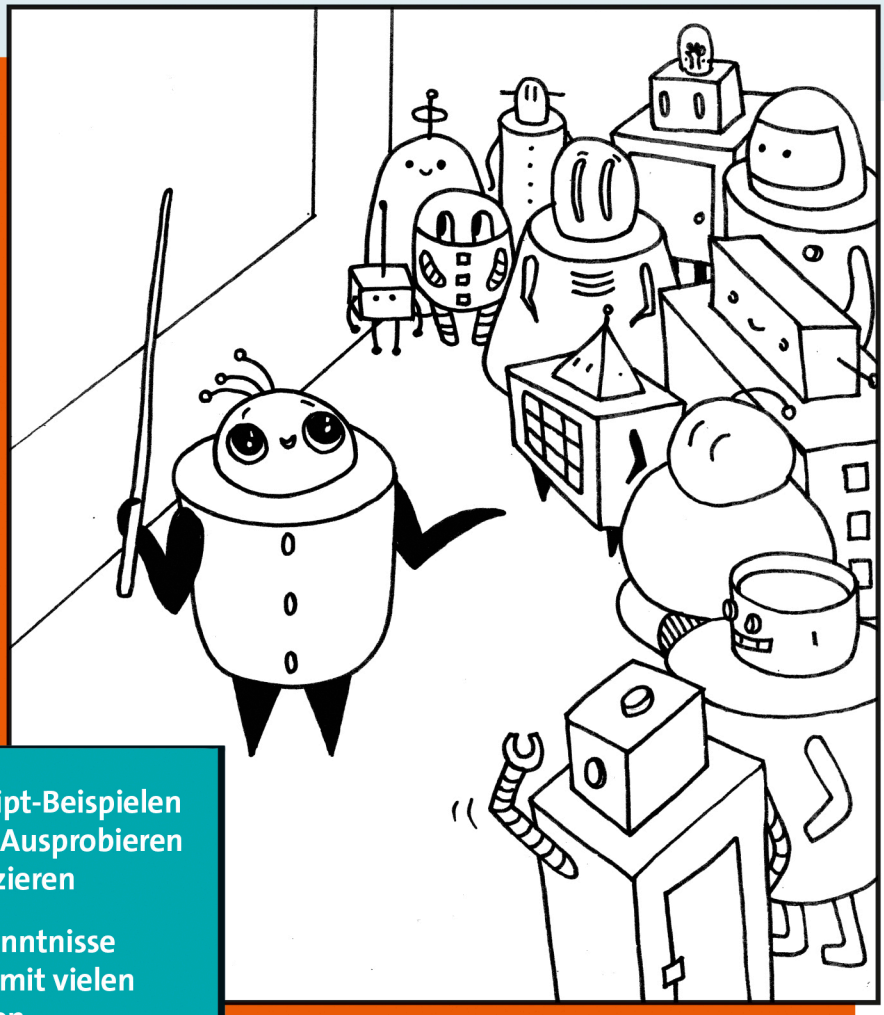
Pit Noack · Sophia Sanner

Künstliche Intelligenz verstehen

Eine spielerische Einführung



+ Mit JavaScript-Beispielen zum online Ausprobieren und Modifizieren
+ Ohne Vorkenntnisse einsteigen, mit vielen Illustrationen
+ Spiele-KI, Graphen, Neuronale Netze u. v. m.

Rheinwerk Computing

Kapitel 1
Einleitung

Es fällt nicht schwer, Namen von historischen Persönlichkeiten zu nennen, die besonders intelligent waren. Wir denken spontan an Köpfe wie Albert Einstein, Hannah Arendt, Leonardo da Vinci oder Simone de Beauvoir. Vielleicht haben wir auch einen Freund – nennen wir ihn Karsten –, der herausragend Schach spielt, oder eine Freundin – nennen wir sie Sabine –, die im Studium immer alles als Erste kapiert hat und im Alltag regelmäßig schlaue Lösungen für vertrackte Probleme aus dem Hut zaubert.

Weitaus schwieriger, als intelligente Menschen zu identifizieren, ist es, zu erklären, was Intelligenz eigentlich ist. Eine allgemein anerkannte Definition fehlt. Somit ist es wenig überraschend, dass auch der Begriff *Künstliche Intelligenz* (KI) höchst umstritten ist.

Den herausragenden Schachspieler Karsten werden wir ohne Umschweife intelligent nennen. Aber viele von uns würden zögern, ein Schachprogramm als intelligent zu bezeichnen, selbst wenn dieses weitaus besser spielen kann als unser Freund. Woher kommt diese Zurückhaltung?

Das dürfte unter anderem daran liegen, dass sich viele Menschen generell schwer damit tun, Maschinen menschliche Eigenschaften zuzuschreiben. Ein weiterer und aus unserer Sicht der entscheidende Grund: Alle Anwendungen, die aktuell unter dem Namen »künstliche Intelligenz« firmieren, sind jeweils auf *eine einzige Klasse von Aufgaben* spezialisiert. Versuchen Sie einmal, mit einem Schachprogramm über Literatur zu diskutieren, probieren Sie, eine Bilderkennungssoftware zu einer Partie Halma zu überreden, oder bitten Sie Ihr Navigationssystem, eine Knobelaufgabe zu lösen oder eine Geschichte zu einem vorgegebenen Thema zu improvisieren.

Diese Spezialisierung markiert einen wesentlichen Unterschied zwischen menschlicher Intelligenz und dem, was Computerprogramme aktuell zu leisten imstande sind: Unsere Freundin Sabine vermag sich auf sehr unterschiedliche Probleme und Aufgaben im kürzester Zeit einzustellen. Auch von Freund Karsten dürfen wir erwarten, dass er seine im Schachspiel erworbenen Fähigkeiten zum strategischen und weit vorausschauenden Denken auch auf andere Brettspiele oder gar auf Alltagssituationen übertragen kann.

Aktuelle KI-Anwendungen sind weit entfernt von der universell anwendbaren Intelligenz, die wir bei Sabine und Karsten beobachten können. Es ist unter Fachleuten umstritten, ob eine solche *Allgemeine Künstliche Intelligenz* (AKI) jemals erreicht werden kann – und wenn ja, auf welchem Wege wir dorthin gelangen könnten. Wir halten uns in dieser Frage mit einer Einschätzung zurück, sind uns aber in einem Punkt sicher: Es genügt nicht, einen Haufen von spezialisierten Programmen zu versammeln und dann mal das eine, mal das andere aufzurufen, je nachdem, was gerade anliegt.

Fassen wir also zusammen: Die Kritik am Begriff *Künstliche Intelligenz* halten wir für berechtigt. Trotzdem lässt sich kaum bestreiten, dass unter diesem Begriff viele erstaunliche Anwendungen versammelt sind, die unser aller Leben bereits maßgeblich bestimmen und verändern – vielleicht sogar mehr, als uns lieb ist. Ziel dieses Buchs ist es, allen interessierten Leserinnen und Lesern ein grundlegendes Verständnis von KI zu vermitteln, ganz unabhängig davon, welche Vorkenntnisse diese mitbringen.

1.1 Worum es uns in diesem Buch geht

Stellen Sie sich vor, Sie besuchten eine Stadt, in der Sie nie zuvor gewesen sind. Sie planen einen längeren Aufenthalt oder wollen diese Stadt in Zukunft öfter besuchen. Es lohnt sich also, Orientierung zu gewinnen! Wie gehen Sie vor? Werden Sie sämtliche Straßen systematisch von Norden nach Süden und von Osten nach Westen durchkämmen? Werden Sie ausgehend von den Hauptverkehrsadern alle Nebenstraßen durchstreifen oder gar den Stadtplan auswendig lernen? Wohl eher nicht! Wenn Sie eine Stadt erkunden, werden Sie sich vermutlich zunächst einige Orientierungspunkte einprägen: den Bahnhof, das gemütliche Café, das Kunstmuseum, das Restaurant mit dem leckeren chinesischen Essen ... Ausgehend von diesen Punkten werden Sie im Laufe der Zeit die Stadt immer besser kennenlernen.

