O'REILLY®

Designing Machine Learning Systems

An Iterative Process for Production-Ready Applications

Chip Huyen

O'REILLY®

Designing Machine Learning Systems

Machine learning systems are both complex and unique. Complex because they consist of many different components and involve many different stakeholders. Unique because they're data dependent, with data varying wildly from one use case to the next. In this book, you'll learn a holistic approach to designing ML systems that are reliable, scalable, maintainable, and adaptive to changing environments and business requirements.

Author Chip Huyen, co-founder of Claypot AI, considers each design decision—such as how to process and create training data, which features to use, how often to retrain models, and what to monitor—in the context of how it can help your system as a whole achieve its objectives. The iterative framework in this book uses actual case studies backed by ample references.

This book will help you tackle scenarios such as:

- Engineering data and choosing the right metrics to solve a business problem
- Automating the process for continually developing, evaluating, deploying, and updating models
- Developing a monitoring system to quickly detect and address issues your models might encounter in production
- Architecting an ML platform that serves across use cases
- Developing responsible ML systems

"This is, simply, the very best book you can read about how to build, deploy, and scale machine learning models at a company for maximum impact."

—Josh Wills
Software Engineer at WeaveGrid and former Director of Data Engineering, Slack

"In a blooming but chaotic ecosystem, this principled view on end-to-end ML is both your map and your compass: a must-read for practitioners inside and outside of Big Tech."

—Jacopo Tagliabue
Director of AI, Coveo

Chip Huyen is co-founder of Claypot AI, a platform for real-time machine learning. Through her work at NVIDIA, Net ix, and Snorkel AI, she's helped some of the world's largest organizations develop and deploy ML systems. Chip based this book on her lectures for CS 329S: Machine Learning Systems Design, a course she teaches at Stanford University.

MACHINE LEARNING

US $59.99 CAN $74.99
ISBN: 978 1 098 10796 3

Twitter: @oreillymedia
linkedin.com/company/oreilly-media
youtube.com/oreillymedia

Praise for *Designing Machine Learning Systems*

There is so much information one needs to know to be an effective machine learning engineer. It's hard to cut through the chaff to get the most relevant information, but Chip has done that admirably with this book. If you are serious about ML in production, and care about how to design and implement ML systems end to end, this book is essential.

—Laurence Moroney, AI and ML Lead, Google

One of the best resources that focuses on the first principles behind designing ML systems for production. A must-read to navigate the ephemeral landscape of tooling and platform options.

—Goku Mohandas, Founder of Made With ML

Chip's manual is the book we deserve and the one we need right now. In a blooming but chaotic ecosystem, this principled view on end-to-end ML is both your map and your compass: a must-read for practitioners inside and outside of Big Tech—especially those working at "reasonable scale." This book will also appeal to data leaders looking for best practices on how to deploy, manage, and monitor systems in the wild.

—Jacopo Tagliabue, Director of AI, Coveo;
Adj. Professor of MLSys, NYU

This is, simply, the very best book you can read about how to build, deploy, and scale machine learning models at a company for maximum impact. Chip is a masterful teacher, and the breadth and depth of her knowledge is unparalleled.

—Josh Wills, So□ware Engineer at WeaveGrid and former
Director of Data Engineering, Slack

This is the book I wish I had read when I started as an ML engineer.

—Shreya Shankar, MLOps PhD Student

Designing Machine Learning Systems is a welcome addition to the field of applied machine learning. The book provides a detailed guide for people building end-to-end machine learning systems. Chip Huyen writes from her extensive, hands-on experience building real-world machine learning applications.

—Brian Spiering, Data Science Instructor at Metis

Chip is truly a world-class expert on machine learning systems, as well as a brilliant writer. Both are evident in this book, which is a fantastic resource for anyone looking to learn about this topic.

—Andrey Kurenkov, PhD Candidate at the Stanford AI Lab

Chip Huyen has produced an important addition to the canon of machine learning literature—one that is deeply literate in ML fundamentals, but has a much more concrete and practical approach than most. The focus on business requirements alone is uncommon and valuable. This book will resonate with engineers getting started with ML and with others in any part of the organization trying to understand how ML works.

—Todd Underwood, Senior Engineering Director for ML SRE, Google, and Coauthor of Reliable Machine Learning

Designing Machine Learning Systems

An Iterative Process for Production-Ready Applications

Chip Huyen

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Designing Machine Learning Systems
by Chip Huyen

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Nicole Butterfield
Development Editor: Jill Leonard
Production Editor: Gregory Hyman
Copyeditor: nSight, Inc.
Proofreader: Piper Editorial Consulting, LLC

Indexer: nSight, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

May 2022: First Edition

Revision History for the First Edition
2022-05-17: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781098107963* for release details.

978-1-098-10796-3

[LSI]

Table of Contents

Preface

Ever since the first machine learning course I taught at Stanford in 2017, many people have asked me for advice on how to deploy ML models at their organizations. These questions can be generic, such as "What model should I use?" "How often should I retrain my model?" "How can I detect data distribution shifts?" "How do I ensure that the features used during training are consistent with the features used during inference?"

These questions can also be specific, such as "I'm convinced that switching from batch prediction to online prediction will give our model a performance boost, but how do I convince my manager to let me do so?" or "I'm the most senior data scientist at my company and I've recently been tasked with setting up our first machine learning platform; where do I start?"

My short answer to all these questions is always: "It depends." My long answers often involve hours of discussion to understand where the questioner comes from, what they're actually trying to achieve, and the pros and cons of different approaches for their specific use case.

ML systems are both complex and unique. They are complex because they consist of many different components (ML algorithms, data, business logic, evaluation metrics, underlying infrastructure, etc.) and involve many different stakeholders (data scientists, ML engineers, business leaders, users, even society at large). ML systems are unique because they are data dependent, and data varies wildly from one use case to the next.

For example, two companies might be in the same domain (ecommerce) and have the same problem that they want ML to solve (recommender system), but their resulting ML systems can have different model architecture, use different sets of features, be evaluated on different metrics, and bring different returns on investment.

