Teil 1

Von null auf Bilderkennung

Dieser Teil gibt eine Einführung in überwachtes Lernen. Innerhalb von nur zwei Kapiteln werden wir ein erstes Machine-Learning-System programmieren, das wir anschließend Schritt für Schritt weiterentwickeln, bis es leistungsfähig genug ist, um handgeschriebene Ziffern zu erkennen.

Ja, Sie haben richtig gelesen: Auf den nächsten *etwa 100 Seiten* werden wir ein Programm zur Bilderkennung schreiben. Was noch besser ist: Wir werden dazu keine Machine-Learning-Bibliothek verwenden. Abgesehen von einigen Allzweckfunktionen für arithmetische Berechnungen und grafische Darstellung schreiben wir den ganzen Code selbst.

Es ist sehr unwahrscheinlich, dass Sie in Ihrer zukünftigen Karriere jemals Machine-Learning-Algorithmen von Grund auf selbst schreiben müssen. Aber es einmal zu tun, um die Grundlagen richtig zu verstehen, ist von unschätzbarem Wert. Sie werden genau wissen, was jede einzelne Zeile des fertigen Programms tut. Danach wird Machine Learning für Sie nie wieder wie schwarze Magie wirken.

1

Einführung in Machine Learning

Softwareentwickler erzählen sich gern Veteranengeschichten. Sobald ein paar von uns in einer Kneipe zusammensitzen, fragt einer: »An was für Projekten arbeitet ihr gerade?« Dann nicken wir heftig und hören uns die amüsanten und teilweise furchtbaren Geschichten der anderen an.

Bei einem dieser abendlichen Geplänkel Mitte der 90er erzählte mir eine Freundin von dem unmöglichen Auftrag, an dem sie gerade arbeitete. Ihre Vorgesetzten wünschten sich von ihr ein Programm, das Röntgenaufnahmen analysieren und dadurch Krankheiten erkennen konnte, etwa eine Lungenentzündung.

Meine Freundin warnte die Geschäftsleitung vor, dass das ein hoffnungsloses Unterfangen sei, aber man wollte ihr nicht glauben. Wenn ein Radiologe das leisten konnte, so argumentierten die Manager, warum dann nicht auch ein Visual-Basic-Programm? Sie stellten ihr sogar einen Radiologen zur Seite, damit sie lernte, wie er vorging, und dies in Code umsetzen konnte. Diese Erfahrung bestärkte sie jedoch nur in ihrer Meinung, dass Radiologie menschliches Urteilsvermögen und menschliche Intelligenz erforderte.

Wir lachten über die Sinnlosigkeit dieser Aufgabe. Ein paar Monate später wurde das Projekt aufgegeben.

Doch kehren wir nun in die Gegenwart zurück. 2017 veröffentlichte ein Forschungsteam der Stanford University einen Algorithmus, um Lungenentzündung anhand von Röntgenbildern zu erkennen.[1] Er erfüllte nicht nur seine Aufgabe, sondern war sogar zuverlässiger als ein professioneller Radiologe. Das hatte als unmöglich gegolten. Wie hatten die Forscher es geschafft, diesen Code zu schreiben?

Die Antwort lautet: gar nicht. Anstatt Code zu schreiben, setzten sie Machine Learning ein. Sehen wir uns an, was das bedeutet.

Programmierung und Machine Learning im Vergleich

Das folgende Beispiel zeigt den Unterschied zwischen Machine Learning (oder kurz ML) und gewöhnlicher Programmierung. Stellen Sie sich vor, Sie sollen ein Programm erstellen, das Videospiele spielt. Bei der traditionellen Programmierung würden Sie dazu Code wie den folgenden schreiben:

```
enemy = get_nearest_enemy()
if enemy.distance() < 100:
  decelerate()
  if enemy.is_shooting():
    raise_shield()
  else:
    if health() > 0.25:
      shoot()
    else:
      rotate_away_from(enemy)
else:
  # ... und noch viel mehr Code
```

Und so weiter. Der Großteil des Codes würde aus einer Riesenmenge von `if... else`-Anweisungen vermischt mit Befehlen wie `shoot()` bestehen.

