Frochte

Maschinelles Lernen

Ihr Plus – digitale Zusatzinhalte!

Auf unserem Download-Portal finden Sie zu diesem Titel kostenloses Zusatzmaterial. Geben Sie dazu einfach diesen Code ein:

plus-iq94e-trn15

plus.hanser-fachbuch.de

Bleiben Sie auf dem Laufenden!

Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter:

www.hanser-fachbuch.de/newsletter

Jörg Frochte

Maschinelles Lernen

Grundlagen und Algorithmen in Python

3., überarbeitete und erweiterte Auflage

HANSER

Autor:

Prof. Dr. rer. nat. Jörg Frochte
Hochschule Bochum
Arbeitsgruppe Angewandte Informatik und Mathematik

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über
http://dnb.d-nb.de abrufbar.

Internet: www.hanser-fachbuch.de

Lektorat: Dipl.-Ing. Natalia Silakova-Herzberg
Herstellung: Anne Kurth
Covergestaltung: Max Kostopoulos
Coverkonzept: Marc Müller-Bremer, www.rebranding.de, München
Titelbild: © shutterstock.com/studiostoks
Satz: Jörg Frochte
Druck und Bindung: CPI books GmbH, Leck
Printed in Germany

Print-ISBN 978-3-446-46144-4
E-Book-ISBN 978-3-446-46355-4

Inhalt

1 Einleitung

Manche sagen, maschinelles Lernen sei ein Teilgebiet der künstlichen Intelligenz, andere ein Hilfsmittel in Disziplinen wie Data Mining oder Information Retrieval. Es hängt von der Sichtweise ab. Für mich ist es das Gebiet, das die interessantesten Aspekte zusammenbringt, nämlich Informatik, Mathematik und Anwendungen, die sehr vielfältig sind. Man kann hier mit Menschen zusammenarbeiten, denen es um autonomes Fahren oder um lernende Roboter in Industrie und Haushalt geht. Andere Anwendungsfelder sind beispielsweise das Verhalten von Menschen in Online-Shops oder Prognosen über Kreditwürdigkeit.
Das Feld ist aber deutlich breiter: Ein Kollege von mir setzt zum Beispiel in einem Projekt maschinelles Lernen ein, um Stimmen bedrohter Vögel in Neuseeland – *http://avianz.massey.ac.nz/* bzw. [PMC18] – zu erkennen und auch um zu ermitteln, wie viele Vögel wirklich auf einer Aufnahme zu hören sind. Das ist nicht leicht, denn es könnte dreimal derselbe Vogel sein, der ruft, oder eben drei verschiedene. Das ist maschinelles Lernen in der Ökologie. Andere Kollegen arbeiten mit Satellitendaten unterschiedlicher Qualität und versuchen so, Aussagen zu treffen, um die ausgebrachte Wasser- und Düngermenge zu optimieren. Das sind zwar alles sehr ernsthafte Fälle. Man darf aber auch einfach Spaß haben und versuchen, die KI in einem Computerspiel besser und unterhaltsamer zu machen. Das maschinelle Lernen ist also eine Art Schweizer Taschenmesser, welches man in den unterschiedlichsten Situationen sinnvoll und unterhaltsam einsetzen kann.

Ich versuche in diesem Buch, die meiner Ansicht nach wichtigsten Techniken und Ansätze darzustellen. Es enthält aber zunächst nur eines von diesen mitteldicken Schweizer Taschenmessern. Wenn Sie das Buch durchgearbeitet haben, schauen Sie mal nach folgenden Begriffen, die es nicht in die dritte Auflage geschafft haben: Outlier Detection, Radiale-Basisfunktionen-Netze, Self Organizing Maps, Recurrent Neural Networks …

Es gibt viele interessante Themen in diesem Feld. Am Ende des Buches haben Sie sich bestimmt die Grundlagen – und auch etwas mehr – angeeignet, um sich schnell in neue Bereiche einarbeiten zu können. Um die ganzen Werkzeuge in diesem Schweizer Taschenmesser sinnvoll nutzen zu können, sollte man einen breiten Überblick über das Gebiet haben und sich nicht auf eine Technik beschränken. Aktuell sind z. B. Convolutional Neural Networks sehr in Mode und natürlich kann man auch diese irgendwie benutzen, ohne verstanden zu haben, wie neuronale Netze überhaupt funktionieren. Ein Keras-Tutorial dazu kann man schnell abtippen und sehen, wie der eigene Computer Ziffern mit hoher Genauigkeit erkennt. Man muss zwar nicht immer alles durchdringen, aber wenn man sich dafür interessiert, will man dort nicht stehen bleiben, oder?
Ich verstehe zum Beispiel nicht viel von meinem Auto, weil es mich nicht interessiert. Meine Ignoranz funktioniert hervorragend, weil das Produkt recht gut entworfen ist und ich nicht den Wunsch habe, an ihm herumzuschrauben. Als reiner User fahre ich bei Problemen in eine Werkstatt. Ich gehe aber davon aus, dass Sie mit dem maschinellen Lernen mehr machen wollen als sich hineinzusetzen und loszufahren. Sie sind Designer von Lösungen, kreativer Kopf hinter neuen Anwendungen oder die Werkstatt, die sich um die rote Warnlampe kümmert.

Wer hier nur ein Werkzeug kennt, läuft in eine Falle, die als *Law of the instrument* oder auch **Maslows Hammer** nach dem Ausspruch *If all you have is a hammer, everything looks like a nail* bekannt ist. Um hier für unterschiedliche Probleme mit großen und kleinen Datenmengen ebenso wie für unterschiedliche Ressourcenlagen Lösungen anbieten zu können, versuchen wir es mit vielen Werkzeugen. Das Ziel ist es, beim Kennenlernen dieser Werkzeuge zwischen den beiden Monstern Skylla (Theorielastigkeit) und Charybdis (flache Unbedeutendheit) möglichst unbeschadet hindurch zu kommen. Das bedeutet, ich möchte, dass Sie am Ende des Buches die mathematischen Hintergründe dieser Technik im Wesentlichen kennen, aber nicht notwendigerweise in der trockenen Form aus Definition, Satz und Beweis. Ich bin überzeugt, dass ein guter Mittelweg darin besteht, möglichst viele Algorithmen aus dem Bereich einmal selbst umzusetzen. Benutzt man nur fertige Bibliotheken, ist es etwa so, als würde jemand glauben, vom Zusehen schwimmen gelernt zu haben. Ich möchte also mit Ihnen gemeinsam wirklich *schwimmen* und Algorithmen basierend auf Verständnis und Theorie umsetzen. Dabei ist es in Ordnung, wenn unsere Umsetzungen nicht das Rennen bzgl. der Performance gewinnen. Es geht darum, die Prinzipien und theoretischen Grundlagen einmal ausprobiert zu haben. Allerdings soll das kein fundamentalistisches Dogma sein. Wer glaubt, er habe den Ansatz gefunden, der immer und für jeden funktioniert, ist im besten Fall eine Gefahr für sich und im schlimmsten für andere Menschen.
Es gibt ein paar Stellen, an denen wir mit der Idee, die Dinge from-scratch umzusetzen, nicht weiterkommen würden: tiefe neuronale Netze (Deep Learning) und Support Vector Machines. Beide benötigen mehr Wissen und Software rund um Optimierung und Co. als in dieses Buch passt. Gleichzeitig sind sie sehr spannend, sodass man sie nicht einfach weglassen sollte, nur weil die Umsetzung nicht gut auf ein paar Buchseiten passt. Wir setzen daher die neuronalen Netze in ihrer klassischen Form zu Fuß um und gehen dann für die tiefen dazu über, Keras als Bibliothek zu verwenden. Im Zusammenhang mit klassischen Netzen handeln wir fast alle Fallen und Begriffe ab und gehen dann mit Keras zu Anwendungen über, die mehr Leistung oder bessere Optimierung brauchen. Dasselbe gilt in gewisser Weise für die Support Vector Machines, die ebenfalls ein Modul aus der quadratischen Optimierung benötigen; nur weglassen sollte man sie nicht. Hier schauen wir uns erst etwas theoretischer die Grundlagen an und greifen dann zum Ausprobieren zur fertigen Umsetzung aus scikit-learn.

Weil wir abseits dieser beiden Fälle tendenziell über Algorithmen gehen, versuchen wir vorzugsweise, einen eher algorithmischen statt einen statistischen Zugang zu nehmen. Das liegt auch daran, für welche Zielgruppe ich gewöhnlich maschinelles Lernen aufbereite. Ich selbst unterrichte das Fach für Ingenieure oder angewandte bzw. technische Informatiker. Die Ingenieure in der Praxis oder an der Hochschule sind oft mit MATLAB vertraut und haben dort ggf. bereits etwas mit maschinellem Lernen versucht. Die Informatiker in den Studiengängen hatten Java, C und ggf. MATLAB. Das meiner Meinung nach beste und kostengünstigste Ökosystem zum maschinellen Lernen findet man jedoch bei Python oder R. Letzteres ist gerade bei Statistikern sehr beliebt. Für Ingenieure ziehe ich Python R vor; u. a. wegen dessen besserer Einbindung, auch in Robotik-Projekten, und wegen des leichten Umstiegs. Der Sprung von MATLAB zu Python ist gering, muss aber immer gemacht werden. Ziemlich genau diese Sprunghöhe, die jemand schaffen muss, der von MATLAB bzw. GNU/Octave zu Python wechseln möchte, ist auch im Buch eingebaut. Es funktioniert aber auch, wenn jemand von Java oder C++ kommt.

Daneben sind Ingenieure oder Ingenieurinformatiker oft sehr gut ausgebildet in linearer Algebra und Analysis, dafür fehlt die Statistik in der Ausbildung. Daher habe ich auch die Statistik

in der Darstellung des maschinellen Lernens möglichst knapp gehalten und den wirklich notwendigen Teil der Mathematik ins Buch integriert.

Die Analysis und lineare Algebra – in Kapitel 5 – sind in weiten Teilen eingebettet, wenn auch teilweise auf dem Niveau einer Erinnerung.

