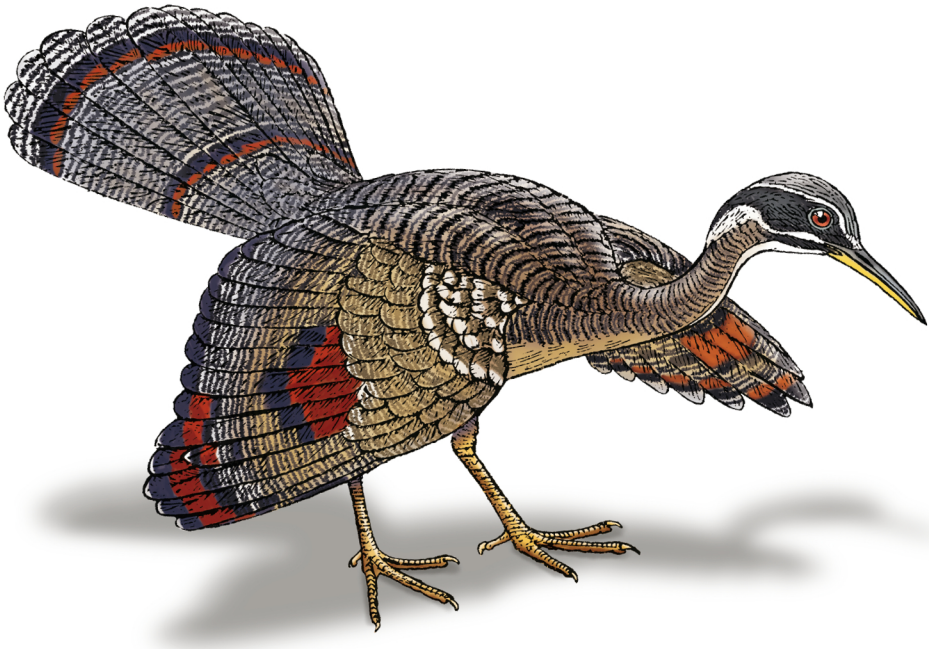
O'REILLY®

Best Practices für die gesamte ML-Pipeline

Design Patterns für Machine Learning

Entwurfsmuster für Datenaufbereitung, Modellbildung und MLOps



Valliappa Lakshmanan,
Sara Robinson & Michael Munn

Übersetzung von Frank Langenau

Design Patterns für Machine Learning

Entwurfsmuster für Datenaufbereitung, Modellbildung und MLOps

Valliappa Lakshmanan, Sara Robinson & Michael Munn

Deutsche Übersetzung von Frank Langenau

O'REILLY®

Valliappa Lakshmanan, Sara Robinson und Michael Munn

Lektorat: Alexandra Follenius
Übersetzung: Frank Langenau
Korrektorat: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Karen Montgomery, Michael Oréal, *www.oreal.de*
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

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:
Print 978-3-96009-164-6
PDF 978-3-96010-596-1
ePub 978-3-96010-597-8
mobi 978-3-96010-598-5

1. Auflage 2022

Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of *Machine Learning Design Patterns*, ISBN 9781098115784 © 2021 Valliappa Lakshmanan, Sara Robinson, and Michael Munn. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«. O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

PEFC zertifiziert
Das Papier für dieses Buch stammt aus nachhaltig bewirtschafteten Wäldern und kontrollierten Quellen.
PEFC™
PEFC/04-31-0810 www.pefc.de

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: *kommentar@oreilly.de*.

5 4 3 2 1 0

Inhalt

Vorwort

Für wen dieses Buch gedacht ist

Einführende Bücher zum Machine Learning konzentrieren sich normalerweise auf das *Was* und *Wie* des maschinellen Lernens (ML). Sie erläutern dann die mathematischen Aspekte neuer Methoden aus KI-Forschungseinrichtungen und lehren, wie sich diese Methoden mithilfe von KI-Frameworks implementieren lassen. Dieses Buch hingegen bringt die hart erarbeiteten Erfahrungen rund um das *Warum* zusammen, das den Tipps und Tricks zugrunde liegt, auf die erfahrene ML-Praktiker:innen setzen, wenn sie maschinelles Lernen auf reale Probleme anwenden.

Wir gehen davon aus, dass Sie über Vorkenntnisse zu maschinellem Lernen und Datenverarbeitung verfügen. Dies ist kein Grundlagenlehrbuch für maschinelles Lernen. Vielmehr richtet sich dieses Buch an Data Scientists oder ML Engineers, die nach einem zweiten Buch über praktisches Machine Learning suchen. Wenn Sie die Grundlagen bereits kennen, präsentiert Ihnen dieses Buch einen Katalog von Ideen, von denen Sie (als ML-Praktiker:in) vielleicht einige wiedererkennen, und gibt diesen Ideen einen Namen, damit Sie zielsicher nach ihnen greifen können.

Wenn Sie Informatikstudent:in sind und einen Job in der Industrie anstreben, wird dieses Buch Ihr Wissen abrunden und Sie auf die Berufswelt vorbereiten. Es wird Ihnen helfen, zu lernen, wie man hochwertige ML-Systeme aufbaut.

Was Sie nicht im Buch finden

In erster Linie richtet sich das Buch an ML Engineers in Unternehmen, nicht an ML-Wissenschaftler:innen in akademischen oder industriellen Forschungslabors.

Wir gehen absichtlich nicht auf Bereiche der aktiven Forschung ein – so werden Sie hier sehr wenig zur Modellarchitektur des maschinellen Lernens finden (wie zum Beispiel bidirektionale Encoder, den Aufmerksamkeitsmechanismus oder Kurzschlussschichten), da wir annehmen, dass Sie mit einer vorgefertigten Modellarchitektur (wie etwa ResNet-50 oder GRUCell) arbeiten und nicht Ihr eigenes Bildklassifizierungs- oder rekurrentes neuronales Netz schreiben werden.

Die folgenden Punkte listen einige konkrete Beispiele für Bereiche auf, von denen wir uns absichtlich fernhalten, da wir glauben, dass diese Themen eher für Hochschulkurse und ML-Forscher geeignet sind:

ML-Algorithmen
: Zum Beispiel behandeln wir nicht die Unterschiede zwischen Random Forests und neuronalen Netzen. Damit beschäftigen sich Lehrbücher, die in das maschinelle Lernen einführen.

Bausteine
: Verschiedene Arten von Optimierern für den Gradientenabstieg oder Aktivierungsfunktionen behandeln wir ebenfalls nicht. Wir empfehlen hier Adam und ReLU – unserer Erfahrung nach ist das Potenzial für Verbesserungen der Performance durch verschiedene Auswahlen bei derartigen Dingen eher gering.

ML-Modellarchitekturen
: Wenn Sie Bilder klassifizieren, verwenden Sie am besten ein Standardmodell wie ResNet oder was auch immer der letzte Schrei ist, wenn Sie dies lesen. Überlassen Sie den Entwurf neuer Modelle für die Bild- oder Textklassifizierung den Forschern, die sich auf derartige Probleme spezialisiert haben.

Modellebenen
: In diesem Buch werden Sie keine Convolutional Neural Networks und keine rekurrenten neuronalen Netze finden, und zwar gleich aus zwei Gründen – erstens, weil sie Bausteine sind, und zweitens, weil sie zu den Dingen gehören, die bereits als Standardlösungen zur Verfügung stehen.

Benutzerdefinierte Trainingsschleifen
: Der einfache Aufruf von `model.fit()` in Keras dürfte den Ansprüchen von Praktikerinnen und Praktikern genügen.

Wir haben versucht, in dieses Buch nur gängige Muster aufzunehmen – wie sie etwa ML Engineers in Unternehmen bei ihrer täglichen Arbeit verwenden.

Nehmen Sie als Analogie dazu Datenstrukturen. Während ein Hochschulkurs über Datenstrukturen eingehend die Implementierungen der verschiedenen Datenstrukturen erläutert und ein Forscher zu Datenstrukturen lernen muss, wie man ihre mathematischen Eigenschaften formal darstellt, können Fachleute pragmatischer herangehen. Entwickler:innen von Unternehmenssoftware müssen einfach wissen, wie sie effektiv mit Arrays, verketteten Listen, Mengen und Bäumen arbeiten können. Dieses Buch ist für pragmatische Fachleute des maschinellen Lernens geschrieben.