Many blog posts and tutorials on ML production focus on answering one specific question. While the focus helps get the point across, they can create the impression that it's possible to consider each of these questions in isolation. In reality, changes in one component will likely affect other components. Therefore, it's necessary to consider the system as a whole while attempting to make any design decision.

This book takes a holistic approach to ML systems. It takes into account different components of the system and the objectives of different stakeholders involved. The content in this book is illustrated using actual case studies, many of which I've personally worked on, backed by ample references, and reviewed by ML practitioners in both academia and industry. Sections that require in-depth knowledge of a certain topic—e.g., batch processing versus stream processing, infrastructure for storage and compute, and responsible AI—are further reviewed by experts whose work focuses on that one topic. In other words, this book is an attempt to give nuanced answers to the aforementioned questions and more.

When I first wrote the lecture notes that laid the foundation for this book, I thought I wrote them for my students to prepare them for the demands of their future jobs as data scientists and ML engineers. However, I soon realized that I also learned tremendously through the process. The initial drafts I shared with early readers sparked many conversations that tested my assumptions, forced me to consider different perspectives, and introduced me to new problems and new approaches.

I hope that this learning process will continue for me now that the book is in your hand, as you have experiences and perspectives that are unique to you. Please feel free to share with me any feedback you might have for this book, via the MLOps Discord server that I run (where you can also find other readers of this book), Twitter, LinkedIn, or other channels that you can find on my website.

Who This Book Is For

This book is for anyone who wants to leverage ML to solve real-world problems. ML in this book refers to both deep learning and classical algorithms, with a leaning toward ML systems at scale, such as those seen at medium to large enterprises and fast-growing startups. Systems at a smaller scale tend to be less complex and might benefit less from the comprehensive approach laid out in this book.

Because my background is engineering, the language of this book is geared toward engineers, including ML engineers, data scientists, data engineers, ML platform engineers, and engineering managers. You might be able to relate to one of the following scenarios:

- You have been given a business problem and a lot of raw data. You want to engineer this data and choose the right metrics to solve this problem.
- Your initial models perform well in offline experiments and you want to deploy them.
- You have little feedback on how your models are performing after your models are deployed, and you want to figure out a way to quickly detect, debug, and address any issue your models might run into in production.
- The process of developing, evaluating, deploying, and updating models for your team has been mostly manual, slow, and error-prone. You want to automate and improve this process.
- Each ML use case in your organization has been deployed using its own workflow, and you want to lay down the foundation (e.g., model store, feature store, monitoring tools) that can be shared and reused across use cases.
- You're worried that there might be biases in your ML systems and you want to make your systems responsible!

You can also benefit from the book if you belong to one of the following groups:

- Tool developers who want to identify underserved areas in ML production and figure out how to position your tools in the ecosystem.
- Individuals looking for ML-related roles in the industry.
- Technical and business leaders who are considering adopting ML solutions to improve your products and/or business processes. Readers without strong technical backgrounds might benefit the most from Chapters 1, 2, and 11.

What This Book Is Not

This book is not an introduction to ML. There are many books, courses, and resources available for ML theories, and therefore, this book shies away from these concepts to focus on the practical aspects of ML. To be specific, the book assumes that readers have a basic understanding of the following topics:

- *ML models* such as clustering, logistic regression, decision trees, collaborative filtering, and various neural network architectures including feed-forward, recurrent, convolutional, and transformer
- *ML techniques* such as supervised versus unsupervised, gradient descent, objective/loss function, regularization, generalization, and hyperparameter tuning
- *Metrics* such as accuracy, F1, precision, recall, ROC, mean squared error, and log-likelihood

- *Statistical concepts* such as variance, probability, and normal/long-tail distribution
- *Common ML tasks* such as language modeling, anomaly detection, object classification, and machine translation

You don't have to know these topics inside out—for concepts whose exact definitions can take some effort to remember, e.g., F1 score, we include short notes as references—but you should have a rough sense of what they mean going in.

While this book mentions current tools to illustrate certain concepts and solutions, it's not a tutorial book. Technologies evolve over time. Tools go in and out of style quickly, but fundamental approaches to problem solving should last a bit longer. This book provides a framework for you to evaluate the tool that works best for your use cases. When there's a tool you want to use, it's usually straightforward to find tutorials for it online. As a result, this book has few code snippets and instead focuses on providing a lot of discussion around trade-offs, pros and cons, and concrete examples.

Navigating This Book

The chapters in this book are organized to reflect the problems data scientists might encounter as they progress through the lifecycle of an ML project. The first two chapters lay down the groundwork to set an ML project up for success, starting from the most basic question: does your project need ML? It also covers choosing the objectives for your project and how to frame your problem in a way that makes for simpler solutions. If you're already familiar with these considerations and impatient to get to the technical solutions, feel free to skip the first two chapters.

Chapters 4 to 6 cover the pre-deployment phase of an ML project: from creating the training data and engineering features to developing and evaluating your models in a development environment. This is the phase where expertise in both ML and the problem domain are especially needed.

Chapters 7 to 9 cover the deployment and post-deployment phase of an ML project. We'll learn through a story many readers might be able to relate to that having a model deployed isn't the end of the deployment process. The deployed model will need to be monitored and continually updated to changing environments and business requirements.

Chapters 3 and 10 focus on the infrastructure needed to enable stakeholders from different backgrounds to work together to deliver successful ML systems. Chapter 3 focuses on data systems, whereas Chapter 10 focuses on compute infrastructure and ML platforms. I debated for a long time on how deep to go into data systems and where to introduce it in the book. Data systems, including databases, data formats,

data movements, and data processing engines, tend to be sparsely covered in ML coursework, and therefore many data scientists might think of them as low level or irrelevant. After consulting with many of my colleagues, I decided that because ML systems depend on data, covering the basics of data systems early will help us get on the same page to discuss data matters in the rest of the book.