Im Buch baue ich die Quelltexte immer (fast) vollständig ein. Sie können die Quellen natürlich auch von meiner Seite *https://joerg.frochte.de* downloaden, aber mir ist es wichtig, dass der Python-Code wirklich ins Buch eingebunden ist. Wenn man einen algorithmischen Zugang versucht, sind die Algorithmen eben wesentlich. Außerdem möchte ich gerne, dass man das Buch sowohl vor dem Computer benutzen kann – direkt alles mitmachen und ausprobieren – als auch in einem Park sitzend und nur lesend. Ich selbst lese gerne auch gedruckte Fachbücher als Entspannung, wenn ich gerade keinen Bildschirm sehen will … und ich vermute, ich bin damit nicht allein. Man kann Algorithmen in Python tatsächlich sehr kompakt notieren und auf die wesentlichen Ideen beschränken, was Python ebenfalls für den Abdruck interessant macht. Bezüglich des Codes wird jemandem, der Python schon länger benutzt, etwas auffallen: Ich benutze kein snake_case, was in Python ungewöhnlich ist. Gründe sind sicherlich, dass meine Kursteilnehmer von C, MATLAB oder Java kommen, ich selbst auch oft zwischen diesen Programmiersprachen wechsle und mir dabei eine Art Crossover-Stil angewöhnt habe. Ich hoffe, es stört niemanden, der an sauberes PEP 8 gewöhnt ist, und bitte falls doch um Nachsicht. Der Stil für die Variablennamen ist hier aber nicht so wichtig. Wir haben es die meiste Zeit mit sehr kurzen Variablen zu tun, wie X für die Trainingsmenge und Y für die Menge der Ziele usw. Da wölbt sich keine Schlange und macht kein Kamel einen Höcker.

In den Python-Codes, die einen wichtigen Teil des Buches ausmachen, müssen wir ohne Pfeile oder Fettdruck für Vektoren und Matrizen auskommen. Entsprechend lassen wir dies auch für die Formeln weg und bemühen uns, so zu klären, was was ist. Formeln haben immer dann eine Nummer, wenn auf diese später noch einmal verwiesen wird. Gleichheitszeichen im Pseudocode sind i. d. R. als *gleichgesetzt*, also als Zuweisung, zu lesen.

Im Buch sind drei Arten von „Boxen“ eingebaut:

Diese hat etwas damit zu tun, wo Sie in **scikit-learn** den Algorithmus, den wir gerade umgesetzt haben, finden können. Es ist lediglich ein Verweis auf professionelle Umsetzungen. **Keras** und scikit-learn sind für mich zwei sehr wichtige Stützpfeiler, um schnell und gut etwas in Python umsetzen zu können. Ich habe die Verweise auf diese beiden Bibliotheken beschränkt.

Die zweite Textbox ist ein allgemeines *Achtung*. Ich setze diese Box nicht so oft ein, weil irgendwie alles oder nichts wichtig ist. Aber manche Dinge sind doch ärgerlicher als andere und kosten sehr viel Zeit, wenn man sie überliest. Wenn ich einen dieser Fälle bereits erlebt habe, taucht diese Box auf.

Die wichtigste Gruppe von Boxen erkennt man an dem Schraubenschlüssel. Hier gibt es Anregungen, was Sie selbst anschließend ausprobieren könnten. Keine von diesen ist nötig, um dem Rest des Buches zu folgen. Es sind einfach Vorschläge, da-

mit Sie selbst etwas mit dem neu Gelernten anfangen können, ohne dass die Lösung sofort danebensteht. Oft gibt es nicht DIE Lösung, sondern eben sehr viele Ansätze.

Wenn ich eine Variable aus einem Quellcode im Fließtext verwende, wird diese als `Schreibmaschinenschrift` gesetzt. Dinge, die fettgedruckt sind, tauchen im Index hinten auf. Der Sinn liegt darin, dass Sie etwas hinten im Index suchen und dann auf der Seite sofort sehen, wo es steht. In der Regel wird etwas nur beim ersten Auftauchen in den Index aufgenommen. Ansonsten bedeutet ein kursiver Druck etwas Ähnliches wie *Eigenname* oder *In-Anführungszeichen*. Der Einstieg in Python ist in Kapitel 3 konzentriert. Wer Python zusammen mit NumPy & Co. schon beherrscht, kann das Kapitel überspringen. Alle anderen erhalten in Kapitel 3 einen schnellen praktischen Einstieg, wie mit Matrizen und Arrays in NumPy gearbeitet wird. Im Unterschied zu Python als Grundlage habe ich beim Aufbau des Buches versucht, auch die Einarbeitung der mathematischen Grundlagen über mehrere Kapitel zu verteilen und nicht in einem Einstiegskapitel oder Anhang zu bündeln; einfach damit es nicht einen langen trockenen Teil gibt und dann viele Passagen, die quasi primär aus Pseudo- bzw. Quellcode bestehen. In Kapitel 4 folgt zusammen mit dem ersten Klassifikator ein guter Teil der minimalisierten statistischen Grundlagen, die wir brauchen. In Kapitel 5 gehen wir noch einmal tiefer auf Vektorräume, Normen und Metriken ein. Diese sind u. a. notwendig, wenn man hinterher, wie in Kapitel 13, versucht, Grade von Ähnlichkeiten zwischen Objekten zu ändern, und zwar durch die Art, wie Abstände definiert werden. In den Kapiteln 7 und 8 ist wiederum etwas Optimierung eingebaut, welche nötig ist, um neuronale Netze zu trainieren.

Die Auswahl von Merkmalen und ihre Reduktion sind das Thema des Kapitels 9. Hier brauchen wir noch einen Nachschlag bzgl. der Statistik und dazu Grundlagen zu Eigenwerten und Eigenvektoren. Diese sind nötig, um die Principal Component Analysis richtig einzuordnen. Vielleicht etwas ungewöhnlich ist die Lage des Kapitels 9 innerhalb des Buches. Man könnte eigentlich annehmen, dass die Diskussion über die Daten weiter vorne kommen sollte. Jedoch wollte ich für einige Demonstrationen schon Lernverfahren zur Verfügung haben, wozu neuronale Netze gehören, da diese z. B. andere Ansprüche haben als ein Entscheidungsbaum. Neu in der dritten Auflage ist, dass ich versuche, in dieses Kapitel Pandas als wohl wichtigste Bibliothek im Umgang mit strukturierten Daten einzubinden.

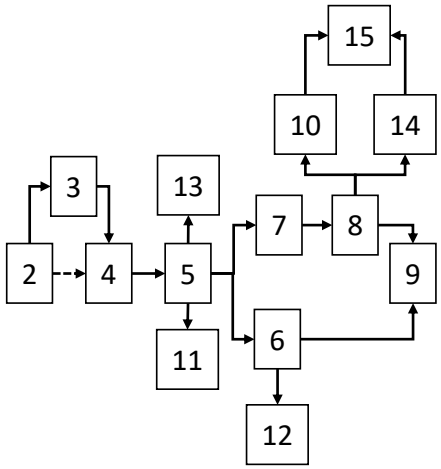

Abbildung 1.1 Abhängigkeiten zwischen den Kapitel des Buches

Wer das Buch in einer Vorlesung einsetzen will und kürzen möchte bzw. muss, für den habe ich die Abbildung 1.1 eingebaut. Sie gibt die wesentlichen Abhängigkeiten wieder. Es kann immer mal irgendwo ein Verweis auch außerhalb dieser Abhängigkeiten auftauchen. Daneben gibt es in jedem Kapitel viele Möglichkeiten zu kürzen, abhängig von den Vorkenntnissen der Teilnehmerinnen und Teilnehmer oder den Zielen zu kürzen. Wer z. B. eigentlich bestärkendes Lernen mit neuronalen Netzen machen möchte, der kann sich im Kapitel 8, auf die ersten zwei beiden Abschnitte konzentrieren. Generell kann man dann vieles auf dem Weg anpassen oder weglassen; dafür sind Dozentinnen und Dozenten eben wichtig.

Jetzt sollten wir aber anfangen, nachdem ich mich bei einigen Menschen bedankt habe. Das sind im Vergleich zur letzten Ausgabe einige mehr geworden. Ich möchte mich wirklich bei Lesern bedanken, die mir Anregungen schicken oder Stellen nennen, bei denen die Erklärungen im Buch nicht rundlaufen. Diese haben mich – ebenso wie die Arbeit zusammen mit Herrn Kaufmann am Projekt Weiterbildung AI (*https://we-ai.de*) – sehr weitergebracht bzgl. der Frage wo man noch was ändern oder ergänzen müsste. Neben den Leserinnen und Lesern, die mir konstruktive Rückmeldungen per E-Mail geben, haben mir auch viele Kolleginnen und Kollegen geholfen. Ich vergesse bestimmt jemanden und möchte mich dafür entschuldigen, jedoch unter anderem möchte ich mich bedanken bei Sabine Weidauer, Benno Stein, Matthias Rottmann, Peter Gerwinski, Herbert Schmidt, Michael Knorrenschild, Peter Beater, Christof Kaufmann, Henrik Blunck, Marco Schmidt, Stefan Müller-Schneiders, René Schoesau, Stefan Bader, Roland Schroth, Annabel Lindner, Julia Sudhoff und Gernot Kucera.

Wenn man ein Buch schreibt, kostet das immer viel zusätzliche Zeit neben dem normalen Beruf. Daher vielen Dank, dass ich mir diese nehmen durfte, an meinen Sohn Laurin und meine Frau Barbara. Letzterer genau wie den Korrektor und Lektorin des Hanser-Verlags vielen Dank für die Anregungen und Überarbeitungen.

Auch für die Zukunft freue ich mich auf jede Rückmeldung an joerg@frochte.de – und nun los!

2 Maschinelles Lernen – Überblick und Abgrenzung

Bevor wir uns mit den einzelnen Techniken, Methoden und Algorithmen beschäftigen, soll es in diesem Kapitel erst einmal darum gehen, einen Überblick zu bekommen. Das bedeutet, wir werden versuchen, eine Taxonomie der Techniken im maschinellen Lernen zu erarbeiten und das maschinelle Lernen im Kontext von *Künstlicher Intelligenz* sowie *Data Mining* bzw. *Knowledge Discovery in Databases* einzuordnen.

