Reproduzierbarkeit oder Audit-Zwecke gespeichert, oft in sicheren, kostengünstigen Umgebungen.
8. Entsorgung / Löschung	Wenn Daten nicht mehr benötigt werden, werden sie sicher gelöscht oder anonymisiert, um Risiko zu reduzieren und Aufbewahrungsrichtlinien einzuhalten.

Tabelle 2.5: Typische Phasen des Datenlebenszyklus.

Schicht	Zweck	Typische Inhalte / Aktionen
Bronze (Roh)	Daten exakt wie empfangen speichern und volle Fidelität bewahren.	Logs, CSVs, JSON-Dateien oder Sensorströme, die aus Quellsystemen ingestiert werden. Minimale Bereinigung; Fokus auf Vollständigkeit und Nachverfolgbarkeit.
Silver (Bereinigt)	Strukturierte, validierte und verbundene Datensätze bereitstellen, die für Analysen bereit sind.	Deduplizierte Tabellen, konsistentes Schema, angereichert mit Metadaten, Einheitenkonversionen und Fehlerbehandlung. Häufig von Data Scientists für Feature-Extraktion verwendet.
Gold (Kuratiert)	Hochwertige, geschäftsreife Datenprodukte liefern.	Aggregierte Datensätze, KPIs, Dashboards und Modell-Trainingstabellen. Dient Entscheidungsträgern oder Produktionssystemen. Betonung auf Genauigkeit, Leistung und Governance.

Tabelle 2.6: Die Medaillon-Architektur: Schichten der Datenveredelung.

“‘latex

Kapitel 3

Maschinelles Lernen in der Praxis: Aufgaben, Daten und Validierung

Contents

Chapter Abstract

Dieses Modul vermittelt ein praktisches Denkmodell dafür, wie Maschinen lernen und wie man Modelle verantwortungsvoll einsetzt. Wir besprechen gängige Aufgabentypen (Klassifikation, Regression, Clustering) und erklären Eingaben (*Features/Merkmale*), Training und Validierung. Wir zeigen, was Modellvalidierung ist und wie man Metriken wählt, die zu den realen Kosten passen (z. B. Precision/Recall für Erkennung, MAE/RMSE für Prognosen), mit einem kurzen fortgeschrittenen Formel-Referenzteil. Wir betonen menschliche Aufsicht, einfache Prüfungen auf Fairness und Kalibrierung sowie klare Kommunikation dessen, was ein Modell kann und nicht kann. Außerdem stellen wir AutoML/No-Code-Werkzeuge vor, wo sie helfen, wo sie scheitern, und schließen mit praktischer Anleitung zur Validierung (Hold-out und intuitive Cross-Validation) und zur Entscheidung, wann ein Modell „gut genug“ für den Einsatz ist.

3.1 Klassifikation, Regression und Clustering

Systeme des maschinellen Lernens können viele unterschiedliche Arten von Fragen bearbeiten. Auf hoher Ebene fallen die meisten Probleme in drei Familien, abhängig davon, was wir vorhersagen wollen: **Klassifikation** (eine Kategorie wählen), **Regression** (eine Zahl vorhersagen) und **Clustering** (natürliche Gruppen finden). Jede hat ein eigenes Ziel, einen eigenen Output und eine eigene Art der Auswertung.

3.1.1 Klassifikation

Bei einem Klassifikationsproblem lernt das Modell, jeden Fall einer von wenigen vordefinierten Kategorien oder Labels zuzuordnen. Typische Fragen sind „Ist diese Transaktion betrügerisch oder nicht?“, „Hat dieser Patient Krankheit A, B oder C?“ oder „Wird dieser Kredit bewilligt oder abgelehnt?“. Der Output ist ein *Label*, also eine diskrete Wahl zwischen möglichen Klassen. Oft gibt das Modell statt einer einzelnen Ja/Nein-Antwort auch eine *Wahrscheinlichkeit* oder *Konfidenzscore* aus. Ein E-Mail-Filter könnte zum Beispiel sagen: „80% Wahrscheinlichkeit, dass diese Nachricht Spam ist.“ Menschen oder nachgelagerte Systeme können dann einen Schwellwert wählen, etwa Nachrichten nur zu markieren, wenn die Wahrscheinlichkeit über 90% liegt.

Definition – Klassifikation

Klassifikation ist eine Art des maschinellen Lernens, mit der Daten diskreten Kategorien zugewiesen werden, etwa um E-Mails als Spam oder Nicht-Spam zu klassifizieren.

Einige Beispiele für Klassifikationsaufgaben sind die folgenden.

- Erkennen von Spam- vs. Nicht-Spam-E-Mails auf Basis von Nachrichteninhalt und Absender.
- Genehmigung oder Ablehnung eines Kredits basierend auf der Historie des Antragstellers.
- Erkennen von Objekten in einem Bild (Katze, Hund, Auto oder Person).
- Vorhersage, ob ein Kunde abwandert (kündigt) oder bleibt.

Arten der Klassifikation

Nicht alle Klassifikationsprobleme sehen gleich aus. Nachfolgend sind gängige Varianten aufgeführt.

Definition – Arten der Klassifikation

Binäre Klassifikation wird verwendet, wenn es nur zwei mögliche Kategorien gibt, z. B. die Entscheidung, ob eine E-Mail Spam ist oder nicht.

Multiklassen-Klassifikation wird verwendet, wenn es mehr als zwei mögliche Kategorien gibt, z. B. das Erkennen, ob ein Bild eine Katze, einen Hund oder ein Pferd zeigt.

Multi-Label-Klassifikation wird verwendet, wenn ein Element gleichzeitig zu mehreren Kategorien gehören kann, z. B. ein Foto sowohl mit „Strand“ als auch mit „Sonnenuntergang“ zu taggen.

Binäre Klassifikation

In der binären Klassifikation, wie der Name schon sagt, gibt es zwei mögliche Kategorien.

- Beispiele: Betrug vs. kein Betrug; Abwanderung vs. Verbleib; Spam vs. Nicht-Spam.

Beispiel – Binäre Klassifikation

Binäre Klassifikation bedeutet, einem Computer beizubringen, zwischen zwei möglichen Ergebnissen zu entscheiden. Zum Beispiel könnte ein Programm lernen, anhand vieler Beispiele zu unterscheiden, ob eine E-Mail „Spam“ oder „Nicht Spam“ ist. Nach dem Lernen aus diesen Beispielen kann das System eine neue E-Mail betrachten und vorhersagen, zu welcher Kategorie sie gehört. Ähnliche Methoden werden in der Medizin (krank oder gesund), im Bankwesen (Betrug oder kein Betrug) und

vielen anderen Bereichen eingesetzt.

Multiklassen-Klassifikation

Bei der Multiklassen-Klassifikation gibt es drei oder mehr *wechselseitig ausschließende* Kategorien. Wichtig ist, dass jeder Fall genau zu einer von ihnen gehört.

- Beispiele: Produktkategorie wählen (`Elektronik / Kleidung / Lebensmittel`); Krankheit A/B/C; Bild ist `Katze / Hund / Auto / Person`.

Beispiel – Multiklassen-Klassifikation

Multiklassen-Klassifikation bedeutet, einem Computer beizubringen, zwischen mehr als zwei möglichen Kategorien zu wählen. Zum Beispiel könnte ein Bildverarbeitungsprogramm lernen, ob ein Bild eine Katze, einen Hund oder ein Pferd zeigt. Nach dem Training mit vielen gelabelten Beispielen kann das System ein neues Bild ansehen und entscheiden, welche Kategorie am besten passt. Dieser Ansatz wird häufig in Anwendungen wie Handschrifterkennung, Produktsortierung und medizinischer Bildanalyse verwendet.

Multi-Label-Klassifikation

Bei der Multi-Label-Klassifikation kann ein einzelner Input zu *mehreren* Kategorien gleichzeitig gehören.

- Beispiele: ein Foto getaggt mit {`Strand`, `Sonnenuntergang`, `Menschen`}; eine E-Mail sowohl mit `Abrechnung` als auch `dringend` markiert; ein Film getaggt mit {`Komödie`, `Romanze`}.

Beispiel – Multi-Label-Klassifikation

Multi-Label-Klassifikation bedeutet, dass jedes Element gleichzeitig zu mehreren Kategorien gehören kann. Zum Beispiel könnte ein Strandfoto sowohl mit „Wasser“ als auch mit „Sonnenuntergang“ gekennzeichnet werden, oder ein medizinischer Bericht könnte mehrere Zustände für einen Patienten angeben. Der Computer lernt aus vielen Beispielen, wie verschiedene Labels gemeinsam auftreten können, und sagt dann für einen neuen Fall alle relevanten Labels voraus. Diese Methode wird häufig in Bereichen wie Musikempfehlung, Bild-Tagging und medizinischer Diagnose verwendet.

Ordinale Klassifikation

Kategorien haben eine natürliche Reihenfolge, sind aber dennoch Labels (keine Zahlen, die man sinnvoll addiert/subtrahiert).

- Beispiele: Kundenzufriedenheit {`niedrig`, `mittel`, `hoch`}; Kreditrating-Bänder; Krankheitsstadien.

Beispiel – Ordinale Klassifikation

Ordinale Klassifikation wird verwendet, wenn die möglichen Kategorien eine natürliche Ordnung haben, der genaue Abstand zwischen ihnen aber nicht bekannt ist. Zum Beispiel könnte ein System lernen, die Kundenzufriedenheit als „schwach“, „durchschnittlich“, „gut“ oder „ausgezeichnet“ vorherzusagen. Diese Kategorien folgen einer klaren Reihenfolge von niedrig nach hoch, aber der Unterschied zwischen „durchschnittlich“ und „gut“ ist nicht notwendigerweise derselbe wie zwischen „gut“ und „ausgezeichnet“. Solche Modelle sind in Umfragen, medizinischen Bewertungen und Risikoabschätzungen verbreitet.

One-vs-Rest (OvR) und One-vs-One (OvO) (Optional)

Manchmal ist es nützlich, Multiklassenprobleme mit einfacheren binären Teilen anzugehen.