While we cover many technical aspects of an ML system in this book, ML systems are built by people, for people, and can have outsized impact on the life of many. It'd be remiss to write a book on ML production without a chapter on the human side of it, which is the focus of Chapter 11, the last chapter.

Note that "data scientist" is a role that has evolved a lot in the last few years, and there have been many discussions to determine what this role should entail—we'll go into some of these discussions in Chapter 10. In this book, we use "data scientist" as an umbrella term to include anyone who works developing and deploying ML models, including people whose job titles might be ML engineers, data engineers, data analysts, etc.

GitHub Repository and Community

This book is accompanied by a GitHub repository that contains:

- A review of basic ML concepts
- A list of references used in this book and other advanced, updated resources
- Code snippets used in this book
- A list of tools you can use for certain problems you might encounter in your workflows

I also run a Discord server on MLOps where you're encouraged to discuss and ask questions about the book.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

As mentioned, supplemental material (code examples, exercises, etc.) is available for download at *https://oreil.ly/designing-machine-learning-systems-code*.

If you have a technical question or a problem using the code examples, please send email to *bookquestions@oreilly.com*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Designing Machine Learning Systems* by Chip Huyen (O'Reilly). Copyright 2022 Huyen Thi Khanh Nguyen, 978-1-098-10796-3."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning

For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *https://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/designing-machine-learning-systems*.

Email *bookquestions@oreilly.com* to comment or ask technical questions about this book.

For news and information about our books and courses, visit *https://oreilly.com*.

Find us on LinkedIn: *https://linkedin.com/company/oreilly-media*

Follow us on Twitter: *https://twitter.com/oreillymedia*

Watch us on YouTube: *https://youtube.com/oreillymedia*

Acknowledgments

This book took two years to write, and many more years beforehand to prepare. Looking back, I'm equally amazed and grateful for the enormous amount of help I received in writing this book. I tried my best to include the names of everyone who has helped me here, but due to the inherent faultiness of human memory, I undoubtedly neglected to mention many. If I forgot to include your name, please know that it wasn't because I don't appreciate your contribution and please kindly remind me so that I can rectify as soon as possible!

First and foremost, I'd like to thank the course staff who helped me develop the course and materials this book was based on: Michael Cooper, Xi Yin, Chloe He, Kinbert Chou, Megan Leszczynski, Karan Goel, and Michele Catasta. I'd like to

thank my professors, Christopher Ré and Mehran Sahami, without whom the course wouldn't exist in the first place.

I'd like to thank a long list of reviewers who not only gave encouragement but also improved the book by many orders of magnitude: Eugene Yan, Josh Wills, Han-chung Lee, Thomas Dietterich, Irene Tematelewo, Goku Mohandas, Jacopo Tagliabue, Andrey Kurenkov, Zach Nussbaum, Jay Chia, Laurens Geffert, Brian Spiering, Erin Ledell, Rosanne Liu, Chin Ling, Shreya Shankar, and Sara Hooker.

I'd like to thank all the readers who read the early release version of the book and gave me ideas on how to improve the book, including Charles Frye, Xintong Yu, Jordan Zhang, Jonathon Belotti, and Cynthia Yu.

Of course, the book wouldn't have been possible with the team at O'Reilly, especially my development editor, Jill Leonard, and my production editors, Kristen Brown, Sharon Tripp, and Gregory Hyman. I'd like to thank Laurence Moroney, Hannes Hapke, and Rebecca Novack, who helped me get this book from an idea to a proposal.

This book, after all, is an accumulation of invaluable lessons I learned throughout my career to date. I owe these lessons to my extremely competent and patient coworkers and former coworkers at Claypot AI, Primer AI, Netflix, NVIDIA, and Snorkel AI. Every person I've worked with has taught me something new about bringing ML into the world.

A special thanks to my cofounder Zhenzhong Xu for putting out the fires at our startup and allowing me to spend time on this book. Thank you, Luke, for always being so supportive of everything that I want to do, no matter how ambitious it is.

CHAPTER 1
Overview of Machine Learning Systems

In November 2016, Google announced that it had incorporated its multilingual neural machine translation system into Google Translate, marking one of the first success stories of deep artificial neural networks in production at scale.[1] According to Google, with this update, the quality of translation improved more in a single leap than they had seen in the previous 10 years combined.

This success of deep learning renewed the interest in machine learning (ML) at large. Since then, more and more companies have turned toward ML for solutions to their most challenging problems. In just five years, ML has found its way into almost every aspect of our lives: how we access information, how we communicate, how we work, how we find love. The spread of ML has been so rapid that it's already hard to imagine life without it. Yet there are still many more use cases for ML waiting to be explored in fields such as health care, transportation, farming, and even in helping us understand the universe.[2]

Many people, when they hear "machine learning system," think of just the ML algorithms being used such as logistic regression or different types of neural networks. However, the algorithm is only a small part of an ML system in production. The system also includes the business requirements that gave birth to the ML project in the first place, the interface where users and developers interact with your system, the data stack, and the logic for developing, monitoring, and updating your models, as well as the infrastructure that enables the delivery of that logic. Figure 1-1 shows you the different components of an ML system and in which chapters of this book they will be covered.

1 Mike Schuster, Melvin Johnson, and Nikhil Thorat, "Zero-Shot Translation with Google's Multilingual Neural Machine Translation System," *Google AI Blog*, November 22, 2016, *https://oreil.ly/2R1CB*.

2 Larry Hardesty, "A Method to Image Black Holes," *MIT News*, June 6, 2016, *https://oreil.ly/HpL2F*.

The Relationship Between MLOps and ML Systems Design

Ops in MLOps comes from DevOps, short for Developments and Operations. To operationalize something means to bring it into production, which includes deploying, monitoring, and maintaining it. MLOps is a set of tools and best practices for bringing ML into production.

ML systems design takes a system approach to MLOps, which means that it considers an ML system holistically to ensure that all the components and their stakeholders can work together to satisfy the specified objectives and requirements.

