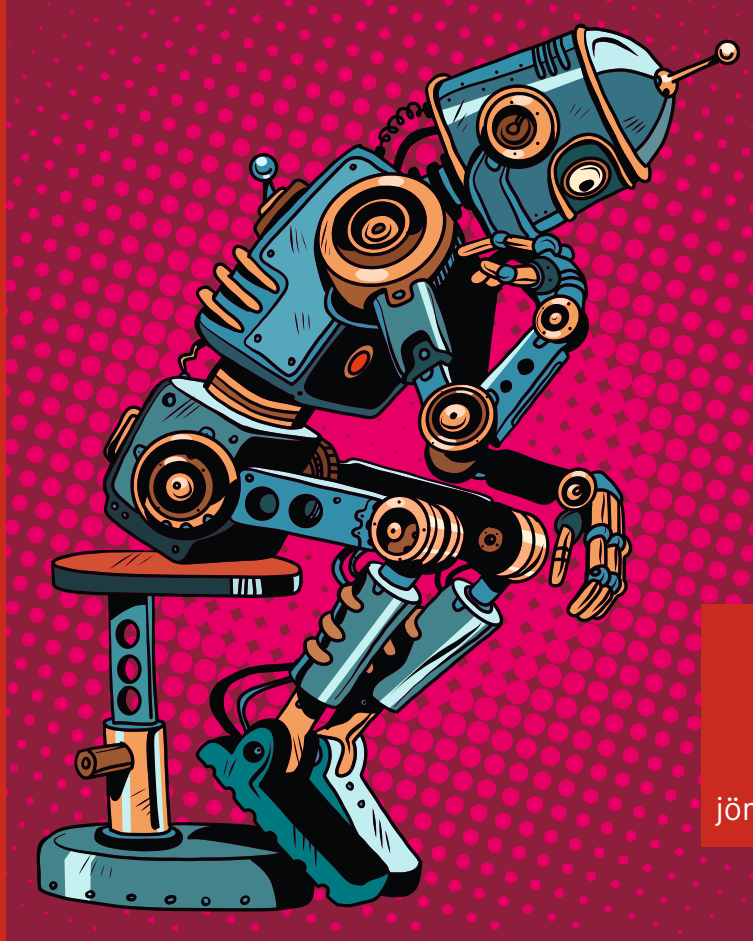

2., aktualisierte Auflage

jörg FROCHTE

Maschinelles Lernen

GRUNDLAGEN UND ALGORITHMEN IN PYTHON

HANSER

Jörg Frochte

Maschinelles Lernen

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

Hanser Update ist der IT-Blog des Hanser Verlags mit Beiträgen und Praxistipps von unseren Autoren rund um die Themen Online Marketing, Webentwicklung, Programmierung, Softwareentwicklung sowie IT- und Projektmanagement. Lesen Sie mit und abonnieren Sie unsere News unter

www.hanser-fachbuch.de/update

Jörg Frochte

Maschinelles Lernen

Grundlagen und Algorithmen in Python

2., aktualisierte Auflage

Mit 146 Abbildungen, 22 Tabellen und zahlreichen Beispielen

HANSER

Prof. Dr. rer. nat. Jörg Frochte

Hochschule Bochum
Arbeitsgruppe Angewandte Informatik und Mathematik

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autoren und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso übernehmen Autoren und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

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.

ISBN: 978-3-446-45996-0
E-Book-ISBN: 978-3-446-45997-7

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung – mit Ausnahme der in den §§ 53, 54 URG genannten Sonderfälle –, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2019 Carl Hanser Verlag München
Internet: http://www.hanser-fachbuch.de

Lektorat: Dipl.-Ing. Natalia Silakova-Herzberg
Herstellung: Anne Kurth
Satz: Jörg Frochte
Coverconcept: Marc Müller-Bremer, www.rebranding.de, München
Coverrealisierung: Stephan Rönigk
Druck und Bindung: Hubert & Co. GmbH & Co. KG BuchPartner, Göttingen
Printed in Germany

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 erste Auflage geschafft haben: Outlier Detection, Radiale-Basisfunktionen-Netze, Self Organizing Maps, XGBoost, Recurrent Neural Networks …

Es gibt viele interessante Themen in diesem Feld. Am Ende des Buches haben Sie sich bestimmt die Grundlagen 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 Unbedeutenheit) 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 zwei 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 immer eher über Algorithmen gehen, versuchen wir vorzugsweise, einen algorithmischen statt einen statischen 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 *www.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 PEP8 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 im 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 11, versucht Grade von Ähnlichkeiten zwischen Objekten zu ändern, und zwar durch die Art, wie Abstände definiert werden. In Kapitel 7 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 Verfahren zur Verfügung haben und auch gerne das Thema Autoencoder betrachten, wozu Neuronale Netze vorher unumgänglich sind.

Wer das Buch in einer Vorlesung einsetzen will und kürzen möchte bzw. muss, für den hier ein paar Hinweise: Die letzten drei Kapitel kann man in einer Veranstaltung weglassen, ohne dass es Probleme mit Abhängigkeiten gibt. Der Abschnitt 8.2 über die Convolutional Neural Networks ist ähnlich isoliert, 8.1 hingegen wird für den Autoencoder in 9.4 und beim Q-Learning in 12.6 und 12.7 verwendet. Diese Abschnitte müsste man dann anpassen oder weglassen.

Jetzt sollten wir aber anfangen, nachdem ich mich bei einigen Menschen bedankt habe: Einmal den Personen die Python und sein Ökosystem voranbringen und den Menschen die zur freien Open Clip Art Library (*openclipart.org*) beitragen, wodurch die Abbildungen 2.5, 12.4, 12.5, 12.8 etwas hübscher wurden. 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 der Korrektorin und Lektorin des Hanser-Verlags vielen Dank für die Anregungen und Überarbeitungen. Rückmeldungen und Anregungen gab es auch von Sabine Weidauer, Benno Stein, Matthias Rottmann, Peter Gerwinski, Herbert Schmidt, Michael Knorrenschild, Peter Beater, Christof Kaufmann, Henrik Blunck, Marco Schmidt und Stefan Müller-Schneiders. Vielen Dank für jeden Fehler, vor dem ihr mich in dieser ersten Auflage bewahrt habt. Damit es in der zweiten noch einmal weniger werden, freue mich über 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 bedeutetet das eigentlich?