- **OvR (One-vs-Rest):** Für jede Klasse einen kleinen Klassifikator trainieren (z. B. „ist es `Elektronik` vs. nicht?“). Bei der Vorhersage die Klasse mit dem stärksten Score wählen. *Analogie:* Jede Klasse hat ihren eigenen „Anwalt“; der lauteste Anwalt gewinnt.
- **OvO (One-vs-One):** Einen Klassifikator für jedes Klassenpaar trainieren (Elektronik vs. Kleidung, Elektronik vs. Lebensmittel, Kleidung vs. Lebensmittel, ...). *Analogie:* Ein Mini-Turnier; die Klasse mit den meisten Paar-„Siegen“ wird gewählt.
- Wann verwenden: OvR ist einfach und skaliert gut; OvO kann gut funktionieren, wenn Klassen insgesamt schwer zu trennen sind, paarweise aber leicht.

In Tabelle 3.1 finden Sie eine knappe Übersicht.

3.1.2 Regression

Bei der Regression ist der Output keine Kategorie, sondern eine kontinuierliche Zahl, die jeden Wert innerhalb eines Bereichs annehmen kann. Diese Modelle versuchen zu schätzen, *wie viel*, *wie lange* oder *welchen Wert*. Anstatt zwischen Optionen zu entscheiden, besteht das Ziel darin, eine numerische Vorhersage zu treffen, die der wahren Antwort so nahe wie möglich kommt.

Typ	Bedeutung	Beispiel
Binär	Genau zwei Kategorien; wähle eine.	Spam vs. Nicht-Spam; Abwanderung vs. Verbleib.
Multiklassen	Eine von vielen Kategorien (wechselseitig exklusiv).	Produktkategorie: Elektronik/-Kleidung/Lebensmittel.
Multi-Label	Viele Labels können gleichzeitig gelten.	Fototags: Strand + Sonnenuntergang + Menschen.
Ordinal	Geordnete Kategorien (Labels mit Rang).	Zufriedenheit: niedrig/mittel/hoch.
One-vs-Rest	Multiklassen über „eine Klasse vs. alle anderen“ Min-Modelle.	Separates „ist es Elektronik?“-Modell usw.
One-vs-One	Multiklassen über „paarweise Duelle“ zwischen Klassen.	Elektronik vs. Kleidung; Gesamtsieger wird gewählt.

Tabelle 3.1: Gängige Klassifikations-Setups und Alltagsbeispiele.

Definition – Regression

Regression ist eine Art des maschinellen Lernens zur Vorhersage kontinuierlicher Werte, wie Temperatur, Preis oder Alter.

Einige Beispiele für Regressionsaufgaben sind die folgenden.

- Prognose der Verkäufe der nächsten Woche für jede Filiale.
- Vorhersage der Lieferzeit basierend auf Entfernung und Verkehr.
- Schätzung von Hauspreisen aus Größe, Lage und Anzahl der Zimmer.
- Vorhersage des Blutzuckerspiegels eines Patienten in einer Stunde.

Example - Regression

Regression wird eingesetzt, wenn wir möchten, dass ein Computer eine Zahl statt einer Kategorie vorhersagt. Zum Beispiel kann ein Modell den Preis eines Hauses basierend auf Größe, Lage und Anzahl der Zimmer schätzen lernen. Nach dem Lernen aus vielen Beispielen kann es für ein neues Haus eine numerische Vorhersage liefern. Regression wird in Bereichen wie Wirtschaft, Ingenieurwesen und Gesundheitswesen breit genutzt, um Werte zu prognostizieren oder Trends zu messen.

3.1.3 Clustering (keine Labels, Gruppen entdecken)

Clustering ist anders: Es gibt keine vordefinierten Labels oder Ergebnisse, die vorhergesagt werden sollen. Stattdessen besteht das Ziel darin, Struktur in den Daten zu finden und Elemente zu gruppieren, die sich ähnlich verhalten oder

aussehen. Das Modell versucht, die Daten in eine kleine Zahl bedeutungsvoller Cluster zu organisieren, sodass Elemente innerhalb eines Clusters einander ähnlicher sind als Elemente in unterschiedlichen Clustern.

Definition – Clustering

Clustering ist eine Form des unbeaufsichtigten Lernens, bei der ähnliche Datenpunkte zu Gruppen zusammengefasst werden, sodass Elemente in derselben Gruppe einander ähnlicher sind als solche in verschiedenen Gruppen.

Dies nennt man **Unsupervised Learning**, weil wir das Modell nicht mit richtigen Antworten unterrichten; es muss Muster selbst „entdecken“. Einige Beispiele für Clustering-Anwendungsfälle sind die folgenden.

- Kunden in wenige Segmente gruppieren, basierend auf Kaufverhalten und Demografie.
- Muster von Website-Besuchern mit ähnlichem Surfverhalten identifizieren.
- Ungewöhnliche Muster (Anomalien) erkennen, die in kein Cluster passen – möglicherweise Hinweise auf Fehler oder Betrug.
- Artikel oder Nachrichten in verwandte Themenbereiche organisieren, ohne vordefinierte Topics.

Tabelle 3.2 fasst zusammen, wie sich Klassifikation, Regression und Clustering darin unterscheiden, was sie vorhersagen, welche Eingaben sie nutzen und wie wir ihre Outputs interpretieren. Zu verstehen, mit welchem Problemtyp Sie es zu tun haben, ist der erste Schritt, bevor Sie ein Modell wählen oder Leistung bewerten. Wenn Sie gelabelte Daten mit bekannten Ergebnissen haben (Spam/Nicht-Spam, Preis etc.), ist die Aufgabe überwachtes Lernen, also Klassifikation oder Regression. Wenn Sie nur ungelabelte Daten haben und Struktur erkunden wollen, ist es unbeaufsichtigt, in diesem Fall Clustering. Diese drei Kategorien decken die meisten praktischen Fälle ab und bilden die Grundlage für fortgeschrittenere Ansätze.

3.2 Empfehlungssysteme

Viele Produkte schlagen (oder empfehlen) Elemente (Filme, Produkte, Nachrichten, Kurse) vor, indem sie für jeden Nutzer einen großen Katalog **ranken**. Empfehlungssysteme sagen voraus, mit welchen Elementen ein Nutzer sich wahrscheinlich als Nächstes beschäftigen wird (zum Beispiel kaufen – denken Sie an Amazon – oder ansehen – denken Sie an Netflix) und geben eine kurze, geordnete Liste zurück. Anders als bei Standard-Klassifikation oder -Regression ist der Output typischerweise eine *Top-k-Liste* oder ein Satz von Scores, mit denen Elemente sortiert werden.

Typ	Welche Frage wird beantwortet?	Typische Eingaben (Beispiele)	Outputs (Beispiele)
Klassifikation	Zu welcher Kategorie gehört dieses Element?	E-Mail-Text, Absender, Links; Patientenakte; Kundenprofil.	{Spam, Nicht-Spam}; {Genehmigen, Ablehnen}; Krankheit A/B/C; Klassenwahrscheinlichkeit.
Regression	Welcher numerische Wert ist zu erwarten?	Vergangene Verkäufe, Wetter, Werbeausgaben; Entfernung, Geschwindigkeit, Zeit.	Verkäufe nächste Woche = 500; Preis = \$350,000; Verzögerung = 12 Minuten.
Clustering	Welche natürlichen Gruppen oder Muster existieren?	Kundendemografie, Kaufhistorie, Webaktivität.	Drei Kundensegmente; fünf Verhaltensgruppen; Ausreißer zur Untersuchung.

Tabelle 3.2: Verschiedene Arten von ML-Aufgaben und ihre Outputs.

Definition – Empfehlungssysteme

Ein Empfehlungssystem schätzt die Nutzerpräferenz für Elemente und erzeugt eine nach Relevanz geordnete Shortlist (Top-k), zugeschnitten auf jeden Nutzer. Es lernt aus historischen Interaktionen (z. B. Klicks, Ansichten, Käufe, Bewertungen) und optionalen Zusatzinformationen (Item-Metadaten, Nutzerattribute, Kontext wie Zeit oder Gerät).

Beispiel – Amazon und Netflix

Amazon (Shopping). Stellen Sie sich vor, Sie schauen sich bei Amazon einen Kaffeemühlen-Artikel an. Unter dem Produkt sehen Sie Vorschläge wie „Häufig zusammen gekauft“ (Mühle + Filter + Bohnen) oder „Kunden, die sich das angesehen haben, sahen auch“ (andere Mühlen zu ähnlichen Preisen). Das Empfehlungssystem von Amazon tut zwei Dinge gleichzeitig: Es bemerkt, was *Sie* sich zuvor angesehen oder gekauft haben (zum Beispiel Küchengeräte), und es lernt aus Mustern von *Millionen* anderer Käufer. Wenn viele Menschen, die diese Mühle gekauft haben, auch Filter und Bohnen gekauft haben, schlägt es Ihnen diese ebenfalls vor. Außerdem mischt es einige neue oder beliebte Produkte bei, damit Sie nicht immer nur denselben Typ Artikel sehen.

Netflix (Streaming). Wenn Sie Netflix öffnen, sind Zeilen wie „Top-Empfehlungen für Sie“ oder „Weil Sie … geschaut haben“ speziell auf Ihre Sehgewohnheiten angeordnet. Wenn Sie abends häufig leichte Komödien zu Ende schauen und am Wochenende Krimiserien, merkt Netflix sich das. Es betrachtet, was Sie gesehen haben, was Sie abgebrochen haben und was Personen mit ähnlichem Geschmack mochten. Daraus baut es eine Shortlist von Titeln, auf die Sie am ehesten klicken – vielleicht eine neue Komödie, einen Klassiker, den Sie verpasst haben, und einen Film mit Ihrem Lieblingsschauspieler. Zudem versucht es, die Vielfalt zu wahren, damit Sie nicht zwölf sehr ähnliche Shows in Folge sehen; die Empfehlungen bleiben so frisch und ansprechend.

3.3 Modellvalidierung

Ein Modell, das auf den Daten, mit denen es trainiert wurde, gut abschneidet, funktioniert nicht automatisch gut auf neuen, ungesehenen Daten. **Modellvalidierung** ist der Prozess zu prüfen, ob das, was das Modell gelernt hat, wirklich über die Trainingsbeispiele hinaus generalisiert. Sie beantwortet die Frage: „Wird dieses Modell morgen, auf zukünftigen oder ungesehenen Daten, immer noch gut funktionieren?“ Ohne korrekte Validierung kann ein Modell auf dem Papier hervorragend aussehen, in der Praxis aber scheitern – ein Problem, das als **Overfitting**[1] bekannt ist, bei dem das Modell Details der Trainingsdaten auswendig lernt, statt allgemeine Muster. Modellvalidierung ist grundlegend, da sie eine ehrliche Schätzung liefert, wie sich das Modell im realen Einsatz verhalten wird. Außerdem verhindert sie überoptimistische Ergebnisse, die entstehen, wenn auf denselben Daten getestet wird, die auch zum Training verwendet wurden.