Codebeispiele

Wir stellen Code für maschinelles Lernen (manchmal in Keras/TensorFlow, manchmal in scikit-learn oder BigQuery ML) und Datenverarbeitung (in SQL) zur Verfügung, um zu zeigen, wie die im Buch beschriebenen Techniken in der Praxis umgesetzt werden. Der gesamte Code, auf den im Buch verwiesen wird, ist Teil unseres

GitHub-Repositorys (*https://github.com/GoogleCloudPlatform/ml-design-patterns*), in dem Sie voll funktionsfähige ML-Modelle finden. Diese Codebeispiele sollten Sie unbedingt ausprobieren.

Gegenüber den behandelten Konzepten und Techniken spielt der Code nur eine untergeordnete Rolle. Unser Ziel ist es gewesen, dass Thema und Prinzipien unabhängig von Änderungen an TensorFlow oder Keras relevant bleiben. Wir können uns durchaus vorstellen, das GitHub-Repository zum Beispiel mit anderen ML-Frameworks zu ergänzen, während der Buchtext unverändert bleibt. Daher sollte das Buch für Sie genauso informativ sein, wenn Ihr primäres ML-Framework PyTorch oder sogar ein Nicht-Python-Framework wie H20.ai oder R ist. Darüber hinaus begrüßen wir es, wenn Sie die Implementierung eines Musters (es dürfen auch mehrere sein) in Ihrem bevorzugten ML-Framework als Beitrag für das GitHub-Repository bereitstellen.

Dieses Buch soll Ihnen bei Ihrer täglichen Arbeit helfen. Falls Beispielcode zum Buch angeboten wird, dürfen Sie ihn im Allgemeinen in Ihren Programmen und für Dokumentationen verwenden. Sie müssen uns nicht um Erlaubnis bitten, es sei denn, Sie kopieren einen erheblichen Teil des Codes. Wenn Sie zum Beispiel ein Programm schreiben, das einige Codeblöcke aus diesem Buch verwendet, benötigen Sie keine Erlaubnis. Sollten Sie aber Beispiele aus O'Reilly-Büchern verkaufen oder verbreiten, ist eine Erlaubnis erforderlich. Wenn Sie eine Frage beantworten und dabei dieses Buch oder Beispielcode aus diesem Buch zitieren, brauchen Sie wiederum keine Erlaubnis. Aber wenn Sie große Teile des Beispielcodes aus diesem Buch in die Dokumentation Ihres Produkts einfließen lassen, ist eine Erlaubnis einzuholen.

Wir schätzen eine Quellenangabe, verlangen sie aber nicht. Eine Quellenangabe umfasst in der Regel Titel, Autor, Verlag und ISBN. Zum Beispiel: »*Design Patterns für Machine Learning* von Valliappa Lakshmanan, Sara Robinson und Michael Munn (O'Reilly). Copyright 2022 dpunkt.verlag, ISBN 978-3-96009-164-6.« Wenn Sie der Meinung sind, dass Sie die Codebeispiele in einer Weise verwenden, die über die oben erteilte Erlaubnis hinausgeht, kontaktieren Sie uns bitte unter *kommentar@oreilly.de*.

Typografischen Konventionen

In diesem Buch folgen wir diesen typografischen Konventionen:

Kursiv
: Kennzeichnet neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.

`Schreibmaschinenschrift`
: Wird in Programmlistings verwendet und im Fließtext für Programmelemente wie zum Beispiel Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

`Schreibmaschinenschrift fett`
: Kennzeichnet Befehle oder andere Texte, die vom Benutzer buchstäblich eingegeben werden sollen.

`Schreibmaschinenschrift kursiv`
: Zeigt Text, der ersetzt werden soll, durch Werte, die der Benutzer bereitstellt, oder Werte, die sich aus dem Kontext ergeben.

Dieses Element kennzeichnet einen Tipp oder Vorschlag.

Dieses Element kennzeichnet einen allgemeinen Hinweis.

Dieses Element kennzeichnet eine Warnung oder einen Achtungshinweis.

Danksagungen

Ein Buch wie dieses wäre nicht möglich ohne die Großzügigkeit zahlreicher Googler, insbesondere unserer Kolleg:innen aus den Teams Cloud AI, Solution Engineering, Professional Services und Developer Relations. Wir sind ihnen dankbar, dass wir ihre Lösungen für die herausfordernden Probleme beim Training, bei der Verbesserung und der Operationalisierung von ML-Modellen beobachten, analysieren und hinterfragen durften. Wir danken unseren Managern Karl Weinmeister, Steve Cellini, Hamidou Dia, Abdul Razack, Chris Hallenbeck, Patrick Cole, Louise Byrne und Rochana Golani dafür, dass sie den Geist der Offenheit bei Google fördern und uns die Freiheit geben, diese Muster zu katalogisieren und dieses Buch zu veröffentlichen.

Salem Haykal, Benoit Dherin und Khalid Salama haben jedes Muster und jedes Kapitel durchgesehen. Sal hat uns auf Feinheiten hingewiesen, die wir übersehen hatten, Benoit hat unsere Behauptungen eingegrenzt, und Khalid hat uns auf relevante Forschungsarbeiten aufmerksam gemacht. Ohne ihre Beiträge wäre dieses Buch bei Weitem nicht so gut geworden. Vielen Dank dafür! Amy Unruh, Rajesh Thallam, Robbie Haertel, Zhitao Li, Anusha Ramesh, Ming Fang, Parker Barnes, Andrew Zaldivar, James Wexler, Andrew Sellergren und David Kanter haben die Teile dieses Buchs, die in ihre Fachgebiete fallen, überprüft und zahlreiche Vorschläge dazu unterbreitet, wie die kurzfristige Roadmap unsere Empfehlungen be-

einflussen würde. Nitin Aggarwal und Matthew Yeager haben das Auge des Lesers in das Manuskript eingebracht und dessen Klarheit verbessert. Besonderer Dank gebührt Rajesh Thallam, der den Prototyp für das Design der allerletzten Abbildung in Kapitel 8 erstellt hat. Alle verbliebenen Fehler gehen natürlich auf unsere Kappe.

O'Reilly ist der Verlag der Wahl für technische Bücher, und die Professionalität unseres Teams zeigt, warum das so ist. Rebecca Novak hat uns an die Hand genommen, um eine überzeugende Gliederung zusammenzustellen, Kristen Brown hat die gesamte inhaltliche Entwicklung souverän gemeistert, Corbin Collins hat uns in jeder Phase hilfreiche Tipps gegeben, die Zusammenarbeit mit Elizabeth Kelly während der Produktion war eine Freude, und Charles Roumeliotis hat das Lektorat mit scharfem Blick begleitet. Danke an alle für diese Hilfe!

Michael: Ich danke meinen Eltern, die immer an mich geglaubt und meine Interessen gefördert haben, nicht nur meine akademischen. Sie werden das Cover, auf dem mein Name steht, genauso zu schätzen wissen wie ich. An Phil: Danke, dass du meinen kaum akzeptablen Zeitplan während der Arbeit an diesem Buch geduldig ertragen hast. Und jetzt werde ich schlafen gehen.

Sara: Jon – du bist ein wichtiger Grund dafür, dass es dieses Buch gibt. Danke, dass du mich ermutigt hast, es zu schreiben, dass du immer gewusst hast, wie du mich zum Lachen bringst, dass du meine Verrücktheit zu schätzen weißt und dass du an mich geglaubt hast, vor allem als ich es nicht tat. Meinen Eltern danke ich, dass sie vom ersten Tag, seit ich mich erinnern kann, meine größten Fans waren und meine Liebe zur Technik und zum Schreiben gefördert haben. An Ally, Katie, Randi und Sophie – danke, dass ihr in diesen unsicheren Zeiten eine ständige Quelle des Lichts und des Lachens seid.