Bevor man zur Klärung gelangt, 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 12 kennen lernen werden – soll aus Erfahrungen lernen und somit Verhaltensänderung 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 hier 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 Lernen 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 dann 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 Laufen 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 doch 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 intelligent 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 nun beantworten müssen, ist, ob eine Maschine, die den **Turing-Test** besteht, nun eine starke oder eine schwache künstliche Intelligenz ist? Der amerikanischer Philosoph John Rogers Searle zum Beispiel geht davon aus, dass hier 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) übernehmen. 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 der 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 doch 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 also 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 nun 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 hier 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

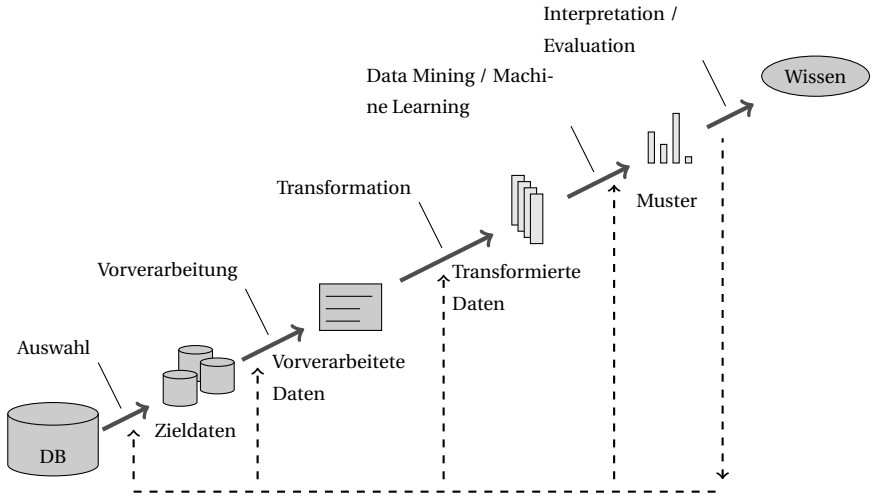

Abbildung 2.1 Knowledge Discovery in Databases als Prozess

einige Schritte ggf. wiederholt werden müssen. In der Abbildung 2.1 wird dies durch die Rückwärtspfeile nach jedem Hauptschritt angedeutet.

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 9.1.2 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 also dann 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 dann schon 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 Fragestellung 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 eben gerade 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

Irgendwie scheint es beim maschinellen Lernen 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 Sache 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							
⋮							
$m-2$							
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.

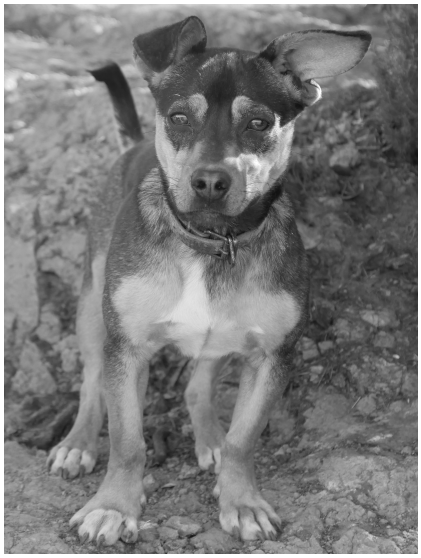

Abbildung 2.2 Unstrukturierte Information, dass ein Hund auf dem Bild ist

Nun sind ja 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? Nun, das Bild-Format als solches ist natürlich strukturiert. Wenn wir also 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 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 schon 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 klar machen, 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 schon 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 alles oben bedeutet ja, 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 jedoch 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 es 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 vielen Daten, die durch mehr Zeilen entstehen, weit weniger problematisch als solche, die durch viele Spalten entstehen. Vielen 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 Mining in Simulationsdaten so ergeben sich z. B. bei Finiten 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 Datensätzen. So wird man vielleicht generell den in Abschnitt 11.3 diskutierten DBSCAN nutzen wollen, weicht jedoch für große Datenmengen eher auf den besser skalierenden k-Means aus Abschnitt 11.1 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 das 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 skalieren 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 Kapitel 12 und Abschnitt 8.1 vor dem Mittagessen zu starten. Bis auf seltene Ausnahmen sollte ein normales Essen, bei dem man nicht schlingt, als Zeitrahmen ausreichen.

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 Funktionen

$$f : X \to Y$$

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, dass einem Element aus X genau ein Element Y zugeordnet wird. In unseren Datenbanken werden wir jedoch auf Grund 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 darum, 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 von i. A. 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 Kapitel 4, 5, 6, 7, 8 und 10.

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. Ein bekanntes Beispiel ist der Iris Dataset. Dieser enthält

Abbildung 2.3 Schwertlilie mit Kelch- und Kronblatt

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* klar zu machen: 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 diskreter 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 kodiert 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

$$c : X \rightarrow C$$

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 sagen, 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 auf Grund 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 diese 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 auch in diese Kategorie. Lediglich in Abschnitt 9.4 besprechen wir mit dem k-Nearest-Neighbor-Algorithmus einen Lazy Learner für Regression und Klassifikation, welcher jedoch wirklich 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 **Bestärkendes Lernen** 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 bestärkten Lernen auch erst am Schluss des Buches.

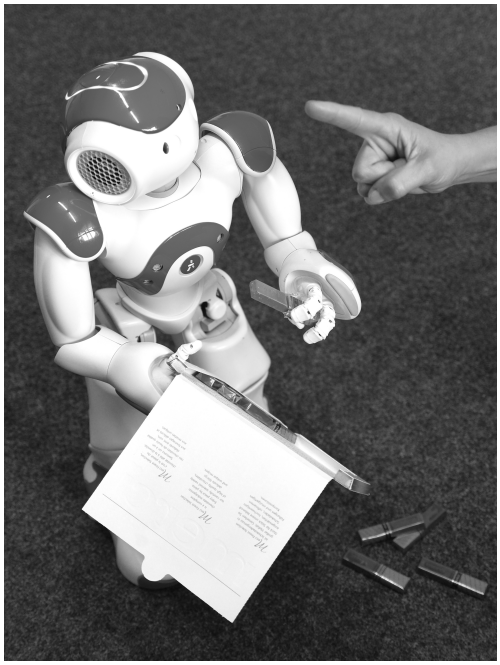

Abbildung 2.4 Nein, der Roboter leidet unter negativem Feedback nicht

Wir werden auch 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 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 werden wir sehen, dass man beweisen kann, 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

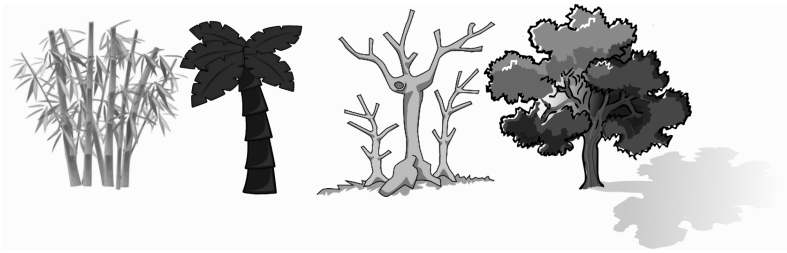

Abbildung 2.5 Zeichnungen verschiedener Pflanzen

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 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 11 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 ja 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 Schleich-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 ja 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 dann drei Gruppen zu unterscheiden – hoffentlich mit wenigen Fehlern. Im unüberwachten Fall haben wir nun ebenfalls Datensätze von Vögeln, Hunden und Affen vorliegen, aber ohne das 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 dann 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.

Das erste ist sicherlich **Pandas** (*https://pandas.pydata.org/*), eine Python-Softwarebibliothek unter der BSD-Lizenz. Diese bietet bessere und komfortablere Möglichkeiten zur Datenmanipulation und -analyse. Hierbei geht es viel um die Vorverarbeitung, den Umgang mit Strings, etc., aber nicht um das Number Crunching in Matrizen, was eine NumPy-Sache ist. Pandas setzt hierfür auch auf NumPy auf. Wer viel mit realistischen Datensätzen aus Datenbanken arbeitet, wird um diese Lib nicht herum kommen wollen. Auf der Berechnungsseite kann **SymPy** (*http://www.sympy.org/en/index.html*) hingegen 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.