Beispiel — Warum Validierung wichtig ist

Ein Händler trainiert ein Modell, das wahrscheinliche Rücksendungen markiert, damit Mitarbeitende diese Bestellungen vor dem Versand noch einmal prüfen. Auf den Daten des letzten Monats wirkt das Modell *fantastisch*: Es „erwischt“ fast jede Rücksendung.
Was lief schief? Der letzte Monat war die *Feiertagssaison*: viele Geschenk-Käufe, viele schnelle Rückgaben und mehrere kurzfristige Aktionen. Das Modell hat stillschweigend diese saisonalen Besonderheiten gelernt (z. B. Geschenkverpackung, Expressversand) statt allgemeiner Muster. Wenn der Januar kommt, ändert sich das Verhalten – weniger Geschenke, andere Artikel – und das Modell verpasst plötzlich viele echte Rücksendungen und schlägt bei normalen Bestellungen falschen Alarm.

[1] Overfitting ist nur ein Aspekt eines Modells, das sich auf neuen Daten schlecht verhält, aber der häufigste.

Es sah in der Vergangenheit großartig aus, scheitert aber im nächsten Monat. Das ist Overfitting.
Was einfache Validierung gezeigt hätte. Hätten wir die *Zukunft simuliert*, indem wir auf früheren Wochen trainiert und auf späteren Wochen *validiert* hätten (zeitbasierter Split), hätten wir den Leistungsabfall außerhalb der Feiertagsperiode gesehen. Eine kleine *abschließende Testmenge*, die bis zum Schluss völlig unberührt bleibt, hätte den Real-World-Score vor dem Einsatz bestätigt.

3.3.1 Der Hold-out-Ansatz

Die gebräuchlichste und praktische Methode für Einsteiger ist der **Hold-out-Ansatz**. Hier teilen wir unsere verfügbaren Daten vor Trainingsbeginn in getrennte Teile:

- eine **Trainingsmenge**, auf der das Modell Muster lernt;
- eine **Validierungsmenge**, mit der Modellentscheidungen getestet und abgestimmt werden;
- und manchmal eine abschließende **Testmenge**, die bis ganz zum Schluss beiseitegelegt wird.

Beispiel — Warum zwei Datensätze beim Tuning nicht reichen

Angenommen, Sie teilen Ihre Daten nur in zwei Teile: *Train* (um dem Modell etwas beizubringen) und *Test* (um zu prüfen, wie gut es ist). Sie beginnen, die *Einstellungen* (Parameter) des Modells anzupassen, um einen besseren Score zu erhalten. Nach jeder Änderung schauen Sie auf das *Test*-Ergebnis, ob es sich verbessert hat. Nach vielen Runden sieht das Modell auf diesem Train/Test-Paar großartig aus. Was passieren kann: Durch das wiederholte Prüfen des Test-Scores während des Tuning haben Sie *unbewusst auf den Test trainiert*. Die Modelleinstellungen beginnen, die *spezifischen Eigenheiten* genau dieser Testmenge zu passen. Wenn Sie das „finale“ Modell dann auf einem *neuen* Datensatz (einer frischen Validierung) ausprobieren, sinkt die Leistung – mitunter stark. Es war nicht wirklich gut; es war nur auf *diesen* einen Test gut.

Stellen Sie sich vor, Sie perfektionieren eine Suppenrezeptur und lassen nach jeder Änderung denselben *Freund* (den Testdatensatz) probieren. Am Ende passen Sie die Suppe an genau diesen Geschmack an. Wenn Sie sie einer neuen Gruppe (dem Validierungsdatensatz) servieren, mögen viele sie nicht. Sie haben keine allgemein schmackhafte Suppe gekocht, sondern eine Suppe für einen einzigen Verkoster (Ihren Freund).
So gehen Sie stattdessen vor:

- **Train:** Hier lernt das Modell Muster.

- **Test:** Einstellungen werden *nur* anhand dieser Menge angepasst (kein Blick auf die Testmenge).
- **Validierung:** Bis zum Ende unberührt; einmalig genutzt, um die *wahre* finale Leistung zu schätzen.

Wenn Sie Modellparameter tunen, reichen zwei Mengen nicht; Sie könnten unbemerkt auf den Test überfitten. Nutzen Sie Train zum Lernen, Test zum Tuning und Validierung einmal am Ende, um zu prüfen, ob es in der realen Welt funktioniert.

Beispiel – Kreditanträge

Stellen Sie sich vor, wir haben 10,000 historische Kreditanträge mit Ergebnissen („genehmigt“ oder „verspätet zurückgezahlt“). Wir könnten sie folgendermaßen aufteilen:

- 70% (7,000 Fälle) für das Training,
- 15% (1,500 Fälle) für den Test während der Entwicklung,
- 15% (1,500 Fälle) für die finale Validierung.

Das Modell wird nur auf den Trainingsdaten trainiert. Nach jeder Trainingsrunde wird es auf den Testdaten ausgewertet, um zu sehen, wie gut es generalisiert. Sobald das Design feststeht, wird es ein letztes Mal auf der unberührten Validierungsmenge getestet, um die Leistung in der Praxis abzuschätzen.

Diese einfache Methode, auch **Train/Validierung/Test-Split** genannt, ist die Grundlage vertrauenswürdiger Modellauswertung. Die wichtigste Regel lautet: Das Modell sollte die Daten, die für die finale Bewertung verwendet werden, niemals „sehen“.

Manchmal findet man in der Literatur die Bezeichnungen *Test* und *Validierung* vertauscht.

Übung (8 Min)

Übung 3.10
Sie haben 12,000 Kundendatensätze, um Abwanderung (gehen/bleiben) vorherzusagen. Schlagen Sie einen Hold-out-Split vor (Prozentsätze oder Anzahlen) und erklären Sie in einem Satz, warum Sie diese Anteile gewählt haben.

3.3.2 (Optional) Fortgeschrittene Validierungsmethoden

Für größere oder kritischere Projekte können wir die Zuverlässigkeit verbessern, indem wir die Validierung mehrfach wiederholen. Zwei gängige Ansätze sind wissenswert.

k-fold-Cross-Validation. Die Daten werden in k gleich große Teile geteilt (z. B. $k = 5$). Wir trainieren das Modell k-mal, wobei jedes Mal ein anderer Teil zum Testen und der Rest zum Trainieren genutzt wird. Die durchschnittliche Leistung über die k Durchläufe liefert eine stabilere Schätzung, weil jeder Datenpunkt einmal zum Testen verwendet wurde. Man kann es sich als mehrere „rotierende Hold-out"-Stichproben vorstellen, um zu sehen, wie konsistent das Modell wirklich ist.

Monte-Carlo (wiederholte Zufallssplits). Statt fester Folds erstellen wir wiederholt zufällige Train/Test-Splits (z. B. 80%/20%) und mitteln die Ergebnisse. Das hilft zu erkennen, ob sich die Modellleistung stark ändert, je nachdem, wie die Daten gesplittet werden – nützlich bei kleinen oder verrauschten Datensätzen.

Beide Methoden sind rechenintensiver, folgen aber demselben Prinzip: *Man hält Daten zurück, testet wiederholt und vertraut nur Ergebnissen, die über Splits hinweg stabil bleiben.*

3.4 Overfitting, Underfitting und Baselines

Modelle können auf viele Arten scheitern, aber zwei sind mit Abstand die häufigsten: Over- und Underfitting. **Overfitting** passiert, wenn ein Modell Eigenheiten des Trainingssets (Rauschen, Zufälligkeiten, Formatartefakte) lernt statt des zugrunde liegenden Musters und deshalb auf vergangenen Daten großartig aussieht, aber *auf neuen Daten scheitert.* **Underfitting** passiert, wenn ein Modell zu einfach oder schlecht spezifiziert ist, um echte Struktur zu erfassen, und deshalb sowohl auf Trainings- als auch auf neuen Daten schlecht abschneidet.

Denken Sie an das Lernen für eine Prüfung.

- **Overfitting** ist, die Fragen vom letzten Jahr Wort für Wort auswendig zu lernen; die alte Klausur gelingt, die neue nicht.
- **Underfitting** ist, nur Überschriften zu überfliegen; man hat nie genug gelernt, um irgendeine Klausur gut zu bestehen.

In der Praxis kann ein überangepasstes Modell in der Produktion viele Fehlalarme auslösen oder wichtige Fälle verpassen, wenn sich die Umgebung ändert. Ein unterangepasstes Modell wird selten nützlich sein: Es fehlt die Nuance, um Entscheidungen besser zu unterstützen als eine einfache Faustregel.

Definition — Overfitting und Underfitting

Overfitting tritt auf, wenn ein Modell die Trainingsbeispiele *zu präzise* lernt, einschließlich ihres Rauschens und zufälliger Details. Es schneidet auf bereits gesehenen Daten extrem gut ab, versagt aber auf neuen Daten, weil es die Vergangenheit auswendig gelernt hat, statt allgemeine Muster.

Underfitting tritt auf, wenn ein Modell *zu einfach* oder nicht flexibel genug ist, um echte Muster in den Daten zu erfassen. Es schneidet sowohl auf Trainingsdaten als auch auf neuen Daten schlecht ab, weil es nicht genug gelernt hat.

3.4.1 Modelldiagnostik

Oft kann man Over- oder Underfitting erkennen, indem man einfache Muster im Verhalten des Modells auf Trainings- und Validierungsdaten betrachtet. Auch ohne fortgeschrittene Tools geben diese Signale ein intuitives Gefühl dafür, ob das Modell echte Struktur gelernt hat oder nur Gesehenes auswendig lernt.