Dieses Buch nimmt sich das Thema KI in ähnlicher Weise vor. Wir erheben nicht den Anspruch, eine systematische oder gar vollständige Darstellung zu liefern. Vielmehr haben wir einige Verfahren herausgepickt, die wir besonders lehrreich, interessant oder unterhaltsam finden.

Beim Schreiben dieses Buchs haben wir großen Wert auf anschauliche Beispiele und eine kurzweilige Darstellung gelegt. Um es mit den Worten einer befreundeten Informatikstudentin zu sagen: »An der Uni sind wir in das Thema Q-Learning mit einem Haufen mathematischer Definitionen eingestiegen – euer Buch startet mit einem Eichhörnchen, das seinen Nussvorrat verbummelt hat.«

Sehr viel Mühe haben wir darauf verwendet, die vorgestellten Verfahren praktisch erfahrbar zu machen: Auf der Webseite zum Buch finden Sie zahlreiche Beispielprogramme, die Sie mit nur einem Mausklick im Browser starten können. Die Programme laden ein zum Probieren und Experimentieren und stellen eine direkte Verbindung zwischen Theorie und Praxis her.

1.2 Für wen wir dieses Buch geschrieben haben

Wir sprechen mit diesem Buch einen großen Kreis von Leserinnen und Lesern an. Wenn Sie sich in einer der folgenden Beschreibungen wiederfinden, dann haben wir dieses Buch auch für Sie geschrieben:

- Sie haben noch nie programmiert und möchten es vielleicht auch nicht lernen, wollen aber mehr über KI erfahren.
- Sie sind Hobbyprogrammiererin oder Freizeit-Coder und suchen Anregungen für eigene Projekte.
- Sie stehen am Beginn einer Ausbildung oder eines Studiums in einem Fachgebiet, in dem Sie mit Code und KI in Berührung kommen werden, und möchten sich einen ersten Überblick über KI-Verfahren verschaffen.
- Sie haben beruflich mit Programmierung zu tun, wissen aber noch wenig über KI und wollen ihr Wissen ausbauen.
- Sie lehren an einer Schule oder einer Hochschule und sind auf der Suche nach anschaulichen Beispielen für Ihren eigenen Unterricht.

Bei allen Verfahren, die wir vorstellen, war unser erstes Anliegen, ein grundlegendes Verständnis der entscheidenden Ideen zu vermitteln. Wir haben mehr Wert auf ein intuitives Verständnis als auf mathematische Präzision gelegt. Denn wir haben uns gedacht: Ausführliche Darstellungen von technischen Details und Spezialfällen nützen wenig, wenn das Wesentliche unklar bleibt.

Das bedeutet aber nicht, dass Sie irgendetwas verlernen müssen, wenn Sie mehr erfahren und über das Darstellungsniveau dieses Buchs hinausgehen wollen. Wann immer wir einen Sachverhalt grob vereinfacht haben, weisen wir deutlich darauf hin.

1.3 Aufbau der einzelnen Kapitel

Die einzelnen Kapitel sind weitgehend einheitlich aus folgenden Bausteinen zusammengesetzt:

- Wir stellen eine Klasse von Problemen vor, die wir lösen wollen.
- Wir erläutern ein Verfahren, das zur Lösung des vorgestellten Problems geeignet ist.
- Wir präsentieren ein Beispielprogramm, in dem das vorgestellte Verfahren zur Anwendung kommt. Das Programm beschreiben wir zunächst aus der Sicht von Nutzerinnen und Nutzern.
- Wir stellen den Programmcode vor, der das jeweilige Verfahren umsetzt.
- Wir geben einen Ausblick auf benachbarte Themen, skizzieren Punkte, die wir weggelassen haben, schlagen Modifikationen des Beispielprogramms sowie Codeexperimente vor und geben Denkanstöße.

1.4 Ein Wort an die Programmierunkundigen

Wir richten uns mit diesem Buch ganz bewusst auch an Personen, die noch nie in ihrem Leben programmiert haben. Trotzdem enthält es einige Passagen, in denen wir Code im Detail diskutieren. In diesen Passagen mussten wir die Kenntnis von Grundbegriffen wie *Schleife*, *Funktion*, *Objekt* oder *Klasse* voraussetzen. Falls Ihnen solche Begriffe noch fremd sind, schlagen wir Ihnen eine dieser drei Vorgehensweisen vor:

- Sie überspringen unsere Codeerläuterungen komplett – das wäre zwar schade, aber zu verschmerzen: Die Passagen, die sich auf Code beziehen, sind für das angestrebte allgemeine Verständnis und die weitere Lektüre nicht zwingend notwendig.
- Sie überfliegen die Codeabschnitte und beachten dabei vor allem die Tabellen und die Infografiken. Auf diese Weise bekommen Sie immerhin eine Ahnung davon, aus welchen Komponenten der Code besteht und wie diese ineinandergreifen.

- Wenn Sie es genauer wissen wollen, finden Sie in Anhang A eine knappe Einführung in die verwendete Programmiersprache JavaScript und die p5.js-Bibliothek. Diese Einführung fängt bei null an, setzt also überhaupt keine Vorkenntnisse voraus. Das Studium dieser Einführung sollte Sie in die Lage versetzen, unseren Codeerläuterungen weitgehend zu folgen.

1.5 Beispielprogramme und die Webseite zum Buch

Alle Beispielprogramme können Sie ohne Download, Installation und Anmeldung direkt im Browser starten. Die Links finden Sie auf der Webseite zum Buch: *https://maschinennah.de/ki-buch*. Profis haben darüber hinaus die Möglichkeit, den Quellcode direkt bei GitHub herunterzuladen: *https://github.com/MaschinenNah/ki-buch*.