Moderne Sprachen bieten uns zwar Möglichkeiten, diese hässlichen, verschachtelten `if`-Anweisungen durch angenehmere Konstruktionen wie Polymorphismus, Mustererkennung oder ereignisgestützte Aufrufe zu ersetzen, aber das Grundprinzip der Programmierung bleibt unverändert: Sie sagen dem Computer, wonach er Ausschau halten und was er tun soll. Dabei müssen Sie jede mögliche Bedingung aufführen und jede mögliche Aktion definieren.

Mit dieser Vorgehensweise sind wir weit gekommen, aber sie hat auch einige Nachteile. Erstens dürfen Sie nichts auslassen. Wahrscheinlich können Sie sich Dutzende oder gar Hunderte von besonderen Situationen vorstellen, die Sie in dem Videospielprogramm berücksichtigen müssen. Was geschieht, wenn sich ein Gegner nähert, sich aber ein Power-up zwischen Ihnen und ihm befindet, das Sie vor

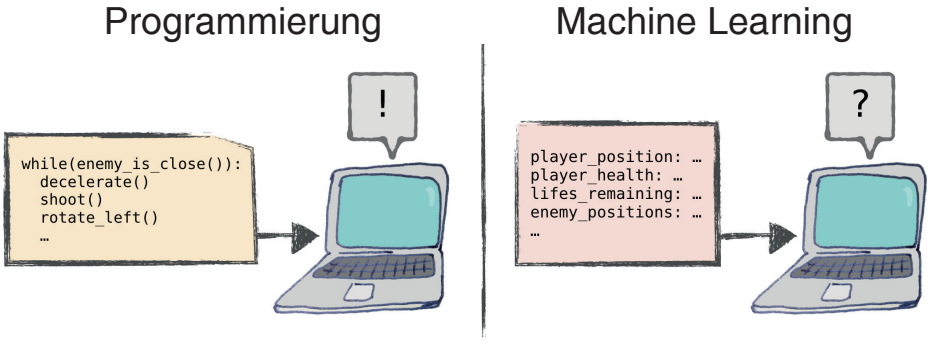
1. *news.stanford.edu/2017/11/15/algorithm-outperforms-radiologists-diagnosing-pneumonia*

seinen Schüssen schützen kann? Ein menschlicher Spieler wird eine solche Situation schnell erkennen und zu seinem Vorteil nutzen. Kann ein Programm das auch? Das hängt ganz von dem Programm ab. Wenn Sie diesen Sonderfall beim Schreiben des Codes berücksichtigt haben, dann kann das Programm damit umgehen. Allerdings wissen wir, wie schwer es ist, selbst in so eng umrissenen Aufgabenfeldern wie der Buchhaltung jegliche Sonderfälle abzudecken. Wenn Sie sämtliche Sonderfälle auf so komplexen Gebieten wie dem Spielen von Videospielen, dem Fahren eines Lkw oder dem Erkennen eines Bilds auflisten wollen, kann ich Ihnen dazu nur viel Glück wünschen.

Es reicht aber nicht nur, all diese Fälle aufzulisten; Sie müssen auch wissen, wie Sie dabei jeweils eine Entscheidung fällen. Das ist die zweite große Einschränkung bei der Programmierung, die in manchen Fachgebieten schon das Aus bedeutet. Betrachten Sie zum Beispiel eine Aufgabe aus dem Bereich des *maschinellen Sehens* wie das bereits erwähnte Erkennen einer Lungenentzündung anhand einer Röntgenaufnahme.

Wir wissen nicht *genau*, wie ein Radiologe eine Lungenentzündung erkennt. Wir haben zwar eine grobe Vorstellung davon, dass er nach undurchsichtigen Bereichen sucht, aber wir wissen nicht, wie sein Gehirn solche undurchsichtigen Bereiche erkennt und bewertet. Manchmal kann ein solcher Experte selbst nicht erklären, wie er zu der Diagnose gekommen ist, sondern nur vage Begründungen geben wie: »Ich weiß aus Erfahrung, dass eine Lungenentzündung nicht so aussieht.« Da wir nicht wissen, wie solche Entscheidungen ablaufen, können wir einen Computer auch nicht anweisen, sie zu fällen. Dieses Problem stellt sich bei allen typisch menschlichen Aufgaben, etwa dem Verkosten von Bier oder dem Verstehen eines Satzes.