2.1 Lernen, was bedeutet das eigentlich?

Vor einer Klärung, was maschinelles Lernen ist, müsste man sich zunächst darüber klar werden, was wir unter Lernen an sich verstehen. Irgendwie hat jeder, der durch Schule und Hochschule gegangen ist, eine Vorstellung davon. Aber immer, wenn ich spontan nachfrage, gehen die Vorstellungen doch sehr auseinander. Eine Idee ist, in diverse Lexika zu schauen. Dabei bin ich auf eine Menge unterschiedlicher Antworten gestoßen. Hier einmal drei zur Auswahl:

> *Jede Form von Leistungssteigerung, die durch gezielte Anstrengung erreicht wurde.*

> *Jede Verhaltensänderung, die sich auf Erfahrung, Übung oder Beobachtung zurückführen lässt.*

> *Durch Erfahrung entstandene, relativ überdauernde Verhaltensänderung bzw. -möglichkeiten.*

Im Prinzip gilt für die Maschine dasselbe, was diese Lexika-Definitionen nahelegen. Die Maschine bzw. das Computerprogramm – oft ein Agent, wie wir ihn in Kapitel 14 kennenlernen werden – soll aus Erfahrungen lernen und somit Verhaltensänderungen bei ihm bewirken.

Statt Maschinen statisch zu programmieren, wollen wir Techniken einsetzen, mit deren Hilfe unsere Computer ein Verhalten aus Daten lernen. Diese Daten stellen die Erfahrungen dar, welche die Maschine macht. Damit ist nicht automatisch gesagt, dass die Maschine immer weiterlernt. Es ist auch denkbar, dass wir ein Verhalten einmal mithilfe von Daten lernen bzw. trainieren und es dann einfrieren. Das sind bewusste Entscheidungen, die Entwickler treffen. Auch ist Lernen damit nicht identisch mit Intelligenz oder Bewusstsein. Eine Maschine kann sehr gut darin sein, ihr Verhalten z. B. bzgl. der Reinigung unserer Wohnungen zu verbessern und dabei nicht einen Krümel Intelligenz oder Bewusstsein besitzen. Sie verändert nur ihr Verhalten in der Umgebung auf der Basis von Algorithmen und Daten; man spricht davon, das Verhalten an die Umgebung zu adaptieren.

Eine genaue und allgemein verbindliche Eingrenzung des Begriffes *Maschinelles Lernen* ist nicht leicht und in der Literatur nicht einheitlich. Das kommt daher, weil das maschinelle Ler-

nen für viele eher ein Werkzeugkasten ist, den sie im Rahmen des sogenannten Data Minings bzw. der Knowledge Discovery in Databases einsetzen. Andere wiederum sehen es als Teil der künstlichen Intelligenz, und entsprechend schauen die Weisen sehr unterschiedlich auf den gleichen Elefanten.

2.2 Künstliche Intelligenz, Data Mining und Knowledge Discovery in Databases

Wenn man von **künstlicher Intelligenz** spricht, steht man vor dem Problem, dass ohne das Adjektiv *künstlich* Intelligenz nicht im Sinne einer mathematischen Definition scharf definiert ist. Der Mensch geht davon aus, dass er – im unterschiedlichen Maße – intelligent ist und danach wird versucht, eine Definition zu erstellen.

Der Begriff *künstliche Intelligenz* spiegelt dabei den Menschen als Maßstab wider. Eine künstliche Intelligenz soll im Wesentlichen die gleichen intellektuellen Tätigkeiten wie ein Mensch ausführen können oder ihn dabei übertreffen. In der Forschung geht man davon aus, dass eine solche künstliche Intelligenz Folgendes leisten können sollte:

1. Logisches Denken
2. Treffen von Entscheidungen bei Unsicherheit
3. Planen
4. Lernen
5. Kommunikation in natürlicher Sprache

All diese Aspekte sollen eingesetzt werden können, um Ziele zu erreichen. Allgemein muss man sagen, dass der erste Punkt *Logisches Denken* zu den härtesten gehört und auch wegen des Wortes *Denken* am schwierigsten zu überprüfen ist.

Wo sind wir dort mit dem maschinellen Lernen zu finden? Nun, augenscheinlich dient das maschinelle Lernen dazu, den Punkt 4 *Lernen* zu bearbeiten. Die Algorithmen des maschinellen Lernens helfen zumindest auch beim Punkt 5 *Kommunikation* und sind ebenfalls in der Lage, den Punkt 3 *Planen* zu verbessern. Wie wir im Laufe des Buches noch sehen werden, fällt der Punkt 2 *Entscheidungen* auch fast vollständig in diesen Bereich. Bei so großen Überschneidungen ist es kein Wunder, dass die Begriffe oft durcheinandergeraten. Es ist allerdings nicht dasselbe, denn *Planen* oder *Entscheidungen bei Unsicherheiten treffen* kann man auch anders. Bis auf den Aspekt des Lernens selbst stellt das maschinelle Lernen oft Ansätze bereit, ist aber nicht der einzige Ansatz.

In der Aufzählung oben fehlen einige besonders schwer zu greifende Begriffe, die oft mit künstlicher Intelligenz verbunden werden, wie *Bewusstsein* und *Empfindungsvermögen*. Hier ist das maschinelle Lernen in dem mir bekannten Stand vollkommen außen vor und kann zumindest aktuell noch keinen Beitrag leisten; allenfalls es vortäuschen. Wenn eine Maschine die obigen Aspekte alle beherrschen würde, spräche man von einer **starken künstlichen Intelligenz**.

Die **schwache künstliche Intelligenz** hingegen beschränkt sich auf konkrete einzelne Anwendungsfelder – ist also keine universale Intelligenz – oder darauf, in gewissen Situationen intel-

ligent zu erscheinen. Letzteres, finde ich, macht sie dann wieder sehr menschlich, denn wer war noch nicht in der Lage, einmal schlauer aussehen zu müssen, als er vermutlich ist?

Der Turing-Test wird oft erwähnt, wenn es um die Überprüfung geht, ob eine Maschine intelligent ist. Er besteht im Wesentlichen daraus, dass ein Mensch nicht mehr in der Lage ist, zu erkennen, ob das Gegenüber bei einem Telefongespräch oder einem Chat eine Maschine oder ein Mensch ist. Hierzu gab es schon viele Tests und Programme, unter anderem **Cleverbot**, der auf Small-Talk spezialisiert ist und als Unterhaltungsmodul von **Hitchbot** diente, der als Anhalter in Kanada unterwegs war.

Die Frage, die Sie sich selbst beantworten müssen, ist, ob eine Maschine, die den **Turing-Test** besteht, eine starke oder eine schwache künstliche Intelligenz ist. Der amerikanischer Philosoph John Rogers Searle zum Beispiel geht davon aus, dass in diesem Fall nur eine Intelligenz vorgetäuscht wird, und hat das in einem Gedankenexperiment, welches als das *Chinesische Zimmer* bekannt ist, ausgearbeitet. Die Frage ist, wie man überhaupt eine starke Intelligenz testen könnte.

Schwache künstliche Intelligenzen, welche quasi Spezialisten auf genau ihrem Gebiet sind, werden jedenfalls aktuell rapide weiterentwickelt und dringen in Bereiche der Produktion, Planung etc. vor. Natürlich sind auch Computerspiele typische Einsatzfelder schwacher künstlicher Intelligenzen. Hierbei muss man zwei Anwendungsfälle unterscheiden: Einmal die KI, die in Spielen – in der Regel mit vollständiger Information wie Schach oder GO – immer besser werden und menschliche Spieler hinter sich lassen. Zum anderen aber die künstliche Intelligenz, die in Computerspielen die Rolle von sogenannten *Non-Player-Characters* (NPC) übernimmt. Hier geht es wieder im Wesentlichen darum, ein gewünschtes Verhalten nachzuahmen und nicht darum, besser als der Spieler zu sein. Wenn dort in einem Rollenspiel ein Dorfdepp vorkommt, soll er nicht zu clever sein, sondern wie ein Schauspieler seine Rolle spielen. Bei NPCs geht es also oft darum, zu unterhalten, was hier dann die eigentliche Leistung ist. Alles andere wäre ökonomisch nicht weise, denn nur wenige von uns sind so veranlagt, dass sie Geld ausgeben, um sich von einem Computer einmal richtig demütigen zu lassen ... die meisten wollen eher selbst gewinnen. Während die Techniken für die Bewältigung von Spielen wie GO in der Regel auf maschinelles Lernen aufbauen, ist dies bei den NPCs in Computerspielen nur sehr selten der Fall.

Fassen wir einmal zusammen, dass maschinelles Lernen und künstliche Intelligenz eine große Überschneidung haben, und viele Techniken des maschinellen Lernens im Bereich der künstlichen Intelligenz eingesetzt werden. Wie sieht es mit dem anderen großen Feld aus, also der **Knowledge Discovery in Databases**?

Um das besser zu verstehen, beginnen wir mit dem Prozess der Knowledge Discovery in Databases, wie er in Abbildung 2.1 dargestellt ist. Diese Version geht auf die Veröffentlichung [FPSS96] zurück.

Am Anfang steht immer eine Sammlung von Daten, die wir einfachheitshalber mit einer großen Datenbank darstellen. Knowledge Discovery in Databases (KDD) ist dabei der Prozess der (semi-)automatischen Extraktion von Wissen aus eben dieser Datenbank. Wie man an dem Begriff *(semi-)automatisch* erkennt, ist hier oft ein Mensch enger Begleiter des Prozesses, und der Prozess läuft nicht immer autonom ab. In seiner (semi-)automatischen Form ist der KDD-Prozess interaktiv und iterativ, was bedeutet, dass der Anwender Entscheidungen trifft und einige Schritte ggf. wiederholt werden müssen. In Abbildung 2.1 wird dies durch die Rückwärtspfeile nach jedem Hauptschritt angedeutet.