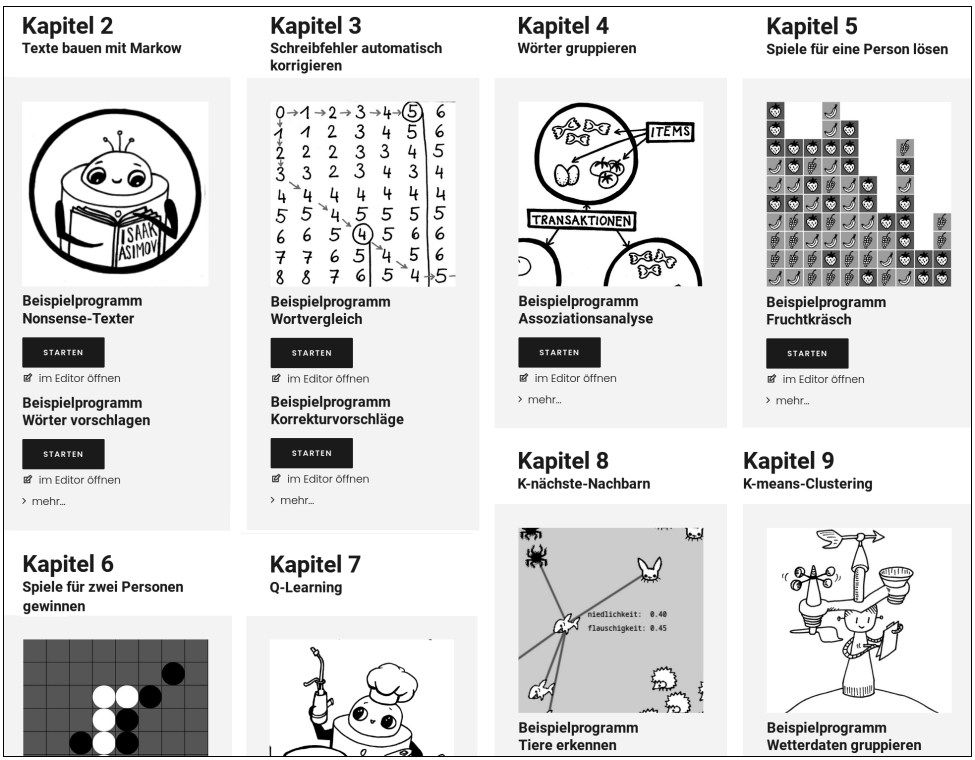


Abbildung 1.1 Die Webseite zum Buch

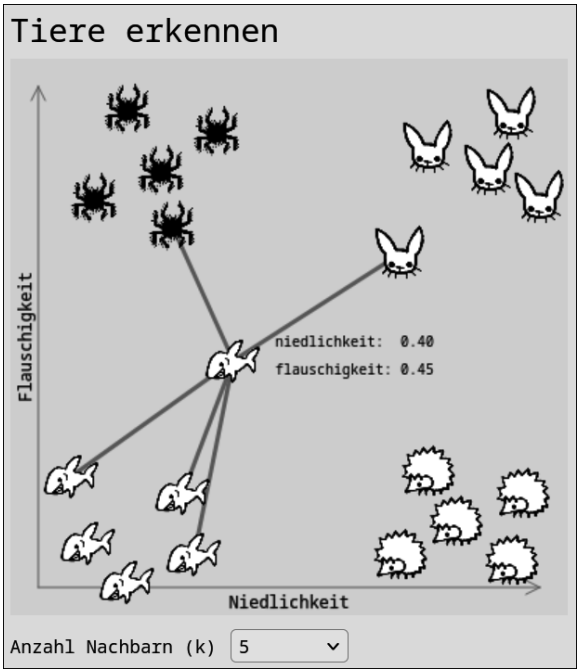


Abbildung 1.2 Das Beispielprogramm »Tiere erkennen«

Experimentierfreudige Leserinnen und Leser können den Code unserer Beispielprogramme eigenhändig modifizieren und das Ergebnis direkt im Browserfenster sehen. Der Online-Editor von p5.js macht es möglich. Abbildung 1.3 zeigt einen Screenshot des Editors: Links sehen Sie den Code, rechts die Ausführung des Programms. Mehr über Nutzung und Funktionen des Online-Editors erfahren Sie in Anhang A im Abschnitt »Der p5.js-Online-Editor«.

Im Buch werden wir ausschließlich jene Codeabschnitte diskutieren, die für die Umsetzung der vorgestellten KI-Verfahren relevant sind. So stellen wir zum Beispiel in Kapitel 6, »Spiele für zwei Personen gewinnen«, nicht dar, wie wir das Spiel Reversi programmiert haben. Die Funktionsweise der grafischen Benutzeroberfläche, die Repräsentation des Spielfelds oder die Ermittlung der jeweils gültigen Züge besprechen wir nicht. Es geht uns allein darum, die Funktionsweise der KI zu erläutern, die gegen uns Reversi spielt.

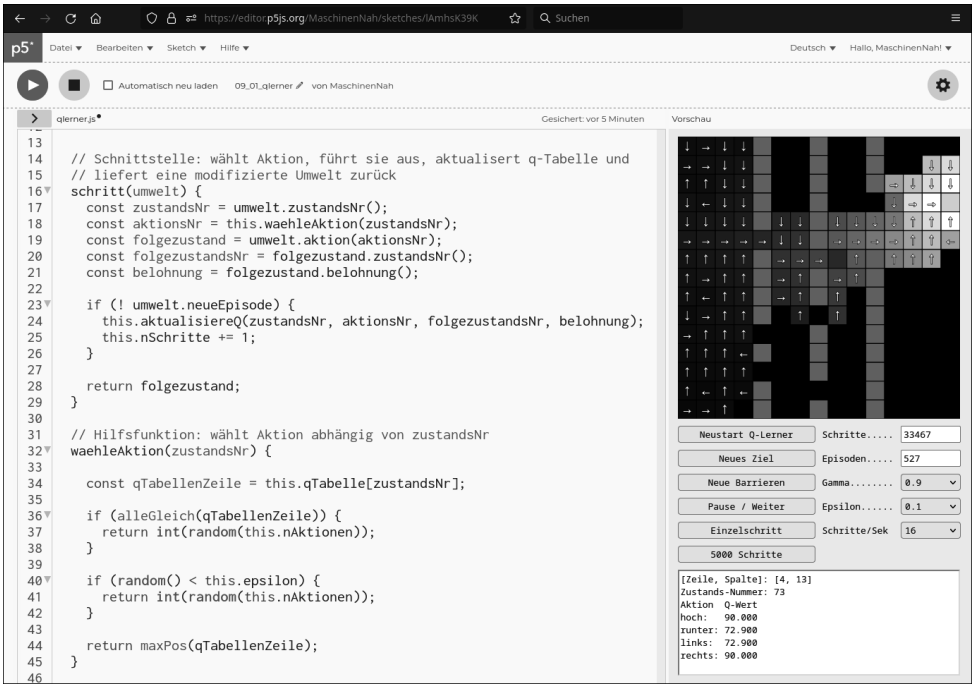


Abbildung 1.3 Im Online-Editor modifizieren Sie auf der linken Seite den Code, auf der rechten Seite sehen Sie das Ergebnis; dafür ist keine Anmeldung notwendig.

Wenn Sie allerdings mehr über das Zusammenspiel der einzelnen Komponenten unserer Beispielprogramme erfahren wollen, stehen Sie nicht mit leeren Händen da: Den online verfügbaren Quellcode haben wir mit erläuternden Kommentaren versehen. Da wir uns mit diesem Buch auch an Anfängerinnen und Einsteiger richten, sind die Kommentare mitunter etwas ausführlicher geraten, als es professionelle Softwareentwicklerinnen und -entwickler für angebracht halten dürften.

1.6 Warum wir JavaScript und p5.js verwendet haben

Programmierprofis werden sich fragen, warum wir für unsere KI-Einführung ausgerechnet JavaScript und p5.js nutzen. Schließlich werden professionelle KI-Anwendungen meist in Python geschrieben. Und in der Tat: Wenn Sie nach der Lektüre dieses Buchs auf den Geschmack gekommen sind und größere KI-Projekte in Angriff nehmen wollen, werden Sie an Python und Python-Umgebungen wie Keras oder PyTorch nicht vorbeikommen.

Das liegt zum einen daran, dass wir waschechte p5.js-Fans sind und diese Sprache seit vielen Jahren erfolgreich in Kursen für jugendliche und erwachsene Programmierneulinge verwenden.

Und auch sonst war unsere Entscheidung didaktisch motiviert: Wir wollen möglichst viele Leserinnen und Leser ansprechen und den Einstieg so unkompliziert wie möglich gestalten. Mit JavaScript, p5.js und dem Online-Editor ist jedes Beispielprogramm nur einen Mausklick entfernt. Die Notwendigkeit eines Downloads, einer Softwareinstallation oder auch nur einer Registrierung würde viele potenziell interessierte Menschen abschrecken.

Zudem programmieren wir in unseren Beispielen vieles »von Hand«, was Keras, PyTorch & Co. als fertige Funktion anbieten. Auch hier gilt: Wenn Sie später auf eine andere Programmiersprache umsteigen, müssen Sie nichts davon verlernen, was Sie in diesem Buch gelernt haben. Im Gegenteil: Die Fachbegriffe, die wir hier vorstellen, bleiben dieselben, ganz unabhängig davon, welche Programmiersprache Sie nutzen. Zudem bieten unsere handprogrammierten Beispiele die Möglichkeit, nachzuvollziehen, was bei Keras und PyTorch hinter den Kulissen passiert.

1.7 Begriffliche Abgrenzung und Fachbegriffe

Wir haben bei der Auswahl der Themen den Begriff *KI* bewusst sehr weit gefasst. Viele der vorgestellten Verfahren werden im strengen Sinne nicht – oder nicht mehr – zum Bereich der KI gezählt. So setzen einige Kapitel einen Schwerpunkt auf *Data-Mining*, also auf die Extraktion von Zusammenhängen aus Datenmengen, deren Umfang das menschliche Fassungsvermögen überschreitet. Die Grenzen zwischen Data-Mining und KI sind aber fließend. Kapitel 9, »K-means-Clustering«, zeigt mit dem k-means-Clustering-Algorithmus ein klassisches Data-Mining-Verfahren. Kapitel 12, »Neuronale Netze III: Fehler zurückverfolgen mit dem Neuronentrainer«, stellt die Verbindung zwischen k-means-Clustering und neuronalen Netzen her: Letztlich geht es in beiden Fällen um die Ziehung von Grenzlinien zwischen Datenpunktmengen.