Der erste und direkteste Hinweis kommt aus dem Vergleich der Modellleistung auf der **Trainingsmenge** (den Daten, aus denen es gelernt hat) mit seiner Leistung auf der **Validierungs- oder Testmenge** (Daten, die es nie zuvor gesehen hat). Wenn das Modell auf den Trainingsdaten extrem gut, auf den Validierungsdaten aber deutlich schlechter ist, bedeutet diese „Lücke“ meist **Overfitting**: Es hat spezifische Details, Rauschen oder Zufälligkeiten aus den Trainingsbeispielen gelernt, die nicht generalisieren. Beispielsweise könnte ein Kreditgenehmigungsmodell lernen, dass Antragsteller aus einer bestimmten Postleitzahl im Training immer zurückgezahlt haben, was für neue Antragsteller nicht gilt. Wenn hingegen sowohl Trainings- als auch Validierungsergebnisse schlecht sind, liegt wahrscheinlich **Underfitting** vor: Das Modell hat das wahre Signal nie erfasst und ist schlicht zu begrenzt, um sinnvolle Muster zu lernen. Zum Beispiel ignoriert die Verwendung nur des durchschnittlichen Umsatzes über alle Filialen zur Prognose der Nachfrage in der nächsten Woche Saisonalität und lokale Unterschiede – zu grob, um irgendwo nützlich zu sein.

Eine zweite Diagnosemöglichkeit sind **Lernkurven**, die darstellen, wie sich der Fehler des Modells verändert, wenn mehr Trainingsdaten hinzukommen. Sinkt der Validierungsfehler konzeptionell weiter, wenn mehr Daten verfügbar werden, ist das Modell in der Lage zu lernen, hat aber einfach noch nicht genug Beispiele; mehr Daten würden wahrscheinlich helfen. Wenn jedoch sowohl Trainings- als auch Validierungsfehler schnell auf hohem Niveau abflachen und auch mit mehr Daten nicht besser werden, ist das Modell für die Aufgabe zu schwach: **Underfitting**. Im Gegensatz dazu ist es ein klares Zeichen für **Overfitting**, wenn der Trainingsfehler weiter besser wird, der Validierungsfehler jedoch schlechter oder stark schwankt. In der Praxis könnte das so aussehen, dass ein Marketing-Prognosemodell gut funktioniert, wenn es auf jüngste Kampagnen trainiert wird, aber scheitert, sobald es auf neue angewendet wird –

es hat kampagnenspezifische Eigenheiten auswendig gelernt statt allgemeiner Kaufmuster.

Eine weitere nützliche Prüfung ist, **Fehler über verschiedene Segmente oder Kohorten** hinweg anzuschauen. Wenn die Leistung zwischen Gruppen stark variiert – z. B. funktioniert das Modell in einem Krankenhaus sehr gut, in einem anderen schlecht oder in einer Region gut, in anderen nicht –, könnte es auf die dominante Gruppe in den Trainingsdaten überangepasst sein. Manchmal passiert das, wenn eine Gruppe im Datensatz überrepräsentiert ist oder wenn kleine Unterschiede in der Datenerhebung versteckte Verzerrungen einführen. Die Untersuchung von Ergebnissen nach Segment hilft, diese Schwächen aufzudecken und sicherzustellen, dass das Modell für alle vorgesehenen Nutzer zuverlässig ist.

Schließlich sollten Sie betrachten, wie das Modell über die Zeit performt, seine **Zeitstabilität**. Ein Modell, das letzten Monat gut war, diesen Monat aber plötzlich schlechter wird, ist wahrscheinlich auf vergangene Bedingungen überangepasst. Vielleicht hat es Beziehungen gelernt, die nicht mehr gelten, etwa Muster im früheren Kundenverhalten oder saisonale Preisregeln. Beispielsweise könnte eine Nachfrageprognose, die in stabilen Monaten trainiert wurde, scheitern, sobald neue Aktionen oder Störungen auftreten. Zeitstabilität zu überwachen hilft, solchen „Daten-Drift“ früh zu erkennen und erinnert daran, dass der Wert eines Modells nicht nur von der Genauigkeit heute abhängt, sondern auch davon, wie lange diese Genauigkeit anhält.

Zusammen bieten diese einfachen Diagnoseprüfungen – Vergleich von Train- und Validierungsresultaten, Beobachtung der Leistung bei mehr Daten, Prüfung der Konsistenz über Gruppen hinweg und Monitoring im Zeitverlauf – ein praktisches Handwerkszeug, um zu verstehen, ob ein Modell die richtigen Muster lernt oder lediglich die Vergangenheit auswendig lernt. In Tabelle 3.3 sehen Sie einen Vergleich von Overfitting, Underfitting und gesundem Modellverhalten.

3.4.2 Praktische Abhilfen

Wenn ein Modell nicht gut performt, liegt es oft daran, dass es entweder **überangepasst** (zu viel auswendig gelernt) oder **unterangepasst** (zu wenig gelernt) hat. Hier sind einfache Wege, jede Situation zu beheben.

- **Wenn das Modell overfittet:** Vereinfachen Sie es (zum Beispiel bei neuronalen Netzen ggf. ein kleineres Netz), damit es sich auf Hauptmuster konzentriert statt auf irrelevante Kleinigkeiten. Entfernen Sie unnötige Eingaben (z. B. zu viele Features), verkürzen Sie die Trainingszeit oder geben Sie ihm mehr frische Daten zum Lernen. Stellen Sie außerdem sicher, dass Sie auf Daten aus unterschiedlichen Zeiträumen oder Gruppen testen, damit es nicht nur ein Muster auswendig lernt.

- **Wenn das Modell underfittet:** Helfen Sie ihm, mehr zu lernen, indem Sie bessere Informationen hinzufügen (z. B. bei Wettervorhersage nicht nur Temperatur an einem Ort, sondern zusätzlich Wind, Niederschlag usw.)

	Symptome	Wahrscheinliche Ursachen	Schnelle Abhilfen
Overfitting	Train-Score exzellent; Validierung/Test deutlich schlechter; Leistung driftet in der Produktion schnell.	Zu komplex für die Daten; Leakage; fehlerhafte Labels; über-getunt auf kleiner Validierung.	Modell vereinfachen; Leakage entfernen; regularisieren/early stop; Daten hinzufügen; zeit-/gruppenbewusste Splits nutzen.
Underfitting	Sowohl Train als auch Validierung schlecht; Fehler wirken „grob“ über alle Fälle.	Modell zu einfach; fehlende Eingaben; schlechte Encodings; zu starke Glättung.	Aussagekräftige Features ergänzen; bessere Encodings; mehr Flexibilität erlauben; Regularisierung abstimmen.
Gesund	Train und Validierung nah beieinander; stabil über Kohorten und Zeit; Baseline klar geschlagen.	Ausgewogene Komplexität; gute Splits; passende Features.	In Produktion überwachen; bei Drift revalidieren; Baseline als Canary behalten.

Tabelle 3.3: Overfitting vs. Underfitting erkennen und beheben; eine Baseline als Anker behalten.

oder es etwas flexibler machen. Geben Sie klarere, nützlichere Eingaben oder erlauben Sie längeres Training (bei neuronalen Netzen entscheiden Sie, wie lange ein Netz lernt; länger ist oft besser), damit es echte Trends statt nur den groben Durchschnitt erfasst.

Beispiel — Overfitting beheben

Ein Unternehmen baut ein Modell, das vorhersagt, welche Kunden ihr Abo kündigen werden. Zunächst wirkt das Modell perfekt – es passt fast genau zu den Daten vom letzten Jahr. Doch bei neuen Kunden in diesem Jahr versagt es deutlich. Es hatte kleine Zufälligkeiten aus den alten Daten auswendig gelernt, zum Beispiel eine einmalige Marketingkampagne. Sie könnten das Modell wie folgt verbessern:

- Modell vereinfachen, damit es sich auf stabile Muster konzentriert (z. B. allgemeine Ausgabemuster statt spezifischer vergangener Daten).

- Eingaben entfernen, die nur für die Vergangenheit relevant sind, wie Rabattcodes vom letzten Jahr.
- Neuere Kundendaten hinzufügen, damit das Modell eine größere Verhaltensvielfalt sieht.
- Über unterschiedliche Zeiträume testen, um Generalisierbarkeit sicherzustellen.

Nach diesen Änderungen memorisiert das Modell die Historie nicht mehr – es lernt die echten Gründe für Abwanderung.

Beispiel — Underfitting beheben

Ein Lieferdienst möchte vorhersagen, wie lange jede Lieferung dauern wird. Er baut ein einfaches Modell mit nur einer Information: der Entfernung zwischen Restaurant und Kunde. Die Vorhersagen sind immer „durchschnittlich“ – manchmal zu schnell, manchmal zu langsam –, weil das Modell andere wichtige Hinweise ignoriert.
Sie könnten das Modell wie folgt verbessern:

- Mehr nützliche Informationen hinzufügen, etwa Tageszeit, Verkehrslage, Wetter und Fahrerauslastung.
- Dem Modell erlauben, etwas mehr Details zu lernen, statt es zu simpel zu halten.
- Ein flexibleres Modell in Betracht ziehen. Wenn Sie lineare Regression verwendet haben, reicht sie evtl. nicht aus, um Feinheiten Ihrer Daten zu erfassen.

Mit diesen Verbesserungen wird das Modell genauer, weil es endlich die realen Faktoren erfasst, die die Lieferzeit beeinflussen.

3.5 Feature Engineering

Jedes ML-Modell lernt Muster aus Informationen, die wir als Eingabe bereitstellen. Diese Eigenschaften der Eingaben nennt man **Features/Merkmale**. Es sind messbare Eigenschaften oder Hinweise, die dem Modell helfen, Vorhersagen zu treffen, wie das Alter einer Person, die Anzahl früherer Käufe, die Zeit seit einem Ereignis oder wie oft ein Wort in einem Text vorkommt. Gute Features enthalten Informationen, die wirklich mit der Entscheidung zusammenhängen, die wir unterstützen wollen, und sind vor allem *vor* dem Vorhersagemoment verfügbar.

Definition — Feature/Merkmal

Ein **Feature/Merkmal** ist ein Informationsbaustein, der ein bestimmtes Objekt oder Ereignis beschreibt, das Sie verstehen möchten. Es beschreibt etwas über jeden Fall in den Daten, z. B. das Alter eines Kunden, den Preis eines Produkts oder die Anzahl der Tage seit dem letzten Kauf. Features sind die Informationen, die das Modell für seine Vorhersagen verwendet.

3.5.1 Was ein gutes Feature ausmacht

Ein nützliches Feature hilft Modellen, Fälle zu unterscheiden. Zum Beispiel:

- Bei der Vorhersage der Kreditrückzahlung sind Einkommenshöhe, Kredit-Score und bestehende Schulden gute Features – sie beschreiben die finanzielle Leistungsfähigkeit.
- Alter oder Wohnort des Antragstellers können ebenfalls helfen, wenn sie mit dem Ausfallrisiko korrelieren.
- Die Haarfarbe aufzunehmen, hilft dem Modell dagegen nicht (wie man sich denken kann). Das ist ein nutzloses, schlechtes Feature.