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 Learning gibt es noch zahlreiche Umgebungen, in denen bzw. mit denen man Agenten trainieren kann. Wir machen das aus später dargelegten Gründen

from scratch, aber hier ein paar interessante Möglichkeiten, die hinterher mindestens schöner aussehen: 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 lieber Fußball spielen will, sollte sich die Software der **RoboCup Simulation League** unter *http://wiki.robocup.org/Soccer_Simulation_League* ansehen. Neben diesen Lösungen, die auf freier Software basieren, gibt es auch noch die spannende Möglichkeit, die neue Schnittstelle der Spieleplattform **Unity3D** für KI zu verwenden: *https://unity3d.com/de/machine-learning*. Hiermit ist dann so ziemlich alles Mögliche zu trainieren, was man vorher in Unity umgesetzt hat. Schließlich gibt es noch OpenAI Gym (*https://gym.openai.com/*). Hierbei handelt es sich um eine Sammlung verschiedenster virtueller Umgebungen. Diese beinhalten simulierte Roboter ebenso wie alte Atari-Spiele.

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 und Datenschutz 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.

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.

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 dann auch 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 allen diesen Fällen ist die KI bzw. das maschinelle Lernen ein wichtiger Baustein eines Produktes, aber nicht das ganze Produkt.

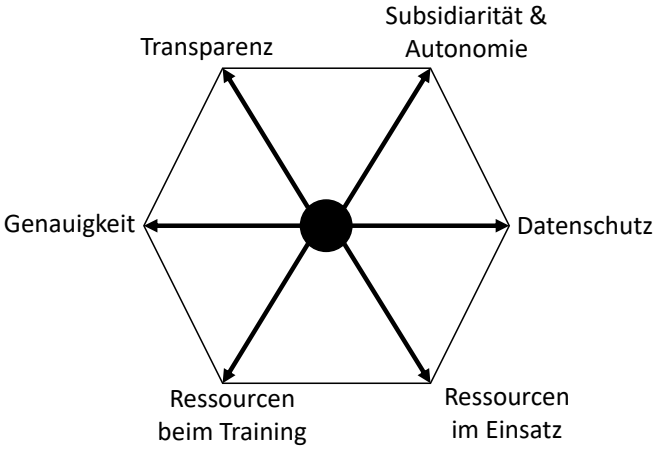

Abbildung 2.6 Unterschiedliche Anforderungen beim Einsatz von maschinellen Lerntechniken

Abbildung 2.6 zeigt ein Spannungsfeld, welche Anforderungen auftreten können. Die Genauigkeit ist meistens als Parameter intuitiv klar. **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, offen legen 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 aus Abschnitt 8.2 ist das fast unmöglich. 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.

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 9.4 behandeln, ist das Training immer wesentlich aufwendiger als die Auswertung. Dafür findet Letztere in vielen Anwendungen sehr viel häufiger statt.

Die Ressourcen gehen oft mit dem **Datenschutz** Hand in Hand. Werden personenbezogene Daten verarbeitet, ist der Transfer zu einem amerikanischen Cloud-Anbieter oft nicht verantwortlich möglich, da der *Privacy Shield* durch eine *Executive Order* (wie am 25. Januar 2017 geschehen) schnell zur löchrigen Angelegenheit wird. Dazu kommt der Aspekt der **Datensicherheit**, der dazu führt, dass viele Verfahren auf eben den verfügbaren Ressourcen eines mittelständischen Unternehmens laufen müssen. Bleiben wir jedoch zunächst beim Datenschutz: Der Name einer Person interessiert uns beim Training maschineller Lernalgorithmen so gut wie nie. Im Prinzip könnten die Spalten mit Namen und Vornamen fast immer gelöscht werden. Haben wir es damit schon erfolgreich mit **Anonymisierung**, also nicht mehr mit personenbezogenen Daten, zu tun und können unbehelligt weiterarbeiten? Leider oft nicht, denn in der harten Auslegung bedeutet Anonymisierung, dass personenbezogene Daten so verändert werden, dass diese einer Person nicht mehr zugeordnet werden können. Je reicher an Merkmalen unsere Datenbank ist, desto wahrscheinlicher ist es jedoch, dass man einen Datensatz einer Person auch ohne ihren Namen zuordnen kann. Nehmen wir mich und eine Datenbank aller Einwohner der Stadt, in der ich wohne, als Beispiel: Der Wohnort hat ca. 50000 Einwohner. Fügen wir das Geschlecht als Merkmal hinzu, haben wir die Zahl halbiert. Nun folgt der Bildungsabschluss: Die Promotionsquote eines Jahrgangs liegt um die 2%. Also sind noch ca. 500 Personen übrig. Was meinen Sie, wie lange brauchen wir, bis wir bei weniger als 10 Personen sind? Nehmen wir noch die Körpergröße dazu und den Jahrgang des Schulabschlusses, so wird es schon sehr eng. Sie arbeiten also oft auch ohne Namen mit Daten, die potenziell personenbezogen sind. Werden diese Daten vor einem Transfer auf externe Server weiter aggregiert, zum Beispiel linear wie in Abschnitt 9.3 oder nicht-linear wie in Abschnitt 9.4, so transferieren Sie dann vermutlich keine personenbezogenen Daten mehr. Hier muss man sich den Einzelfall genau ansehen. Was Sie in jedem Fall erreicht haben ist der Status der **Pseudonymisierung**. Im Gegensatz zur Anonymisierung existiert bei der Pseudonymisierung ein Rückweg, wenn man einen Schlüssel hat. Das wäre hier der Decoder, der auf den lokalen Systemen verbleibt.

Das Thema *Datenschutz* ist natürlich im Umfeld einer Technologie, die auf Daten basiert, wesentlich. Trotzdem werden wir darauf außerhalb dieses Abschnitts nicht mehr eingehen und uns mehr um die technischen und mathematischen Aspekte kümmern.

Spricht man in Deutschland über Datenschutz, wird dieser entweder als positiver Schutz von Individuen oder als Arbeitshemmnis im Alltag wahrgenommen. Ich möchte Sie ermuntern, sich zu diesem Aspekt einmal selber Gedanken zu machen. Was ist mit Daten über den Verlauf von Krankheiten? Ist es ethisch akzeptabel, diese der Allgemeinheit vorzuenthalten, obwohl damit anderen geholfen werden könnte? Wie kann man sichergehen, dass eine solche Datenfreigabe sich dann nicht z. B. bei der Vergabe von Krediten gegen den Einzelnen wendet? Der Gesundheitszustand könnte durchaus in ein Scoring für Langzeitkredite eingehen. Andere Fragestellung; man könnte ggf. den Verkehr und die Straßenführung besser planen, wenn man mehr über das Pendelverhalten der Bevölkerung auf Straßenzug-Niveau hätte. Das würde vielleicht Menschen mehr Lebenszeit abseits des Staus schenken.