Beim Schreiben dieses Buchs haben wir uns sehr viele Gedanken darüber gemacht, welche Fachbegriffe wir vorstellen und welche wir unerwähnt lassen. Einerseits helfen Fachbegriffe, Ordnung in ein Themengebiet zu bringen und Unterschiede zu markieren. Andererseits kann eine zu große Genauigkeit beim Einstieg eher abschrecken und den Blick auf das Wesentliche verstellen.

1.8 Inhalte, Themen, Kapitel

Im Folgenden skizzieren wir, was Sie in den einzelnen Kapiteln erwartet. Viele der Kapitel können Sie unabhängig voneinander lesen. Wo diese inhaltlich aufeinander aufbauen, haben wir das markiert.

In **Kapitel 2 bis Kapitel 4** stellen wir drei Verfahren vor, die sich besonders gut auf Texte anwenden lassen. Sie stehen in einem losen Zusammenhang, können aber auch unabhängig voneinander gelesen werden:

- Mittels sogenannter *Markow-Prozesse* lassen sich Texte im Hinblick auf die Häufigkeit bestimmter Buchstaben- und Wortfolgen untersuchen. Anschließend können wir diese Eigenschaften im Sinne einer Parodie reproduzieren, um abhängig vom Ausgangsmaterial etwa deutsch-, englisch- oder französischsprachigen Unsinn zu produzieren. Wir zeigen auch, wie Sie den Markow-Prozess zur Programmierung einer einfachen Textvervollständigung nutzen können, wie wir sie von unseren Smartphones kennen.
- Der *Levenshtein-Algorithmus* berechnet die Ähnlichkeit zweier Wörter, indem er die kürzeste Schrittfolge ermittelt, die nötig ist, um ein Wort in ein anderes zu verwandeln. Wir nutzen dieses Verfahren zur Programmierung einer einfachen Rechtschreibprüfung, die etwa das Wort »Schampingjong« zu »Champignon« korrigiert.
- Das Data-Mining-Verfahren *Assoziationsanalyse* findet inhaltliche Zusammenhänge in Playlisten, Warenkörben oder Texten. Unser Beispielprogramm »Begriffsnetz« navigiert durch ein Netz von Assoziationen, die wir aus Sendungsmanuskripten des Deutschlandfunks gewonnen haben. Das Programm lernt etwa, dass die Begriffe »Urknall«, »Weltformel« und »Relativitätstheorie« zusammengehören, ohne eine Idee von Astrophysik oder von überhaupt irgendeinem Thema zu haben.

Kapitel 5 und Kapitel 6 sind der klassischen Spiele-KI gewidmet. Sie bauen inhaltlich aufeinander auf und sollten daher unbedingt hintereinander gelesen werden:

- Wir haben uns extra für Sie, liebe Leserinnen und Leser, das Ein-Personen-Spiel »Fruchtkräsch« ausgedacht und programmiert. Dabei haben wir uns von Puzzlespielklassikern wie »Candy Crush« oder »Collapse« inspirieren lassen. Wir zeigen, wie eine KI mittels eines Verfahrens namens *Breitensuche* zukünftige Spielzustände vorausberechnen und auf diese Weise den vielversprechendsten Zug ermitteln kann. Sie können Ihre eigenen Fruchtkräsch-Skills direkt mit jenen unserer KI messen.
- Bei Spielen für zwei Personen reicht es nicht, die denkbaren Spielverläufe per Breitensuche zu berechnen, um den jeweils besten Zug zu finden. Die KI muss die den eigenen Interessen entgegengesetzten Absichten einer Gegnerin oder eines Gegners

berücksichtigen. Dies leisten die *Tiefensuche* und der *Minimax-Algorithmus*. Im Beispielprogramm können Sie »Reversi« gegen unsere selbst programmierte KI spielen, die Minimax und eine Optimierung namens *Alpha-Beta-Pruning* nutzt.

Kapitel 7 stellt einen grundlegenden Vertreter des *bestärkenden Lernens* vor:

- *Q-Learning* ist ein automatisches Lernverfahren, bei dem ein Agent (in unserem Fall ein Eichhörnchen) mit einer Umwelt (dem Garten) interagiert, um ein Ziel (das vergessene Nussdepot) zu erreichen. Wir führen den Spezialfall des Q-Learnings mittels *Q-Tabelle* im Detail vor.

In **Kapitel 8 und Kapitel 9** zeigen wir zwei Verfahren, die auf Geometrie beruhen. Sie bauen aufeinander auf und sollten daher gemeinsam gelesen werden:

- Der *k-nächste-Nachbarn-Algorithmus* ist ein grundlegendes Verfahren zur *Klassifikation* von Daten. Wir lernen, Daten als geometrische Punkte zu interpretieren. Dann können wir nicht klassifizierte *Datenpunkte* einer Klasse zuordnen, indem wir die Entfernungen zu anderen Datenpunkten berechnen, die bereits klassifiziert wurden. Wir führen das praktisch vor mit einer Anwendung, bei der wir anhand wissenschaftlich objektiv gemessener Werte für »Flauschigkeit« und »Niedlichkeit« entscheiden, ob wir es mit einem Häschen, einem Hai, einem Igel oder einer Vogelspinne zu tun haben.
- Beim *k-means-Clustering-Algorithmus* geht es ebenfalls um die Klassifizierung von Datenpunkten. Allerdings starten wir in diesem Fall mit einer vollständig unklassifizierten Punktmenge. Der Algorithmus ordnet die Datenpunkte in Klassen ein, indem er Gruppen von Punkten findet, die besonders nahe beieinanderstehen. Die praktische Anwendung gruppiert auf diese Weise Messdaten von Wetterstationen.

In **Kapitel 10 bis Kapitel 13** geben wir eine Einführung in die Programmierung *neuronaler Netze*. Dieser Abschnitt stellt inhaltliche Bezüge zu Kapitel 8 und Kapitel 9 her, die Sie also vorher gelesen haben sollten:

- Sie lernen, was ein *künstliches Neuron* ist, wie Sie aus Neuronen ein Netz spinnen und wie ein solches *künstliches neuronales Netz* Berechnungen durchführt. Bei dieser Gelegenheit treffen wir das »Häschenproblem« aus Kapitel 8 wieder und lösen es mittels eines neuronalen Netzes.
- Wir erklären, wie Netze per *überwachtem Lernen* darauf trainiert werden, ein bestimmtes Problem zu lösen, und stellen das *Gradientenabstiegsverfahren* vor, das wir für dieses Training benötigen.

- Das Beispielprogramm »Neuronentrainer« erlaubt es Ihnen, einem neuronalen Netz beim Lernen zuzuschauen, und lädt zu eigenen Experimenten ein.
- Im abschließenden Kapitel bieten wir eine flotte Übersicht weiterer Typen und Bauarten neuronaler Netze. Wir erklären Ihnen, was *Faltungsnetze* sind, und führen die Arbeit von *Filterkerneln* anhand eines praktischen Beispiels vor. Zudem skizzieren wir, was es mit *Autoencodern*, *GANs* und *Deep Q-Learning* auf sich hat. Zuletzt kommen wir auf die in dieser Einleitung aufgeworfene Frage nach einer allgemeinen künstlichen Intelligenz zurück.

Jetzt bleibt uns nur noch, Ihnen unterhaltsame und lehrreiche Stunden mit unserem Buch und den Beispielprogrammen zu wünschen. Möge Ihnen das eine oder andere Licht aufgehen! Wenn Sie bei der Lektüre auch nur annähernd so viel Spaß haben, wie uns das Schreiben gebracht hat, und wenn Sie nur einen Bruchteil von dem lernen, was wir beim Schreiben gelernt haben – dann dürfen wir uns sehr glücklich schätzen.