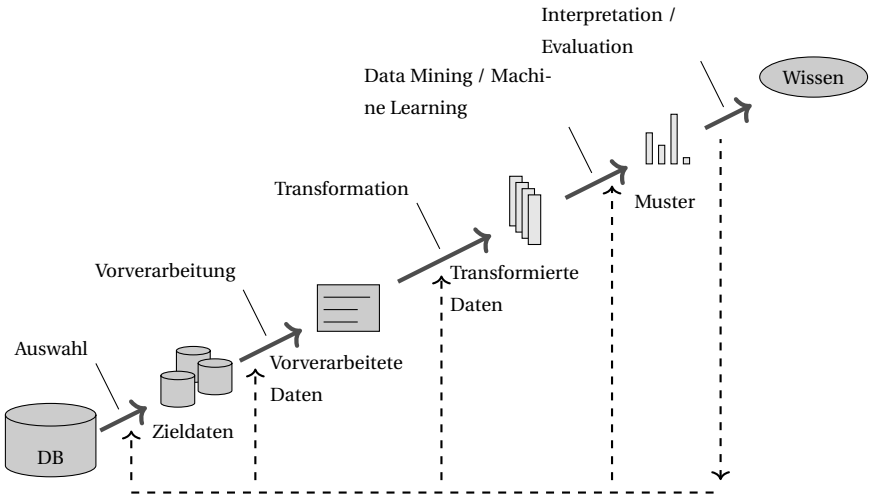

Abbildung 2.1 Knowledge Discovery in Databases als Prozess

Beim KDD-Prozess geht es zunächst darum, die Anwendungsdomäne zu verstehen und Daten für die definierten Ziele auszuwählen. Der Grund ist, dass die Ausgangsdatensammlung nicht speziell für unser Ziel angefertigt wurde, sondern eben viele Dinge enthält, die für uns keine Rolle spielen. Im nächsten Schritt der Vorverarbeitung werden die Daten bereinigt. Das meint unter anderem den Umgang mit fehlenden Daten, wie wir ihn in Abschnitt 9.2.2 diskutieren. Als Nächstes müssen die Daten transformiert und ggf. auch reduziert werden. Die Transformation kann dabei z. B. darin liegen, Daten, die als Strings vorliegen, numerischen Werten zuzuordnen, da die meisten Methoden auf numerischen Werten basieren und eben nicht auf symbolischen Größen. Das Reduzieren meint dann z. B., Daten zusammenzulegen, wie wir es in Kapitel 9 besprechen werden. Nehmen wir als Beispiel an, in einer Datenbank sind Informationen zu Länge, Breite und Gewicht eines Fahrzeuges enthalten. Wollen wir nicht mit so vielen Merkmalen arbeiten, fassen wir diese drei Merkmale irgendwie zu einem Meta-Merkmal *Groß* zusammen. Wir haben im weiteren Verlauf nicht mehr Länge, Breite und Gewicht, sondern einen Wahrheitswert für *Groß*, der z. B. zwischen 0 und 1 liegt. Für die vorliegenden Daten wählen wir dann eine geeignete Methode aus dem Bereich des maschinellen Lernens – oder wie andere sagen würden Data Minings – aus und nutzen diese. Das Resultat sind neue Ergebnisse (Muster) in den Daten. Das wäre im Wesentlichen das Wissen. Wenn es aber rein um Muster geht, kann es auch sein, dass diese wieder von einem Menschen interpretiert werden und dann als Output dieses Prozesses verwendet werden, um z. B. Abläufe in Firmen oder Institutionen zu verbessern.

Solange es um die Erkennung von Mustern in Daten geht, fallen das unüberwachte Lernen als Teil des Machine Learnings, über das wir gleich noch sprechen werden, und das Data Mining als Schritt innerhalb des KDD-Prozesses eigentlich zusammen. Es gibt manchmal den Versuch, eine Abgrenzung über die Menge der Daten zu machen. Das bedeutet, Sie lesen vielleicht irgendwo, dass es sich um Machine Learning handele, wenn es in den Hauptspeicher passt. Wenn es hingegen sequenziell aus Datenbank-Anwendungen kommen muss, sei es Data Mining. Das Problem an diesem Ansatz ist, dass es vorkommen kann, dass eine Fragestel-

lung im Jahr 2005 Data Mining war, weil diese nicht in den Hauptspeicher passte. Aber im Jahr 2015 ist es dann Machine Learning, weil der durchschnittliche Hauptspeicher in Computern sich verändert hat. Das ist, wie ich finde, unglücklich. Es spricht nichts dagegen, Data Mining zu sagen, wenn man hervorheben möchte, dass eine klassische Datenbank Ausgangspunkt der Arbeit war und man nichts mit künstlicher Intelligenz zu tun hat. Wirklich sinnvoll zu unterscheiden sind jedoch primär der KDD-Prozess und das maschinelle Lernen als Werkzeug innerhalb dieses Prozesses.

2.3 Strukturierte und unstrukturierte Daten in Big und Small

Beim maschinellen Lernen scheint es immer darum zu gehen, aus Daten Wissen zu generieren, und zwar unabhängig davon, ob im Umfeld der künstlichen Intelligenz Systeme trainiert werden oder im Rahmen eines KDD-Prozesses Wissen aus einer Datenbank erzeugt wird. Da also Daten der Dreh- und Angelpunkt der ganzen Angelegenheit sind, sollte man sich die unterschiedlichen Arten von Daten einmal ansehen.

Zunächst gilt es, zwischen strukturierten und unstrukturierten Daten zu unterscheiden. **Strukturierte Daten** kann man sich fast immer in Form einer Tabelle vorstellen. Jede Spalte stellt dabei ein **Merkmal** oder eben englisch **Feature** dar und jede Zeile einen Eintrag, der mehrere dieser Merkmale in einem **Datensatz** oder **Record** kombiniert. Ein Problem mit der Fachsprache und der Umgangssprache ist, dass *Datensatz* oft eher als *Datenbestand*, also mehr im Sinne von *ein Satz/eine Sammlung von Daten*, benutzt wird.

Tabelle 2.1 Strukturierter Datenbestand in einer Tabellenform

Merkmale / Datensatz	f_1	f_2	f_3	…	f_{n-2}	f_{n-1}	f_n
1							
2							
⋮							
$m-1$							
m							

Nehmen wir an, die Tabelle enthält Informationen zu PKW, dann kann jede Zeile für ein konkretes Auto stehen, und in den Spalten finden sich die Eigenschaften, wie die Anzahl der Türen (f_1), der kombinierte Verbrauch(f_2), der Anschaffungspreis(f_3) etc. Solche Tabellen sind auch die Grundlage von relationalen Datenbankanwendungen wie SQL etc., sodass viele Daten, die wir in Unternehmen finden, strukturiert sind. Das bedeutet, wenn wir uns für eine Eigenschaft eines Objektes interessieren, wissen wir genau, wo wir diese finden. Die Informationen sind für uns in strukturierter Weise abrufbar.

Nun sind irgendwie alle Daten, die in einem Computer verarbeitet werden, strukturiert; also auch Bilder, die oft als Beispiel für **unstrukturierte Daten** genannt werden. Wie kommt das?

Abbildung 2.2 Unstrukturierte Information, dass ein Hund auf dem Bild ist

Das Bild-Format als solches ist natürlich strukturiert. Wenn wir z. B. ein Bild im PNG-Format in Python laden, werden wir drei Matrizen mit RGB (Rot, Gelb, Blau)-Werten erhalten und können dadurch, dass bekannt ist, wie das Bildformat aufgebaut ist, dieses Bild auch anzeigen. Die Information, was z. B. auf dem Foto in Abbildung 2.2 zu sehen ist, können wir jedoch nicht strukturiert abgreifen. Ist eine Katze auf dem Bild oder ein Hund oder keines von beiden? Die Information ist irgendwie im Bild enthalten, jedoch nicht für uns direkt zugreifbar. Dasselbe gilt für freie Texte wie E-Mails, denn sie sind bzgl. der Informationen, die uns interessieren, unstrukturiert.

Es ist einsichtig, dass es für uns leichter ist, strukturierte Daten als Grundlagen für Lernalgorithmen zu verwenden als unstrukturierte. Ebenso muss man sich klarmachen, dass die Frage, ob etwas als strukturiert oder unstrukturiert gilt, manchmal von der Anwendung bzw. Frage abhängt. Nehmen wir an, Sie haben eine Aufnahme einer Wärmebildkamera. Wenn die gesuchte Information die Wärme an einem Bildpunkt ist, so ist dieses Bild als Informationsquelle sehr strukturiert. Wollen wir hingegen auf dem Bild erkennen, ob etwas ein Gesicht ist oder nicht, dann ist die Datenquelle unstrukturiert.

Ein anderer Begriff, der in letzter Zeit im Zusammenhang mit dem maschinellen Lernen durch die Presse wirbelt, ist **Big Data**. Was meint man damit und will man das eigentlich haben? Generell meint Big Data Datenbestände, die bzgl. ihrer Menge, Komplexität, schwachen Strukturierung und/oder Schnelllebigkeit ein Problem für die herkömmliche Datenverarbeitung bzw. Datenanalyse sind. Eine recht akzeptierte Definition von Big Data bezieht das *big* auf drei Dimensionen

- Volume – großes Datenvolumen
- Velocity – große Geschwindigkeit, in der neue Daten generiert werden
- Variety – große Bandbreite der Datentypen und -quellen

Wenn man das zunächst so liest, dann ist Big Data nichts, was man haben möchte, denn das oben Aufgelistete bedeutet, dass man ein Problem hat, mit etwas umzugehen. Neben dem Punkt, dass Big Data aktuell durchaus ein Hype-Begriff ist, um Dinge zu verkaufen, ist es je-

doch tatsächlich so, dass man hofft, in großen Datenbeständen Schätze zu haben, die es ermöglichen sollen, neue Geschäftsmodelle und -ideen zu entwickeln. Oft werden die Begriffe jedoch falsch genutzt, und es gibt den Wunsch, *irgendetwas mit Big Data* zu machen, obwohl die Fragestellungen und/oder Datenbestände zwar nicht per Hand, jedoch mit Standardmethoden des maschinellen Lernens bearbeitet werden könnten.