Ich bin gespannt, zu welchen Schlüssen Sie kommen. Das Kuriose im Fall mit den Bewegungsdaten ist, dass durch die Verwendung von *Google Maps* zur Navigation vermutlich in Kalifornien alle nötigen Daten liegen. Wissenschaftler hier haben hingegen ggf. keine Arbeitsgrundlage auf der Basis frei zugänglicher Datenbestände. Die Open Data Ansätze werden dadurch

eingeschränkt, dass die Veröffentlichung nicht stattfindet, wenn ein Rückschluss von den Daten auf natürliche Personen nicht ausgeschlossen werden kann. Wenn Sie an das Beispiel der Mittelstadt oben denken, erkennen Sie das Problem. Aggregierte Daten in Excel-Tabellen oder Diagrammen, wie sie als Open Data herausgegeben werden, sind hingegen für das maschinelle Lernen stark entwertet. Daten sind wichtig und sensibel; ihr Schutz ist auch ein Schutz des Bürgers vor Konzernen und staatlichen Zugriffen auf Privates. Gleichzeitig sind Daten der Rohstoff für das maschinelle Lernen, den man nicht leichtfertig verknappen möchte. Es ist also oft nicht alles einfach schwarz und weiß, sondern manches eben doch grau.

Der Datenschutz spielt auch mit, wenn es um Subsidiarität geht. Der Begriff Subsidiarität betrifft meistens gesellschaftliche Zusammenhänge. Das Prinzip ist, dass jeweils die kleinste gesellschaftliche Einheit, die eine Aufgabe erledigen kann, dies auch tun sollte. Nur wenn eine kleinere Einheit dazu nicht in der Lage ist, wird die jeweils übergeordnete Instanz aktiv. Was hat das nun mit maschinellem Lernen zu tun? Nehmen wir an, es handelt sich um lernende Agenten. Wo sollen die von den Agenten neu gefundenen Daten gespeichert werden, wo sollen die Agenten trainiert werden, wem gegenüber sollen sie loyal sein? Fangen wir mit der ersten Frage an, die auch noch an den Aspekt des Datenschutzes anknüpft: Man muss sich dazu klar machen, dass ein Roboter, der *sehen kann*, auch nichts Anderes ist als eine mobile Kamera. Werden die Bilder lokal durch den Roboter verarbeitet und nach angemessener Zeit gelöscht, haben nur wenige damit ein Problem. Wie sieht es jedoch aus, wenn die Bilder aus dem eigenen Haus zum weiteren Training auf einem Server des Herstellers geladen werden? Das eigene Bild, halbnackt im Morgenmantel, möchte man sich dort sicherlich nicht vorstellen. Auch bzgl. des Datenschutzes als Gesetz stellen beide Szenarien unterschiedliche Hürden dar. Hier wäre es also wünschenswert, wenn ohne Datentransfer etwas aus den Daten gemacht werden könnte.

Und was bedeutet nun *loyal*? Als Beispiel nehmen wir einmal den Fall, in dem Amazon im Juli 2009 legal erworbene eBooks von den Geräten seiner Kunden gelöscht hat. Grund war ein Rechtsstreit, aber das Wesentliche hier ist, dass das Gerät nicht vom Kunden kontrolliert wurde, sondern von dem, der es verkauft hat. Stellen Sie sich vor, dass immer mehr Geräte, die lernen und sich entwickeln können sollen, Sie persönlich umgeben. Ein lernender Putzroboter, ein lernendes Smart Home und ein autonom fahrendes Auto. Die meisten von uns wären der Ansicht, dass ein Gerät, das uns gehört, auch uns gegenüber loyal sein muss. Jedenfalls legt dies der in *Science* veröffentlichte Artikel [BSR16] nahe. Grob zusammengefasst war ein Ergebnis dieser Studie über Moralvorstellungen, dass autonome Autos Autoinsassen opfern sollen, wenn dadurch mehr Passanten geschützt werden als Leute, die im Auto sitzen. Diese Aussage gilt aber nur solange, wie man nicht selbst im Wagen sitzt. Dann ist doch irgendwie das Gefühl vorherrschend, dass der Wagen seinen Besitzer schützen sollte. Das lässt sich fortsetzen. Der Putzroboter soll nicht gegen seinen Besitzer aussagen, wann dieser wo im Haus war – oder eben auch nicht – und das Smart Home nicht für den Stromerzeuger optimieren, sondern für seinen Besitzer. Das bedeutet für das maschinelle Lernen, dass es ggf. wünschenswert ist, wenn der Anwender mehr Kontrolle über das Lernen hat, beispielsweise darüber, welche Daten transferiert werden. Dann muss aber ggf. auf eine zentrale Rechenanlage verzichtet und lokal gelernt werden. Das bedeutet, dass nach einem initialen Werkstraining mit Ressourcen gearbeitet werden muss, die eher in der Größenordnung eines PCs oder sogar eines Raspberry Pi liegen und nicht in der eines Rechenzentrums.

Während die Aspekte oben eher die Probleme von Endkunden waren, ergeben sich analoge Probleme für Betriebe: Ist es akzeptabel, wenn Daten aus der Produktion wegen der lernenden

Smart Factory an einen externen Anbieter abfließen? Wie viel Risiko ist tragbar, dass die Anlage nicht mehr uneingeschränkt läuft, wenn die Verbindung zum externen Rechenzentrum länger abbricht? In Star Wars Episode I wirkt es lustig, wenn, nachdem das Kontrollschiff der Druiden zerstört wurde, sich diese einklappen und den Kampf einstellen. Aber das gleiche Szenario, wenn ein Anbieter – technisch oder vertraglich – ausfällt, würde man doch in keiner Produktion oder Dienstleistung witzig finden. Das bedeutet, dass man in vielen Szenarien auf ein größeres Maß an Autonomie achten sollte und dafür auch ggf. Einbußen bei anderen Kriterien wie der Genauigkeit akzeptieren muss. Natürlich gibt es keine optimale Mischung, die für jede Anwendung gilt. Es geht vielmehr darum, immer einen guten Kompromiss für die konkrete Anwendung zu finden. Wenn beim autonomen Fahren Menschenleben gefährdet sind, wird man eher auf mehr Genauigkeit drängen, in anderen Szenarien auf mehr Datenschutz oder Autonomie und das Subsidiaritätsprinzip.

Der letzte Punkt betrifft die Entwicklung und Wartbarkeit von Software. Er ist weniger politisch oder philosophisch als die vorangegangen Aspekte, aber in der Praxis doch oft wichtig. Hier treffen die Ansprüche eines Software-Architekten auf die eines KI-Enthusiasten. Für jemanden, der sich mit künstlicher Intelligenz beschäftigt, ist es wissenschaftlich und technisch faszinierend, je mehr man von der schwachen künstlichen Intelligenz, die auf ein Spezialgebiet fokussiert ist, hin zu einer kommt, die stärker generalisieren kann. Daher war man im Jahr 2015 auch davon begeistert, als die Google-Tochter DeepMind eine einzelne KI vorstellen konnte, die sich das Spielen von 49 unterschiedlichen Atari-Konsolen-Games selbst beibrachte. Die Resultate und Grundlagen dieser KI mit dem Deep-Q-Network wurden in der Fachzeitschrift *Nature* [MKS+15] veröffentlicht.