Lak: Ich habe dieses Buch in Angriff genommen in der Annahme, ich könnte daran arbeiten, um Wartezeiten auf Flughäfen sinnvoll zu nutzen. COVID-19 hat es »möglich gemacht«, dass ich einen Großteil der Arbeit zu Hause erledigen konnte. Danke Abirami, Sidharth und Sarada für all eure Nachsicht, als ich mich wieder einmal zum Schreiben hingehockt habe. Mehr Wanderungen an den Wochenenden jetzt!

Wir drei spenden 100 % der Tantiemen aus diesem Buch an »Girls Who Code« (*https://girlswhocode.com/*), eine Organisation, deren Mission es ist, die weltweit größte Pipeline an zukünftigen Ingenieurinnen aufzubauen. Vielfalt, Gleichberechtigung und Inklusion sind beim maschinellen Lernen besonders wichtig, um sicherzustellen, dass KI-Modelle bestehende Vorurteile in der menschlichen Gesellschaft nicht noch untermauern.

KAPITEL 1

Der Bedarf an Entwurfsmustern für maschinelles Lernen

In technischen Disziplinen erfassen Entwurfsmuster Best Practices und Lösungen für häufig auftretende Problemstellungen. Sie kodifizieren das Wissen und die Erfahrung von Expertinnen und Experten in Ratschlägen, die alle Praktiker befolgen können. Dieses Buch ist ein Katalog von Entwurfsmustern, auch Design Patterns genannt, die wir im Laufe unserer Arbeit mit Hunderten von Teams für Machine Learning beobachtet haben.

Was sind Entwurfsmuster?

Christopher Alexander und fünf Mitautoren haben die Idee der Muster (engl. *Patterns*) und einen Katalog bewährter Muster im Bereich der Architektur in einem weltweit anerkannten Buch mit dem Titel *A Pattern Language* (Oxford University Press, 1977) eingeführt. In ihrem Buch stellen sie 253 Muster folgendermaßen vor:

> Jedes Muster beschreibt zum einen ein Problem, das in unserer Umgebung immer wieder auftritt, und zum anderen das Prinzip der Problemlösung. Das geschieht so, dass man diese Lösung unzählige Male nutzen kann, ohne sie jemals auf exakt dieselbe Weise anzuwenden.
>
> ...
>
> Jede Lösung gibt das wesentliche Beziehungsfeld an, das für die Lösung des Problems erforderlich ist, aber in einer sehr allgemeinen und abstrakten Form. Damit können Sie das Problem in eigener Regie und auf Ihre eigene Art und Weise angehen, indem Sie die Lösung an Ihre Vorstellungen und die vor Ort herrschenden Bedingungen anpassen.

Beispielsweise lauten verschiedene Muster, die persönliche Besonderheiten beim Bau von Wohnungen einbeziehen, *Tageslicht auf zwei Seiten jedes Raums* und *Zwei-Meter-Loggia*. Denken Sie an Ihr Lieblingszimmer in Ihrem Haus und an den Raum, den Sie am wenigsten mögen. Hat Ihr Lieblingsraum zwei Fenster in zwei Wänden? Wie steht es mit Ihrem unbeliebtesten Raum? Nach Alexander:

> In Räumen, in denen von zwei Seiten natürliches Licht einfallen kann, ist die Blendwirkung um Personen und Objekte herum geringer, und vor allem können wir die Mimik in den Gesichtern der Menschen in allen Einzelheiten erkennen ...

Ein Name für dieses Muster erspart einem Architekten, dieses Prinzip ständig neu entdecken zu müssen. Doch wo und wie man zwei Lichtquellen in einer bestimmten örtlichen Situation herbekommt, bleibt dem Geschick des Architekten überlassen. Ähnlich verhält es sich beim Entwurf einer Loggia: Wie groß sollte sie sein? Alexander empfiehlt eine Größe von 2 × 2 Metern als ausreichend für zwei (nicht unbedingt zusammenpassende) Stühle und einen Beistelltisch, während die Loggia 4 × 4 Meter groß sein sollte, wenn Sie sowohl einen überdeckten Sitzplatz haben als auch in der Sonne sitzen möchten.

Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides übertrugen die Idee auf Software, indem sie 1994 im Buch *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1995) 23 objektorientierte Entwurfsmuster katalogisierten. Die in ihrem Katalog enthaltenen Muster wie Proxy, Singleton und Decorator haben das Gebiet der objektorientierten Programmierung nachhaltig beeinflusst. Im Jahr 2005 verlieh die *Association of Computing Machinery* (ACM) ihren jährlichen Programming Languages Achievement Award an die Autoren und würdigte damit den Einfluss ihrer Arbeit »auf die Programmierpraxis und das Design von Programmiersprachen.«

Modelle für maschinelles Lernen in der Produktion zu erstellen, wird zunehmend zu einer Engineering-Disziplin. Man greift dabei auf bewährte ML-Methoden aus Forschungsumgebungen zurück und wendet sie auf Geschäftsprobleme an. Da maschinelles Lernen immer mehr zum Mainstream wird, sollten Praktiker unbedingt die Vorteile bewährter Methoden nutzen, um damit wiederkehrende Probleme zu lösen.

Unsere Arbeit im kundenorientierten Teil von Google Cloud hat den Vorteil, dass wir mit unterschiedlichsten Teams für maschinelles Lernen und Data Science sowie einzelnen Entwickler:innen aus der ganzen Welt in Kontakt kommen. Gleichzeitig arbeitet jeder von uns eng mit internen Google-Teams zusammen, die hochmoderne Probleme des maschinellen Lernens lösen. Schließlich sind wir in der glücklichen Lage, mit den Teams von TensorFlow, Keras, BigQuery ML, TPU und Cloud AI Platform zusammenzuarbeiten, die die Demokratisierung der Forschung und Infrastruktur für maschinelles Lernen vorantreiben. Dies alles gibt uns eine ziemlich einzigartige Perspektive, von der aus wir die Best Practices katalogisieren können, die wir bei diesen Teams beobachtet haben.

Dieses Buch ist ein Katalog von Entwurfsmustern oder wiederholbaren Lösungen für häufig auftretende Probleme im ML-Engineering. Zum Beispiel erzwingt das Muster *Transformation* (Kapitel 6) die Trennung von Eingaben, Features und Transformationen. Außerdem macht es die Transformationen persistent, um die Überführung eines ML-Modells in die Produktion zu vereinfachen. In ähnlicher Weise ist *Keyed Predictions* in Kapitel 5 ein Muster, das die Verteilung von Batch-Vorhersagen im großen Maßstab ermöglicht, wie zum Beispiel für Empfehlungsmodelle.

Für jedes Muster beschreiben wir das häufig auftretende Problem, das angesprochen wird, gehen dann verschiedenartige mögliche Lösungen für das Problem durch, erläutern Kompromisse dieser Lösungen und geben Empfehlungen für die Auswahl zwischen diesen Lösungen. Der Implementierungscode für diese Lösungen ist angegeben in SQL (was sinnvoll ist, wenn Sie Vorverarbeitungen und andere ETL[1]-Operationen in Spark SQL, BigQuery usw. ausführen), scikit-learn und/oder Keras mit einem TensorFlow-Backend.

Wie Sie dieses Buch verwenden

Vor Ihnen liegt ein Katalog von Entwurfsmustern, die wir in der Praxis beobachtet haben, und zwar bei mehreren Teams. In einigen Fällen sind die zugrunde liegenden Konzepte schon seit vielen Jahren bekannt. Wir erheben nicht den Anspruch, diese Muster erfunden oder entdeckt zu haben. Vielmehr hoffen wir, einen gemeinsamen Bezugsrahmen und einen Satz von Werkzeugen für ML-Praktiker:innen bereitzustellen. Das ist uns dann gelungen, wenn dieses Buch Ihnen und Ihrem Team ein Vokabular an die Hand gibt, um über Konzepte zu sprechen, die Sie in Ihren ML-Projekten bereits intuitiv umgesetzt haben.