Generell ist *Volumen* für uns im Bereich des maschinellen Lernens nicht per se ein Problem. Sie werden im Laufe des Buches feststellen, dass wir uns z. B. im Rahmen strukturierter Daten sehr darüber freuen, viele Datensätze zu haben, es jedoch ungünstig finden, viele Merkmale zu haben, von denen wir nicht wissen, ob diese relevant sind bzw. ob diese miteinander stark korrelieren. In Sinne der Tabelle 2.1 sind also viele Daten, die durch mehr Zeilen entstehen, weit weniger problematisch als solche, die durch viele Spalten entstehen. Viele Spalten führen uns auf den **Curse of Dimensionality** oder **Fluch der Dimensionalität**, den wir in Abschnitt 5.3 besprechen werden. Nimmt man den Bereich des Data Minings in Simulationsdaten, so ergeben sich z. B. bei Finite-Element-Simulationen so viele Daten pro Simulation, dass man schnell viele Spalten erhält. Jedoch ist jede Simulation recht rechenintensiv, sodass viele Zeilen aufwendig zu erhalten sind. Das Thema **Simulation Data Mining** wird z. B. in [BY05] und [BSF+11] vertieft diskutiert. Haben wir viele Daten, müssen wir natürlich mehr auf das Laufzeitverhalten der Algorithmen achten als bei kleineren Datenbeständen. So wird man vielleicht generell den in Abschnitt 13.3 diskutierten DBSCAN nutzen wollen, weicht jedoch für große Datenmengen eher auf den besser skalierenden k-Means aus Abschnitt 13.1 aus. Es geht also darum, welche Algorithmen wie gut skalieren, damit diese für Anwendungsfälle mit großem Datenvolumen taugen. Dafür bekommen wir durch mehr Datensätze oft mehr Qualität für unsere Prognosen. Wenn es hingegen um unstrukturierte Daten wie Bilder geht, sind große Mengen an Datensätzen sogar oft bitter nötig, um die dann oft verwendeten tiefen neuronalen Netze wie in Kapitel 8 trainieren zu können.

Fazit ist: Man will kein Big Data – abseits von Marketinginteressen – um seiner selbst Willen, da es oft Schwierigkeiten macht, wie die Diskussion oben nahelegt. Wenn man einen Anwendungsfall bzw. eine Fragestellung mit einer einfachen strukturierten Datenbank, die auf einer normalen Workstation läuft, beantworten kann, sollte man froh sein. Wenn die Anwendung wirklich Daten im Sinne des Big Data benötigt, dann schränkt dies die Auswahl von Algorithmen auf diejenigen ein, die gut mit der Anzahl an Datensätzen skalieren. Was natürlich immer steigt, sind die Anforderungen an die Hardware. Aber auch hier bieten Mietmodelle zunehmend Möglichkeiten, kurzzeitig große Leistung anzufragen. Ein größeres Problem für den Bereich des maschinellen Lernens ist tatsächlich ein Aspekt, der sich indirekt in *Velocity* und *Variety* verbirgt: Die meisten Algorithmen sind darauf angewiesen, dass die Merkmale als solche konstant bleiben. Wird irgendwo auf einmal ein neuer Sensor eingebaut und liefert Daten, die zuvor nicht zur Verfügung standen, ist das erst einmal eine Herausforderung. In der Tabelle 2.1 würde das einer neuen Spalte entsprechen, die jedoch für alle alten Datensätze keinen Eintrag beinhaltet.

Wir thematisieren immer die Laufzeit der Algorithmen, die dann Rückschlüsse darauf zulässt, ob diese gut oder schlecht skaliert werden. Ansonsten sind jedoch alle Beispiele, die wir in diesem Buch adressieren, weit davon ab, unter den Begriff Big Data zu fallen. Sie werden alles auf einem normalen PC oder Notebook berechnen können. Ein paar Beispiele sind leider nicht ganz so klein zu kriegen und brauchen ggf. wenige Stunden Rechenzeit. Davor *warne* ich Sie dann. Mein Tipp ist, alles durchzuarbeiten und den Quellcode dieser komplexeren Anwendungen in den Kapiteln 11 und 14 vor dem Mittagessen zu starten. Bis auf seltene Ausnahmen

sollte ein normales Essen, bei dem man nicht schlingt, als Zeitrahmen ausreichen. Lediglich Abschnitt 15.6 fällt aus dem Rahmen. Hier ist im Vorteil, wer einen Gaming PC oder ähnliches besitzt und diesen auch mal länger entbehren kann.

2.4 Überwachtes, unüberwachtes und bestärkendes Lernen

Die Lernalgorithmen lassen sich im Wesentlichen in drei Kategorien einteilen:

- **„Überwachtes Lernen“**
- **„Bestärkendes Lernen“**
- **„Unüberwachtes Lernen“**

Tatsächlich geht es bei allen diesen Dingen darum, eine mathematische Funktion

$$f : X \to Y$$

zu konstruieren (lernen). Der Unterschied liegt in den Mengen X und Y, um die es geht, sowie darum, wie die Daten aussehen müssen, um diese Funktion zu lernen. Wichtig ist dabei, sich in Erinnerung zu rufen, dass das wesentliche Merkmal einer Funktion in der Mathematik ist, einem Element aus X genau ein Element Y zuzuordnen. In unseren Datenbanken werden wir jedoch aufgrund von Messfehlern, statistischen Effekten etc. oft den Fall haben, dass für einen Wert in X mehrere Aussagen über den Wert in Y vorliegen. Diese Widersprüche müssen die Algorithmen dann so auflösen, dass möglichst viele Einträge richtig durch die Funktion wiedergegeben werden.

2.4.1 Überwachtes Lernen

Das überwachte Lernen bedarf eines Lehrers. Den darf man sich jetzt allerdings nicht als eine Person vorstellen, welche die ganze Zeit den Algorithmus überwacht. Es geht darin, der Methode eine hinreichend große Menge von Ein- und Ausgaben zur Verfügung zu stellen, die bereits über den korrekten Funktionswert verfügen. Bei Datensätzen spricht man von gelabelten oder markierten Datensätzen. Ein Beispiel kann ein großer Datenbestand von Bildern mit Hunden und Katzen sein. Für jedes Bild hat jemand die Information hinterlegt, ob auf dem Bild ein Hund oder eine Katze abgebildet ist. Diese Information nutzen wir dann, um unseren Computer zu trainieren, auf Bildern Hunde und Katzen auseinanderzuhalten. Wenn dann neue Bilder kommen, zu denen diese Information nicht vorliegt, haben wir dann die begründete Hoffnung, dass der Computer diese selbstständig einordnen kann. Diese Art von Daten ist quasi die Daten-Gold-Klasse, da diese im Allgemeinen von Menschen mit Informationen veredelt wurden, und wenn man nicht gerade viele Leute dazu bringen kann, umsonst zu arbeiten, ist es durchaus teuer, die Daten so aufzuwerten. In diesem Lichte wird auch klarer, warum diverse Internetkonzerne froh sind, wenn wir als Verbraucher unsere Fotos beschreiben und sie ihnen zur Verfügung stellen. Die Klasse von Algorithmen beschäftigt uns in dem größten Teil des Buches, u. a. in den Kapiteln 4, 5, 6, 7, 8 und 12.

Im Wesentlichen geht es beim überwachten Lernen um zwei Problemstellungen, nämlich die Regression und die Klassifikation.

2.4.1.1 Klassifikation

Wie erwähnt, läuft es immer darauf hinaus, eine Funktion $f : X \rightarrow Y$ zu lernen. Bei der Klassifikation ist die Zielmenge Y diskret.

Abbildung 2.3 Schwertlilie mit Kelch- und Kronblatt

Ein bekanntes Beispiel ist der Iris Dataset. Dieser enthält Messwerte für das Kelch- und Kronblatt einer Schwertlilie (Iris) wie in Abbildung 2.3 dargestellt. Zu jedem Satz von Messwerten liegt eine Einschätzung eines Experten – in diesem Fall R. Fisher, der die Daten 1936 publizierte – vor, um welche Art von Schwertlilie es sich handelt. Die Datensammlung enthält dabei drei verschiedene Arten von Schwertlilien, nämlich *Iris setosa*, *Iris versicolor* und *Iris virginica*. Diese Werte bilden unsere Zielmenge:

$$Y = \{\text{Iris setosa, Iris versicolor , Iris virginica}\}$$

Um nicht zu tief in die Mathematik einzutauchen, versuchen wir uns *diskret* einmal im Unterschied zu *kontinuierlich* klarzumachen: Wenn unsere Menge aus dem Intervall von null bis zehn besteht, also $Y = [0, 10]$, dann gibt es direkt neben jedem Element y in diesem Intervall wieder andere Elemente. Man kann keine Umgebung um y finden, in der nicht auch ein anderes Element liegt. Bei diskreten Mengen liegen die Elemente so vereinzelt, dass man Umgebungen finden kann, in denen niemand liegt. Beispielsweise nehmen wir einmal die ganzen Zahlen zwischen null und zehn, also $Y = \{0, 1, 2, \ldots 10\} \subset \mathbb{R}$. Wenn wir uns nun 0.5 weit rechts und links von z. B. 3 umsehen, finden wir dort nichts außer eben 3. Diese Menge ist diskret. Wir wollen also eine Abbildung in eine solche diskrete Zielmenge lernen und nennen dieses Vorhaben **Klassifikation**:

$$\text{Classification} = \text{gelernteFunktion}(\text{Features})$$

Die Merkmale, im Beispiel der Iris also die Messwerte für die Blätter, werden als Input gegeben, und der Output ist dann die Art der Schwertlilie. Im Unterschied zur Regression, über die wir gleich reden werden, können wir hierbei keine Zwischenwerte als Ausgabe akzeptieren. Das bedeutet, wenn die drei Typen von Schwertlilien mit den Zahlen von 1 bis 3 codiert werden, muss auch wirklich eine dieser drei Zahlen ausgegeben werden und nicht 2.5, weil der Algorithmus zwischen mehreren Möglichkeiten schwankt. Das klingt zunächst komplizierter,

jedoch sind Klassifikationen in der Regel nicht schwerer als Regressionen, zu denen wir jetzt kommen, weil die Mengen oft deutlicher voneinander abgegrenzt sind.