Diese Faszination wird im Butter&Brot-Geschäft nicht immer geteilt. Hier kann es eher sinnvoll sein, spezialisierte lernende Einheiten bzw. KIs zu verschalten, obwohl ein umfassender lernender Ansatz möglich wäre.

3 Python, NumPy, SciPy und Matplotlib – in a nutshell

Der Ansatz in dieser kurzen Einführung ist es, jemanden, der noch keinen Kontakt mit der Kombination aus Python, NumPy, SciPy und Matplotlib hatte, in einen im Wesentlichen für das Buch arbeitsfähigen Zustand zu versetzen. Dieses kurze Kapitel hat nicht den Anspruch, diese Werkzeuge umfassend zu behandeln. Python allein füllt dicke Bücher und die Kombination aus NumPy, SciPy und Matplotlib als Zusatz oft ein weiteres. Es ist aber – u. a. auf Grund der Einsteigerfreundlichkeit von Python – sehr schnell und mit wenigen Seiten möglich, für den Stoff dieses Buches arbeitsfähig zu werden, wenn man Vorkenntnisse in anderen Programmiersprachen hat. Besonders einfach geht der Umstieg, wenn Kenntnisse in MATLAB bzw. GNU Octave vorliegen, aber auch C/C++ oder Java können eine gute Ausgangsbasis für einen Einstieg sein.

3.1 Installation mittels Anaconda und die Spyder-IDE

Python kann unter Linux mit den Systemwerkzeugen wie z. B. `apt` installiert werden. Unter Windows ist es ggf. komplex, die benötigten Pakete direkt von den jeweiligen Seiten zu installieren. Hier bietet es sich an, auf eine Python-Distribution wie Anaconda zurückzugreifen.

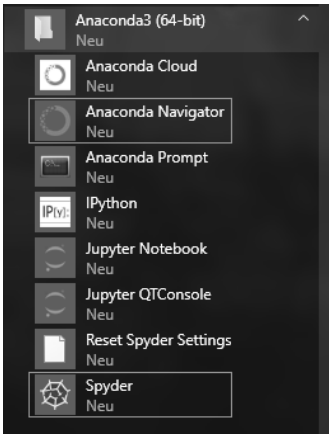

Abbildung 3.1 Anaconda im Windows-Menü

Wir gehen kurz die Installation mittels Anaconda unter Windows durch, wobei der Einsatz von Anaconda auch unter Linux möglich ist. Durch die Installation von Anaconda werden automa-

tisch Python 3.x (64 bit), NumPy, SciPy, IPython, matplotlib, Spyder, scikit-learn und Pandas installiert.

Zur Installation laden Sie Anaconda (Python 3.x version) als 64-bit Installer von *https://www.continuum.io/downloads* herunter. Anschließend installieren Sie nun Anaconda (benutzer- oder systemweit). Es ist oft vorteilhaft, wenn der Pfad keine Leerzeichen enthält, weil weitere Pakete aus der Python-Familie wie z. B. `Nuitka` damit öfter Probleme haben. Ein Ansatz ist sicherlich `C:\Anaconda3`, während hingehen `C:\Programme` ungeeignet ist, da dies nur ein Alias für `C:\Program Files` ist und somit Leerzeichen enthält. Nach der Installation gibt es im Startmenü die Einträge wie in Abbildung 3.1 gezeigt.

Ich nutze beim Schreiben des Buches die Versionen NumPy 1.12.1, SciPy 1.0.0, Matplotlib 2.1.0 und Python 3.6.3, wobei ich meines Wissens nur Features bis Version 3.5.1 wirklich verwende. Falls Sie z. B. eine spezielle Python-Version verwenden wollen, gehen Sie wie folgt vor: Um die aktuell verwendete Version zu überprüfen, öffnen Sie den *Anaconda Navigator*. Hier zu *Environments*, danach rechts zu *Python* herunterscrollen. Dort steht dann rechts die installierte Version. Falls eine nicht gewünschte Version installiert ist, klicken Sie auf die grüne Checkbox und stellen dann bei *Mark for specific version installation* die gewünschte Version ein, z. B. 3.5.1. Anschließend unten rechts auf *Apply* klicken und bestätigen. In vielen Fällen ist so eine Down- oder Upgrade möglich. Daneben gibt es in Python noch die Möglichkeit von *Virtual Environments*, auf die wir hier jedoch nicht eingehen.

Es gibt für Python sehr viele verschiedene Editoren und Entwicklungsumgebungen. Sie können nach eigenem Geschmack wählen. Nach diesem Kapitel wird auch nicht mehr auf unterschiedliche Umgebungen eingegangen.

Generell haben haben zwei Umgebungen einen besonderen Charme. Das ist zum einen das **Jupyter-Notebook**. Es kann in einem Browser als webbasiertes interaktives Notizbuch genutzt werden. Der Server dazu kann auf dem eigenen Rechner unter localhost:8888 laufen oder eben auch auf einem Rechner ganz woanders. Hier liegt ein besonderer Reiz, weil Sie auf einem z. B. starken Rechner irgendwo anders über den Webbrowser arbeiten können, während das lokale Notebook nur wenig Ressourcen bereitzustellen braucht. Das Jupyter-Notebook ist besonders stark, wenn es darum geht, Datensätze mit fertigen Klassen zu bearbeiten. Wenn ich selbst Klassen implementiere, ziehe ich eine echte IDE vor.

Hier bietet sich die Python IDE **Spyder** an. Der große Vorteil für Umsteiger ist, dass diese in ihrem Design MATLAB bzw. GNU Octave sehr ähnlich ist. Spyder braucht beim ersten Starten in einer Sitzung unter Windows recht lange.

Wer bereit mit GNU Octave oder MATLAB gearbeitet hat, erkennt den Aufbau in Abbildung 3.2. In dem mit [1] gekennzeichneten Bereich steht Ihnen eine iPython-Console zur Verfügung. Hier können Befehle eingeben werden, die dann sofort ausgeführt werden. Werden dabei Variablen erzeugt oder verändert, können Sie dies im **Variable Explorer** [3] verfolgen. Dieser entspricht in seiner Funktionalität in etwa dem Workspace von Octave/MATLAB. Eigenen Funktionen und Klassen können unter [2] geschrieben werden. Spyder hat dabei eine Anbindung an einen Debugger integriert.