Wir gehen nicht davon aus, dass Sie dieses Buch der Reihe nach durchlesen (obwohl nichts dagegenspricht!). Stattdessen nehmen wir an, dass Sie das Buch überfliegen, einige Abschnitte eingehender als andere lesen, die Ideen in Gesprächen mit Kolleginnen und Kollegen erwähnen und auf das Buch zurückgreifen, wenn Sie mit Problemen konfrontiert werden, von denen Sie hier bereits gelesen haben. Falls Sie so vorgehen möchten, empfehlen wir, mit Kapitel 1 und Kapitel 8 zu beginnen, bevor Sie sich einzelnen Mustern zuwenden.

Zu jedem Muster gehört eine kurze Problemaussage, eine kanonische Lösung und eine Erklärung dazu, warum die Lösung funktioniert, sowie eine mehrteilige Diskussion über Kompromisse und Alternativen. Wir empfehlen, den Diskussionsabschnitt zu lesen und dabei die kanonische Lösung fest im Hinterkopf zu behalten, um zu vergleichen und gegenüberzustellen. Die Musterbeschreibung enthält Codefragmente aus der Implementierung der kanonischen Lösung. Den vollständigen Code finden Sie in unserem GitHub-Repository (*https://github.com/GoogleCloudPlatform/ml-design-patterns*). Es empfiehlt sich, den Code durchzugehen, während Sie die Musterbeschreibung lesen.

Terminologie für maschinelles Lernen

Da Praktikerinnen und Praktiker im Bereich des maschinellen Lernens heutzutage aus den unterschiedlichsten Fachgebieten – Softwaretechnik, Datenanalyse, DevOps oder Statistik – stammen können, gibt es subtile Unterschiede in der Verwen-

1 ETL – Extraktion, Transformation, Laden; ein Prozess, um Daten, die aus mehreren Quellen stammen können, in einer Zieldatenbank zusammenzufassen (Anm. d. Übers.).

dung bestimmter Begriffe. In diesem Abschnitt definieren wir die Terminologie, die wir im gesamten Buch verwenden.

Modelle und Frameworks

In seinem Kern ist *maschinelles Lernen* ein Prozess, der Modelle erstellt, die aus Daten lernen. Dies steht im Gegensatz zur herkömmlichen Programmierung, bei der wir explizite Regeln schreiben, die den Programmen sagen, wie sie sich verhalten sollen. *Modelle für maschinelles Lernen* sind Algorithmen, die Muster aus Daten lernen. Diesen Punkt wollen wir anhand einer Firma veranschaulichen, die Umzugskosten für potenzielle Kunden abschätzen muss. In der herkömmlichen Programmierung könnten wir dies mit einer `if`-Anweisung lösen:

```
if num_bedrooms == 2 and num_bathrooms == 2:
estimate = 1500
elif num_bedrooms == 3 and sq_ft > 2000:
estimate = 2500
```

Man kann sich vorstellen, wie schnell dies kompliziert wird, wenn wir weitere Variablen (Anzahl großer Möbelstücke, Umfang der Kleidung, zerbrechliche Gegenstände usw.) hinzufügen und versuchen, Sonderfälle zu behandeln. Und wenn man all diese Informationen im Voraus von den Kunden abfragt, kann das vor allem dazu führen, dass die Firma den Schätzprozess aufgibt. Stattdessen können wir ein maschinelles Lernmodell trainieren, um die Umzugskosten basierend auf den Daten früherer Umzüge unseres Unternehmens zu schätzen.

In den Beispielen, die das Buch vorstellt, verwenden wir hauptsächlich neuronale Feedforward-Netze, doch ziehen wir auch Modelle der linearen Regression, Entscheidungsbäume, Clustering-Modelle und andere heran. *Neuronale Feedforward-Netze*, die wir üblicherweise kurz als *neuronale Netze* bezeichnen, stellen einen Algorithmentyp für maschinelles Lernen dar, bei dem mehrere Schichten (engl. *Layers*) mit jeweils vielen Neuronen Informationen analysieren und verarbeiten und dann diese Informationen an die nächste Schicht senden, wobei schließlich die letzte Schicht eine Vorhersage als Ausgabe produziert. Obwohl sie keineswegs identisch sind, werden neuronale Netze oft mit den Neuronen in unserem Gehirn verglichen, und zwar aufgrund der Konnektivität zwischen den Knoten und der Art und Weise, wie sie verallgemeinern und neue Vorhersagen aus den verarbeiteten Daten bilden können. Neuronale Netze mit mehr als einem *Hidden Layer* (einer versteckten Schicht, d.h. einer Schicht, die weder Eingabe- noch Ausgabeschicht ist) werden als *Deep Learning* klassifiziert (siehe Abbildung 1-1).

Modelle für maschinelles Lernen sind – unabhängig davon, wie man sie visuell darstellt – mathematische Funktionen und lassen sich demzufolge mit einem numerischen Softwarepaket von Grund auf neu erstellen. Allerdings greifen ML Engineers in der Industrie gern zu einem von mehreren Open-Source-Frameworks, die konzeptionell intuitive APIs für das Erstellen von Modellen anbieten. Die Mehrheit unserer Beispiele verwendet *TensorFlow*, ein Open-Source-Framework für maschi-

nelles Lernen, das von Google mit Schwerpunkt auf Deep-Learning-Modelle geschaffen wurde. Innerhalb der TensorFlow-Bibliothek verwenden wir für unsere Beispiele die *Keras*-API, die sich über `tensorflow.keras` importieren lässt. Bei Keras handelt es sich um eine Higher-Level-API zum Erstellen von neuronalen Netzen. Von den verschiedenen Backends, die Keras unterstützt, haben wir uns für TensorFlow entschieden. Andere Beispiele arbeiten mit den ebenfalls beliebten Open-Source-Frameworks *scikit-learn*, *XGBoost* und *PyTorch*, die neben APIs für das Erstellen von linearen und tiefen Modellen auch Hilfsprogramme enthalten, mit denen Sie Ihre Daten vorbereiten können. Maschinelles Lernen wird immer zugänglicher, und eine spannende Entwicklung ist die Verfügbarkeit von Modellen für maschinelles Lernen, die sich in SQL ausdrücken lassen. Als Beispiel hierfür setzen wir *BigQuery ML* ein, insbesondere in Situationen, in denen wir Datenvorverarbeitung und Modellerstellung kombinieren möchten.

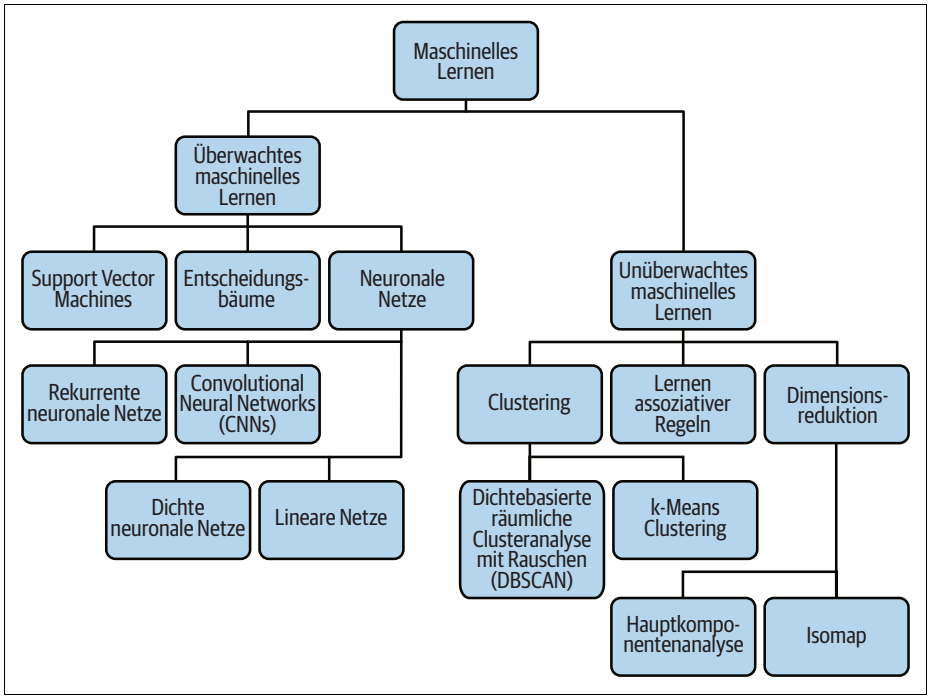


Abbildung 1-1: Eine Aufschlüsselung der verschiedenen Arten von maschinellem Lernen mit jeweils einigen Beispielen. Obwohl sie nicht in der Darstellung enthalten sind, können auch neuronale Netze wie Autoencoder für unüberwachtes Lernen eingesetzt werden.