Machine Learning dagegen stellt das Prinzip der traditionellen Programmierung auf den Kopf: Der Computer erhält keine *Anweisungen*, sondern *Daten*, und wird aufgefordert, selbst herauszufinden, was er tun soll.



Dass der Computer selbst etwas herausfinden soll, klingt wie Wunschdenken. Allerdings gibt es tatsächlich einige Möglichkeiten, um das zu erreichen. Für alle diese Vorgehensweisen ist nach wie vor die Ausführung von Code erforderlich, allerdings ist dieser Code im Gegensatz zur herkömmlichen Programmierung keine schrittweise Anleitung, um das vorliegende Problem zu lösen. Beim Machine Learning teilt der Code dem Computer mit, wie er die Daten verarbeiten soll, um das Problem selbst zu lösen.

Betrachten wir als Beispiel wieder einen Computer, der herausfinden soll, wie man ein Videospiel spielt. Stellen Sie sich einen Algorithmus vor, der durch Versuch und Irrtum zu spielen lernt. Zu Anfang gibt er zufällige Befehle: schießen, bremsen, drehen usw. Er merkt sich, wann die Befehle zum Erfolg führen, etwa zu einem höheren Punktestand, aber auch, wann sie in einem Fehlschlag wie dem Tod der Spielfigur resultieren. Außerdem registriert er den Spielzustand: wo sich die Gegner, die Hindernisse und die Power-ups befinden, wie viele Gesundheitspunkte die Figur hat usw.

Wenn der Algorithmus später einem ähnlichen Spielzustand begegnet, wird er mit höherer Wahrscheinlichkeit erfolgreiche Aktionen durchführen. Eine Vielzahl solcher Trial-and-Error-Durchläufe kann aus dem Programm einen fähigen Spieler machen. Mit dieser Vorgehensweise erreichte ein System im Jahr 2013 geradezu übermenschliche Fähigkeiten bei einer Reihe alter Atari-Spiele.[2]

Diese Form von ML wird als *bestärkendes Lernen* oder *Reinforcement Learning* bezeichnet. Es funktioniert fast genauso wie Hundetraining: »Gutes« Verhalten wird belohnt, sodass der Hund es häufiger an den Tag legt. (Ich habe das auch mit meiner Katze versucht, aber bislang ohne Erfolg.)

Reinforcement Learning ist jedoch nur eine Möglichkeit, mit der ein Computer die Lösung eines Problems herausfinden kann. Der Schwerpunkt dieses Buchs liegt dagegen auf einer anderen Form von Machine Learning, nämlich der wahrscheinlich am häufigsten verwendeten. Sehen wir uns das genauer an.

Überwachtes Lernen

Unter den verschiedenen Ansätzen des ML hat das *überwachte Lernen* (*Supervised Learning*) bisher die beeindruckendsten Ergebnisse hervorgebracht. Damit lassen sich auch Aufgaben wie die Diagnose einer Lungenentzündung lösen.

Ausgangspunkt des überwachten Lernens ist eine Reihe von *Beispielen*, die jeweils mit *Labels* versehen sind, aus denen der Computer etwas lernen kann:

2. *deepmind.com/research/publications/playing-atari-deep-reinforcement-learning*