Features können Zahlen (wie Einkommen), Kategorien (wie Beruf), Zeitdauern (wie Tage seit dem letzten Kauf) oder Flags (Ja/Nein-Werte) sein. Einige Features werden direkt gemessen, andere sind *abgeleitet* aus Rohdaten, z. B. Mittelwerte, Zählwerte oder Zeitdifferenzen.

Ein subtiles, aber sehr häufiges Problem in realen Projekten ist die Verwendung **uneinheitlicher oder unklarer Einheiten**. Daten werden oft aus verschiedenen Systemen, Teams oder Ländern gesammelt, die jeweils ihre eigenen Konventionen nutzen. Ein Datensatz kann Körpergröße in Zentimetern erfassen, ein anderer in Metern; eine Unternehmensniederlassung speichert Umsätze in Euro, eine andere in Dollar. Wenn diese Unterschiede vor dem Training nicht harmonisiert werden, interpretiert das Modell sie als echte Variationen statt als Skalierungsunterschiede. Das kann zu bizarren Ergebnissen führen, etwa dass das Modell denkt, eine 1,8 Meter große Person sei kleiner als jemand mit Angabe 180 (weil es annimmt, beide Zahlen seien in derselben Einheit). Ähnlich kann ein Finanzmodell Gewinne stark falsch prognostizieren, wenn einige Daten in Tausend Euro und andere in Einzel-Euro vorliegen. Modelle können nicht „raten“, welches Maßsystem korrekt ist; sie nehmen Zahlen einfach beim Wort.

Ein weiterer häufiger Fehler ist, **zu viele schwache oder redundante Features** aufzunehmen. Es ist verlockend zu glauben, dass mehr Spalten ein Modell automatisch klüger machen, doch das ist selten der Fall. Features, die wenig einzigartige Information liefern oder stark mit anderen überlappen, können die Leistung sogar verschlechtern, indem sie Rauschen und Ablenkung einbringen. Wenn wir zum Beispiel bereits „Gesamtausgaben im letzten Monat“ und „Anzahl der Käufe im letzten Monat“ haben, bringt „durchschnittlicher Betrag

pro Kauf“ keine neue Einsicht – es ist nur eine rechnerische Kombination der ersten beiden. Ebenso kann die Aufnahme Hunderter kleiner, schlecht verstandener Indikatoren (wie Dutzende Social-Media-Metriken) das Modell empfindlich gegenüber Zufallsschwankungen statt sinnvollen Trends machen. Praktisch kann ein mit redundanten Features überladenes Modell langsamer, schwerer interpretierbar und eher zum Overfitting neigend werden. Eine sinnvolle Faustregel ist, einfach zu beginnen: Nutzen Sie einen kleinen Satz an Features, die für das Problem intuitiv plausibel sind und die Sie einem Laien klar erklären können. Fügen Sie nur dann neue hinzu, wenn Sie begründen können, warum sie helfen könnten. Gute Modelle basieren auf klaren, relevanten Signalen, nicht auf einem Berg kaum zusammenhängender Zahlen.

Anschauliches Beispiel

Beispiel – Anschauliches Beispiel

Stellen Sie sich ein Modell vor, das vorhersagt, wie lange eine Essenslieferung dauern wird. Nützliche Features könnten die Entfernung zum Kunden, Tageszeit, Wochentag und die Anzahl aktiver Fahrer in der Nähe sein. Die Verpackungsart als Feature zu verwenden, wäre jedoch nutzlos. Ein sauberes, realistisches Feature-Set konzentriert sich nur auf das, was *vor* Beginn der Lieferung bekannt sein kann.

3.5.2 Warum Feature Engineering wichtig ist

Feature Engineering ist der Bereich, in dem ein Großteil der eigentlichen Expertise im maschinellen Lernen liegt. Selbst einfache Modelle können überraschend gut abschneiden, wenn die richtigen Features gewählt und bereinigt werden. Umgekehrt scheitern auch fortgeschrittene Algorithmen, wenn die Eingabedaten verrauscht, uneinheitlich oder unrealistisch sind. In den meisten realen Projekten hat die investierte Zeit in das Verstehen, Auswählen und Verbessern von Features mehr Einfluss als der Wechsel zu einem komplexeren Algorithmus. Kurz: *Bessere Daten schlagen schickere Modelle.*

Übung (10–12 Min): Finde das Problem

Übung 3.11
Sie sollen ein Modell bauen, das vorhersagt, ob ein Patient innerhalb von 30 Tagen nach Entlassung die Notaufnahme aufsucht. Vorgeschlagene Features sind: Alter, Diagnosen, durchschnittliche Tagestemperatur in der nächsten Woche, Anzahl früherer Besuche und Entlassungsnotizen. Welche Features sind gut oder falsch und warum?

3.6 Grenzen und Risiken von No-Code-Tools

In den letzten Jahren sind **No-Code**- und **AutoML**-Plattformen zunehmend populär geworden. Sie versprechen, dass jeder über eine grafische Oberfläche ML-Modelle bauen kann – ohne Code zu schreiben – und viele technische Aufgaben wie Feature-Auswahl, Modelltraining und Auswertung zu automatisieren. Diese Tools können für schnelle Experimente oder Teams ohne Programmierkenntnisse äußerst nützlich sein, bringen jedoch ernsthafte Einschränkungen mit sich, die Nicht-Spezialisten in die Irre führen und verborgene Risiken im realen Einsatz schaffen können.

Was No-Code-Tools sind

No-Code-Tools sind visuelle Umgebungen, in denen Nutzer Daten per Drag-and-Drop laden, ein Ziel wählen (z. B. „Umsatz nächsten Monat vorhersagen“) und das System automatisch mehrere Modelle im Hintergrund trainieren und testen lassen. Sie übernehmen typischerweise Aufgaben wie Datenbereinigung, Feature Engineering, Modellvergleich und Performance-Reporting, ohne dass eine einzige Zeile Code nötig ist.

Beispiele umfassen:

- **Google Cloud AutoML** – baut Bild-, Text- oder Tabellendaten-Modelle automatisch aus hochgeladenen Daten.
- **Microsoft Azure ML Designer** – bietet eine Drag-and-Drop-Oberfläche zum Verbinden von Datenquellen, Trainieren von Modellen und Deployen.
- **DataRobot** oder **H2O.ai Driverless AI** – liefern automatisierte Pipelines für Business-User mit Dashboards und Leistungs-Leaderboards.
- **RapidMiner** oder **KNIME** – ältere, aber weiterhin verbreitete visuelle Workflow-Tools für ML und Analytics.

Im Kern versuchen diese Plattformen, manuelles Codieren und Konfigurieren durch geführte Automatisierung zu ersetzen. Ein Nutzer lädt Daten hoch, wählt eine Zielvariable, und das System wählt automatisch Algorithmen, stimmt Parameter ab und gibt Metriken wie Accuracy oder Recall aus. Das kann magisch wirken und spart in der Tat enorm Zeit für Prototyping oder frühe Exploration. Diese scheinbare Einfachheit kann jedoch leicht tiefe Komplexität verbergen und zu Überconfidence in Ergebnisse führen, die nicht korrekt validiert oder interpretiert wurden.

Übersimplifizierung komplexer Probleme

No-Code-Systeme lassen ML oft einfacher erscheinen, als es ist. Mit wenigen Klicks erhält man ein buntes Dashboard mit hoher Accuracy oder ein bereit zum Deployen scheinendes Modell – unsichtbar bleibt jedoch, ob die Daten geeignet,

balanciert oder zeitlich konsistent waren. So könnte ein Krankenhausadministrator Patientendaten hochladen, um Wiedereinweisungen vorherzusagen, und beeindruckende Ergebnisse erhalten, ohne zu merken, dass das Modell versehentlich Zukunftsinformationen genutzt hat (z. B. Medikamente, die *nach* der Entlassung verschrieben wurden). Das System sähe genau aus, würde aber im Live-Einsatz völlig scheitern. Indem Schritte wie Feature-Auswahl, Leakage-Checks und Validierungsstrategie verborgen werden, können No-Code-Tools falsches Vertrauen und schlecht generalisierende Modelle erzeugen.

Mangel an Transparenz und Kontrolle

Ein weiteres ernstes Problem: Nutzer können oft nicht leicht einsehen, was das Tool intern getan hat. Die meisten Plattformen legen nicht offen, welche Features verwendet wurden, wie mit fehlenden Werten umgegangen wurde oder welche Algorithmen ausprobiert und warum ausgewählt wurden. Dieser „Black-Box"-Ansatz erschwert das Verstehen oder Erklären der Ergebnisse. Wenn sich ein Modell seltsam verhält, z. B. Kreditbewerber aus einer Region deutlich häufiger ablehnt, fehlt Nutzern der Weg zurück zur Ursache. Dieser Mangel an Transparenz verhindert effektives Debugging, macht Reproduzierbarkeit unmöglich und untergräbt Vertrauen – besonders in sensiblen Bereichen wie Gesundheitswesen oder öffentlicher Verwaltung.

Vendor Lock-in und Nachhaltigkeit

No-Code-Systeme sind meist proprietär. Ist ein Workflow oder Modell einmal in ihrer Plattform erstellt, lässt er sich oft nicht leicht exportieren oder anderswo wiederverwenden. Das bedeutet, dass die Organisation von diesem Anbieter abhängig wird – für Zugang, Updates und Wartung. Wenn das Unternehmen die Preise ändert, einen Dienst einstellt oder den Zugriff beschränkt, kann kritische Logik verloren gehen. In der Praxis kann das „technische Schulden" erzeugen: Systeme, die nur funktionieren, solange ein spezifisches Produkt verfügbar ist. Beispielsweise könnte eine Stadtverwaltung, die ein Wohnungsvergabemodell in einem geschlossenen Cloud-Tool baut, später feststellen, dass sie das Modell ohne Neuaufbau nicht zu einem anderen Anbieter migrieren kann.