Umgekehrt bilden neuronale Netze mit nur einer Eingabe- und einer Ausgabeschicht eine andere Teilmenge des maschinellen Lernens, die sogenannten *linearen Modelle*. Diese stellen die aus den Daten gelernten Muster mithilfe einer linearen Funktion dar. *Entscheidungsbäume* sind Modelle des maschinellen Lernens, die Ihre Daten verwenden, um eine Teilmenge von Pfaden mit verschiedenen Verzwei-

gungen zu erzeugen. Diese Verzweigungen stellen eine Annäherung an die Ergebnisse verschiedener Ausgaben aus Ihren Daten dar. Schließlich suchen *Clustering*-Modelle nach Ähnlichkeiten zwischen verschiedenen Teilmengen Ihrer Daten und gruppieren die Daten anhand der identifizierten Muster in Clustern.

Die Probleme des maschinellen Lernens (siehe Abbildung 1-1) lassen sich in zwei Typen unterteilen: überwachtes und unüberwachtes Lernen. *Überwachtes Lernen* definiert Probleme, bei denen Sie die Ground-Truth-Labels (auch *Label der Grundwahrheit* genannt) für Ihre Daten im Voraus kennen. Zum Beispiel könnte dies die Benennung eines Bilds als »Katze« oder die Benennung eines Babys als »2300 Gramm bei Geburt« sein. Diese benannten Daten speisen Sie in Ihr Modell ein in der Hoffnung, dass es genügend lernen kann, um neue Beispiele zu benennen. Beim unüberwachten Lernen kennen Sie die Bezeichnungen für Ihre Daten nicht im Voraus, und das Ziel besteht darin, ein Modell zu erstellen, das natürliche Gruppierungen der Daten finden (*Clustering* genannt), den Informationsgehalt komprimieren (*Dimensionsreduzierung*) oder Assoziationsregeln ableiten kann. Der größte Teil dieses Buchs konzentriert sich auf überwachtes Lernen, da die in der Produktion verwendeten Modelle für maschinelles Lernen vorwiegend überwacht arbeiten.

Beim überwachten Lernen können Probleme typischerweise entweder als Klassifizierung oder als Regression definiert werden. *Klassifizierungsmodelle* ordnen Ihren Eingabedaten aus einer diskreten, vordefinierten Menge von Kategorien ein oder mehrere Labels zu. Zu den Klassifizierungsproblemen gehört es zum Beispiel, eine Tierrasse auf einem Bild zu bestimmen, ein Dokument zu kennzeichnen oder vorherzusagen, ob eine Transaktion betrügerisch ist oder nicht. *Regressionsmodelle* ordnen Ihren Eingaben kontinuierliche Zahlenwerte zu. Regressionsmodelle verwendet man zum Beispiel, um die Dauer einer Fahrradtour, den zukünftigen Umsatz eines Unternehmens oder den Preis eines Produkts vorherzusagen.

Daten und Feature Engineering

Daten stehen bei jedem Problem des maschinellen Lernens im Mittelpunkt. Wenn wir von *Datasets* oder *Datensätzen* sprechen, meinen wir Daten, die zum Trainieren, Validieren und Testen eines ML-Modells verwendet werden. Den Hauptteil Ihrer Daten werden die *Trainingsdaten* ausmachen: die Daten, die Sie Ihrem Modell beim Training zuführen. Die *Validierungsdaten* sind separate Daten, die nicht zu den Trainingsdaten gehören und die Sie heranziehen, um die Performance des Modells nach jeder *Trainingsepoche* (oder jedem Durchlauf durch die Trainingsdaten) zu bewerten. Anhand der Performance des Modells auf den Validierungsdaten entscheiden Sie, wann der Trainingslauf zu beenden ist, und wählen *Hyperparameter* aus wie zum Beispiel die Anzahl der Bäume in einem Random-Forest-Modell. *Testdaten* sind Daten, die im Trainingsprozess überhaupt nicht verwendet werden und die dazu dienen, die Performance des trainierten Modells zu bewerten. Berichte zur Performance des ML-Modells müssen auf den unabhängigen Testdaten

beruhen, aber weder auf den Trainings- noch auf den Validierungstests. Wichtig ist auch, die Daten so aufzuteilen, dass die statistischen Eigenschaften aller drei Datensätze (Training, Test, Validierung) ähnlich sind.

Die Daten, mit denen Sie Ihr Modell trainieren, können je nach Modelltyp viele Formen annehmen. Wir definieren *strukturierte Daten* als numerische und kategoriale Daten. Die numerischen Daten umfassen Ganzzahl- und Gleitkommawerte, während sich kategoriale Daten in eine endliche Menge von Gruppen wie zum Beispiel Autotyp oder Bildungsniveau unterteilen lassen. Strukturierte Daten können Sie sich auch als solche Daten vorstellen, wie Sie sie häufig in einer Tabellenkalkulation finden. Im Buch verwenden wir den Begriff *tabellarische Daten* synonym zu strukturierten Daten. Hingegen umfassen *unstrukturierte Daten* solche Daten, die sich nicht so übersichtlich darstellen lassen. Dazu zählen typischerweise formatfreier Text, Bilder, Videos und Audiodaten.

Numerische Daten können oft direkt in ein ML-Modell eingespeist werden, während andere Daten verschiedene *Vorverarbeitungsschritte* benötigen, bevor sie an ein Modell gesendet werden können. Typischerweise werden dabei die numerischen Werte skaliert oder nicht numerische Daten in ein numerisches Format konvertiert, das Ihr Modell dann auch verstehen kann. Ein anderer Begriff für Vorverarbeitung ist *Feature Engineering*. Diese beiden Begriffe verwenden wir im Buch gleichberechtigt nebeneinander.

Es gibt verschiedene Begriffe, um Daten zu beschreiben, die den Feature-Engineering-Prozess durchlaufen. So steht *Eingabe* für eine einzelne Spalte in Ihrem Datensatz, *bevor* sie verarbeitet wurde, und *Feature* beschreibt eine einzelne Spalte, *nachdem* sie die Aufbereitung passiert hat. Wenn etwa ein Zeitstempel Ihre Eingabe ist, könnte der Wochentag das Feature sein. Um die Daten von einem Zeitstempel in einen Wochentag umzuwandeln, müssen Sie die Daten aufbereiten. Dieser Aufbereitungsschritt kann auch als *Datentransformation* bezeichnet werden.

Eine *Instanz* ist ein Element, das Sie an Ihr Modell zur Vorhersage senden möchten. Dabei könnte es sich um eine Zeile in Ihrem Testdatensatz (ohne die Label-Spalte) handeln, ein Bild, das Sie klassifizieren möchten, oder ein Textdokument, das an ein Sentimentanalysemodell gesendet werden soll. Mit einem Satz von Features über die Instanz berechnet das Modell einen vorhergesagten Wert. Hierfür wird das Modell auf *Trainingsbeispielen* trainiert, die einer Instanz ein *Label* zuordnen. Ein *Trainingsbeispiel* bezieht sich auf eine einzelne Instanz (Zeile) von Daten aus Ihrem Datensatz, die Ihrem Modell zugeführt werden. Basierend auf dem Use Case »Zeitstempel«, könnte ein vollständiges Trainingsbeispiel »Wochentag«, »Stadt« und »Autotyp« enthalten. Ein *Label* ist die Ausgabespalte in Ihrem Datensatz – das Element, das Ihr Modell vorhersagt. *Label* kann sich sowohl auf die Zielspalte in Ihrem Datensatz (*Label der Grundwahrheit* bzw. engl. *Ground-Truth-Label*) als auch auf die von Ihrem Modell gelieferte Ausgabe (auch als *Vorhersage* bezeichnet) beziehen. Ein Beispiellabel für das oben umrissene Trainingsbeispiel könnte »Fahrtdauer« lauten – in diesem Fall ein Gleitkommawert, der Minuten angibt.