Kapitel 8
K-nächste-Nachbarn

Eine Tafel Schokolade hat mehr gemeinsame Eigenschaften mit einem Stück Torte als mit einer Karotte. Eine Tomate wiederum hat viel mehr Gemeinsamkeiten mit der Karotte als mit Schokolade. In diesem Kapitel zeigen wir, wie Sie solche Ähnlichkeiten und Nachbarschaftsverhältnisse in Zahlen fassen und diese nutzen können, um Ordnung in große Datenbestände zu bringen und Zusammengehörigkeiten darin zu erkennen.

Worum es in diesem Kapitel geht

- Zahlenförmige Daten lassen sich durch *Datenpunkte* darstellen.
- Distanzen zwischen Datenpunkten können wir mit einem einfachen geometrischen Verfahren berechnen.
- Der *k-nächste-Nachbarn-Algorithmus* nutzt solche Distanzmessungen, um Datenpunkte zu klassifizieren.

Angenommen, ein Datenbestand versammelt Nährwertangaben, die jeweils zwei Werte enthalten, einen für den Zucker- und einen für den Fettgehalt. Dann kann es sehr hilfreich sein, jedes einzelne dieser Wertepaare als einen *Datenpunkt* auf eine Fläche zu zeichnen. Dabei können wir den Zuckergehalt als Position des Punkts in der Waagerechten und den Fettgehalt als Position des Punkts in der Senkrechten nehmen.

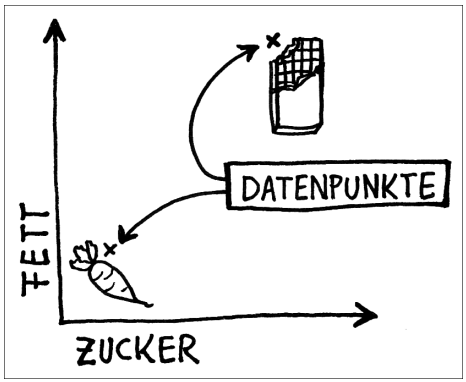


Abbildung 8.1 Messwerte lassen sich als Punkte darstellen.

Ein riesiger Vorteil dieser Darstellung: *Ähnlichkeit* zwischen Datenpunkten lässt sich jetzt *als räumliche Nähe* ablesen und berechnen: Je näher sich zwei Punkte auf der Fläche sind, umso so größer ist die Ähnlichkeit:

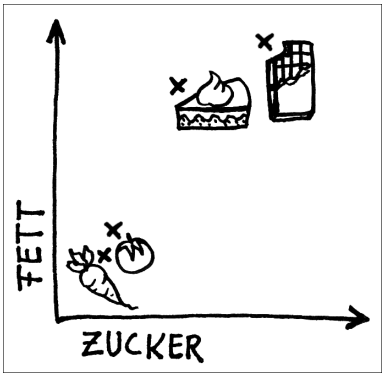


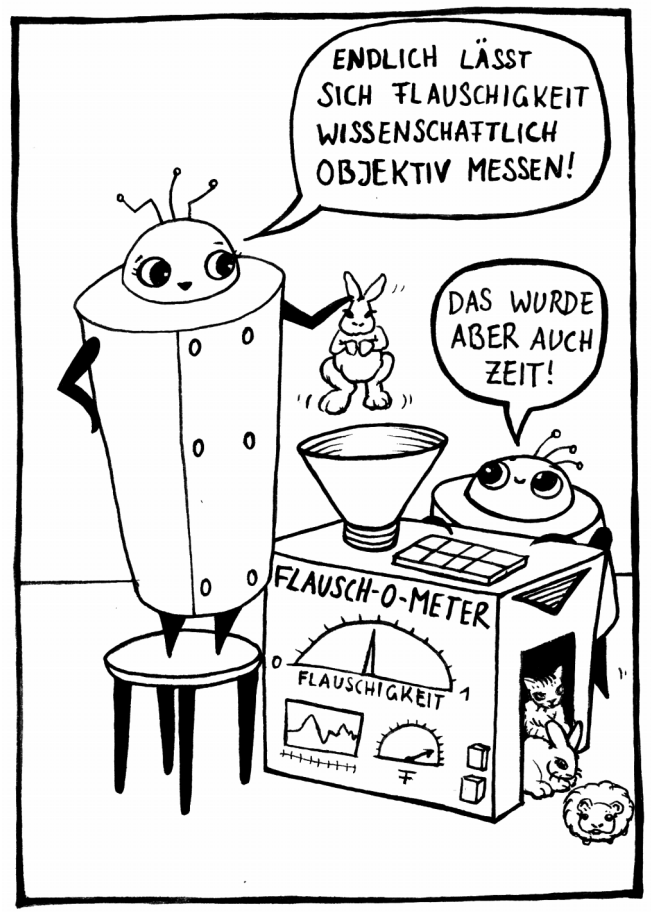
Abbildung 8.2 Räumlich nahegelegene Punkte haben ähnliche Eigenschaften.

Für dieses Kapitel haben wir uns ein alltagsnahes Anwendungsbeispiel ausgedacht, das insbesondere Tierfreundinnen und -freunden anschaulich sein wird: Es geht um die Identifizierung von Tierarten anhand der Eigenschaften *Niedlichkeit* und *Flauschigkeit*.

8.1 Häschen, Igel, Vogelspinne oder Hai?

Das Beispiel geht von folgenden Annahmen aus:

- Sie haben zwei Messgeräte konstruiert, mit denen Sie die Eigenschaften *Flauschigkeit* und *Niedlichkeit* jeweils auf einer Skala von 0 bis 1 präzise messen können. Das Messverfahren ist selbstverständlich ganz und gar harmlos, es kitzelt höchstens ein wenig.
- Mittels dieser Geräte haben Sie jeweils fünf Tiere der Spezies Vogelspinne, Häschen, Hai und Igel vermessen und entsprechende Datenpunkte erzeugt: `{niedlichkeit: 0.90, flauschigkeit: 0.90, spezies: "haeschen"}`. Sie besitzen also einen Datenbestand mit 20 Datenpunkten.
- Es gibt Tiere, von denen Sie nicht wissen, zu welcher Spezies sie gehören. Anhand der Werte für *Flauschigkeit* und *Niedlichkeit* wollen Sie diese einer der vier Ihnen bekannten Spezies zuordnen. Für diese Zuordnung wollen Sie die räumliche Nähe zu bereits klassifizierten Datenpunkten nutzen.



8.2 Das Beispielprogramm Tiere erkennen

Das Programm »Tiere erkennen« finden Sie wie alle anderen hier besprochenen Beispielprogramme auf der Webseite zum Buch: *https://maschinennah.de/ki-buch*. Wenn Sie das Programm starten, sehen Sie die vorliegenden 20 Datenpunkte und deren Klassifizierung, eingetragen im Koordinatensystem. Die waagerechte Achse steht für *Niedlichkeit*, die senkrechte Achse für *Flauschigkeit*. Sie können an dieser grafischen Darstellung ihres Datenbestands Folgendes ablesen:

- Vogelspinnen sind flauschig, aber nicht besonders niedlich.
- Häschen zeichnen sich durch hohe Flauschigkeit und Niedlichkeit aus.

- Haie sind weder besonders flauschig noch allzu niedlich.
- Igel wiederum sind außerordentlich niedlich, eine ausgeprägte Flauschigkeit hingegen gehört nicht zu ihren hervorstechenden Eigenschaften.

Wir müssen hier den Einwand gelten lassen, dass in unserer Darstellung auch die Haie recht niedlich geraten sind. Zudem gibt es nachweislich Personen, die sogar Vogelspinnen süß finden.