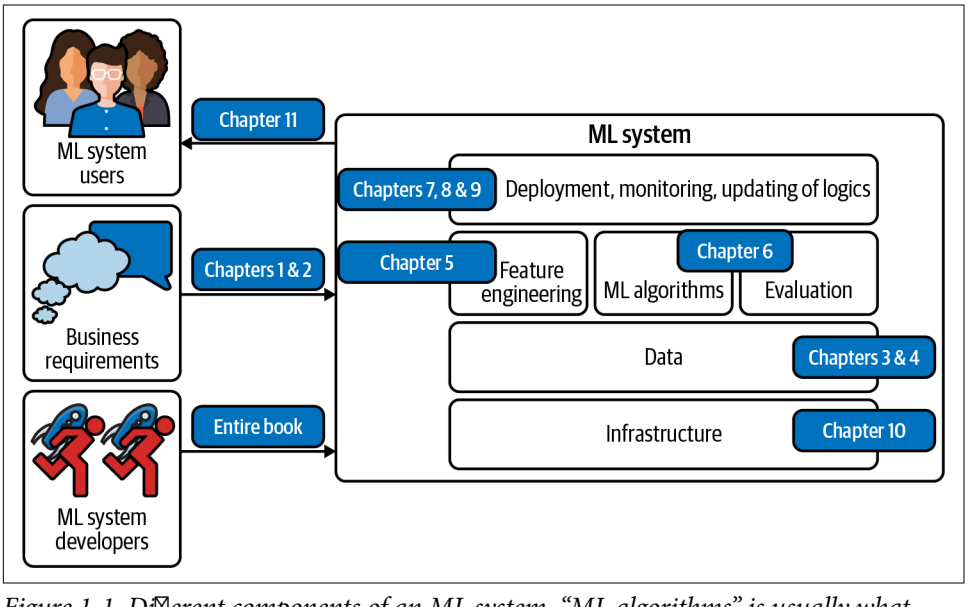

Figure 1-1. Different components of an ML system. "ML algorithms" is usually what people think of when they say machine learning, but it's only a small part of the entire system.

There are many excellent books about various ML algorithms. This book doesn't cover any specific algorithms in detail but rather helps readers understand the entire ML system as a whole. In other words, this book's goal is to provide you with a framework to develop a solution that best works for your problem, regardless of which algorithm you might end up using. Algorithms might become outdated quickly as new algorithms are constantly being developed, but the framework proposed in this book should still work with new algorithms.

The first chapter of the book aims to give you an overview of what it takes to bring an ML model to production. Before discussing how to develop an ML system, it's

important to ask a fundamental question of when and when not to use ML. We'll cover some of the popular use cases of ML to illustrate this point.

After the use cases, we'll move on to the challenges of deploying ML systems, and we'll do so by comparing ML in production to ML in research as well as to traditional software. If you've been in the trenches of developing applied ML systems, you might already be familiar with what's written in this chapter. However, if you have only had experience with ML in an academic setting, this chapter will give an honest view of ML in the real world and set your first application up for success.

When to Use Machine Learning

As its adoption in the industry quickly grows, ML has proven to be a powerful tool for a wide range of problems. Despite an incredible amount of excitement and hype generated by people both inside and outside the field, ML is not a magic tool that can solve all problems. Even for problems that ML can solve, ML solutions might not be the optimal solutions. Before starting an ML project, you might want to ask whether ML is necessary or cost-effective.[3]

To understand what ML can do, let's examine what ML solutions generally do:

> Machine learning is an approach to (1) *learn* (2) *complex patterns* from (3) *existing data* and use these patterns to make (4) *predictions* on (5) *unseen data*.

We'll look at each of the italicized keyphrases in the above framing to understand its implications to the problems ML can solve:

1. *Learn: the system has the capacity to learn*

   A relational database isn't an ML system because it doesn't have the capacity to learn. You can explicitly state the relationship between two columns in a relational database, but it's unlikely to have the capacity to figure out the relationship between these two columns by itself.

   For an ML system to learn, there must be something for it to learn from. In most cases, ML systems learn from data. In supervised learning, based on example input and output pairs, ML systems learn how to generate outputs for arbitrary inputs. For example, if you want to build an ML system to learn to predict the rental price for Airbnb listings, you need to provide a dataset where each input is a listing with relevant characteristics (square footage, number of rooms, neighborhood, amenities, rating of that listing, etc.) and the associated output is the rental price of that listing. Once learned, this ML system should be able to predict the price of a new listing given its characteristics.

3 I didn't ask whether ML is sufficient because the answer is always no.

2. Complex patterns: there are patterns to learn, and they are complex

ML solutions are only useful when there are patterns to learn. Sane people don't invest money into building an ML system to predict the next outcome of a fair die because there's no pattern in how these outcomes are generated.[4] However, there are patterns in how stocks are priced, and therefore companies have invested billions of dollars in building ML systems to learn those patterns.

Whether a pattern exists might not be obvious, or if patterns exist, your dataset or ML algorithms might not be sufficient to capture them. For example, there might be a pattern in how Elon Musk's tweets affect cryptocurrency prices. However, you wouldn't know until you've rigorously trained and evaluated your ML models on his tweets. Even if all your models fail to make reasonable predictions of cryptocurrency prices, it doesn't mean there's no pattern.

Consider a website like Airbnb with a lot of house listings; each listing comes with a zip code. If you want to sort listings into the states they are located in, you wouldn't need an ML system. Since the pattern is simple—each zip code corresponds to a known state—you can just use a lookup table.

The relationship between a rental price and all its characteristics follows a much more complex pattern, which would be very challenging to manually specify. ML is a good solution for this. Instead of telling your system how to calculate the price from a list of characteristics, you can provide prices and characteristics, and let your ML system figure out the pattern. The difference between ML solutions and the lookup table solution as well as general traditional software solutions is shown in Figure 1-2. For this reason, ML is also called Software 2.0.[5]

ML has been very successful with tasks with complex patterns such as object detection and speech recognition. What is complex to machines is different from what is complex to humans. Many tasks that are hard for humans to do are easy for machines—for example, raising a number of the power of 10. On the other hand, many tasks that are easy for humans can be hard for machines—for example, deciding whether there's a cat in a picture.