Nachdem Sie Ihr Dataset zusammengestellt und die Features für Ihr Modell bestimmt haben, ist die *Datenvalidierung* der Prozess, Statistiken für Ihre Daten zu berechnen, Ihr Schema zu verstehen und den Datensatz zu bewerten, um Probleme wie Drift und Training-Serving-Verzerrung zu identifizieren. Die Auswertung verschiedener Statistiken für Ihre Daten kann Ihnen helfen, sicherzustellen, dass der Datensatz eine ausgewogene Darstellung jedes Features enthält. Dort, wo es nicht möglich ist, weitere Daten zu sammeln, hilft Ihnen das Verständnis der Datenausgewogenheit, Ihr Modell dahin gehend zu entwerfen. Zum Verständnis Ihres Schemas gehört es auch, den Datentyp für jedes Feature zu definieren und die Trainingsbeispiele zu identifizieren, bei denen bestimmte Werte falsch sind oder fehlen. Schließlich kann die Datenvalidierung Inkonsistenzen identifizieren, die die Qualität Ihrer Trainings- und Testsets beeinträchtigen können. Vielleicht enthält zum Beispiel der Großteil Ihres Datensatzes für das Training *Wochentag*-Beispiele, während Ihr Testdatensatz hauptsächlich aus *Wochenende*-Beispielen besteht.

Der Prozess des maschinellen Lernens

Der erste Schritt in einem typischen Workflow des maschinellen Lernens ist das *Training* – die Übergabe der Trainingsdaten an ein Modell, sodass es lernen kann, Muster zu identifizieren. Nach dem Training wird im nächsten Prozessschritt getestet, was das Modell auf Daten außerhalb des Trainingssets leistet. Dies ist die *Bewertung* des Modells. Training und Bewertung können Sie mehrmals ausführen, dabei zusätzliches Feature Engineering realisieren und Ihre Modellarchitektur optimieren. Wenn Sie mit der Performance Ihres Modells während der Bewertung zufrieden sind, werden Sie Ihr Modell wahrscheinlich bereitstellen wollen, damit andere darauf zugreifen und Vorhersagen treffen können. Mit *Serving* (Bereitstellen) meinen wir, dass eingehende Anforderungen akzeptiert und Vorhersagen zurückgesendet werden, indem das Modell als Microservice bereitgestellt wird. Die Infrastruktur hierfür könnte in der Cloud, lokal oder auf einem mobilen Gerät untergebracht sein.

Den Prozess, der neue Daten an Ihr Modell sendet und dessen Ausgabe verwendet, nennen wir *Vorhersage*. Dabei kann es sich sowohl um das Generieren von Vorhersagen aus lokalen Modellen, die noch nicht bereitgestellt wurden, als auch um das Abrufen von Vorhersagen aus bereitgestellten Modellen handeln. Bei bereitgestellten Modellen beziehen wir uns sowohl auf Online- als auch auf Batch-Vorhersagen. Die *Online-Vorhersage* wird verwendet, wenn Vorhersagen bei nur wenigen Beispielen in nahezu Echtzeit gefragt sind. Dabei liegt die Betonung auf geringer Latenz. Dagegen generiert eine *Batch-Vorhersage* die Vorhersagen offline für eine große Datenmenge. Zwar dauern die Jobs der Batch-Vorhersage länger als die der Online-Vorhersage, doch eignen sie sich insbesondere, um Vorhersagen im Voraus zu berechnen (wie in Empfehlungssystemen) und die Vorhersagen eines Modells über eine große Stichprobe neuer Daten zu analysieren.

Der Begriff *Vorhersage* ist treffend, wenn es um die Prognose zukünftiger Werte geht, etwa um vorherzusagen, wie lange eine Fahrradtour dauern wird oder ob ein Kunde den Inhalt seines Einkaufswagens wieder verwirft. Weniger intuitiv ist es bei Modellen für die Bild- und Textklassifizierung. Wenn ein ML-Modell eine Textrezension verarbeitet und ausgibt, dass die Stimmung positiv ist, stellt das nicht wirklich eine »Vorhersage« dar (es gibt kein zukünftiges Ergebnis). Daher werden Sie auch auf die Begriffe *Schlussfolgerung* oder *Inferenz* treffen, wenn es um derartige Vorhersagen geht. Der Begriff Inferenz ist aus der Statistik entlehnt, wobei es hier aber nicht wirklich um Schlussfolgerungen geht.

Oftmals wird das Sammeln von Trainingsdaten, das Feature Engineering, das Training und die Bewertung des Modells getrennt von der Produktionspipeline behandelt. In derartigen Fällen werden Sie Ihre Lösung neu bewerten, sobald Sie über genügend zusätzliche Daten verfügen, um eine neue Version Ihres Modells zu trainieren. In anderen Situationen kann es sein, dass neue Daten kontinuierlich eingelesen und sofort verarbeitet werden müssen, bevor sie zum Training oder zur Vorhersage an das Modell gehen. Dies wird als *Streaming* bezeichnet. Um Streaming-Daten zu verarbeiten, brauchen Sie eine mehrstufige Lösung, die Feature Engineering, Training, Bewertung und Vorhersagen umfasst. Derartige mehrstufige Lösungen nennt man *ML-Pipelines*.

Tools für Daten und Modelle

Es gibt verschiedene Google-Cloud-Produkte, auf die wir uns beziehen und die Tools bereitstellen, um Probleme mit Daten und maschinellem Lernen zu lösen. Die angegebenen Produkte sind lediglich Vorschläge, um die in diesem Buch vorgestellten Entwurfsmuster zu implementieren, und keine vollständige Liste. Alle hier aufgeführten Produkte sind serverlos, sodass wir uns mehr auf die Implementierung von Mustern für maschinelles Lernen konzentrieren können und weniger auf die dahinterstehende Infrastruktur.

BigQuery (*https://oreil.ly/7PnVj*) ist ein Data Warehouse für Unternehmen, das dafür konzipiert ist, große Datensätze mit SQL schnell zu analysieren. In unseren Beispielen verwenden wir BigQuery für das Sammeln von Daten und das Feature Engineering. Die Daten in BigQuery sind in Datensätzen organisiert, und ein Datensatz kann mehrere Tabellen enthalten. Viele unserer Beispiele nutzen Daten von *Google Cloud Public Datasets* (*https://oreil.ly/AbTaJ*), einem Satz kostenloser, öffentlich verfügbarer Daten, die in BigQuery gehostet werden. Google Cloud Public Datasets besteht aus Hunderten verschiedener Datensätze, unter anderem den NOAA-Wetterdaten seit 1929, den Fragen und Antworten von Stack Overflow, Open-Source-Code von GitHub, Geburtendaten und mehr. Um einige der Modelle in unseren Beispielen zu erstellen, stützen wir uns auf *BigQuery Machine Learning* (oder *BigQuery ML*, *https://oreil.ly/_VjVz*). BigQuery ML ist ein Tool, um Modelle aus Daten zu erstellen, die in BigQuery gespeichert sind. Mit BigQuery können wir unsere Modelle mittels SQL trainieren, bewerten und für Vorhersagen nutzen. Es

unterstützt Klassifizierungs- und Regressionsmodelle neben Modellen für unüberwachtes Clustering. Zudem ist es möglich, zuvor trainierte TensorFlow-Modelle für Vorhersagen in BigQuery ML zu importieren.

Zu *Cloud AI Platform* (*https://oreil.ly/90KLs*) gehört eine breite Palette von Produkten für das Training und das Bereitstellen von benutzerdefinierten Modellen des maschinellen Lernens auf Google Cloud. In unseren Beispielen verwenden wir *AI Platform Training* und *AI Platform Prediction*. AI Platform Training bietet eine Infrastruktur für das Training von Modellen des maschinellen Lernens auf Google Cloud. Mit AI Platform Prediction können Sie Ihre trainierten Modelle bereitstellen und mittels einer API Vorhersagen auf ihnen generieren. Beide Dienste unterstützen TensorFlow-, scikit-learn- und XGBoost-Modelle sowie benutzerdefinierte Container für Modelle, die mit anderen Frameworks erstellt wurden. Darüber hinaus verweisen wir auf das Tool *Explainable AI* (*https://oreil.ly/lDocn*), mit dem Sie die Ergebnisse aus den Vorhersagen Ihres Modells interpretieren können. Das Tool ist für Modelle verfügbar, die auf AI Platform bereitgestellt werden.