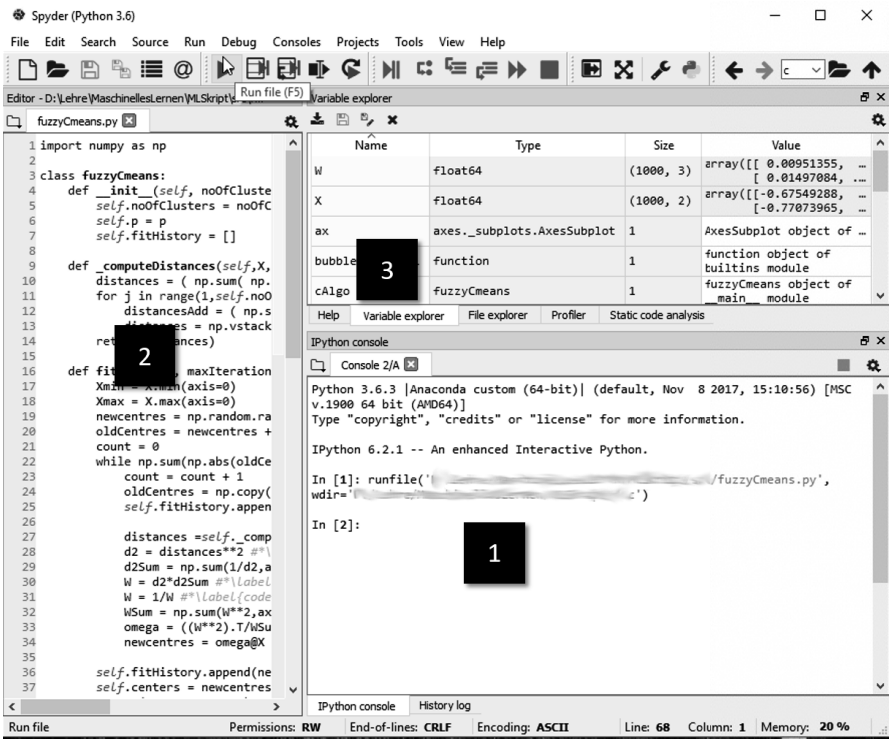

Abbildung 3.2 Spyder IDE Version 3.2.4

Ebenfalls analog zu MATLAB etc. gibt es ein Working-Directory. Es ist u. a. dafür entscheidend, wo nach Klassen etc. gesucht wird. Wenn Sie das Working-Directory unter *Tools* bzw. *Preferences* ändern, wirkt sich das erst auf eine neu geöffnete Konsole aus, nicht jedoch auf eine bereits geöffnete.

Ebenfalls für Umsteiger oder Benutzer der Bash angenehm ist die **Tab Completion**. Mit der Taste Tab kann nach einer bekannten Variablen oder Methode mit den entsprechenden Anfangsbuchstaben gesucht werden. Ähnliches funktioniert auch mit Dateipfaden. Wenn etwas eingegeben wird, was wie der Anfang eines Dateipfads aussieht (enthält „/“ bei Linux und Mac), wird eine Liste der passenden Dateinamen ausgegeben. Natürlich gibt es auch die Möglichkeit der Suche in der **Command History**. Mit `Strg + P` (oder Pfeil nach unten), `Strg + N` (oder Pfeil nach oben) und `Strg + R` (reverse Suche) kann bereits eingegebener Code erneut ausgeführt werden oder darin gesucht werden. Ebenfalls angenehm ist die **Introspection**. Ein Fragezeichen nach einer Variablen oder Funktion gibt Informationen darüber aus.

Oft sehr nützlich sind die sogenannten **magic commands**. Mit vorangestelltem `%` und `%%` können diese spezielle IPython-Befehle ausgeführt werden. Ein Beispiel ist `%run`, wodurch ein Python-Code in einer Datei innerhalb von **IPython** ausgeführt wird. Es sind viele spezielle Tastenkürzel und ähnliches vorhanden. Am einfachsten wird das mit „%quickref“ und „h“ für Hilfe aufgelistet.

Zum Schluss noch eine angenehme Sache besonders unter Linux. Ein beliebiger Konsolenbefehl wird mit einem vorangestellten „!“ ausgeführt, beispielsweise: `!ping 192.168.1.42`

Das war natürlich nur eine grobe Übersicht zur IDE. Die jeweilige Umgebung spielt ab jetzt auch keine Rolle mehr in den Ausführungen.

3.2 Python Grundlagen

Python ist eine Interpretersprache. Dies bedeutet, dass Programme – wie auch bei MATLAB oder Octave – nicht kompiliert, sondern interpretiert werden. Es gibt jedoch Projekte wie z. B. *Nuitka* (*http://nuitka.net/*), die versuchen, auch Möglichkeiten anzubieten, Python zu kompilieren, um eine kompaktere Distribution oder Performance-Verbesserungen zu erreichen. Einer der Vorteile einer Interpretersprache ist es, dass sich zwei Modi aufdrängen: einmal der Einsatz in einem interaktiven Kommando-Fenster – i. d. R. mittels IPython – und zum anderen die Organisation in Skripten, Funktionen und Bibliotheken. Im interaktiven Kommando-Fenster kann man Befehle eingeben und bekommt die Resultate direkt angezeigt. Wenn wir das im Buch tun, deuten wir es mit den drei Zeichen >>> an. Darüber hinaus können wir natürlich wie in jeder anderen Programmiersprache Funktionen designen und diese im Kommando-Fenster oder aus anderen Funktionen heraus aufrufen.

3.2.1 Basis-Datentypen und Lists

Beginnen wir mit den Variablen. Eine Variable in Python zu erzeugen ist einfach: Man vergibt einen Namen und weist einen Wert zu. Variablentypen müssen nicht deklariert werden. Wird beispielsweise

```
>>> a = 42
```

ausgeführt, so entsteht eine Integer-Variable `a`. Seit Python 3 müssen wir als Personen, die eher mit Float-Werten arbeiten, uns glücklicherweise nicht mehr so viele Gedanken darüber machen, was passiert, wenn wir als nächstes

```
>>> b = a / 5
```

eingeben. In Python 2.x wäre das Resultat noch 8 gewesen und vom Typ Integer, aber in Python 3 ist das Ergebnis 8.4 und ein Float, wie wir durch den Befehl `type(b)` überprüfen können.

Mathematische Operationen werden auf den Variablen wie gewohnt aufgeführt, also `+`, `-`, `*`, `/`. Lediglich das Potenzieren ist – je nachdem, von welcher Programmiersprache man kommt – etwas gewöhnungsbedürftig. Soll das Quadrat der Variablen `a` berechnet werden, so nutzt man die Syntax `a**2` oder `pow(a,2)` und nicht das Symbol `^`.

Wie z. B. in MATLAB oder C/C++ werden die Bool-Werte `True` und `False` als 1 und 0 codiert. Diese entstehen z. B. als Resultat von Vergleichsoperatoren wie ==, <, <=, >, >=. Ungleich kann sowohl mittels != als auch mit <> ausgedrückt werden. Hier ein kurzes Beispiel:

```
>>> z=4 < 6
>>> b= z+2
>>> b !=3
False
```

b hat den Wert 3, da z `True` war – also 1 – und dieser Wert um zwei erhöht b zugewiesen wurde. Vom Typ her ist z allerdings als Bool mit dem Wert `True` angeben, nicht z. B. als `Integer` mit dem Wert 1.

Neben diesen Basisdatentypen gibt es noch viele weitere wie z. B. **Tuples** oder **Dictionaries**. Für uns wirklich wichtig sind allerdings nur **Lists** und noch ein wenig **Strings**.