Ethische Risiken und Verantwortlichkeit

Vielleicht am wichtigsten ist das Ethische. Da diese Systeme Modellentscheidungen automatisieren, merken Nutzer womöglich nicht, wenn sie auf verzerrten, unvollständigen oder sensiblen Daten trainieren. Ein No-Code-HR-Screening-Tool, das auf historischen Einstellungsdaten trainiert wurde, könnte bestehende Geschlechter- oder ethnische Ungleichgewichte reproduzieren, selbst wenn die Oberfläche neutral wirkt. Ähnlich könnte ein Kreditrisikomodell Menschen aus bestimmten Postleitzahlen benachteiligen, wenn das Tool sozioökonomische Verzerrungen nicht korrigiert. Auch wenn das Modell automatisch gebaut wurde,

bleiben Organisationen rechtlich und moralisch für die Ergebnisse verantwortlich. No-Code-Plattformen nehmen keine Verantwortung ab; sie machen es nur leichter, das menschliche Urteil zu übersehen, das automatisierte Entscheidungen begleiten muss.

Wann und wie No-Code-Tools sicher nutzen

No-Code-Systeme sollten nach Möglichkeit vermieden werden. Für Hochrisiko- oder regulierte Entscheidungen sollten sie ohne Expertenreview nicht eingesetzt werden. ML umfasst statistisches Denken, ethisches Urteil und Domänenexpertise – nichts davon lässt sich automatisieren. Menschliche Aufsicht stellt sicher, dass Modelle fair, transparent und robust sind und ihrem vorgesehenen Zweck verantwortungsvoll dienen.

3.7 Evaluationsmetriken in der Praxis

Modelle machen auf unterschiedliche Weise Fehler. Ziel der Auswertung ist nicht, eine perfekte Zahl zu finden, sondern zu verstehen, *wie* das Modell scheitert und ob diese Fehler für die jeweilige Entscheidung akzeptabel sind. Jede Metrik ist eine Art, Fehler zu zählen. Wählen Sie Metriken, die zu den Kosten des Fehlers in Ihrem Kontext passen – etwas Wichtiges zu verpassen vs. zu viele Fehlalarme auszulösen. In Abschnitt 3.9 finden Sie als Referenz die Formeln für die in ML am häufigsten verwendeten Metriken.

3.7.1 Für Ja/Nein- oder Kategorien-Entscheidungen (Klassifikation)

Wenn Sie etwas klassifizieren (z. B. Bilder von Katzen und Hunden oder E-Mails als Spam oder Nicht-Spam), möchten Sie im Allgemeinen wissen, wie viele Fälle richtig und wie viele falsch sind. Wahrscheinlich die am häufigsten verwendeten Metriken zur Messung, wie gut (oder schlecht) ein Modell ist, sind die folgenden vier. In den nächsten Abschnitten besprechen wir jede im Detail.

- **Accuracy (Genauigkeit)** – der Anteil korrekter Antworten insgesamt. Leicht zu verstehen, kann aber irreführend sein, wenn ein Ergebnis sehr häufig ist (z. B. allen „kein Betrug“ zuzuordnen ergibt 99% Accuracy, hat aber keinen Wert).
- **Precision** – unter den vom Modell als positiv markierten Elementen: wie viele waren tatsächlich richtig. Nützlich, wenn Fehlalarme teuer sind (z. B. gute Kreditanträge fälschlich ablehnen).
- **Recall (Sensitivität)** – von allen echten Positiven: wie viele hat das Modell gefunden. Wichtig, wenn es teuer ist, einen Fall zu verpassen (z. B. eine Krankheit nicht zu erkennen).

- **F1-Score** – ein einfacher Ausgleich zwischen Precision und Recall. Praktisch, wenn sowohl Verpassen als auch Überalarmierung schlecht sind.

3.7.2 Accuracy verstehen (und seine Grenzen)

Accuracy ist die einfachste Metrik bei der Klassifikation: Sie misst den Anteil aller korrekten Vorhersagen. Formal gilt:

$$\text{Accuracy} = \frac{\text{Anzahl korrekt erkannter Fälle}}{\text{Gesamtzahl der Fälle}} \tag{3.1}$$

Einfach ausgedrückt: Wenn ein Modell 100 Vorhersagen macht und 95 richtig sind, beträgt seine Accuracy 95%. Obwohl das leicht zu verstehen ist, kann es stark in die Irre führen, wenn ein Ergebnis viel häufiger ist als das andere. In der Betrugserkennung sind z. B. nur etwa 1% der Transaktionen tatsächlich betrügerisch. Ein Modell, das immer „kein Betrug“ vorhersagt, läge zu 99% richtig und hätte eine sehr hohe Accuracy, würde aber nie echten Betrug erkennen. In solchen **unausgewogenen Datensätzen**[2] verbirgt Accuracy die wahre Schwäche des Modells, weil sie das Raten der Mehrheitsklasse belohnt. Um die Leistung fair zu beurteilen, brauchen wir Metriken, die sagen, wie gut das Modell Minderheitsfälle erkennt: **Recall** (wie viele echte Betrugsfälle es erkannt hat) und **Precision** (wie oft seine Betrugsalarme korrekt waren) sind die gebräuchlichsten und werden in Abschnitt 3.7.3 besprochen. Accuracy ist nur informativ, wenn beide Klassen ungefähr ausgeglichen sind; bei seltenen Ereignissen erzeugt sie ein falsches Qualitätsgefühl.

3.7.3 Precision und Recall verstehen

Wenn ein Modell Kategorien vorhersagt (ein Klassifikationsproblem), wie Spam/Nicht-Spam, Betrug/Nicht-Betrug oder Krankheit vorhanden/nicht vorhanden, kann es zwei Arten von Fehlern machen: *Fehlalarme* (etwas wird markiert, das tatsächlich in Ordnung ist) und *verpasste Erkennungen* (etwas wird nicht markiert, das tatsächlich ein Problem ist). **Precision** und **Recall** sind zwei komplementäre Metriken, die diese Fehler beschreiben und helfen, das Verhalten des Modells über die Accuracy hinaus zu verstehen.

Precision: „Wenn das Modell Ja sagt, wie oft liegt es richtig?“ Precision sagt uns, wie vertrauenswürdig eine positive Vorhersage ist. Sie misst den Anteil der als positiv markierten Elemente, die wirklich positiv waren. Sie ist gegeben durch:

$$\text{Precision} = \frac{\text{Anzahl korrekter positiver Vorhersagen}}{\text{Gesamtzahl positiver Vorhersagen des Modells}} \tag{3.2}$$

[2] Ein unausgewogener Datensatz ist einer, in dem es viel mehr Fälle in einer Klasse als in der anderen gibt.

Einfach gesagt: Hohe Precision bedeutet, dass die meisten vom Modell ausgelösten Alarme korrekt sind. Wenn ein Betrugserkennungsmodell 100 Transaktionen als verdächtig markiert und 90 davon tatsächlich betrügerisch sind, beträgt seine Precision 90%. Wenn es jedoch häufig Fehlalarme auslöst (viele legitime Transaktionen markiert), sinkt die Precision. Das ist in Anwendungen kritisch, in denen jeder Fehlalarm Kosten verursacht, etwa legitime Zahlungen zu blockieren oder unnötige medizinische Alarme zu senden.

Recall: „Von allen echten Positiven – wie viele hat das Modell erwischt?“ Recall (auch **Sensitivität**) misst, wie gut das Modell alle tatsächlich positiven Fälle findet.

$$\text{Recall} = \frac{\text{Anzahl korrekter positiver Vorhersagen}}{\text{Gesamtzahl tatsächlich positiver Fälle in der Realität}} \tag{3.3}$$

Hoher Recall bedeutet, dass das Modell die meisten echten Positiven erkennt; niedriger Recall bedeutet, es verpasst viele. Wenn es z. B. insgesamt 200 betrügerische Transaktionen gab und das Modell 90 korrekt markiert, beträgt der Recall 45%. In Problemen wie Krankheitsdetektion oder Sicherheitsüberwachung kann es viel schlimmer sein, einen echten Fall zu verpassen, als zusätzliche Fehlalarme zu erzeugen – daher zählt Recall hier am meisten.

Definition — Accuracy, Precision und Recall

Accuracy sagt uns, wie oft das Modell insgesamt richtig liegt. Sie misst den Anteil aller korrekten Vorhersagen – sowohl der „Ja“- als auch der „Nein“-Fälle. Sie ist nützlich, wenn beide Outcomes gleich häufig sind.

Precision beantwortet die Frage: „Wenn das Modell *Ja* sagt, wie oft liegt es richtig?“ Sie konzentriert sich auf die Qualität positiver Vorhersagen. Hohe Precision bedeutet wenige Fehlalarme – die meisten markierten Fälle sind wirklich positiv.

Recall beantwortet die Frage: „Von allen echten *Ja*-Fällen – wie viele hat das Modell gefunden?“ Sie konzentriert sich auf Vollständigkeit. Hoher Recall bedeutet, das Modell erwischt die meisten echten Positiven, auch wenn es manchmal zusätzliche Fehlalarme auslöst.

Kurz:

- **Accuracy** — wie oft das Modell insgesamt richtig liegt.
- **Precision** — wenn es etwas vorhersagt, wie oft es korrekt ist.
- **Recall** — wie viele der echten Fälle es findet.

Der Zielkonflikt zwischen Precision und Recall. Eine Verbesserung der einen verringert oft die andere. Wenn Sie das Modell vorsichtiger machen (nur markieren, wenn es sehr sicher ist), gibt es *weniger Fehlalarme*, Precision steigt,

aber Sie verpassen mehr echte Fälle – Recall sinkt. Wenn Sie es sensibler machen (auch schwache Signale markieren), erwischen Sie mehr Positive (Recall steigt), erzeugen aber mehr Fehlalarme (Precision sinkt). Das richtige Gleichgewicht hängt vom Kontext ab. Zum Beispiel:

- In **medizinischem Screening** oder **Sicherheitsüberwachung** ist es teuer, echte Fälle zu verpassen ⇒ Recall priorisieren.
- In **Spam-Filterung** oder **Kreditgenehmigung** sind Fehlalarme teuer ⇒ Precision priorisieren.

Example - Precision vs. Recall

Stellen Sie sich einen Spam-Filter vor. Betrachten Sie folgende Situation.

- Von 1000 E-Mails sind 100 wirklich Spam.
- Von diesen 1000 markiert das Modell 120 als Spam, 80 davon korrekt.

Wir können Precision und Recall berechnen.

$$\text{Precision} = \frac{80}{120} = 0{,}67\ (67\%), \qquad \text{Recall} = \frac{80}{100} = 0{,}80\ (80\%) \quad (3.4)$$

Das Modell erwischt also 80% allen Spams (guter Recall), aber jede dritte markierte Nachricht ist tatsächlich kein Spam (mäßige Precision). Wenn dieser Filter in einem Unternehmen eingesetzt würde, in dem es schlimmer ist, Spam zu übersehen, als einige normale E-Mails falsch zu klassifizieren, könnte dieser Trade-off akzeptabel sein.