Rollen

Innerhalb einer Organisation gibt es viele verschiedene Jobrollen, die sich auf Daten und maschinelles Lernen beziehen. Nachfolgend definieren wir einige gängige Rollen, auf die im Buch häufig verwiesen wird. Da sich dieses Buch in erster Linie an Data Scientists, Data Engineers und ML-Engineers richtet, soll es mit diesen losgehen.

Data Scientists sind Personen, die sich auf das Erfassen, Interpretieren und Verarbeiten von Datensätzen konzentrieren. Zu ihren Aufgaben gehört es, statistische und explorative Analysen von Daten durchzuführen. In Bezug auf maschinelles Lernen kann ein Data Scientist unter anderem an der Datenerfassung, dem Feature Engineering und der Modellerstellung arbeiten. Oftmals arbeiten Data Scientists in Python oder R in einer Notebook-Umgebung, und sie sind üblicherweise die ersten, die die ML-Modelle eines Unternehmens ausarbeiten.

Data Engineers befassen sich mit der Infrastruktur und den Workflows, die den Daten eines Unternehmens Kraft verleihen. Unter anderem haben sie Einfluss darauf, wie eine Firma Daten erfasst, Datenpipelines einrichtet und wie Daten gespeichert und übertragen werden. Data Engineers implementieren Infrastruktur und Pipelines rund um die Daten.

Machine Learning Engineers führen ähnliche Aufgaben wie Data Engineers aus, allerdings für ML-Modelle. Für die von den Data Scientists entwickelten Modelle richten sie die Infrastruktur und den Betrieb rund um Training und Bereitstellung (engl. *Deploying*) dieser Modelle ein. ML Engineers helfen beim Aufbau von Produktionssystemen, um das Aktualisieren der Modelle, die Versionierung von Modellen und das Bereitstellen von Vorhersagen für Endbenutzer handhabbar zu machen.

Je kleiner und agiler das Data-Science-Team in einem Unternehmen ist, desto wahrscheinlicher ist es, dass ein und dieselbe Person mehrere Rollen einnimmt. Wenn Sie sich in einer derartigen Situation befinden, werden Sie sich vermutlich in allen drei der oben beschriebenen Kategorien zumindest teilweise wiederfinden. Üblicherweise beginnen Sie ein ML-Projekt als Data Engineer und erstellen Datenpipelines, um die Eingabe der Daten zu operationalisieren. Dann wechseln Sie in die Rolle des Data Scientists und erstellen das/die ML-Modell(e). Schließlich setzen Sie sich den Hut des ML Engineers auf und überführen das Modell in die Produktion. In größeren Unternehmen können ML-Projekte die gleichen Phasen durchlaufen, wobei aber an jeder Phase verschiedene Teams beteiligt sind.

Wissenschaftler:innen, Datenanalyst:innen und Entwickler:innen können ebenfalls diese KI-Modelle erstellen und verwenden, doch aufgrund ihrer Arbeitsrollen gehören sie nicht zur Zielgruppe dieses Buchs.

Wissenschaftler:innen konzentrieren sich vor allem darauf, neue Algorithmen zu entwickeln, um die Disziplin ML voranzubringen. Das könnte eine breite Palette an Teilgebieten innerhalb des maschinellen Lernens umfassen, etwa Modellarchitekturen, Verarbeitung natürlicher Sprache, Computervision, Optimierung von Hyperparametern, Interpretierbarkeit von Modellen und mehr. Im Unterschied zu den anderen hier besprochenen Rollen verbringen Wissenschaftler:innen die meiste Zeit damit, Prototypen zu entwickeln und neue Konzepte für maschinelles Lernen zu bewerten, anstatt produktionsreife ML-Systeme zu erstellen.

Datenanalyst:innen bewerten und sammeln Erkenntnisse aus Daten und fassen dann diese Erkenntnisse für andere Teams innerhalb ihres Unternehmens zusammen. In der Regel arbeiten sie in SQL und Tabellenkalkulationen und erstellen mithilfe von Business-Intelligence-Tools Datenvisualisierungen, um ihre Resultate zu teilen. Datenanalyst:innen arbeiten eng mit Produktteams zusammen, um zu verstehen, wie ihre Erkenntnisse dabei helfen, geschäftliche Probleme anzugehen und Mehrwert zu schaffen. Während sich Datenanalyst:innen darauf konzentrieren, Trends in vorhandenen Daten zu identifizieren und Erkenntnisse daraus abzuleiten, geht es Data Scientists darum, anhand dieser Daten Prognosen zu erstellen und das Generieren von Erkenntnissen zu automatisieren oder zu skalieren. Mit der wachsenden Demokratisierung des maschinellen Lernens können sich Datenanalyst:innen selbst zu Data Scientists weiterbilden.

Entwickler:innen sind dafür zuständig, Produktionssysteme aufzubauen, die Endbenutzern den Zugriff auf ML-Modelle ermöglichen. Oft sind sie beteiligt am Entwurf der APIs, die in einem benutzerfreundlichen Format über eine Web- oder mobile Anwendung Modelle abfragen und Vorhersagen zurückgeben. Dabei kann es sich um Modelle handeln, die in der Cloud gehostet oder geräteseitig bereitgestellt werden. Auf der von den ML Engineers implementierten Infrastruktur erstellen Entwickler:innen Anwendungen und Benutzeroberflächen, um den Modellbenutzern die Vorhersagen zu präsentieren.

Abbildung 1-2 veranschaulicht, wie diese verschiedenen Rollen im Entwicklungsprozess für Modelle des maschinellen Lernens in einer Organisation zusammenarbeiten.

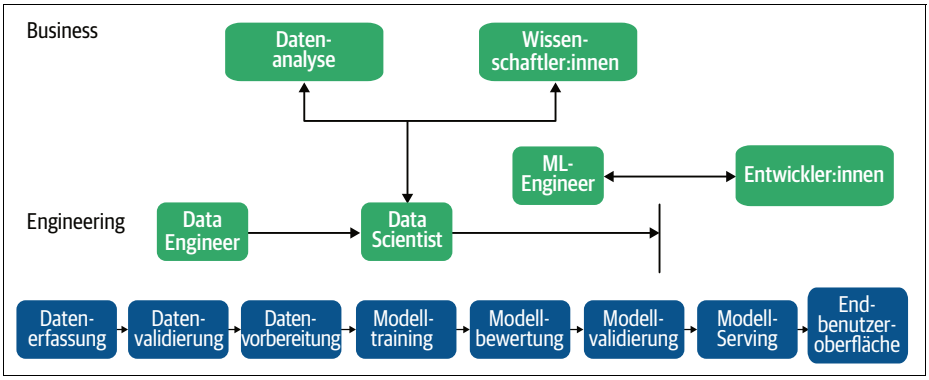


Abbildung 1-2: Viele verschiedene Jobrollen beziehen sich auf Daten und maschinelles Lernen. Und diese Rollen wirken gemeinsam im ML-Workflow von der Datenerfassung bis zur Modellbereitstellung und der Endbenutzeroberfläche. Zum Beispiel beschäftigt sich der Data Engineer mit der Datenerfassung und Datenvalidierung, wobei er eng mit Data Scientists zusammenarbeitet.

Allgemeine Herausforderungen beim maschinellen Lernen

Warum brauchen wir ein Buch über Entwurfsmuster für maschinelles Lernen? Will man ML-Systeme erstellen, bekommt man es mit einer Vielzahl einzigartiger Herausforderungen zu tun, die das ML-Design beeinflussen. Diese Herausforderungen zu verstehen, hilft Ihnen als ML-Praktiker:in, einen Bezugsrahmen für die im Buch vorgestellten Lösungen zu entwickeln.