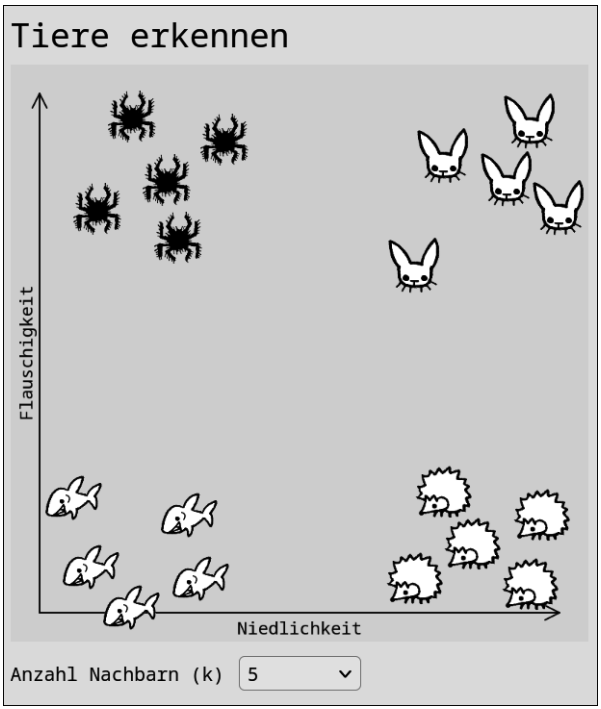


Abbildung 8.3 Das Beispielprogramm »Tiere erkennen«: Die 20 bekannten Datenpunkte sind hier abhängig von den Parametern »Niedlichkeit« und »Flauschigkeit« im Koordinatensystem angeordnet.

Mit dem Drop-down-Menü ANZAHL NACHBARN legen Sie fest, wie viele nächste Nachbarn zur Einordnung eines nicht klassifizierten Datenpunkts genutzt werden sollen.

Die Erzeugung eines neuen Datenpunkts ist ganz einfach: Klicken Sie mit der Maus auf eine beliebige Stelle im Koordinatensystem. Das Programm ordnet den neuen Datenpunkt sogleich einer Spezies zu. *Die Zuordnung passiert abhängig davon, welcher Spezies die meisten der benachbarten Datenpunkte angehören.* Abbildung 8.4, Abbildung 8.5 und Abbildung 8.6 zeigen Beispiele für solche Zuordnungen.

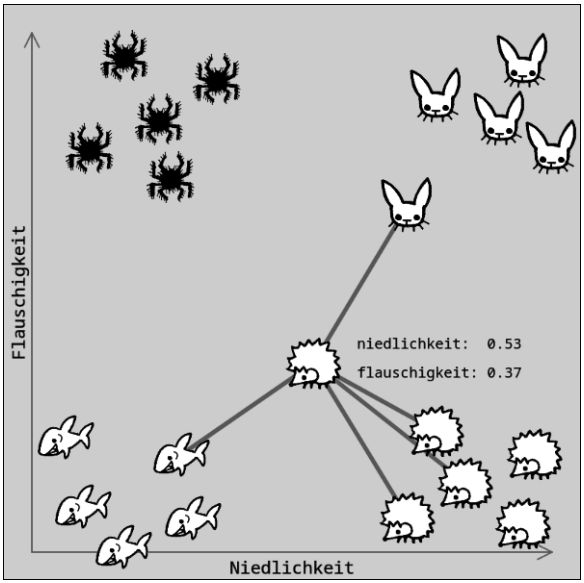


Abbildung 8.4 Das Programm ordnet unbekannte Datenpunkte den vier Spezies Vogelspinne, Häschen, Hai und Igel zu – hier wurde ein Igel erkannt, weil drei der fünf nächsten Nachbarn Igel sind.

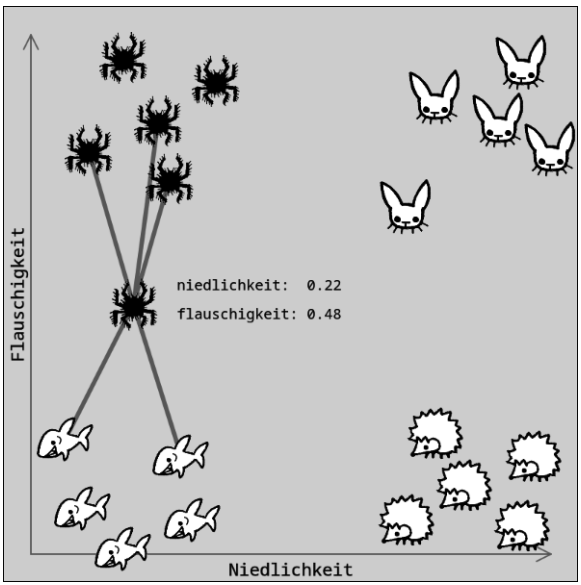


Abbildung 8.5 In diesem Fall hat das Programm eine Vogelspinne identifiziert, weil drei der fünf nächsten Nachbarn zu dieser Spezies gehören.

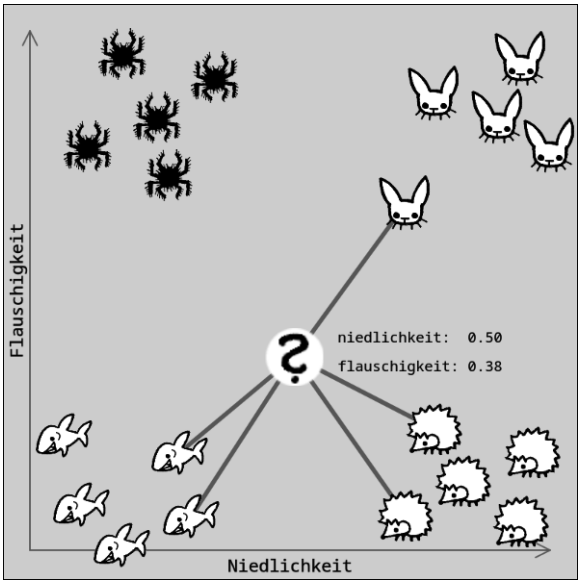


Abbildung 8.6 Weil es keinen eindeutigen »Sieger« gibt, hält sich das Programm mit einer Beurteilung zurück.

8.3 Entfernungen bestimmen mit Pythagoras

Jetzt kennen Sie das Prinzip des k-nächste-Nachbarn-Algorithmus. Es bleibt zu klären, *wie* wir die Entfernung zwischen zwei Datenpunkten bestimmen. Hier hilft uns ein alter Bekannter aus dem Matheunterricht weiter – der *Satz des Pythagoras*. Diese Gleichung macht eine einfache und nützliche Aussage über die Längenverhältnisse der Seiten eines rechtwinkligen Dreiecks. Dabei ist c die längste Seite, a und b die beiden kürzeren Seiten:

$$a^2 + b^2 = c^2$$

Abbildung 8.7 zeigt, wie Sie durch eine einfache Umformung dieser Gleichung die Länge von c berechnen können, wenn Ihnen die Längen von a und b bekannt sind.

In der Abbildung haben wir das Dreieck abweichend von der üblichen Darstellung des Pythagoras-Satzes so gedreht, *dass der rechte Winkel des Dreiecks unten links steht*. Damit wollen wir deutlich machen, dass wir die Formel zur Berechnung der Länge von c

auch nutzen können, um die Entfernung zwischen zwei Punkten P1 und P2 im Koordinatensystem zu ermitteln: Wir benennen die Seiten *a*, *b* und *c* schlichtweg um:

- Aus *a* wird die Distanz der beiden Punkte auf der x-Achse: Δx.
- Aus *b* wird die Distanz der beiden Punkte auf der y-Achse: Δy.
- Aus *c* wird die Distanz zwischen den beiden Punkten P1 und P2.