Ein String gehört in Python zu den Basisdatentypen oder **Built-in Types**. Der Operator + ist für Strings überladen, was zu sehr angenehmen Möglichkeiten führt, um Strings zusammenzufügen.

```
>>> a="Hallo"
>>> b='Welt'
>>> c = a + ' ' + b
>>> print(c)
Hallo Welt
>>> b == 'Welt'
True
```

Wie man sieht, können Strings durch einfache oder doppelte Anführungszeichen erzeugt werden. Auch hier können wir die Vergleichsoperatoren verwenden. Für die zahlreichen weiteren Operationen auf Strings sei hier auf die Python-Dokumentation (*https://docs.python.org/3/library/string.html*) verwiesen. Wir werden davon nur wenig Gebrauch machen.

Wir wenden uns nun dem Typ `List` zu. Hierbei muss man zunächst eine kleine Warnung voranstellen. Während in Python die Basisdatentypen wie z. B. `char`, `float`, `complex`, ... praktisch analog zu dem meisten Programmiersprachen bei einer Zuweisung `a = b` kopiert werden, ist das bei den Objekten, zu denen wir nun kommen, nicht mehr der Fall. Die Variablennamen sind hier lediglich **Referenzen** auf ein Objekt, und die Zuweisung erzeugt dann wiederum nur eine Referenz auf dasselbe Objekt. Wir schauen uns das gleich direkt einmal im Kontext von Listen an.

Eine Liste ist in Python einfach eine sehr flexible Kombination von Variablen beliebiger – auch unterschiedlicher – Typen, die durch eckige Klammern zusammengeschlossen werden. Damit ist Folgendes eine quasi typische Liste:

```
>>> TolkinListe=[3, 'Elben', 7, 'Zwerge', 9, 'Menschen', 1, 'Dunkler Herrscher']
>>> TolkinListeNat  = TolkinListe
>>> TolkinListeNat[1]=3.14
>>> TolkinListeNat[5]=9.81
>>> print(TolkinListe)
[3, 3.14, 7, 'Zwerge', 9, 9.81, 1, 'Dunkler Herrscher']
```

Wie man sieht, ist `TolkinListeNat` nur eine Referenz, also ein anderer Name für das gleiche Objekt, das auch mit `TolkinListe` bezeichnet wird. Änderungen in `TolkinListeNat` wirken sich somit direkt auf `TolkinListe` aus. Darüber hinaus erkennt man in dem Code oben eine weitere wichtige Eigenheit von Python:

Python startet in der Nummerierung für den Zugriff in Listen, Arrays etc. bei 0 wie C/C++ und nicht bei 1 wie MATLAB oder GNU Octave!

Kommen wir zurück zu den Referenzen. Unser Problem wird dadurch verkompliziert, dass Listen auch z. B. wieder Listen enthalten können. Es wäre sehr unerfreulich, diese Objekte per Hand mit unbestimmter Tiefe zu durchlaufen und quasi elementweise neu aufzubauen und zu kopieren. Zum Glück gibt es eine fertige Routine, die sich um tiefe Kopien bzw. *deep copy* kümmert:

```
>>> autoren = ["Douglas Adams","Isaac Asimov", "Terry Pratchett"]
>>> buecher = ["Per Anhalter durch die Galaxis", "Foundation-Zyklus", "Fliegende Fetzen"]
>>> empfehlungen = [autoren, buecher]
>>> print(empfehlungen)
[['Douglas Adams', 'Isaac Asimov', 'Terry Pratchett'], ['Per Anhalter durch die Galaxis', '
     Foundation-Zyklus', 'Fliegende Fetzen']]
>>> import copy
>>> mycopy = copy.deepcopy(empfehlungen)
```

Die Funktion `deepcopy` ist keine Build-In-Funktion von Python, weshalb wir die Bibliothek `copy` einbinden müssen. Darauf gehen wir noch einmal detaillierter im Abschnitt 3.2.3 ein.

Da die Liste einer der zentralen Datentypen von Python ist, gibt es auch entsprechend viele Methoden, die auf dieses Objekt angewendet werden können. Ein wichtiger Aspekt ist, dass diese Methoden i. d. R. kein Objekt zurückgeben, sondern nur das Objekt selbst modifizieren.

```
>>> liste=[ 1, -2, 3.5 , 1.4]
>>> myliste = liste.sort()
>>> print(myliste)
None
>>> print(liste)
[-2, 1, 1.4, 3.5]
```

Eigentlich ist das Verhalten nicht ungewöhnlich, wenn man an Java oder C++ denkt. Probleme entstehen hingegen manchmal daraus, dass Python mit seinem Laissez-faire-Stil Zuweisungen wie in der zweiten Zeile zulässt und `myliste` eben nur einfach den Wert `None` zuweist. Neben `sort` gibt natürlich noch viele weitere Funktionen zur Manipulation von Listen, die in der Online-Dokumentation von Python nachgelesen werden können: *https://docs.python.org/3/tutorial/datastructures.html*.

Wenn wir mit Kopien von Objekten arbeiten, stellt sich oft die Frage, ob diese bereits modifiziert wurden. Hier gilt, dass der Vergleichs-Operator == dies Element für Element überprüft:

```
>>> mycopy == empfehlungen
True
>>> mycopy[1][2]='Ab die Post'
>>> mycopy == empfehlungen
False
```

Ein weiterer wichtiger Operator, der gleichfalls auf Listen und Arrays funktioniert, ist der Slice-Operator. Wir werden ihn im Zusammenhang mit Arrays in Abschnitt 3.2.5 besprechen.

3.2.2 Funktionen

Bis jetzt haben wir Python nur interaktiv auf der Kommandozeile genutzt, was oft auch nützlich ist. Das primäre Ziel in diesem Buch ist es hingegen, Funktionen zu schreiben, die wiederverwertet werden können. Hierzu benötigen wir dann einen Editor, bzw. in Spyder wird unter `File → New File` eine leere Vorlage erzeugt, die i. A. in etwa folgendermaßen heißt: `untitled0.py`. Damit haben wir mit `.py` auch schon die gewünschte Endung für alle Python-Dateien, die wir erstellen. Alle Kommentare, die in dieser Vorlage stehen, können Sie ruhig löschen und die Datei mittels *save as* nach Bedarf benennen. Wir beginnen nun damit, eine kleine eigene Funktion zu schreiben.

Funktionen werden in Python entsprechend dem Schema:

```
def funktionsName(Parameterliste):
    Anweisungen
```

definiert. Zuerst steht also das Schlüsselwort **def**, gefolgt von dem Funktionsnamen. Diesem schließen sich die Funktionsparameter an, die innerhalb von runden Klammern übergeben werden.

Wer von einer Programmiersprache wie C/C++ oder Java kommt, wird den Typ für den Rückgabewert u. U. vermissen. Hier muss man sich zum einen daran erinnern, dass Python eine **dynamische Typisierung** (engl. *dynamic typing*) verwendet. Das bedeutet, dass Typprüfungen hier zur Laufzeit durchgeführt werden. Zum anderen handelt es sich um eine Prüfung, die man als **Duck-Typing** bezeichnet. Das bedeutet, dass die Objekte – zu denen wir gleich noch kommen – nicht primär durch ihre Klasse beurteilt werden, sondern nur danach, ob geeignete Methoden umgesetzt wurden. Der Ausdruck *Duck-Typing* geht dabei auf den recht bekannten Ausspruch

> *When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.*

zurück.