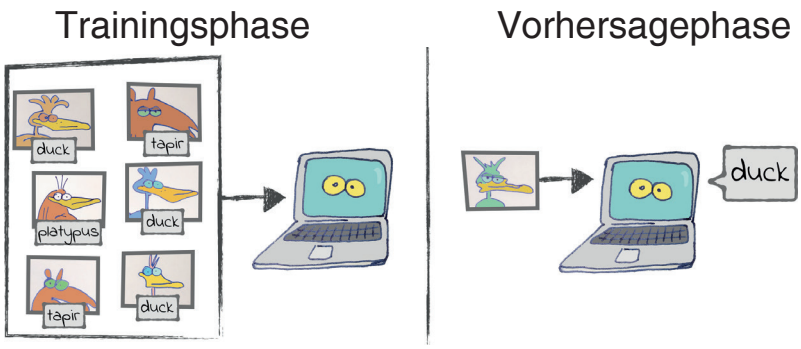
Zweck	Beispiel	Label
Ein System, das aus dem Bellen eines Hundes die Hunderasse bestimmt	`.wav`-Datei mit Hundegebell	Die Hunderasse, also etwa »Windhund«, »Beagle« usw.
Ein System, das eine Lungenentzündung erkennen kann	Röntgenaufnahme	Ein boolesches Flag: 1, wenn die Aufnahme eine Lungenentzündung zeigt, anderenfalls 0
Ein System, das die Einnahmen einer Eisdiele anhand des Wetters vorhersagt	Aufgezeichnete Temperaturen an einem vergangenen Tag	Die aufgezeichneten Einnahmen an diesem Tag
Ein System, das erkennt, welche Stimmung der Tweet eines Politikers vermittelt	Ein Tweet	Eine Stimmung wie »entrüstet«, »verärgert« oder »wütend«

Wie Sie sehen, können als Beispiele viele verschiedene Dinge genutzt werden: Daten, Text, Klänge, Videos usw. Es gibt auch zwei Arten von Labels: *Numerische Labels* sind einfach nur Zahlen, wie es etwa bei dem Temperatur-Eisdielenumsätze-Umrechner der Fall ist. *Kategorielabels* dagegen stehen wie bei dem Hunderassendetektor für eine Kategorie (oder Klasse) aus einer vordefinierten Menge. Wenn Sie Ihre Fantasie spielen lassen, werden Ihnen noch viele weitere Möglichkeiten einfallen, um etwas – seien es Zahlen oder Kategorien – aus etwas anderem zu bestimmen.

Nehmen wir an, dass wir bereits eine Reihe solcher Beispiele mit Labels zusammengestellt haben. Daran schließen sich nun die beiden folgenden Phasen des überwachten Lernens an:

1. In der ersten Phase speisen wir die mit Labeln gekennzeichneten Beispiele in einen Algorithmus ein, der Muster erkennen kann. Ein solcher Algorithmus kann beispielsweise bemerken, dass alle Aufnahmen, die eine Lungenentzündung zeigen, gemeinsame Merkmale aufweisen, etwa undurchsichtige Bereiche, die in den Aufnahmen ohne Lungenentzündung nicht zu finden sind. Dies ist die sogenannte *Trainingsphase*, in der der Algorithmus die Beispiele immer wieder durchsieht, um zu lernen, solche Muster zu erkennen.
2. Wenn der Algorithmus weiß, wie eine Lungenentzündung aussieht, gehen wir zur *Vorhersagephase* über, in der wir die Früchte unserer Arbeit ernten. Wir zeigen dem trainierten Algorithmus eine Röntgenaufnahme *ohne Label*, woraufhin uns der Algorithmus sagt, ob sie Anzeichen einer Lungenentzündung aufweist oder nicht.

Auch das folgende System zur Erkennung von Tierarten ist ein Beispiel für überwachtes Lernen. Hier sind die einzelnen Eingaben die Bilder von Tieren mit der Angabe der Art als Label. In der Trainingsphase zeigen wir dem Algorithmus Bilder mit Labels und in der Vorhersagephase Bilder ohne Labels, wobei der Algorithmus die Labels rät:



Ich habe bereits gesagt, dass der Computer beim Machine Learning etwas aus den Daten »herausfindet«. Das überwachte Lernen ist ein Beispiel eines solchen Vorgangs. Bei der herkömmlichen Programmierung führen Sie den Computer mit Ihrem Code von der Eingabe zur Ausgabe. Beim überwachten Lernen dagegen stellen Sie dem Computer Beispiele von Ein- und Ausgaben zur Verfügung, anhand derer er selbst herausfindet, wie er von den einen zu den anderen kommt.