Zur Erinnerung: Accuracy allein kann nicht sagen, ob ein Modell die *richtige Art* von Fehlern macht. Precision und Recall zusammen beschreiben die *Qualität der Vorhersagen* und ob das Modell vorsichtig oder aggressiv positive Fälle markiert. Ein ausgewogenes System hängt vom Kontext ab, und Sie entscheiden, ob es besser ist, einige echte Fälle zu verpassen oder einige zusätzliche Fehlalarme zu erzeugen. Zusammengefasst:

- **Precision** beantwortet: „Wenn ich Alarm schlage, wie oft liege ich richtig?“
- **Recall** beantwortet: „Von allen echten Alarmen – wie viele habe ich erwischt?“

In vielen realen Systemen werden beide Metriken zusammen überwacht und manchmal in einer einzigen Balance-Zahl (dem **F1-Score**) kombiniert, die widerspiegelt, wie gut das Modell Precision und Recall ausbalanciert. Eine Diskussion des F1-Scores geht jedoch über den Rahmen dieses Kurses hinaus. Eine kurze Übersicht steht in Tabelle 3.4.

Metrik	Aussage
Precision	„Wenn ich Alarm schlage, wie oft liege ich richtig?“ (Fehlalarme vermeiden)
Recall	„Von allen echten Alarmen – wie viele habe ich erwischt?“ (Verpasser vermeiden)

Tabelle 3.4: Precision und Recall und was jede Metrik aussagt.

3.7.4 Für Zahlen und Prognosen (Regression)

Wenn das Modell statt einer Kategorie eine Zahl vorhersagt, wie Verkaufszahlen nächste Woche, Lieferzeit oder Energieverbrauch, müssen wir messen, wie nah seine Vorhersagen an den erwarteten Werten (Labels) liegen. Das tun wir mit **Regressionsmetriken**, die unterschiedliche Arten beschreiben, Vorhersagefehler zu charakterisieren. Einige fokussieren auf die *durchschnittliche Größe* der Fehler, andere darauf, wie stark große Fehler ins Gewicht fallen, und manche darauf, wie gut das Gesamtschwankungsmuster in den Daten erklärt wird. Die gebräuchlichsten sind unten zusammengefasst. Für die Formeln bezeichnen wir mit y_i die wahren Labels (was wir erwarten) und mit $\hat{y}_i$ den vorhergesagten Output für den i-ten Input.

- **Mean Absolute Error (MAE)**: wie stark die Vorhersagen des Modells im Durchschnitt danebenliegen. Leicht in denselben Einheiten wie das Ziel zu interpretieren (z. B. „die Temperaturprognose liegt im Schnitt um $5°C$ daneben“). Die Formel lautet

$$\text{MAE} = \frac{1}{n}(|y_1 - \hat{y}_1| + |y_2 - \hat{y}_2| + \cdots + |y_n - \hat{y}_n|) \tag{3.5}$$

  wobei wir n Dateneingaben haben. Das Symbol $|\cdot|$ bezeichnet den Absolutwert, d. h. jede Zahl in ihren positiven Betrag umzuwandeln. Also ist z. B. $|3| = 3$, aber $|-3| = 3$. Anschaulich misst die MAE, wie weit das Modell im Durchschnitt vom erwarteten Output entfernt ist[3].

- **Mean Squared Error (MSE)**: Diese Metrik beruht auf einer ähnlichen Idee wie MAE, summiert aber nicht die Absolutwerte der Fehler, sondern deren Quadrate gemäß

$$\text{MSE} = \frac{1}{n}((y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \cdots + (y_n - \hat{y}_n)^2) \tag{3.6}$$

  Diese Art von Metrik betont große Fehler. Große Fehler werden durch das Quadrieren noch größer!

- **Erklärte Varianz (R^2)**: Das R-Quadrat (R2 oder R^2) ist eine Zahl, die angibt, wie gut die unabhängigen Variablen (Inputs) in einem statistischen Modell die Variation in der abhängigen Variable (Vorhersagen) erklären.

[3] Denken Sie daran, wir sind im überwachten Lernen; unsere Datensätze bestehen aus Eingaben und Labels bzw. erwarteten Outputs.

Eine höhere Zahl bedeutet, dass das Modell besser funktioniert, eine niedrigere, dass es schlecht funktioniert (ein intuitives Verständnis genügt; für eine mathematischere Diskussion siehe Abschnitt 13.2.2 in [34]).

Definition — MAE und MSE (Vorhersagefehler messen)

Wenn ein Modell eine Zahl vorhersagt (etwa einen Preis, eine Verzögerung oder eine Temperatur), können wir messen, wie weit seine Vorhersagen von den echten Werten entfernt sind. Zwei gebräuchliche Wege hierfür sind der **Mean Absolute Error (MAE)** und der **Mean Squared Error (MSE)**.

Mean Absolute Error (MAE) misst, wie groß die Fehler im Durchschnitt sind, und behandelt alle Fehler gleich. Definiert ist er durch:

$$\text{MAE} = \frac{1}{n}\left(|y_1 - \hat{y}_1| + |y_2 - \hat{y}_2| + \cdots + |y_n - \hat{y}_n|\right)$$

Dabei gilt:

- y_i ist der wahre (reale) Wert,
- $\hat{y}_i$ ist der vom Modell vorhergesagte Wert,
- n ist die Gesamtzahl der Vorhersagen.

Das Ergebnis sagt uns, wie weit das Modell im Durchschnitt danebenliegt – in denselben Einheiten wie die Daten (z. B. „im Schnitt 5 Minuten daneben“).

Mean Squared Error (MSE) misst ebenfalls die Distanz zwischen Vorhersagen und Realität, gewichtet *große Fehler* aber stärker, indem jeder Fehler quadriert wird:

$$\text{MSE} = \frac{1}{n}\left((y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \cdots + (y_n - \hat{y}_n)^2\right)$$

Weil große Fehler quadriert werden, zählen sie im Endergebnis viel stärker. Das macht MSE nützlich, wenn große Ausreißer besonders kostspielig oder riskant sind.

Kurz:

- **MAE** — durchschnittliche Größe der Fehler, alle Fehler gleich gewichtet.
- **MSE** — gleiche Idee, aber große Fehler zählen deutlich mehr.

3.7.5 Die richtige Metrik wählen

Im Allgemeinen bestimmt das zu lösende Problem, welche Metrik die richtige ist. Tabelle 3.5 gibt eine kurze Übersicht, die bei der Wahl helfen kann.

Szenario (Beispiel)	**Fokus (Primäre Metrik)**	**Sekundäre Metrik**
Krankheitsscreening (selten)	**Recall:** so viele echte Fälle wie möglich erwischen.	**Precision:** zu viele Fehlalarme verschwenden Zeit und Ressourcen.
Inhaltsmoderation	**Precision:** vermeiden, sichere oder neutrale Inhalte zu entfernen.	**Recall:** dennoch schwerwiegende Schäden oder Missbrauch erfassen.
Kreditrisikomodellierung	**Kalibrierung:** vorhergesagte Wahrscheinlichkeiten müssen das echte Ausfallrisiko widerspiegeln.	**Fairness:** Leistung und Genehmigungsraten sollten über Gruppen hinweg ausgewogen sein.
Betrugserkennung	**Recall:** die meisten betrügerischen Fälle identifizieren.	**Precision:** Fehlalarme bei legitimen Kunden begrenzen.
Nachfrageprognose	**MAE:** typische Fehlergröße in verkauften Einheiten.	**Bias:** auf systematische Unter-/Überprognosen nach Region oder Saison prüfen.
Kostenschätzung oder Energieverbrauch	**MSE/RMSE:** große Fehlgriffe stärker bestrafen.	**Ausreißer:** extreme Werte können diese Metrik dominieren.
Modell-Fit (Regression)	R^2**:** Anteil der Varianz im Ziel, den das Modell erklärt.	**Datenqualität:** hohes R^2 kann dennoch Leakage oder Bias verbergen.

Tabelle 3.5: Beispiele passender Metriken und ergänzender Schutzgeländer in verschiedenen Modellsituationen.

3.8 Interpretierbarkeit und Mensch-in-der-Schleife

Modelle unterstützen menschliche Entscheidungen; sie ersetzen nicht Verantwortung oder Urteil. Selbst wenn ein Modell gut performt, müssen wir noch verstehen, *warum* es bestimmte Antworten gibt, und Menschen eingebunden halten, wo Ergebnisse für Sicherheit, Fairness oder Ethik relevant sind.

Ein Modell kann Muster finden, die Menschen nicht sehen, aber wenn sein Denken nicht verstanden werden kann, ist es schwer zu vertrauen oder zu verbessern. **Interpretierbarkeit** bedeutet, in einfachen Worten erklären zu können, was eine Vorhersage getrieben hat. Ein Triage-Tool im Krankenhaus könnte z. B. erklären, dass ein Patient vor allem wegen Alter, jüngsten Aufnahmen und bestimmten Symptomen als Hochrisiko markiert wurde. Ziel ist nicht, die

Mathematik zu lesen, sondern das Verhalten für Fachexperten prüfbar und plausibel zu machen. Statt sich auf abstrakte „Features“ oder technische Grafiken zu konzentrieren, denken Sie an Erklärungen als kurze Geschichten darüber, was dem Modell aufgefallen ist. Wir können fragen:

- Welche Informationsstücke haben diese spezielle Vorhersage am meisten beeinflusst?
- Sind diese Faktoren für diesen Kontext vernünftig?
- Werden ähnliche Fälle konsistent behandelt?

Diese Prüfungen helfen Menschen zu erkennen, wenn sich das Modell auf irreführende oder unfairen Muster stützt.

Interpretierbarkeit bedeutet auch, die Leistung über verschiedene **Segmente oder Kohorten** hinweg zu prüfen (z. B. nach Alter, Standort, Region oder Saison). Wenn ein Modell für eine Gruppe gut, für eine andere schlecht performt, kann es unfair oder unzuverlässig sein. Einfache Tabellen oder Grafiken mit Ergebnissen nach Gruppe können solche Lücken sichtbar machen und Verbesserungen lenken. Kein automatisiertes System sollte sensible Entscheidungen völlig allein treffen. Menschliche Aufsicht bleibt wesentlich, um:

- **Ungewöhnliche oder Grenzfälle zu prüfen**, mit denen Modelle kämpfen könnten.
- **Entscheidungen zu übersteuern oder anzupassen**, wenn lokales Wissen nahelegt, dass das Modell falsch liegt.
- **Feedback zu geben**, um künftige Versionen zu verbessern, z. B. durch Korrektur falscher Labels oder Identifikation neuer Faktoren.