4 Patterns are different from distributions. We know the distribution of the outcomes of a fair die, but there are no patterns in the way the outcomes are generated.

5 Andrej Karpathy, "Software 2.0," *Medium*, November 11, 2017, *https://oreil.ly/yHZrE*.

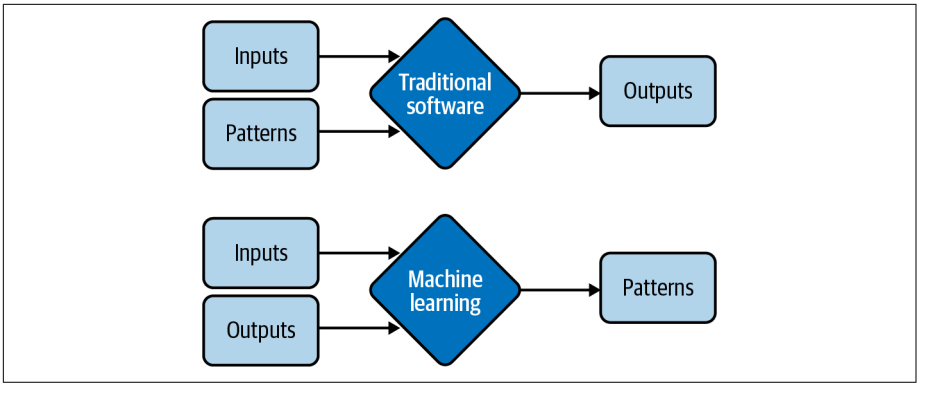

Figure 1-2. Instead of requiring hand-specified patterns to calculate outputs, ML solutions learn patterns from inputs and outputs

3. *Existing data: data is available, or it's possible to collect data*

   Because ML learns from data, there must be data for it to learn from. It's amusing to think about building a model to predict how much tax a person should pay a year, but it's not possible unless you have access to tax and income data of a large population.

   In the zero-shot learning (sometimes known as zero-data learning) context, it's possible for an ML system to make good predictions for a task without having been trained on data for that task. However, this ML system was previously trained on data for other tasks, often related to the task in consideration. So even though the system doesn't require data for the task at hand to learn from, it still requires data to learn.

   It's also possible to launch an ML system without data. For example, in the context of continual learning, ML models can be deployed without having been trained on any data, but they will learn from incoming data in production.[6] However, serving insufficiently trained models to users comes with certain risks, such as poor customer experience.

   Without data and without continual learning, many companies follow a "fake-it-til-you make it" approach: launching a product that serves predictions made by humans, instead of ML models, with the hope of using the generated data to train ML models later.

6 We'll go over online learning in Chapter 9.

4. *Predictions: it's a predictive problem*

   ML models make predictions, so they can only solve problems that require predictive answers. ML can be especially appealing when you can benefit from a large quantity of cheap but approximate predictions. In English, "predict" means "estimate a value in the future." For example, what will the weather be like tomorrow? Who will win the Super Bowl this year? What movie will a user want to watch next?

   As predictive machines (e.g., ML models) are becoming more effective, more and more problems are being reframed as predictive problems. Whatever question you might have, you can always frame it as: "What would the answer to this question be?" regardless of whether this question is about something in the future, the present, or even the past.

   Compute-intensive problems are one class of problems that have been very successfully reframed as predictive. Instead of computing the exact outcome of a process, which might be even more computationally costly and time-consuming than ML, you can frame the problem as: "What would the outcome of this process look like?" and approximate it using an ML model. The output will be an approximation of the exact output, but often, it's good enough. You can see a lot of it in graphic renderings, such as image denoising and screen-space shading.[7]

5. *Unseen data: unseen data shares patterns with the training data*

   The patterns your model learns from existing data are only useful if unseen data also share these patterns. A model to predict whether an app will get downloaded on Christmas 2020 won't perform very well if it's trained on data from 2008, when the most popular app on the App Store was Koi Pond. What's Koi Pond? Exactly.

   In technical terms, it means your unseen data and training data should come from similar distributions. You might ask: "If the data is unseen, how do we know what distribution it comes from?" We don't, but we can make assumptions—such as we can assume that users' behaviors tomorrow won't be too different from users' behaviors today—and hope that our assumptions hold. If they don't, we'll have a model that performs poorly, which we might be able to find out with monitoring, as covered in Chapter 8, and test in production, as covered in Chapter 9.

7 Steke Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle, "Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings," *ACM Transactions on Graphics* 36, no. 4 (2017): 97, *https://oreil.ly/EeI3j*; Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, Hans-Peter Seidel, and Tobias Ritschel, "Deep Shading: Convolutional Neural Networks for Screen-Space Shading," *arXiv*, 2016, *https://oreil.ly/dSspz*.

Due to the way most ML algorithms today learn, ML solutions will especially shine if your problem has these additional following characteristics:

6. It's repetitive

Humans are great at few-shot learning: you can show kids a few pictures of cats and most of them will recognize a cat the next time they see one. Despite exciting progress in few-shot learning research, most ML algorithms still require many examples to learn a pattern. When a task is repetitive, each pattern is repeated multiple times, which makes it easier for machines to learn it.

7. The cost of wrong predictions is cheap

Unless your ML model's performance is 100% all the time, which is highly unlikely for any meaningful tasks, your model is going to make mistakes. ML is especially suitable when the cost of a wrong prediction is low. For example, one of the biggest use cases of ML today is in recommender systems because with recommender systems, a bad recommendation is usually forgiving—the user just won't click on the recommendation.

If one prediction mistake can have catastrophic consequences, ML might still be a suitable solution if, on average, the benefits of correct predictions outweigh the cost of wrong predictions. Developing self-driving cars is challenging because an algorithmic mistake can lead to death. However, many companies still want to develop self-driving cars because they have the potential to save many lives once self-driving cars are statistically safer than human drivers.