Möglicherweise hat dieser grobe Überblick über das überwachte Lernen mehr Fragen aufgeworfen als beantwortet. Wir haben gesagt, dass ein Programm zum überwachten Lernen Muster und gemeinsame Merkmale in den Daten »erkennt«. Aber wie macht es das? Schauen wir uns also genauer an, wie dieses Zauberkunststück tatsächlich funktioniert.

Die Mathematik hinter dem Zaubertrick

Um die Beziehung zwischen Daten und ihren Labels zu erkennen, nutzt ein System zum überwachten Lernen das mathematische Prinzip der *Näherungsfunktion* aus. Sehen wir uns an einem konkreten Beispiel an, was das bedeutet.

Nehmen wir an, Sie haben eine Solaranlage auf Ihrem Dach und wollen ein System zum überwachten Lernen konstruieren, das lernt, in welchem Maße diese Anlage Energie erzeugt, und die in Zukunft generierten Energiemengen vorhersagen kann.

Der Ertrag einer Solaranlage hängt von verschiedenen Variablen ab, darunter der Tageszeit, dem Wetter usw. Besonders wichtig scheint die Tageszeit zu sein, weshalb Sie sich darauf konzentrieren. In der typischen Vorgehensweise für über-

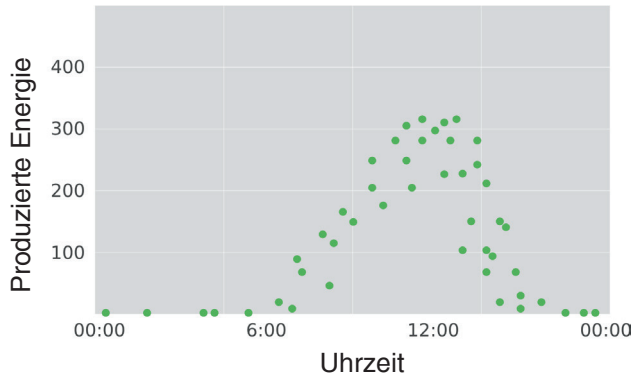
wachtes Lernen sammeln Sie Beispiele für die an verschiedenen Tageszeiten produzierte Energie. Nach einigen Wochen, in denen Sie zufällige Stichproben aufgenommen haben, liegt Ihnen die folgende Tabelle vor:

Uhrzeit	Energie (Wattstunden)
09:01	153
11:48	280
05:20	0

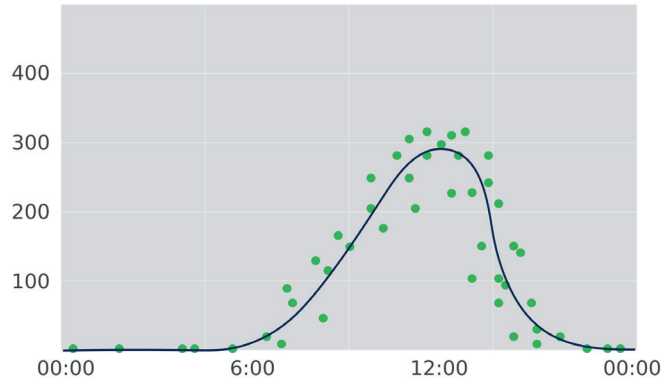
... usw.

Jede Zeile dieser Tabelle ist ein Beispiel, das jeweils aus einer *Eingabevariablen* (der Uhrzeit) und einem *Label* (der produzierten Energie) besteht. Das ist genauso wie bei dem System zum Erkennen von Tieren, bei denen ein Bild als Eingabe diente und der Name des Tiers als Label angegeben wurde.