Datenqualität

Modelle für maschinelles Lernen sind nur so zuverlässig wie die Daten, mit denen sie trainiert werden. Wenn Sie ein ML-Modell auf einem unvollständigen Datensatz trainieren, auf Daten mit schlecht ausgewählten Features oder auf Daten, die nicht genau die Population darstellen, die das Modell verwendet, werden die Vorhersagen Ihres Modells diese Daten direkt widerspiegeln. Daher tituliert man ML-Modelle oftmals als »Garbage In, Garbage Out«[2]. Die folgenden Abschnitte beleuchten vier wichtige Komponenten der Datenqualität: Genauigkeit, Vollständigkeit, Konsistenz und Aktualität.

2 Garbage In, Garbage Out: Ungültige Eingaben liefern ungültige Ausgaben (wörtlich: »Müll rein, Müll raus«).

Datengenauigkeit bezieht sich sowohl auf die Features Ihrer Trainingsdaten als auch auf die Labels der Grundwahrheit, die diesen Features entsprechen. Um die Feature-Genauigkeit sicherzustellen, ist es hilfreich, wenn Sie die Quelle Ihrer Daten und potenzielle Fehler bei der Datenerhebung kennen. Nachdem Sie Ihre Daten zusammengetragen haben, ist es wichtig, eine gründliche Analyse durchzuführen, um Tippfehler, doppelte Einträge, inkonsistente Messwerte in tabellarischen Daten, fehlende Features und andere Fehler zu erkennen, die die Datenqualität beeinträchtigen können. Zum Beispiel können doppelte Einträge in Ihrem Trainingsdatensatz dazu führen, dass Ihr Modell diesen Datenpunkten fälschlicherweise ein größeres Gewicht zuweist.

Genaue Datenlabels sind ebenso wichtig wie genaue Features. Das Modell stützt sich ausschließlich auf die Labels der Grundwahrheiten in Ihren Trainingsdaten, um seine Gewichte zu aktualisieren und den Verlust zu minimieren. Folglich können falsch gelabelte Trainingsbeispiele eine irreführende Modellgenauigkeit bewirken. Nehmen wir zum Beispiel an, dass Sie ein Stimmungsanalysemodell erstellen und 25% Ihrer »positiven« Trainingsbeispiele sind fälschlicherweise als »negativ« gelabelt worden. Dann wird Ihr Modell ein ungenaues Bild davon haben, was als negative Stimmung angesehen werden sollte, und dies wird sich direkt in seinen Vorhersagen widerspiegeln.

Um *Datenvollständigkeit* zu verstehen, nehmen wir an, Sie trainierten ein Modell, das Katzenrassen erkennen soll. Das Modell trainieren Sie auf einem umfangreichen Datensatz an Katzenbildern, und das fertige Modell ist in der Lage, Bilder mit 99%iger Genauigkeit einer von zehn möglichen Kategorien (»Bengal«, »Siam« usw.) zuzuordnen. Als Sie jedoch das Modell in die Produktion überführen, stellen Sie fest, dass nicht nur Katzenfotos zur Klassifizierung hochgeladen werden, sondern auch viele Ihrer Benutzer Fotos von Hunden hochladen und von den Ergebnissen des Modells enttäuscht sind. Da das Modell nur daraufhin trainiert wurde, zehn verschiedene Katzenrassen zu erkennen, kennt es auch nichts anderes. Diese zehn Rassenkategorien sind praktisch das gesamte »Weltbild« des Modells. Egal, was Sie dem Modell schicken, es wird das Bild einer dieser zehn Kategorien zuordnen – und möglicherweise sogar mit großer Überzeugung für ein Bild, das überhaupt nicht wie eine Katze aussieht. Darüber hinaus wird Ihr Modell nicht in der Lage sein, »keine Katze« zurückzugeben, wenn derartige Daten und Labels im Trainingsdatensatz nicht enthalten waren.

Des Weiteren ist zu gewährleisten, dass Ihre Trainingsdaten verschiedene Darstellungsvarianten zu jedem Label enthalten. Wenn im Beispiel der Katzenrassen alle Bilder nur Nahaufnahmen von Katzengesichtern zeigen, wird Ihr Modell später keine Katze korrekt erkennen können, wenn ihm ein Bild präsentiert wird, das eine Katze von der Seite oder als Ganzkörperdarstellung zeigt. Oder nehmen wir ein Beispiel mit tabellarischen Daten: Wenn Sie ein Modell entwickeln, das Immobilienpreise in einer bestimmten Stadt vorhersagen soll, Ihre Trainingsbeispiele aber

nur Häuser von mehr als 180 Quadratmetern umfassen, wird das Modell letztlich bei kleineren Häusern schlecht abschneiden.

Der dritte Aspekt der Datenqualität ist die *Datenkonsistenz*. Bei großen Datasets ist es üblich, das Erfassen und das Labeln der Daten auf eine Gruppe von Personen aufzuteilen. Wenn man eine Reihe von Standards für diesen Prozess entwickelt, kann das dazu beitragen, die Konsistenz über den Datensatz sicherzustellen, da jede beteiligte Person unweigerlich eigene Verzerrungen in den Prozess einbringt. Wie bei der Datenvollständigkeit können Dateninkonsistenzen sowohl bei den Features als auch den Labels der Daten auftreten. Nehmen wir als Beispiel für inkonsistente Features an, dass Sie atmosphärische Daten von Temperatursensoren sammeln. Wurde jeder Sensor nach anderen Standards kalibriert, führt das zu ungenauen und unzuverlässigen Modellvorhersagen. Inkonsistenzen können auch mit dem Datenformat zu tun haben. Wenn Sie Standortdaten erfassen, müssen Sie damit rechnen, dass die einen den Straßennamen vollständig schreiben, wie etwa »Hauptstraße«, aber andere den Straßennamen in der Adresse mit »Hauptstr.« abkürzen. Auch Maßeinheiten wie Meilen und Kilometer werden weltweit nicht einheitlich verwendet.

Was die Inkonsistenzen beim Labeling angeht, kommen wir auf das Beispiel der Textstimmung zurück. In diesem Fall ist es wahrscheinlich, dass die Meinungen darüber auseinandergehen, was als positiv und was als negativ anzusehen ist, wenn die Trainingsdaten gelabelt werden. Um dieses Problem zu lösen, können Sie jedes Beispiel in Ihrem Datensatz von mehreren Personen beurteilen lassen und dann das am häufigsten verwendete Label für jedes Element verwenden. Wenn Sie sich der potenziellen Voreingenommenheit der am Labeling beteiligten Personen bewusst sind und Systeme implementieren, die dies berücksichtigen, gewährleisten Sie Label-Konsistenz in Ihrem gesamten Datensatz. Das Konzept der Verzerrungen untersuchen wir in Kapitel 7 im Abschnitt »Entwurfsmuster 30: Fairness Lens« auf Seite 376.

Aktualität bei Daten bezieht sich auf die Latenz zwischen dem Zeitpunkt, zu dem ein Ereignis aufgetreten ist, und dem Zeitpunkt, zu dem es Ihrer Datenbank hinzugefügt wurde. Wenn Sie beispielsweise Daten in Anwendungsprotokollen sammeln, kann es Stunden dauern, bis ein Fehlerprotokoll in Ihrer Protokolldatenbank auftaucht. Bei einem Datensatz, der Kreditkartentransaktionen aufzeichnet, kann ab dem Zeitpunkt, zu dem die Transaktion stattgefunden hat, ein Tag vergehen, bis sie Ihrem System gemeldet wird. Für den Umgang mit Aktualität ist es zweckmäßig, möglichst viele Informationen über einen bestimmten Datenpunkt aufzuzeichnen und sicherzustellen, dass diese Informationen berücksichtigt werden, wenn Sie Ihre Daten in Features für ein ML-Modell transformieren. Im Besonderen können Sie anhand des Zeitstempels verfolgen, wann ein Ereignis aufgetreten ist und wann es in Ihren Datensatz aufgenommen wurde. Wenn Sie dann Feature Engineering betreiben, können Sie diese Differenzen entsprechend berücksichtigen.