8. It's at scale

ML solutions often require nontrivial up-front investment on data, compute, infrastructure, and talent, so it'd make sense if we can use these solutions a lot.

"At scale" means different things for different tasks, but, in general, it means making a lot of predictions. Examples include sorting through millions of emails a year or predicting which departments thousands of support tickets should be routed to a day.

A problem might appear to be a singular prediction, but it's actually a series of predictions. For example, a model that predicts who will win a US presidential election seems like it only makes one prediction every four years, but it might actually be making a prediction every hour or even more frequently because that prediction has to be continually updated to incorporate new information.

Having a problem at scale also means that there's a lot of data for you to collect, which is useful for training ML models.

9. *Patterns are constantly changing*

   Cultures change. Tastes change. Technologies change. What's trendy today might be old news tomorrow. Consider the task of email spam classification. Today an indication of a spam email is a Nigerian prince, but tomorrow it might be a distraught Vietnamese writer.

   If your problem involves one or more constantly changing patterns, hardcoded solutions such as handwritten rules can become outdated quickly. Figuring how your problem has changed so that you can update your handwritten rules accordingly can be too expensive or impossible. Because ML learns from data, you can update your ML model with new data without having to figure out how the data has changed. It's also possible to set up your system to adapt to the changing data distributions, an approach we'll discuss in the section "Continual Learning" on page 264.

The list of use cases can go on and on, and it'll grow even longer as ML adoption matures in the industry. Even though ML can solve a subset of problems very well, it can't solve and/or shouldn't be used for a lot of problems. Most of today's ML algorithms shouldn't be used under any of the following conditions:

- It's unethical. We'll go over one case study where the use of ML algorithms can be argued as unethical in the section "Case study I: Automated grader's biases" on page 341.
- Simpler solutions do the trick. In Chapter 6, we'll cover the four phases of ML model development where the first phase should be non-ML solutions.
- It's not cost-effective.

However, even if ML can't solve your problem, it might be possible to break your problem into smaller components, and use ML to solve some of them. For example, if you can't build a chatbot to answer all your customers' queries, it might be possible to build an ML model to predict whether a query matches one of the frequently asked questions. If yes, direct the customer to the answer. If not, direct them to customer service.

I'd also want to caution against dismissing a new technology because it's not as cost-effective as the existing technologies at the moment. Most technological advances are incremental. A type of technology might not be efficient now, but it might be over time with more investments. If you wait for the technology to prove its worth to the rest of the industry before jumping in, you might end up years or decades behind your competitors.

Machine Learning Use Cases

ML has found increasing usage in both enterprise and consumer applications. Since the mid-2010s, there has been an explosion of applications that leverage ML to deliver superior or previously impossible services to consumers.

With the explosion of information and services, it would have been very challenging for us to find what we want without the help of ML, manifested in either a *search engine* or a *recommender system*. When you visit a website like Amazon or Netflix, you're recommended items that are predicted to best match your taste. If you don't like any of your recommendations, you might want to search for specific items, and your search results are likely powered by ML.

If you have a smartphone, ML is likely already assisting you in many of your daily activities. Typing on your phone is made easier with *predictive typing*, an ML system that gives you suggestions on what you might want to say next. An ML system might run in your photo editing app to suggest how best to enhance your photos. You might authenticate your phone using your fingerprint or your face, which requires an ML system to predict whether a fingerprint or a face matches yours.

The ML use case that drew me into the field was *machine translation*, automatically translating from one language to another. It has the potential to allow people from different cultures to communicate with each other, erasing the language barrier. My parents don't speak English, but thanks to Google Translate, now they can read my writing and talk to my friends who don't speak Vietnamese.

ML is increasingly present in our homes with smart personal assistants such as Alexa and Google Assistant. Smart security cameras can let you know when your pets leave home or if you have an uninvited guest. A friend of mine was worried about his aging mother living by herself—if she falls, no one is there to help her get up—so he relied on an at-home health monitoring system that predicts whether someone has fallen in the house.

Even though the market for consumer ML applications is booming, the majority of ML use cases are still in the enterprise world. Enterprise ML applications tend to have vastly different requirements and considerations from consumer applications. There are many exceptions, but for most cases, enterprise applications might have stricter accuracy requirements but be more forgiving with latency requirements. For example, improving a speech recognition system's accuracy from 95% to 95.5% might not be noticeable to most consumers, but improving a resource allocation system's efficiency by just 0.1% can help a corporation like Google or General Motors save millions of dollars. At the same time, latency of a second might get a consumer distracted and opening something else, but enterprise users might be more tolerant of high latency. For people interested in building companies out of ML applications,

consumer apps might be easier to distribute but much harder to monetize. However, most enterprise use cases aren't obvious unless you've encountered them yourself.

According to Algorithmia's 2020 state of enterprise machine learning survey, ML applications in enterprises are diverse, serving both internal use cases (reducing costs, generating customer insights and intelligence, internal processing automation) and external use cases (improving customer experience, retaining customers, interacting with customers) as shown in Figure 1-3.[8]