Fassen wir es einmal etwas formaler zusammen:

Klassifizierungsproblem: Sei X der Raum der Featurevektoren und C eine Menge von Klassen. Darüber hinaus soll es eine Funktion c geben – die wir i. d. R. nicht kennen –, welche die fehlerfreie Klassifizierung

$$c : X \to C$$

vornimmt. Uns ist i. A. nur eine Menge von Beispielen bekannt:

$$D = \{(x_1, c(x_1),), (x_2, c(x_2),), \ldots (x_n, c(x_n),)\} \subseteq X \times C$$

Das Konstruieren dieser Funktion c ist die Lösung des Klassifizierungsproblems.

2.4.1.2 Regression

Die Regression funktioniert im Grundsatz recht analog zur Klassifikation. Auch hier wird im Wesentlichen eine Funktion gelernt; nur dass hier mit $Y \subseteq \mathbb{R}^n$ eine andere Zielmenge

$$D = \{(x^1, y^1),\ (x^2, y^2), \ldots, (x^n, y^n)\} \subset X \times Y$$

bereitgestellt wird. Es geht hierbei um Werte aus einem in der Regel kontinuierlichen Bereich; beispielsweise darum, einen optimalen Drehwinkel zu lernen oder die Höhe eines Kreditrahmens, der als sicher bzgl. der Rückzahlung gelten kann. Das Ergebnis ist dann eine Funktion

$$y = \text{gelernteFunktion(Features)}.$$

Wie schon gesagt, ist y dabei in der Regel eine kontinuierliche Zielgröße. Aber es gibt auch Fälle, z. B. wenn wir die Anzahl der Fahrräder lernen wollen, die abhängig von Wochentag, Witterung etc. ausgeliehen werden, in denen die Zielgröße in den natürlichen Zahlen $\mathbb{N}$ liegt. Der Unterschied liegt also primär in den Skalenniveaus, die wir in Abschnitt 4.2.2 diskutieren werden. Grob gesprochen liegt es daran, dass die Klassifikation in Räume abbildet, in denen die Werte nominal sind. Das meint, dass wir unterschiedliche Gruppen angeben und ihre Vertreter zählen, jedoch nicht z. B. ordnen können. Katzen sind nicht besser als Hunde, jedoch vielleicht häufiger auf Bildern. Zwanzig Fahrräder, die in einer Stunde ausgeliehen werden, sind aber mehr als eines oder keines.

Vereinheitlicht kann man festhalten, dass es sowohl bei der Regression als auch bei der Klassifikation darum geht, eine Funktion $f(x)$ aus Beispielen zu lernen. Wir werden noch feststellen, dass man sogar Techniken aus der Regression dazu verwenden kann, um Probleme der Klassifikation zu lösen. In der Theorie geht es, wie oben beim Klassifizierungsproblem, beim **Regressionsproblem** darum, die hypothetisch existierende perfekte Funktion zu finden bzw. zu konstruieren. Rein praktisch können wir das natürlich nicht, weil wir immer zu wenige Beispiele, ein zu schlechtes Modell und/oder unsere Beispiele eine zu schlechte Qualität haben. Wir werden jedoch in der Praxis mit der Zeit immer mehr Beispiele ansammeln, daher sollte man sich die Funktionsannäherung bzw. Funktionsapproximation als etwas vorstellen, was kontinuierlich verbessert werden kann.

2.4.1.3 Lazy Learning und Eager Learning

Bei den Verfahren unterscheidet man zwei Arten von Ansätzen: den wesentlich häufigeren **Eager Learner** und den **Lazy Learner**. Beim Eager Learner ist der Lernprozess, also das Training, in der Regel wesentlich aufwendiger als die spätere Abfrage der gelernten Funktion. Man trainiert also beispielsweise über Stunden oder Tage neuronale Netze, um in Bildern dieses oder jenes zu finden. Ist das Netz trainiert und legt man ihm ein Bild vor, so kommt die Antwort vergleichsweise schnell. Grundlage dieser schnellen Antwort ist ein globales Modell, das in dieser vergleichsweise aufwendigen Trainingsphase erzeugt wurde. Der Lazy Learner hingegen investiert kaum Arbeit in das Training; kommt jedoch eine Abfrage, dann schaut er sich seinen Datenbestand an, baut ein lokales Modell und nutzt dieses für eine Aussage. Das Training ist also billiger als beim Eager Learner, die Abfrage jedoch teuer. Schon aufgrund der Begriffe, *lazy* heißt ja nichts anderes als *faul*, ist man geneigt, die zweite Gruppe für weniger sinnvoll zu halten. Das ist jedoch so allgemein betrachtet sicherlich falsch. Ein großer Vorteil ist nämlich das lokale Modell, das passgenau gebildet werden kann. Geht es um Regression, sind diese lokalen Modelle oft weit genauer als die globalen Modelle der Eager Learner. Wie wir im Laufe dieses Buches sehen werden, sind die globalen Modelle immer in einem stärkeren Maße Kompromisse, und ihre Qualität schwankt von Region zu Region. So konnte z. B. in [BFV+13] lediglich eine lokale Approximation die für numerische Anwendungen nötige Genauigkeit bringen, während alle globalen Ansätze keine Fortschritte erreichen konnten. Da Abfragen im Einsatz häufiger sind als die Trainingsphasen, wird man trotz allem aus ökonomischen Gründen versuchen, wenn möglich auf Eager Learner zurückzugreifen. Fast alle im Buch vorgestellten Verfahren fallen in diese Kategorie. Lediglich in Abschnitt 5.4 besprechen wir mit dem k-Nearest-Neighbor-Algorithmus einen Lazy Learner für Regression und Klassifikation, welcher jedoch oft sehr gute Resultate liefert, wenn globale Ansätze versagen.

2.4.2 Bestärkendes Lernen

Da man oft nicht weiß, was richtig oder falsch ist, kann man die Daten nicht entsprechend labeln. Man weiß z. B. nicht, wie die optimale Strategie aussieht, um ein Gebäude in kurzer Zeit mit wenig Energie zu reinigen. Man weiß aber, was ein wünschenswerter und was ein unerwünschter Ausgang ist. Letzteres kann z. B. sein, wenn der Putzroboter am Treppenabsatz Selbstmord begeht oder immer wieder die Erbstücke aus weichem Holz rammt. Für solche Problemstellen sind Techniken aus dem Bereich des **bestärkenden Lernens** bzw. englisch **Reinforcement Learning** die Lösung des Problems. Diese Methoden sind fast immer mit agentenbasierten Ansätzen verknüpft. Hierbei enthält ein Agent von uns kontinuierliche Rückmeldungen in Form von Belohnung und Bestrafung, wodurch er mit der Zeit eine (möglichst) optimale Strategie für unser Problem lernen soll. Wie wir sehen werden, brauchen wir, um diese Strategie lernen zu können, jedoch wieder die Möglichkeit, eine Funktion zu konstruieren, wobei wir dabei auf Techniken zurückgreifen, die aus dem Bereich der überwachten Methoden kommen. Daher beschäftigen wir uns mit dem bestärkenden Lernen auch erst am Schluss des Buches.

Wir werden feststellen, dass viele Analogien bzgl. Menschen oder Tieren, die zum bestärkenden Lernen außerhalb der Fachpresse publiziert sind, in die Irre führen. Im Gegensatz zu einem Hund oder Pferd, das man aus ethischen Gründen – und weil die Resultate, wie man mir sagte, oft schlecht sind – nie mit Strafen erziehen sollte, spricht bei einem Softwareagenten

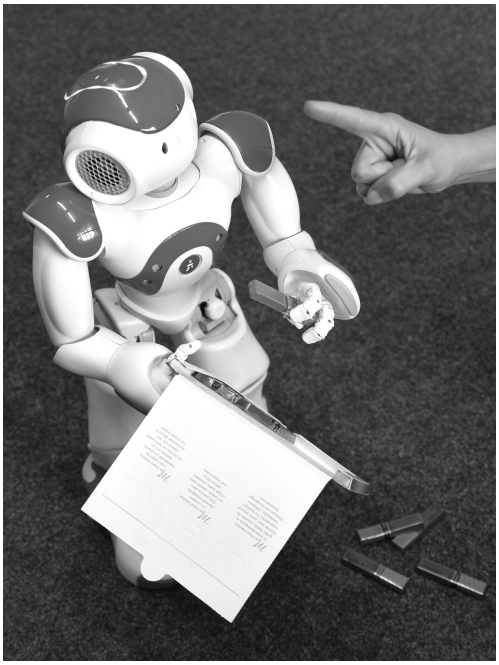

Abbildung 2.4 Nein, der Roboter leidet unter negativem Feedback nicht

nichts gegen negatives Feedback. Er leidet nicht darunter. Was er tut, ist auf der Basis der Werte und Techniken des überwachten Lernens eine Nutzenfunktion zu approximieren, die ihm sagen soll, welche Aktionen zu dem höchsten Nutzen führen. Es ist also eine Optimierungsaufgabe, deren Grundlagen mathematische Reihen bilden. Für einfache Fälle kann man beweisen, dass etwas Sinnvolles dabei herauskommt, und dann feststellen, dass es in den komplexen Fällen leider immer Raum für Unsicherheiten geben wird. In der Praxis funktionieren die Lösungen jedoch oft vollkommen ausreichend.

2.4.3 Unüberwachtes Lernen

Während wir beim überwachten Lernen dem Algorithmus eine Sammlung von Zielwerten zur Verfügung stellen und beim bestärkenden Lernen immerhin noch die Ergebnisse eines Verhaltens positiv bzw. negativ bewerten können, gibt es Fälle, in denen beides nicht möglich ist. Wir haben einfach nur eine Menge an Daten und wollen mittels **unüberwachtem Lernen** versuchen, versteckte Strukturen in unmarkierten Daten zu finden. Da die Beispiele für den Lernalgorithmus unmarkiert sind, kann kein Fehler berechnet oder eine Belohnung verteilt werden. In diesem Sinne ist es nicht direkt möglich, Lösungen des Computers zu bewerten.

Als Beispiel benutze ich gerne die vier Pflanzen-Skizzen aus Abbildung 2.5 und lege diese meinen Mitmenschen vor. Wenn Sie zwei Gruppen bilden müssten – wobei sowohl eine Gruppe à 3 und eine à 1 Pflanze als auch zwei Gruppen mit jeweils zwei Elementen akzeptiert werden –, zu welchem Ergebnis würden Sie kommen und warum?