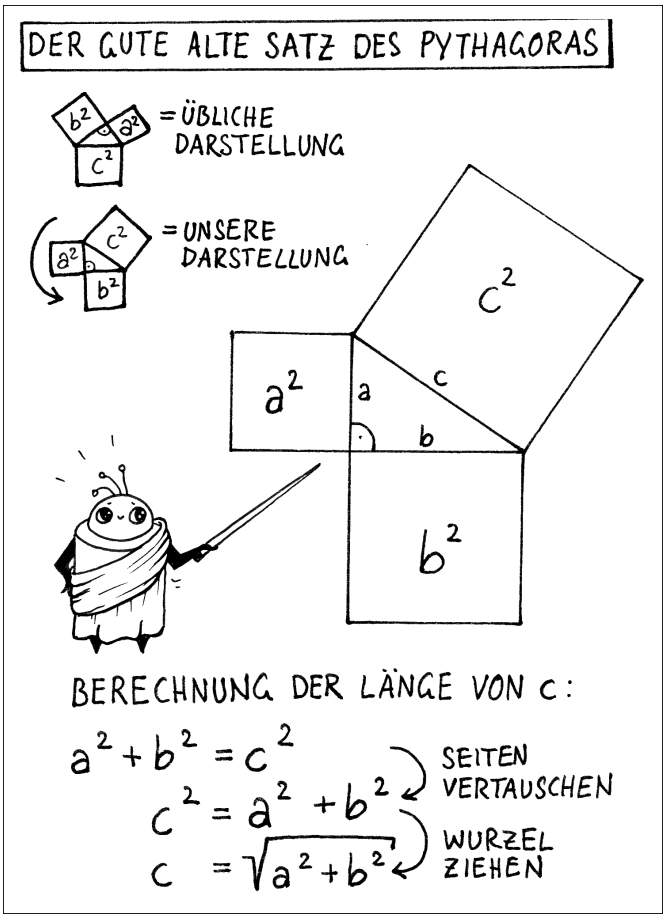


Abbildung 8.7 Mit dem Satz des Pythagoras können wir die Länge der Seite c berechnen …

Das Zeichen Δ ist der griechische Buchstabe Delta, der in Formeln und Gleichungen häufig für Differenzen steht.

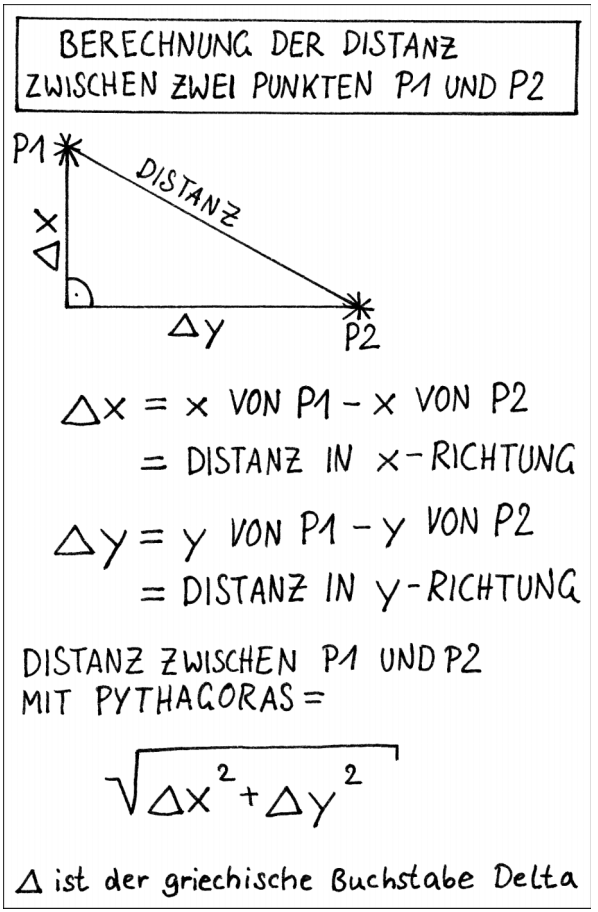


Abbildung 8.8 ... und dies für die Ermittlung der Distanz zwischen zwei Punkten nutzen.

8.4 Der Code im Detail

Im Code unseres Beispielprogramms sind Objekte der Klasse `KNN` für die Umsetzung des **k**-**n**ächste-**N**achbarn-Algorithmus zuständig. Sie besitzt eine Eigenschaft und vier Funktionen, die in Tabelle 8.1 aufgelistet sind.

Name	Aufgabe

`datenpunkte`	Ein Array mit den bereits klassifizierten Datenpunkten in diesem Format:
	`{niedlichkeit: 0.90,`
	`flauschigkeit: 0.90,`
	`spezies: "haeschen"}`

`klassifiziere(datenpunktA, k)`	Klassifiziert einen Datenpunkt und liefert diesen zurück. `k` bestimmt die Anzahl der Nachbarn, die zur Klassifizierung genutzt werden. Ruft die Funktionen `nachbarn()` und `zaehlung()` auf.
`nachbarn(datenpunktA, k)`	Liefert ein Array mit den `k` nächsten Nachbarn von `datenpunktA`. Ruft die Funktion `distanz()` auf.
`distanz(datenpunktA, datenpunktB)`	Ermittelt die Distanz zwischen `datenpunktA` und `datenpunktB`.

`zaehlung(nachbarn)`	Zählt die Häufigkeit der einzelnen Spezies in dem Datenpunkt-Array `nachbarn`.
	Rückgabewert ist eine Rangfolge im folgenden Format:
	`[["igel", 3],`
	`["hai", 1],`
	`["haeschen", 1]]`

Tabelle 8.1 Eine Eigenschaft und vier Funktionen der Klasse KNN

Listing 8.1 zeigt die Funktion zur Berechnung der Distanz zwischen zwei Datenpunkten, so wie wir sie in Abbildung 8.8 dargestellt haben. `deltaX` und `deltaY` sind die Distanzen zwischen den beiden Punkten in x- und y-Richtung. Die beiden Deltas werden anschließend quadriert und summiert. Die Quadratwurzel des Ergebnisses ist die Distanz:

```
distanz(datenpunktA, datenpunktB) {
  const deltaX = datenpunktA.niedlichkeit - datenpunktB.niedlichkeit;
  const deltaY = datenpunktA.flauschigkeit - datenpunktB.flauschigkeit;
  return sqrt(deltaX * deltaX + deltaY * deltaY);
}
```

Listing 8.1 Die Funktion knn.distanz() berechnet die Entfernung zwischen zwei Datenpunkten.

Die Funktion `nachbarn()` liefert die `k` nächsten Nachbarn zu einem gegebenen `datenpunktA` – im Fall unserer Anwendung ist `datenpunktA` ein noch nicht klassifizierter Punkt. Um die Nachbarn zu finden, sortiert `sort()` die `datenpunkte` abhängig von der Distanz zu `datenpunktA`. Anschließend wählt `slice()` die ersten `k` Datenpunkte aus:

```
nachbarn(datenpunktA, k) {
  this.datenpunkte.sort((L, R) =>
    this.distanz(datenpunktA, L) - this.distanz(datenpunktA, R));
  return datenpunkte.slice(0, k);
}
```

Listing 8.2 Die Funktion knn.nachbarn() liefert benachbarte Datenpunkte.

Jetzt fehlt noch die Auswertung des Rückgabewerts von `nachbarn()`: Sind die meisten der benachbarten Datenpunkte Vogelspinnen, Häschen, Haie oder Igel? Diese Auswertung liefert die Funktion `zaehlung()`.

Eine `for`-Schleife durchläuft alle im Argument übergebenen Datenpunkte und zieht jeweils die `spezies` heraus. Ist die jeweilige `spezies` im Objekt `zaehlung` noch nicht als Schlüssel vorhanden, wird der Schlüssel angelegt und mit dem Wert 1 initialisiert. Andernfalls wird der unter diesem Schlüssel eingetragene Wert um 1 erhöht. Dies ähnelt übrigens dem Vorgehen, das wir in Abschnitt 2.2, »Der Code des Nonsense-Texters unter der Lupe«, angewendet haben, als es um die Zählung von Worthäufigkeiten im Rahmen eines Markow-Prozesses ging. Abschließend wird das Objekt `zaehlung` per `Object.entries()` in ein Array umgewandelt. Die Funktion `sort()` sortiert das Array nach Anzahl. Das sortierte Array ist der Rückgabewert.

```
zaehlung(nachbarn) {
  const zaehlung = {};

  for (let datenpunkt of nachbarn) {
    const spezies = datenpunkt.spezies;
    if (!zaehlung[spezies]) {
      zaehlung[spezies] = 1;
    } else {
      zaehlung[spezies] += 1;
    }
  }
```

```
  const alsArray = Object.entries(zaehlung);
  alsArray.sort((L, R) => R[1] - L[1]);
  return alsArray;
}
```

Listing 8.3 Die Funktion knn.zaehlung() liefert eine Rangfolge der Spezies der in der Nachbarschaft gefundenen Datenpunkte.