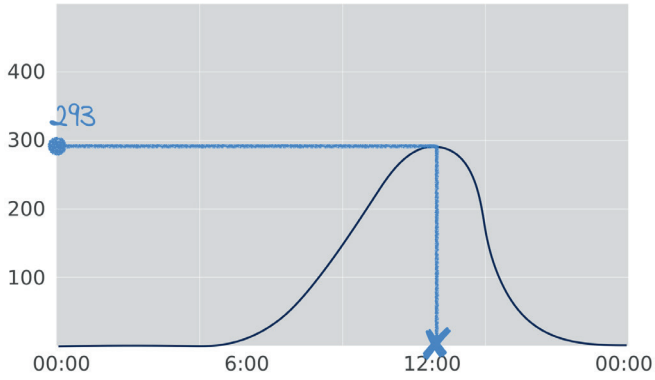
Wenn Sie die Beispiele wie folgt in ein Diagramm einzeichnen, können Sie unmittelbar ablesen, in welchem Zusammenhang die Uhrzeit zur produzierten Energie steht:



Unser menschliches Gehirn erkennt auf den ersten Blick, dass die Solaranlage nachts keine Energie produziert und dass der Ertrag um die Mittagszeit einen Spitzenwert annimmt. Einem System zum überwachten Lernen fehlt eine Luxuseinrichtung wie unser Gehirn, allerdings kann es die Daten verstehen, indem es sie durch eine Funktion annähert, wie Sie im folgenden Diagramm sehen:



Die passende Näherungsfunktion zu den Beispielen zu finden, ist der schwierige Teil. Dies ist, was ich früher als »Trainingsphase« bezeichnet habe. Die anschließende Vorhersagephase dagegen ist einfach: Das System kümmert sich dann nicht mehr um die Beispiele, sondern nutzt die Funktion, um die von der Solaranlage produzierte Energie vorherzusagen, etwa zur Mittagszeit an beliebigen Tagen:



Das war es, was ich weiter vorn mit der Aussage meinte, beim überwachten Lernen werde das Prinzip der Näherungsfunktion genutzt: Das System erhält reale Daten, die gemeinhin unsauber und unvollständig sind. In der Trainingsphase nähert es diese komplizierten Daten durch eine relativ einfache Funktion an, die es dann in der Vorhersagephase nutzt, um unbekannte Daten zu prognostizieren.

Als Programmierer sind Sie es gewohnt, an all die verschiedenen Dinge zu denken, die schiefgehen können. Wahrscheinlich sind Ihnen deshalb bereits zahlreiche Möglichkeiten eingefallen, um unser Beispiel komplizierter zu machen. Erstens hängt der Ertrag einer Solaranlage noch von anderen Variablen ab als der Uhrzeit, etwa der Bewölkung und der Jahreszeit. Wenn wir auch diese Variablen erfassen, haben wir eine mehrdimensionale Wolke aus Datenpunkten, die wir nicht mehr in einem Diagramm wie dem vorherigen grafisch darstellen können. Außer-

dem wollen wir im Fall unserer Solaranlage ein *numerisches* Label voraussagen. Wahrscheinlich fragen Sie sich, wie sich die hier gezeigte Methode auch für nicht numerische Labels wie Tiernamen, also für *Kategorielabels*, umsetzen lässt.

All diese Dinge werden wir uns in diesem Buch noch ansehen. Für den Augenblick wollen wir jedoch nur festhalten, dass das Grundprinzip des überwachten Lernens unabhängig von allen zusätzlichen Komplikationen der hier gegebenen Beschreibung folgt: Wir gehen von einer Reihe von Beispielen aus und suchen eine Näherungsfunktion dafür.

Moderne Systeme für überwachtes Lernen sind sehr gut darin, diese Annäherung vorzunehmen. Sie können selbst komplizierte Beziehungen wie die zwischen Röntgenbildern und einer medizinischen Diagnose annähern. Die Annäherungsfunktion dafür sieht für Menschen zwar wahnsinnig kompliziert aus, ist für solche Systeme aber nichts Ungewöhnliches.

Und das ist auch schon kurz und bündig, worum es beim überwachten Lernen geht. Die Einzelheiten ... nun, die sehen wir uns im Verlaufe dieses Buchs an. Im Folgenden wollen wir daher unsere Computer einrichten, damit wir mit dem Programmieren beginnen können.

Und was ist mit unüberwachtem Lernen?