Es werden sehr unterschiedliche Antworten gegeben. Gerade Kinder sortieren die tote Pflanze aus, andere wiederum separieren den Bambus, weil er kein Gehölz ist usw. Der wichtige Punkt

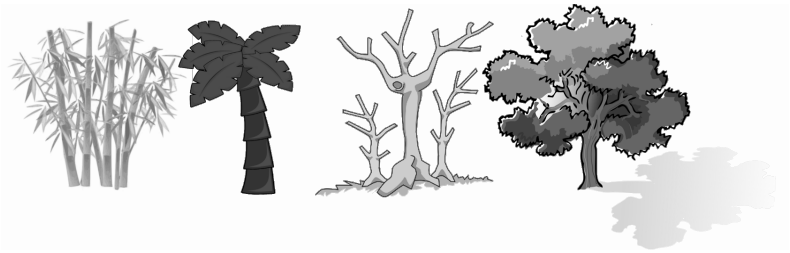

Abbildung 2.5 Zeichnungen verschiedener Pflanzen

ist: alle haben Recht. Jeder Datensatz – also jede Pflanze – hat verschiedene Merkmale. Man untersucht diese Datensätze auf Ähnlichkeiten, wobei jeder die Merkmale unterschiedlich gewichtet und entsprechend gruppiert. Genauso gehen die Algorithmen in Kapitel 13 vor. Ein häufigeres Anwendungsgebiet kennt man vom Online-Shopping, das sich in Meldungen äußert wie *Kunden, die diesen Artikel gekauft haben, kauften auch …* Es reicht im Allgemeinen, Kunden oder auch Business-Partner in ähnliche Gruppen zusammenzufassen, um Mehrwerte zu erzeugen. Ein Label im Sinne des überwachten Lernens brauchen diese Gruppen nicht, denn es ist gleichgültig, wie man eine Käufergruppe nennt; es reicht, wenn diese sich ähnlich verhält. Wie komplex und undankbar solche Aufgaben sein können, wird klar, wenn man überlegt, dass jemand nicht nur für sich über einen Account einkauft. Ich persönlich habe als Deko für ein Event einmal etwas gekauft, was eigentlich in ein Aquarium gehört. Es dauerte Monate, bis der Online-Shop aufhörte, mich mit den neuesten Trends für Aquaristik zu behelligen. Die Dinosaurier-Phase meines Sohnes hingegen war so intensiv, dass ich auch jetzt noch durch den Online-Shop in Versuchung gebracht werden soll. Der Kern dieser Geschichten ist ein sehr handfester, nämlich der nach Gewichtungen. Oft ist es sinnvoll, Datensätze nach ihrem Alter zu gewichten, damit der Computer die Möglichkeit erhält, zu vergessen, da Menschen keine statischen Objekte sind. Gleichzeitig hat man unter Umständen pro Kunde nicht so viele Datensätze, sodass man diese nicht zu schnell entwerten möchte. Sie sehen schon, dass es hier viele Anpassungsmöglichkeiten gibt, die nicht nur auf der Auswahl der Algorithmen, sondern auch auf der Aufbereitung der Daten basieren.

Die großen Unterschiede vom unüberwachten Clustering zur überwachten Klassifikation sind das Ziel und die Datenlage. Datenlage heißt, dass wir einmal Daten vorliegen haben, die annotiert sind. Das bedeutet, bei der Klassifikation liegen z. B. Datensätze von Vögeln, Hunden und Affen vor, und wir trainieren den Algorithmus mit den Zielwerten für diese Tiere. Er lernt, drei Gruppen zu unterscheiden – hoffentlich mit wenigen Fehlern. Im unüberwachten Fall haben wir ebenfalls Datensätze von Vögeln, Hunden und Affen vorliegen, aber ohne dass diese den drei Gruppen zugeordnet wären bzw. sein müssen. Wir benutzen deren Merkmale, wie z. B. *kann fliegen*, um diese in Gruppen zusammenzufassen. Je nachdem, wie der Clusteralgorithmus ausgelegt wurde, entstehen vielleicht dieselben drei Gruppen wie bei der Klassifikation, jedoch heißen diese hier nur Gruppe 1, 2 und 3, weil es nur darum ging, ähnliche Dinge zusammenzurücken. Je nachdem, welche Merkmale wir verwendet haben, ist der Kaiserpinguin ggf. zu Recht mit dem Bonobo in Gruppe 2, weil dort alle Tiere mit zwei Beinen hineingekommen sind, die nicht fliegen können. Vielleicht sind aber auch alle genau nach Vögeln, Hunden und Affen getrennt, weil die Merkmale das eben so hergegeben haben.

Eng verwandt mit dem Clustering, quasi die Umkehrung davon, ist die **Outlier Detection**; die Identifizierung von Datensätzen, die nicht mit einem erwarteten Muster oder anderen

Elementen in einer Datenbank übereinstimmen. Das können natürlich einfach dramatische Messfehler oder Fehlklassifikationen sein. Dann geht es darum, Daten von diesen zu befreien, was in den Kontext von Kapitel 9 fällt. Die häufigere Anwendung ist jedoch, dass ein solcher Eintrag mit einem Problem einhergeht, wie beispielsweise Betrug, einem technischen Defekt, medizinischen Problemen oder einem grammatikalischen Fehler in einem Text. Während Messfehler oder Fehlklassifikationen sehr vereinzelt sind, ist dies leider zum Beispiel bei Angriffen auf Netzinfrastrukturen nicht der Fall. Hier bilden die Outlier bereits wieder eigene, wenn auch wesentlich kleinere, Strukturen. Diese stimmen nicht ganz mit der üblichen statistischen Definition eines Ausreißers als seltenes Objekt überein. Oft sind hier Cluster-Algorithmen in der Lage, die durch diese Muster gebildeten Mikrocluster zu erkennen und bei solchen Anwendungen hilfreich zu sein.

2.5 Werkzeuge und Ressourcen

Im Internet gibt es viele Quellen, Werkzeuge und Hilfen zum Thema *maschinelles Lernen*. Manches ist flüchtig, aber vieles ist eine bewährte Anlaufstelle. Neben den IDEs wie Spyder oder Jupyter Notebooks gibt es eine Reihe von Ressourcen, die hier nicht thematisiert werden, die Sie sich aber ansehen sollten, sobald Sie wirklich mit maschinellem Lernen arbeiten. Auf der Berechnungsseite kann **SymPy** (*http://www.sympy.org/en/index.html*) oft eine sinnvolle Ergänzung sein. Die Verwendung dieser Python-Bibliothek erlaubt das Arbeiten mit symbolischen Berechnungen als Ergänzungen zu den numerischen in NumPy. Der Funktionsumfang ist im Wesentlichen der eines Computeralgebra-Systems. Oft nett ist die Fähigkeit, das Ergebnis der Berechnungen als LaTeX-Code auszugeben. SymPy ist ebenfalls freie Software unter der BSD-Lizenz. Mein Ziel ist es primär, Prinzipen und Ideen zu erklären und weniger, aktuelle Bibliotheken, entsprechend versuche ich daher mit Keras als API auszukommen und auch hier konservativ vorzugehen. Das Buch sollte halt länger sinnvoll sein als die API. Praktisch lohnt sich aber im Anschluss ein tieferer Einstieg in Keras und das darunterliegende Tensorflow.

Über die Matplotlib hinaus bietet sich **Seaborn** (*https://seaborn.pydata.org/*) zur Visualisierung an. Es baut auf die Matplotlib auf und bietet eine High-Level-Schnittstelle zum Erstellen anspruchsvoller statistischer Grafiken. Wer mit Bildern, besonders im Umfeld von Real-Time-Anwendungen, arbeitet, wird auf **OpenCV**(*https://opencv.org/*) stoßen. Bei OpenCV handelt es sich um eine freie Programmbibliothek unter der BSD-Lizenz. Sie beinhaltet Algorithmen für die Bildverarbeitung und im Rahmen von Computer Vision – wofür auch das CV in OpenCV steht – auch für maschinelles Lernen. Eine wichtige und häufige Anwendung ist die Gesichtserkennung.

Im Bereich des Reinforcement Learnings gibt es noch zahlreiche Umgebungen, in denen bzw. mit denen man Agenten trainieren kann. Wir machen das aus später dargelegten Gründen *from scratch* und im letzten Kapitel mit OpenAI Gym (*https://gym.openai.com/*), aber hier ein paar interessante Möglichkeiten, die sich anbieten, hinterher weiterzumachen. Einmal gibt es die Möglichkeit, mit **TORCS** bzw. *The Open Racing Car Simulator* (*http://torcs.sourceforge.net/*) einen Agenten für Autorennen zu trainieren. Es ist kein reines *Spiel*, sondern enthält eine gute Portion Physik. Es stehen Modelle von einigen Formel-1- und Geländefahrzeugen zur Verfügung. Wer eher an richtiges autonomes Fahren denkt, sollte einen Blick auf CARLA (*https://carla.org/*), einen Open-Source-Simulator werfen, welcher u. a. von Toyota und Intel

mit unterstützt wird. Wer lieber Fußball spielen will, sollte sich die Software der **RoboCup Simulation League** unter *https://www.robocup.org/leagues/23* ansehen.

Wenn die Daten nicht wie beim Reinforcement Learning in einer Simulation quasi automatisch entstehen, sind sie die letzte noch fehlende Ressource, die man nicht unterschätzen darf. Sie finden eine Reihe interessanter Datensätze, um Ihre Fähigkeiten zu trainieren, im **UCI Machine Learning Repository** unter *https://archive.ics.uci.edu/ml/*. Die Plattform **Kaggle** (*https://www.kaggle.com*) wiederum bietet die Möglichkeit, sich mit anderen in dem Bereich maschinelles Lernen und Data Mining zu messen. Ziel ist, das beste Modell für die in dem Wettbewerb zur Verfügung gestellten Daten bereitzustellen.