In der Funktion `klassifiziere()` greifen alle vorgestellten Komponenten ineinander. Die Funktion erwartet als erstes Argument einen nicht klassifizierten `datenpunktA`. Das zweite Argument `k` bestimmt die Anzahl der Nachbarn, die für die Klassifikation relevant sind.

Die eben vorgestellten Funktionen liefern die entsprechenden nächsten Nachbarn und deren Auswertung nach Häufigkeit. Dann bleibt nur noch eine Sache zu tun, nämlich, die Auswertung in eine erkannte Spezies zu übersetzen – oder eben nicht:

- Die Bedingung `zaehlung.length == 1` ist genau dann erfüllt, wenn alle Nachbarn derselben Spezies angehören. In diesem Fall ist die Zuordnung eindeutig.
- Ein wenig komplizierter wird es, wenn die Auswertung mehrere Einträge enthält. Ist die Häufigkeit des ersten Eintrags größer als die des zweiten, dann ist die Zuordnung ebenfalls klar: Die Spezies des vorne stehenden Eintrags wurde identifiziert.
- Wenn keine dieser beiden Bedingungen erfüllt ist, gibt es keinen klaren Spitzenreiter: Die Spezies von `datenpunktA` erhält den Wert `"unbekannt"`.

```
klassifiziere(datenpunktA, k) {
  const nachbarn = this.nachbarn(datenpunktA, k);
  const zaehlung = this.zaehlung(nachbarn);

  if (zaehlung.length == 1 || zaehlung[0][1] > zaehlung[1][1]) {
    datenpunktA.spezies = zaehlung[0][0];
  } else {
    datenpunktA.spezies = "unbekannt";
  }
  return datenpunktA;
}
```

Listing 8.4 Die Funktion knn.klassifiziere() ordnet den unbekannten datenpunktA einer Spezies zu.

8.5 Ideen zum Weitermachen

Der k-nächste-Nachbarn-Algorithmus ist nicht besonders anspruchsvoll. Es sollte nicht schwer sein, unser Beispielprogramm zu modifizieren.

Sie könnten etwa eine weitere Tierart in das Programm aufnehmen. Hierfür sind lediglich zwei Ergänzungen notwendig:

- Zeichnen Sie das Tier und speichern Sie die Zeichnung im Ordner `daten` unter dem entsprechenden Namen. Ideal ist hier das PNG-Format mit einem Alpha-Kanal, also einem durchsichtigen Hintergrund.
- Schreiben Sie in die Datei *daten/datenpunkte.js* einige Datenpunkte, die Vertreter der neu hinzugefügten Tierart repräsentieren.

Alternativ könnten Sie Tiere auch nach anderen Eigenschaften ordnen: Wie wäre es, wenn sie Tiere nach Gewicht und Zutraulichkeit charakterisierten? Sie könnten auch ein ganz anderes Thema bearbeiten, etwa indem Sie Nahrungsmittel nach ihrem Fett- und Zuckergehalt klassifizieren. Überlegen Sie dabei, welche Eigenschaften sich überhaupt sinnvoll auf einer Skala eintragen lassen.

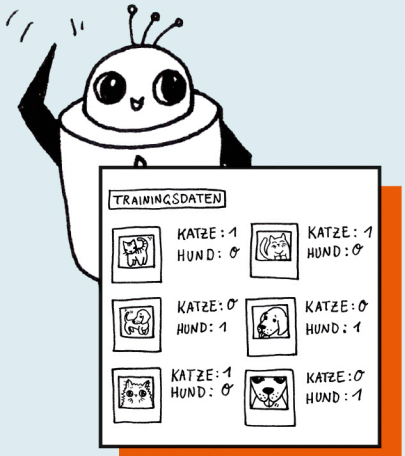
8.6 Zusammenfassung und Ausblick

Der k-nächste-Nachbarn-Algorithmus ist ein einfaches Verfahren, um zahlenförmige Daten zu klassifizieren. Der Cartoon mit der falsch klassifizierten Katze zeigt die Grenzen auf: Es können nur Klassen erkannt werden, die schon vorher bekannt waren. Vielen ist die Erfahrung geläufig, von einem Computerprogramm oder einem formalen Prozess in ein Schema hineingepresst zu werden, das offensichtlich nicht passt.

Die in Abschnitt 8.3, »Entfernungen bestimmen mit Pythagoras«, vorgestellte Formel eignet sich ebenfalls für Daten mit mehr als zwei Dimensionen. Auch wenn wir uns das bildlich nur schwer vorstellen können: Wenn die Nährwertangaben eines Fertiggerichts jeweils Werte für Kohlenhydrate, Fett, Eiweiß, Ballaststoffe und Salz enthalten, können wir jedes Fertiggericht als Datenpunkt in einem fünfdimensionalen Raum behandeln und damit rechnen. Die Formel zur Berechnung der Entfernung wird nach demselben Prinzip aufgebaut.

Inhalt

Künstliche Intelligenz verstehen



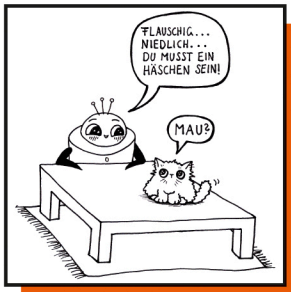
Hereinspaziert! Über ein Dutzend interaktive Projekte warten auf Ihre Neugierde. Schauen Sie zu, wie ein Algorithmus Gedichte schreibt, Wetterdaten gruppiert oder Reversi lernt – und spielen Sie gleich mit.

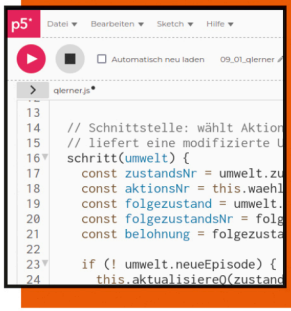
Im Buch erfahren Sie, was jeweils dahinter steckt. Lernen Sie einschlägige KI-Verfahren praktisch kennen und programmieren Sie selbst: ohne Installation im einsteigerfreundlichen Online-Editor.

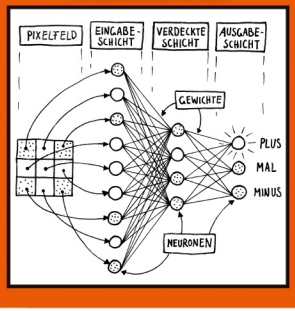
+ Grundlagen aus der Informatik
+ Spiele gewinnen
+ K-means-Clustering
+ Q-Learning
+ Neuronale Netze

Alle Beispielprogramme online:

+ Ausprobieren, mitspielen und mit dem Code experimentieren
+ Grafisch programmieren mit der JavaScript-Bibliothek p5.js







Das Autorenteam verbindet Kunst mit KI, Sound mit Grafik und Spaß mit Code. Sie arbeiten digital, auf Leinwand, im Projektraum Keller Drei in Hannover und in immer neuen Formaten. **Pit Noack** (Text & Code) begeistert in seinen Workshops für KI und Programmierung. Der Künstlerin **Sophia Sanner** (Illustration) ist kein Thema zu komplex, um es mit Cartoons und Infografiken auf humorvolle Weise anschaulich zu machen.

Rheinwerk Computing

€ 29,90 [D]
€ 30,80 [A]

ISBN 978-3-8362-8467-7
KI

9 783836 284677