Es gibt verschiedene Spielarten des Machine Learning. In den meisten Lehrbüchern werden drei erwähnt: bestärkendes Lernen (Reinforcement Learning), überwachtes Lernen (Supervised Learning) und als dritte Variante das *unüberwachte Lernen* (*Unsupervised Learning*). Wir haben zur Einführung in die Grundlagen von ML Reinforcement Learning herangezogen und werden uns im Rest dieses Buchs mit überwachtem Lernen beschäftigen. Was aber ist unüberwachtes Lernen?

Theoretisch besteht der Unterschied zwischen überwachtem und unüberwachtem Lernen darin, dass die Beispieldaten in letzterem Fall keine Labels haben. In der Praxis hat unüberwachtes Lernen jedoch nicht viel mit unserer üblichen Vorstellung von »Lernen« zu tun, sondern wirkt eher wie eine äußerst anspruchsvolle Art von Datenverarbeitung.

Als Beispiel für unüberwachtes Lernen stellen Sie sich vor, dass Sie Marktforschung für einen Onlineshop betreiben und dabei alle Daten über die Kunden vorliegen haben: wie viel sie ausgeben, wie oft sie den Shop besuchen usw. Ein Algorithmus für unüberwaches Lernen kann Ihnen helfen, sinnvolle Zusammenhänge aus diesen Daten abzulesen, indem er ähnliche Kunden zu Gruppen zusammenfasst – ein Vorgang, der als *Clustering* bezeichnet wird.

Unüberwachtes Lernen kann sehr praktisch sein, steht heutzutage aber nicht so sehr im Rampenlicht wie die anderen Varianten von ML. Für dieses Buch musste ich mich auf ein Thema beschränken und habe mich deshalb auf überwachtes Lernen konzentriert.

Das System einrichten

Den ersten Teil dieses Buchs nimmt ein ML-Tutorial ein. Wir fangen dabei bei null an und haben am Ende ein funktionierendes Programm für maschinelles Sehen, das uns später als Ausgangspunkt dient, um die höheren Weihen zu empfangen: neuronale Netze in Teil II und Deep Learning in Teil III.

Am besten ist es, das Tutorial gleich bei der Lektüre durchzuarbeiten, also den angegebenen Quellcode auszuführen und die Übungsaufgaben zu lösen, die am Ende der meisten Kapitel stehen. Es ist jedoch auch möglich, erst alles zu lesen, um sich einen Überblick zu verschaffen, und erst dann zur Tastatur zu greifen. Beide Vorgehensweisen sind geeignet, wobei Programmierer jedoch gewöhnlich die erste bevorzugen.

Es dauert auch nicht lange, Ihr System so einzurichten, dass Sie den angegebenen Code ausführen können. ML erfordert zwar gewöhnlich eine hohe Rechenleistung, doch der Code in diesem Buch läuft auch auf einem normalen Laptop sehr gut. Sie müssen lediglich etwas Software installieren.

Vor allem benötigen Sie Python, die am häufigsten für ML verwendete Sprache, in der auch die Beispiele in diesem Buch geschrieben sind. Machen Sie sich keine Sorgen, falls Sie noch nie in Python programmiert haben – Sie werden überrascht sein, wie gut lesbar diese Sprache ist. Wenn Sie Schwierigkeiten haben sollten, den Python-Code zu verstehen, lesen Sie Anhang 1, »*Grundlagen von Python*«. Darin erfahren Sie alles, was Sie für dieses Buch benötigen.

Hinweis für erfahrene Pythonistas

Wenn Sie sich mit Python auskennen, werden Sie bemerken, dass der Code in diesem Buch teilweise von den üblichen Konventionen abweicht. Beispielsweise habe ich bewusst auf einige sprachspezifische Konstruktionen wie die Listennotation verzichtet, die den Code für Einsteiger unnötig kompliziert machen. In demselben Bemühen, mich möglichst verständlich auszudrücken, verwende ich auch manche Fachbegriffe etwas lockerer, z. B. »Funktion« statt »Methode«. Für meine Verstöße gegen die Python-Etikette möchte ich mich bereits im Voraus entschuldigen.