2.6 Anforderungen im praktischen Einsatz

Kaggle ist eine interessante Plattform, führt aber, wenn man nur diese im Blick auf maschinelles Lernen hat, oft zu stark verkürzten Schlüssen. So wurde mitunter die Meinung geäußert, es reiche, Bücher und Vorlesungen auf das Thema *Deep Learning* zu begrenzen und die anderen Methoden auszulassen. Immerhin würde auf Kaggle doch fast jeder Wettbewerb seit ein paar Jahren durch Deep Learning gewonnen. Zunächst ist die Aussage inhaltlich nicht ganz glücklich, weil hier unstrukturierte Daten überrepräsentiert werden. Bei den strukturierten Daten liegt in der Tendenz eher ein Random Forest bzw. seit einiger Zeit XGboost in der Genauigkeit vorne. Zwei Algorithmen, die wir in Kapitel 10 kennenlernen werden.

Das Problem ist aber nicht der Punkt strukturierte vs. unstrukturierte Daten, sondern dass Kaggle im maschinellen Lernen so etwas wie die Formel 1 ist. Es geht bei dem, was man entwickelt, als einziges Kriterium um das Maximum an Genauigkeit, welches mit einem Ansatz erreicht werden kann. Aber ebenso wie die Formel 1 nicht viel mit dem Weg zur Arbeit zu tun hat, stellt die Realität auch öfter eher mehrdimensionale Anforderungen. Es gibt natürlich Rennwagen, es gibt aber auch Laster, Traktoren und Familienautos. Alle haben in ihrem Anwendungsbereich ihre Berechtigung. Bei Wettbewerben wie Kaggle liegen die 10 oder 20 besten Lösungsansätze oft nur in den hinteren Dezimalstellen auseinander. Es kommt aber nur in wenigen praktischen Anwendungen auf diese kleinen Unterschiede in der Genauigkeit an. Dafür gibt es andere Anforderungen, die unter den letzten Prozentpunkten leiden, die man hier versucht herauszuquetschen.

2.6.1 Mehrdimensionale Anforderungen

Viele Anwendungen im maschinellen Lernen liegen im Bereich der Vorhersage von Verbraucherverhalten, Predictive Maintenance, Supply-Chain-Optimierungen etc. Die letzten Anwendungen gehen wieder auf Zeitreihendaten zurück usw. Das findet in Deutschland viel in Mittelstandsunternehmen statt, die maschinelles Lernen eben nur als einen Teil ihrer IT-Infrastruktur sehen. Dazu kommen lernende autonome Systeme in Bereichen, wo es auch um Zertifizierungen geht; beispielsweise in der Produktion. In all diesen Fällen ist die KI bzw. das maschinelle Lernen ein wichtiger Baustein eines Produktes, aber nicht das ganze Produkt.

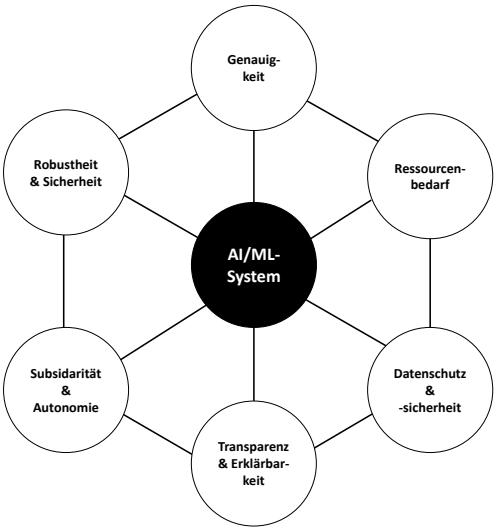

Abbildung 2.6 Anforderungshexagon für den Einsatz von ML bzw. KI-basierenden Systemen

Abbildung 2.6 zeigt ein Spannungsfeld, welche Anforderungen auftreten können. Natürlich wiegen nicht in jedem Einsatzfeld alle diese Anforderungen gleich viel. Beim Anwendungsgebiet Predictive Maintenance auf der Basis von Maschinendaten ist Datenschutz kein Thema und bei einem Empfehlungssystem für einen Online-Shop sind die Anforderungen an die Sicherheit andere als beim autonomen Fahren.

2.6.1.1 Genauigkeit

Die Genauigkeit ist meistens als Parameter intuitiv klar. Wir werden im Buch bei jedem Verfahren am meisten darüber reden. Halten wir es hier also kurz. Natürlich ist Genauigkeit wichtig. Unterhalb einer gewissen Schwelle nimmt dieser Parameter eine Königsrolle ein. Wenn die nötige Genauigkeit für den Einsatz nicht erreicht wird, braucht man sich um die anderen Probleme keine Gedanken zu machen. Ist die zwingend nötige Schwelle jedoch einmal übertreten, gewinnen andere Anforderungen an Bedeutung.

2.6.1.2 Transparenz und Erklärbarkeit

Transparenz hingegen ist oft schwerer zu fassen. Es gibt zum Beispiel viele Fälle, in denen von den Entwicklern erwartet wird, die Grundlage, auf der Entscheidungen durch den Algorithmus getroffen werden, offenlegen zu können. Dabei kann es um Anforderungen aus rechtlichen Rahmenbedingungen gehen oder um Nachweise in einem Zertifizierungsprozess in der Industrie. Wenn man z. B. einen CART-Algorithmus aus Abschnitt 6.3 eingesetzt hat, hat man eine Chance, diese Anforderungen umzusetzen. Mit einem Convolutional Neural Network (CNN) aus Abschnitt 11.2 ist das aktuell in dieser Qualität noch nicht möglich. Daneben kann die Analyse der eigenen strukturierten Daten, wie man diese im Vorfeld vieler Verfahren wie in Kapitel 9 durchführt, ebenfalls für Transparenz sorgen; hierbei sogar sowohl innerhalb eines Betriebs als auch nach außen. Werden die Einflussfaktoren besser durchdrungen, versteht man z. B. seine eigenen Kunden, Maschinen etc. besser. Daneben kann nach außen auf Anfrage transparent gemacht werden, was im Allgemeinen die größten Einflussfaktoren für eine Entscheidung

sind. Wie sehr das Thema der Transparenz und Erklärbarkeit allgemein, aber besonders CNN und andere Deep-Learning-Techniken betreffend – einfach weil es hier noch weniger möglich ist – an Bedeutung gewinnt, erkennt man z. B. an den Aktivitäten auf EU-Ebene.

Im April 2019 hat die damalige EU-Kommission die *Ethik-Richtlinien für vertrauenswürdige KI* in ihrer überarbeiteten Form veröffentlicht. Den Begriff *vertrauenswürdige KI* wird man in dem anglophon geprägten Umfeld sicherlich seltener lesen als die englische Version **Trustworthy AI**. Unabhängig von der Bezeichnung umfassen diese Richtlinien viele Aspekte, die in dem über 50 Seiten langen Bericht diskutiert werden. Wohlgemerkt, die Anforderung und die Rechtfertigung der Anforderungen werden dargestellt, nicht die Lösungen. Diese sind nämlich nicht leicht und oft – abhängig davon, wie man die Anforderungen interpretiert – im Stand von Wissenschaft und Technik nicht vollständig umsetzbar. Ein Bereich, der besonders hervorgehoben wird, ist eben die Transparenz und Erklärbarkeit. Das bedeutet unter anderem, dass ein Mensch das Zustandekommen der Entscheidungen der KI verstehen können sollte. Selbst wenn man annimmt, dass mit *Mensch* hinreichend vorgebildetes Fachpersonal gemeint ist, sollte klar sein: Das wird nicht leicht. Generell ist zu hoffen, dass diese Begriffe nicht absolut zu verstehen sind. Es gibt sonst die Tendenz, von einer Maschine zu erwarten, was Menschen untereinander nicht schaffen. Der deutsch-britische Hirnforscher John-Dylan Haynes hat einmal gesagt:

> *Man weiß generell, dass Menschen nicht gut darin sind, zu sagen, warum sie sich auf eine bestimmte Art und Weise entschieden haben. Wir handeln oft intuitiv, ganz oft können wir gar nicht die Gründe angeben, warum wir eine Entscheidung gefällt haben, und selbst wenn wir diese Gründe angeben, sind sie oft falsch, wie sich zeigt.*

Nun ist aber der Stand der Verständigung zwischen Mensch und Maschine noch etwas schlechter als der unter Menschen, sodass es sich lohnt zu verstehen, warum das so ist und wo man ansetzen kann. Nehmen wir ein Beispiel, bei dem es Menschen eher leicht fällt, sich objektiv über Entscheidungsgrundlagen auszutauschen. Wenn wir Dinge bestimmen sollen, können wir recht gut erklären, warum wir glauben, dass es dieses oder jenes ist. Auch hier ist nicht immer alles leicht. Nehmen wir zum Beispiel die Klassen *Stuhl* und *Sessel*. Manche Dinge können wir sehr scharf einer Klasse zuordnen, aber bei manchem Schreibtischstuhl oder Schaukelstuhl beginnt dann schnell die Diskussion. Tiere sind hingegen eindeutiger klassifiziert und die Tendenz, als Beispiel die Klassifizierung von Hunden und Katzen auf Bildern als Einführungsbeispiel zu nehmen, ist so klischeehaft, dass wir dies auch in Kapitel 11 machen werden, um dem Klischee Genüge zu tun. Hier könnte eine Begründung lauten, dass Katzenaugen ein Indiz für eine Katze sind, genau wie Pfoten ohne Krallen, da eine Katze die Krallen einziehen kann. Jedoch ist es fraglich, ob eine in Bruchteilen einer Sekunde getroffene Klassifizierung eines Tieres so zu erklären ist oder ob das nicht viel direkter passiert.

2.6.1.3 Ressourcen

Der nächste Punkt betrifft die Ressourcen. Das sind u. a. reale Zeit sowie Rechenleistung und damit quasi Energie. Zeit ist immer ein Faktor; geht es um Echtzeitanwendungen, wird dieses Anforderungskriterium ggf. deutlich dominanter als die Genauigkeit. Hierbei muss man bei sehr vielen Methoden zwischen Training und Auswertung unterscheiden. Außer bei den Lazy-Learnern, die wir in Abschnitt 5.4 behandeln, ist das Training immer wesentlich aufwendiger als die Auswertung. Dafür findet Letztere in vielen Anwendungen sehr viel häufiger statt. Bzgl.