Diese Zusammenarbeit zwischen Menschen und Modellen kombiniert die Geschwindigkeit und Konsistenz des Modells mit menschlichem Kontext, Ethik und gesundem Menschenverstand.

Wichtig ist: Interpretierbarkeit bedeutet nicht technische Transparenz. Es geht darum, *das Modellverhalten ausreichend verständlich zu machen, um es verantwortlich zu nutzen.* Das Modell kann unterstützen, aber Menschen müssen für Entscheidungen und dafür verantwortlich bleiben, dass das System weiterhin sicher und fair agiert.

3.9 Fortgeschrittenes Referenzkapitel: Metrikdefinitionen (Optional)

Zweck. Diese Formeln sind für Leser gedacht, die die formalen Definitionen hinter den zuvor eingeführten Metriken sehen möchten. Sie müssen sie nicht auswendig lernen; sie sind als mathematische Referenz enthalten.

Klassifikationsmetriken

Seien:

- TP = True Positives (Modell sagt positiv und es war positiv)
- FP = False Positives (Modell sagt positiv, aber es war negativ)
- TN = True Negatives (Modell sagt negativ und es war negativ)
- FN = False Negatives (Modell sagt negativ, aber es war positiv)

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

$$\textbf{Recall (Sensitivität)} = \frac{TP}{TP + FN}$$

$$\textbf{Spezifität} = \frac{TN}{TN + FP}$$

$$\textbf{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Regressionsmetriken

Seien y_i der wahre Wert, $\hat{y}_i$ der vorhergesagte Wert und n die Anzahl der Stichproben.

$$\textbf{Mean Absolute Error (MAE)} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$\textbf{Mean Squared Error (MSE)} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\textbf{Root Mean Squared Error (RMSE)} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$\textbf{Bestimmtheitsmaß } (R^2) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

wobei $\bar{y}$ der Mittelwert der wahren Werte ist.

Interpretationszusammenfassung.

- ↑ Precision: weniger Fehlalarme.
- ↑ Recall: weniger verpasste Erkennungen.
- ↑ F1: ausgewogener Trade-off.
- ↓ MAE/MSE/RMSE: kleinere durchschnittliche bzw. quadrierte Fehler.
- $R^2 \in (-\infty, 1]$: höher ist besser; kann negativ sein, wenn das Modell schlechter ist als der Mittelwert.

Glossar

Klassifikation
: Aufgaben, bei denen der Output eine *Kategorie* ist (z. B. Spam vs. Nicht-Spam; Genehmigen vs. Ablehnen). Das Modell gibt oft auch eine Wahrscheinlichkeit für jede Klasse aus.

Regression
: Aufgaben, bei denen der Output eine *Zahl* ist (z. B. Preis, Lieferzeit, Nachfrage nächste Woche).

Clustering
: Ähnliche Elemente ohne Labels gruppieren, um natürliche Segmente oder Muster in den Daten zu entdecken.

Feature/Merkmal
: Eine Eingangsvariable (Signal), die vom Modell verwendet wird (z. B. Alter, Entfernung, Tage seit letztem Kauf, Wortanzahl).

Label / Target
: Das Ergebnis, das wir vorhersagen möchten (Klasse bei Klassifikation; numerischer Wert bei Regression).

Wahrscheinlichkeitsschwelle
: Ein Cut-off auf der vorhergesagten Wahrscheinlichkeit, um Scores in Ja/Nein-Entscheidungen zu verwandeln (z. B. markieren, wenn $p \geq 0.8$).

Konfusionsmatrix
: Eine 2×2-Tabelle, die korrekte/falsche Entscheidungen (True/False Positives/Negatives) für einen binären Klassifikator zählt.

Accuracy
: Anteil aller Vorhersagen, die korrekt sind. Kann bei seltenen Ereignissen irreführend sein.

Precision
: Von den vom Modell als positiv markierten Elementen der Anteil, der wirklich positiv war (*„Wenn ich Alarm schlage, wie oft liege ich richtig?“*).

Recall (Sensitivität)
: Von allen wirklich positiven Elementen der Anteil, den das Modell erfolgreich markiert hat (*„Von allen echten Alarmen – wie viele habe ich erwischt?“*).

F1-Score
: Eine einzelne Zahl, die Precision und Recall balanciert (harmonisches Mittel). Nützlich, wenn beide Fehlertypen wichtig sind.

Kalibrierung
: Wie gut vorhergesagte Wahrscheinlichkeiten die Realität abbilden (z. B. unter Fällen mit 70% Score sollten etwa 70% positiv sein).

MAE (Mean Absolute Error)
: Durchschnittliche absolute Abweichung zwischen Vorhersagen und echten Werten (gleiche Einheiten wie das Ziel).

MSE / RMSE
: Mittel der quadrierten Fehler / dessen Quadratwurzel. Bestrafen große Fehler stärker als MAE; RMSE hat Zieleinheiten.

R^2 **(Erklärte Varianz)**
: Anteil der Variation im Ziel, den das Modell erklärt (1 ist perfekt; kann negativ sein, wenn schlechter als der Mittelwert).

Baseline
: Eine einfache Referenz, die es zu schlagen gilt (z. B. Mehrheitsklasse bei Klassifikation; Wert der letzten Woche oder Durchschnitt bei Forecasting).

Overfitting
: Modell lernt Trainings-Eigenheiten und performt auf neuen Daten schlecht (Train-Score $\gg$ Test-Score).

Underfitting
: Modell ist zu einfach, um echte Struktur zu erfassen (schwach auf Train und Test).

Modellvalidierung
: Prüfen, wie gut ein Modell auf ungesehenen Daten generalisiert; das Gegenmittel zu überoptimistischen Ergebnissen.

Hold-out-Split
: Ein einzelner Split der Daten in Train/Validierung(/Test); die Testmenge bleibt bis zum Ende unberührt.

Train/Validierung/Test
: *Train* zum Lernen; *Validierung* zum Abstimmen und Schwellwerte wählen; *Test* zur Schätzung der finalen Leistung.

Cross-Validation
: Training/Testing mehrfach auf verschiedenen Splits wiederholen, um eine stabilere Leistungsschätzung zu erhalten (z. B. k-fold; wiederholte Zufallssplits).

K-Fold Daten in k Teile teilen; k-mal trainieren, jedes Mal einen anderen Teil zum Testen zurückhalten; Scores mitteln.

Monte Carlo CV
: Wiederholte zufällige Train/Test-Splits (z. B. 80/20) mit gemittelter Leistung über Wiederholungen.

Data Leakage
: Verwendung von Informationen, die zum Entscheidungszeitpunkt nicht verfügbar sind (oder aus dem Outcome abgeleitet wurden) und die Validierungsergebnisse künstlich erhöhen, aber in der Praxis scheitern.

Split-Strategien
: **Zufällig** (unabhängige Fälle), **zeitbasiert** (früher trainieren, später testen), **gruppenbewusst** (Entitäten nicht über Splits hinweg mischen).

Segment / Kohorte
: Eine Teilgruppe (z. B. Standort, Region, Gerät, Altersgruppe). Ergebnisse nach Kohorten zu prüfen hilft, Fairness-Lücken oder Drift zu erkennen.

Fairness (High Level)
: Leistung und Entscheidungen sollten geschützte oder bedeutende Gruppen nicht systematisch benachteiligen; Metriken über Kohorten prüfen.

Human-in-the-Loop
: Menschen prüfen Randfälle, können Entscheidungen übersteuern und liefern Korrekturen zur Verbesserung künftiger Versionen.

Interpretierbarkeit
: In einfachen Worten erklären, was eine Vorhersage beeinflusst hat (lokal) oder wie sich das Modell insgesamt verhält.

Drift
: Wenn sich Daten oder Beziehungen im Zeitverlauf ändern und die Leistung abnimmt; erfordert Monitoring und periodische Neukalibrierung/Neutrainierung.

Ordinal / Multiklassen / Multi-Label
: **Ordinal:** geordnete Labels (niedrig/mittel/hoch). **Multiklassen:** eine von vielen exklusiven Labels. **Multi-Label:** mehrere Labels können gleichzeitig gelten.

One-vs-Rest / One-vs-One
: Schemata, um Multiklassenprobleme mit binären Bausteinen zu lösen: OvR trainiert ein Modell pro Klasse vs. alle; OvO trainiert eines pro Klassenpaar.

No-Code / AutoML
: Tools, die Teile von ML (Feature-Handling, Modellsuche) über eine grafische Oberfläche automatisieren. Schnell für Exploration; erfordern sorgfältige Validierung und Aufsicht.

Metrik (Primär & Guardrail)
: **Primär** spiegelt die Hauptkosten wider (z. B. Recall für Sicherheit, MAE für Geld/Einheiten). **Guardrail** beobachtet Nebenwirkungen (z. B. Precision, Fairness, Kalibrierung).

‟‟

Literaturverzeichnis

[1] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[2] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[3] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[4] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[6] Terrence J Sejnowski and Charles R Rosenberg. Parallel networks that learn to pronounce english text. *Complex systems*, 1(1):145–168, 1987.

[7] Yann LeCun et al. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2:396–404, 1990.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] Vladimir N Vapnik and Alexey Ya Lerner. A class of algorithms for learning pattern recognition. *Automation and Remote Control*, 24(6), 1964.

[11] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[13] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[14] Yoav Freund and Robert E Schapire. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14:771–780, 1999.

[15] Jerome H Friedman. Greedy function approximation: A gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[16] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[17] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[18] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[19] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[20] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12:1057–1063, 2000.

[21] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[22] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[25] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[28] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[29] Ian J. Goodfellow et al. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[30] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[31] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[32] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Enhanced higgs boson to $\tau+ \tau-$ search with deep learning. *Physical review letters*, 114(11):111801, 2015.

[33] Andrew W. Senior et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577:706–710, 2020.

[34] Umberto Michelucci. *Statistics for Scientists: A Concise Guide for Data-driven Research*. Springer Nature Switzerland, Cham, 2025.

Index