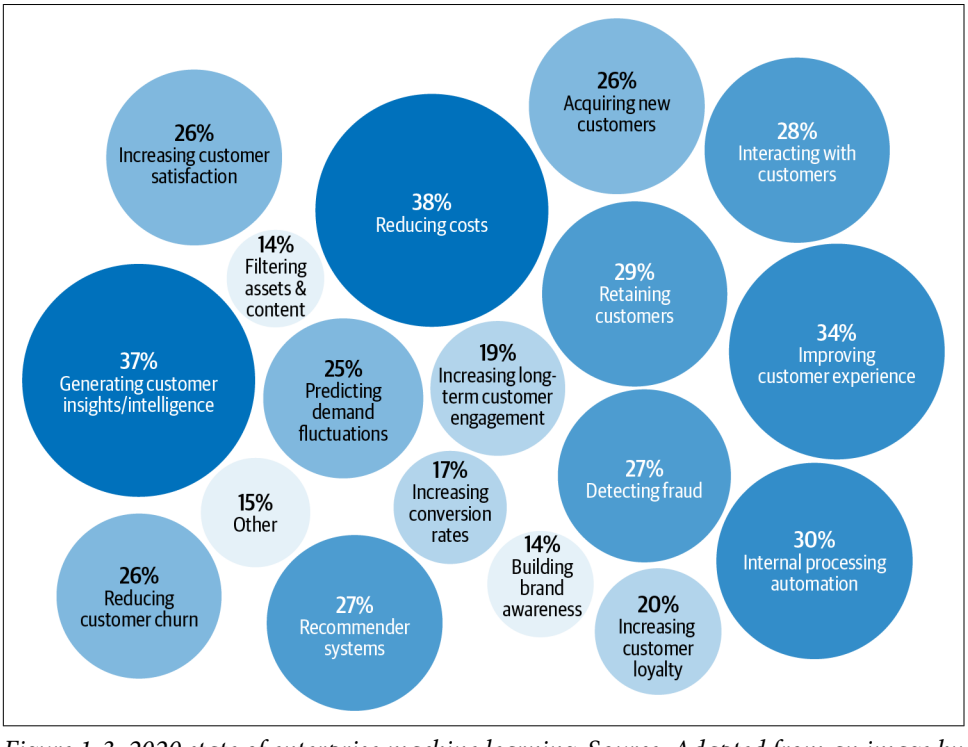

Figure 1-3. 2020 state of enterprise machine learning. Source: Adapted from an image by Algorithmia

8 "2020 State of Enterprise Machine Learning," *Algorithmia*, 2020, *https://oreil.ly/wKMZB*.

Fraud detection is among the oldest applications of ML in the enterprise world. If your product or service involves transactions of any value, it'll be susceptible to fraud. By leveraging ML solutions for anomaly detection, you can have systems that learn from historical fraud transactions and predict whether a future transaction is fraudulent.

Deciding how much to charge for your product or service is probably one of the hardest business decisions; why not let ML do it for you? *Price optimization* is the process of estimating a price at a certain time period to maximize a defined objective function, such as the company's margin, revenue, or growth rate. ML-based pricing optimization is most suitable for cases with a large number of transactions where demand fluctuates and consumers are willing to pay a dynamic price—for example, internet ads, flight tickets, accommodation bookings, ride-sharing, and events.

To run a business, it's important to be able to forecast customer demand so that you can prepare a budget, stock inventory, allocate resources, and update pricing strategy. For example, if you run a grocery store, you want to stock enough so that customers find what they're looking for, but you don't want to overstock, because if you do, your groceries might go bad and you lose money.

Acquiring a new user is expensive. As of 2019, the average cost for an app to acquire a user who'll make an in-app purchase is $86.61.[9] The acquisition cost for Lyft is estimated at $158/rider.[10] This cost is so much higher for enterprise customers. Customer acquisition cost is hailed by investors as a startup killer.[11] Reducing customer acquisition costs by a small amount can result in a large increase in profit. This can be done through better identifying potential customers, showing better-targeted ads, giving discounts at the right time, etc.—all of which are suitable tasks for ML.

After you've spent so much money acquiring a customer, it'd be a shame if they leave. The cost of acquiring a new user is approximated to be 5 to 25 times more expensive than retaining an existing one.[12] *Churn prediction* is predicting when a specific customer is about to stop using your products or services so that you can take appropriate actions to win them back. Churn prediction can be used not only for customers but also for employees.

9 "Average Mobile App User Acquisition Costs Worldwide from September 2018 to August 2019, by User Action and Operating System," *Statista*, 2019, *https://oreil.ly/2pTCH*.

10 Jeff Henriksen, "Valuing Lyft Requires a Deep Look into Unit Economics," *Forbes*, May 17, 2019, *https://oreil.ly/VeSt4*.

11 David Skok, "Startup Killer: The Cost of Customer Acquisition," *For Entrepreneurs*, 2018, *https://oreil.ly/L3tQ7*.

12 Amy Gallo, "The Value of Keeping the Right Customers," *Harvard Business Review*, October 29, 2014, *https://oreil.ly/OlNkl*.

To prevent customers from leaving, it's important to keep them happy by addressing their concerns as soon as they arise. Automated support ticket classification can help with that. Previously, when a customer opened a support ticket or sent an email, it needed to first be processed then passed around to different departments until it arrived at the inbox of someone who could address it. An ML system can analyze the ticket content and predict where it should go, which can shorten the response time and improve customer satisfaction. It can also be used to classify internal IT tickets.

Another popular use case of ML in enterprise is brand monitoring. The brand is a valuable asset of a business.[13] It's important to monitor how the public and your customers perceive your brand. You might want to know when/where/how it's mentioned, both explicitly (e.g., when someone mentions "Google") or implicitly (e.g., when someone says "the search giant"), as well as the sentiment associated with it. If there's suddenly a surge of negative sentiment in your brand mentions, you might want to address it as soon as possible. Sentiment analysis is a typical ML task.

A set of ML use cases that has generated much excitement recently is in health care. There are ML systems that can detect skin cancer and diagnose diabetes. Even though many health-care applications are geared toward consumers, because of their strict requirements with accuracy and privacy, they are usually provided through a health-care provider such as a hospital or used to assist doctors in providing diagnosis.

Understanding Machine Learning Systems

Understanding ML systems will be helpful in designing and developing them. In this section, we'll go over how ML systems are different from both ML in research (or as often taught in school) and traditional software, which motivates the need for this book.

Machine Learning in Research Versus in Production

As ML usage in the industry is still fairly new, most people with ML expertise have gained it through academia: taking courses, doing research, reading academic papers. If that describes your background, it might be a steep learning curve for you to understand the challenges of deploying ML systems in the wild and navigate an overwhelming set of solutions to these challenges. ML in production is very different from ML in research. Table 1-1 shows five of the major differences.