Diese liberale Umgang mit Typen ist manchmal sehr hilfreich und gleichzeitig für Umsteiger oft auch gewöhnungsbedürftig, u. a. wenn man wie hier nicht mehr an der ersten Zeile der Funktion erkennen kann, welcher Typ zurückgeliefert wird. Das geht so weit, dass eine Funktion sogar unterschiedliche Typen zurückliefern kann. Die Rückgabe erfolgt mit dem Schlüsselwort **return**. Wenn nach return kein Ausdruck steht oder falls return gänzlich fehlt, liefert die Funktion **None** zurück.

Nach dieser Kopfzeile folgt der Funktionskörper, der durch die Anweisungen gebildet wird. Alle Anweisungen müssen eingerückt sein. Wir machen uns das einmal an einem sehr einfachen Beispiel klar:

```
1  def mystemp ():
2      """ Gibt den aktuellen Zeitstempel im intern. Format als String zurueck"""
3      import time
4
5      today = time.localtime()
6
7      InternDatum = str(today.tm_year)+'-'+str(today.tm_mon)+'-'+str(today.tm_mday)
8      InternZeit  = str(today.tm_hour)+':'+str(today.tm_min)+':'+str(today.tm_sec)
9      InternDatumsformat = InternDatum + 'T' + InternZeit
10     return(InternDatumsformat)
```

```
11
12  print(mystemp())
```

Die Funktion mystemp erstreckt sich von Zeile 1 bis Zeile 10. Wird dieses Skript gestartet, wird Zeile 12 ausgeführt, welche die oben definierte Funktion aufruft. Dass Zeile 12 nicht mehr zu der Funktion gehört, wird alleine durch die Einrückung deutlich gemacht. Aus dem Code ist bei komplexeren Funktionen oft nur schwer zu erkennen, was zurückgeben wird. Um so wichtiger ist der in Zeile 2 eingebaute **doc String**. Er erlaubt eine kurze Dokumentation, die mithilfe von `help(mystemp)` aufgerufen werden kann. Darüber hinaus kann diese Technik in Zusammenhang mit dem Python-Dokumentations-Generator *pydoc* verwendet werden. In unserem Fall soll ein String zurückgeben werden, weshalb wir die von `time.localtime()` zurückgegebene Zeit erst einmal mittels **str** konvertieren müssen. Generell werden mittels der Funktion `str()` beliebige Python-Objekte in Strings umgewandelt.

Um die Funktion `time.localtime()` nutzen zu können, müssten wir `import time` in Zeile 3 notieren und so das Zeitmodul einbinden. Wie diese Module und die *Namespaces* in Python funktionieren, wird im folgenden Abschnitt kurz angesprochen.

3.2.3 Modularisierung

Eine `.py`-Datei, die nur Funktionen enthält (bzw. nur Klassendefinitionen, über die wir später kurz sprechen), ist quasi bereits eine Bibliothek. Eine solche Bibliothek – selbst erstellt oder vorgefertigt – wird mittels der **import-Anweisung** eingebunden. Wir haben das schon für `copy` und `time` gemacht. Eine weitere für uns wichtige Standardbibliothek ist `math`. Diese könnten wir mittels `import math` einbinden. Nach dem Schlüsselwort `import` können auch mehrere durch Komma getrennte Bibliotheken aufgeführt werden wie z. B. `import math, random`. Theoretisch können import-Anweisungen an jeder Stelle des Quellcodes stehen, i. A. werden diese jedoch direkt zu Beginn eingebunden, um die Übersichtlichkeit zu verbessern.

Ist die Bibliothek einmal importiert, stehen die Funktionen dieser Bibliothek in einem eigenen, geschützten Namensraum zur Verfügung. Das bedeutet: Nach `import math` kann auf den Kosinus nur über die Kombination aus Namensraum und Funktionsnamen zugegriffen werden, hier also `math.cos(x)`.

Man kann auch einzelne Funktionen aus der Bibliothek in den globalen Namensraum importieren:

```
>>> from math import cos, pi
>>> cos(-pi)
-1.0
```

Wem das zu umständlich ist, der kann die Funktionen der Bibliothek auch vollständig in den globalen Namensraum importieren, z. B. wie folgt:

```
>>> from math import *
>>> cos(-pi)
-1.0
```

Das Problem ist, dass dadurch auch der Schutz des Namensraumes ausgehebelt wird und vorher vorhandene Funktionen mit identischen Namen überschrieben werden. Ein guter Kom-

promiss sind häufig Abkürzungen, wie wir es für die NumPy später auch in Abschnitt 3.3 diskutieren werden. Hier ein Beispiel für ein Vorgehen mit Abkürzung:

```
>>> import math as Q
>>> Q.cos(Q.pi)
-1.0
```

Darüber hinaus gibt es noch eine weitere Sache, die Umsteigern ggf. Ärger bereiten kann: Python verwendet die **PYTHONPATH**-Variable als Angabe, wo nach Modulen gesucht werden soll. Ist das aktuelle Verzeichnis nicht in diesem Pfad, wird dort nicht gesucht. Mit `Spyder` wird das meiste hierbei sehr komfortabel abgefangen; so gibt es z. B. unter `Tools` im Menü den `PYTHONPATH-Manager`, der die Verwaltung erleichtert. Wer das hingegen von Hand tun will oder tun muss, kann den Pfad wie folgt um ein weiteres Verzeichnis erweitern:

```
>>> import sys
>>> sys.path.append('meinVerzeichnis')
```

Wenn man selbst an einer Bibliothek arbeitet, kommt es naturgemäß häufig zu Änderungen. Das Problem ist, dass diese Änderungen sich teilweise nur nach dem Importieren auswirken. Will man nicht ständig eine Python-Konsole öffnen und schließen, bietet sich **importlib** an.

```
>>> import importlib
>>> importlib.reload(NameDerLib)
```

Durch dieses Vorgehen wird `NameDerLib` erneut eingelesen und werden Änderungen propagiert.

Um Funktionen wirklich mit Leben zu erfüllen, braucht man im Allgemeinen Kontrollstrukturen, über die wir noch gar nicht gesprochen haben.

3.2.4 Kontrollstrukturen und Schleifenformen

Wie schon bei den Funktionen zuvor kommt der Einrückung des Quellcodes in Python eine strukturierende Rolle zu, was bedeutet, dass z. B. C-Programmierer ihre Klammern { } zu Gunsten einer konsequenten Einrückung eintauschen müssen.

Generell stehen die klassischen Kontrollstrukturen in Python zur Verfügung, und zumindest die `if`-Abfrage und die `While`-Schleife unterscheiden sich kaum von anderen Sprachen. Hier ein Beispiel, in dem ein wenig mit `if` und `while` herumgespielt wird.

```
1  foo = 21;
2  if foo == 42:
3      print("This is the answer")
4  elif foo == 21:
5      print('At least ...')
6      print('it is half of the truth')
7  else:
8      print("I'm sorry, Dave, I'm afraid I can't do that.")
9
10 wurzel = foo
11 while abs(wurzel**2 - foo) > 10**-7 :
12     wurzel = 0.5 * (wurzel + foo/wurzel)
13
14 print("Die Wurzel von %e ist %e" % (foo,wurzel) )
```