Fangen wir an! Als Erstes müssen Sie prüfen, ob Python bei Ihnen installiert ist. Führen Sie folgenden Befehl aus:

```
python3 --version
```

Wenn Sie Python 3 nicht haben, dann beschaffen Sie es sich, bevor Sie weiterlesen. In Anhang 1, »*Grundlagen von Python*«, erfahren Sie, wie Sie die Sprache installieren.

Achtung: Auf manchen Systemen können Sie Python 3 auch mit `python` ohne die angehängte `3` ausführen, während dieser Befehl auf anderen Systemen stattdessen Python 2 ausführt. Um Verwirrung und mysteriöse Fehler aufgrund der Verwendung einer älteren Python-Version zu verhindern, sollten Sie immer den Befehl `python3` verwenden.

Wenn Sie die Sprache installiert haben, müssen wir noch ein Wort über die Bibliotheken verlieren. Für den Anfang benötigen Sie drei davon. An erster Stelle steht dabei NumPy, eine Bibliothek für wissenschaftliche Berechnungen. Außerdem verwenden wir zwei Bibliotheken zur Ausgabe von Diagrammen. Matplotlib ist der De-facto-Standard für Diagramme in Python. Seaborn baut auf Matplotlib auf und dient vor allem dazu, Diagrammen ein angenehmes Erscheinungsbild zu verleihen.

Um diese Bibliotheken zu installieren, können Sie entweder `pip` verwenden, den offiziellen Paketmanager von Python, oder Conda, einen anspruchsvollen Umgebungsmanager, der insbesondere im Umfeld von ML sehr beliebt ist. Im Abschnitt »*Pakete mit Conda installieren*« auf *Seite 340* werden die Unterschiede zwischen `pip` und Conda genauer beleuchtet. Im Zweifelsfall nehmen Sie `pip`.

Zur Installation der Bibliotheken mit `pip` führen Sie die folgenden Befehle aus:

```
pip3 install numpy==1.15.2
pip3 install matplotlib==3.1.2
pip3 install seaborn==0.9.0
```

Das war es auch schon! Wenn Sie lieber Conda verwenden, richten Sie sich nach den Anweisungen in der Datei `readme.txt`, die Sie im Wurzelverzeichnis des Quellcodes finden.

Schließlich brauchen Sie auch noch eine Programmierumgebung. In vielen ML-Tutorials wird Jupyter Notebook verwendet, womit Sie den Code im Browser bearbeiten und ausführen können. Für die Beispiele in diesem Buch ist es jedoch nicht unbedingt erforderlich, Jupyter Notebook einzusetzen. Als Entwickler sind Sie ja schon damit vertraut, Programme zu schreiben und auszuführen. Nutzen Sie also einfach den Texteditor oder die IDE Ihrer Wahl. Für den Fall, dass Sie Jupyter bereits kennen und verwenden, finden Sie im Verzeichnis `notebooks` die Jupyter-Version des Codes zu diesem Buch.

Gehen wir die Liste noch einmal durch: Sie haben Python, drei wichtige Bibliotheken und Ihren bevorzugten Editor. Mehr brauchen Sie nicht, um anfangen zu können.

Als Nächstes wollen wir damit beginnen, ein Programm zu schreiben, das lernen kann.

Gewöhnen Sie sich an ML

Als Entwickler sind Sie es gewohnt, rasch zu lernen. Bei ML jedoch wagen Sie sich auf völlig neues Terrain vor. Ich will es nicht beschönigen: Die nächsten drei oder vier Kapitel sind schwer. Bei der Lektüre werden Sie sich womöglich wie ein blutiger Anfänger vorkommen, was zwar durchaus spannend, aber manchmal auch sehr frustrierend sein kann.

Ich kenne dieses Gefühl, aber ich kann Ihnen versichern, dass es sich lohnt, diese Hürden zu überwinden. Ich kann mich noch gut daran erinnern, wie faszinierend es war, als ich zum ersten Mal ein Machine-Learning-Programm ausführte, das eine korrekte Vorhersage machte. Halten Sie durch, und schon bald werden auch Sie dieses Glücksgefühl erleben!