13 Marty Swant, "The World's 20 Most Valuable Brands," *Forbes*, 2020, *https://oreil.ly/4uS5i*.

Table 1-1. Key differences between ML in research and ML in production

	Research	Production
Requirements	State-of-the-art model performance on benchmark datasets	Different stakeholders have different requirements
Computational priority	Fast training, high throughput	Fast inference, low latency
Data	Static[a]	Constantly shifting
Fairness	Often not a focus	Must be considered
Interpretability	Often not a focus	Must be considered

[a] A subfield of research focuses on continual learning: developing models to work with changing data distributions. We'll cover continual learning in Chapter 9.

Different stakeholders and requirements

People involved in a research and leaderboard project often align on one single objective. The most common objective is model performance—develop a model that achieves the state-of-the-art results on benchmark datasets. To edge out a small improvement in performance, researchers often resort to techniques that make models too complex to be useful.

There are many stakeholders involved in bringing an ML system into production. Each stakeholder has their own requirements. Having different, often conflicting, requirements can make it difficult to design, develop, and select an ML model that satisfies all the requirements.

Consider a mobile app that recommends restaurants to users. The app makes money by charging restaurants a 10% service fee on each order. This means that expensive orders give the app more money than cheap orders. The project involves ML engineers, salespeople, product managers, infrastructure engineers, and a manager:

ML engineers
: Want a model that recommends restaurants that users will most likely order from, and they believe they can do so by using a more complex model with more data.

Sales team
: Wants a model that recommends the more expensive restaurants since these restaurants bring in more service fees.

Product team
: Notices that every increase in latency leads to a drop in orders through the service, so they want a model that can return the recommended restaurants in less than 100 milliseconds.

ML platform team
: As the traffic grows, this team has been woken up in the middle of the night because of problems with scaling their existing system, so they want to hold off on model updates to prioritize improving the ML platform.

Manager
: Wants to maximize the margin, and one way to achieve this might be to let go of the ML team.[14]

"Recommending the restaurants that users are most likely to click on" and "recommending the restaurants that will bring in the most money for the app" are two different objectives, and in the section "Decoupling objectives" on page 41, we'll discuss how to develop an ML system that satisfies different objectives. Spoiler: we'll develop one model for each objective and combine their predictions.

Let's imagine for now that we have two different models. Model A is the model that recommends the restaurants that users are most likely to click on, and model B is the model that recommends the restaurants that will bring in the most money for the app. A and B might be very different models. Which model should be deployed to the users? To make the decision more difficult, neither A nor B satisfies the requirement set forth by the product team: they can't return restaurant recommendations in less than 100 milliseconds.

When developing an ML project, it's important for ML engineers to understand requirements from all stakeholders involved and how strict these requirements are. For example, if being able to return recommendations within 100 milliseconds is a must-have requirement—the company finds that if your model takes over 100 milliseconds to recommend restaurants, 10% of users would lose patience and close the app—then neither model A nor model B will work. However, if it's just a nice-to-have requirement, you might still want to consider model A or model B.

Production having different requirements from research is one of the reasons why successful research projects might not always be used in production. For example, ensembling is a technique popular among the winners of many ML competitions, including the famed $1 million Netflix Prize, and yet it's not widely used in production. Ensembling combines "multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone."[15] While it can give your ML system a small performance improvement, ensembling tends to make a system too complex to be useful in production, e.g.,

14 It's not unusual for the ML and data science teams to be among the first to go during a company's mass layoff, as has been reported at IBM, Uber, Airbnb. See also Sejuti Das's analysis "How Data Scientists Are Also Susceptible to the Layoffs Amid Crisis," *Analytics India Magazine*, May 21, 2020, *https://oreil.ly/jobmz*.

15 Wikipedia, s.v. "Ensemble learning," *https://oreil.ly/5qkgp*.

slower to make predictions or harder to interpret the results. We'll discuss ensembling further in the section "Ensembles" on page 156.

For many tasks, a small improvement in performance can result in a huge boost in revenue or cost savings. For example, a 0.2% improvement in the click-through rate for a product recommender system can result in millions of dollars increase in revenue for an ecommerce site. However, for many tasks, a small improvement might not be noticeable for users. For the second type of task, if a simple model can do a reasonable job, complex models must perform significantly better to justify the complexity.

Criticism of ML Leaderboards

In recent years, there have been many critics of ML leaderboards, both competitions such as Kaggle and research leaderboards such as ImageNet or GLUE.

An obvious argument is that in these competitions many of the hard steps needed for building ML systems are already done for you.[16]

A less obvious argument is that due to the multiple-hypothesis testing scenario that happens when you have multiple teams testing on the same hold-out test set, a model can do better than the rest just by chance.[17]

The misalignment of interests between research and production has been noticed by researchers. In an EMNLP 2020 paper, Ethayarajh and Jurafsky argued that benchmarks have helped drive advances in natural language processing (NLP) by incentivizing the creation of more accurate models at the expense of other qualities valued by practitioners such as compactness, fairness, and energy efficiency.[18]

Computational priorities

When designing an ML system, people who haven't deployed an ML system often make the mistake of focusing too much on the model development part and not enough on the model deployment and maintenance part.

During the model development process, you might train many different models, and each model does multiple passes over the training data. Each trained model then generates predictions on the validation data once to report the scores. The validation data is usually much smaller than the training data. During model development,

16 Julia Evans, "Machine Learning Isn't Kaggle Competitions," 2014, *https://oreil.ly/p8mZq*.

17 Lauren Oakden-Rayner, "AI Competitions Don't Produce Useful Models," September 19, 2019, *https://oreil.ly/X6RlT*.

18 Kawin Ethayarajh and Dan Jurafsky, "Utility Is in the Eye of the User: A Critique of NLP Leaderboards," EMNLP, 2020, *https://oreil.ly/4Ud8P*.

training is the bottleneck. Once the model has been deployed, however, its job is to generate predictions, so inference is the bottleneck. Research usually prioritizes fast training, whereas production usually prioritizes fast inference.

One corollary of this is that research prioritizes high throughput whereas production prioritizes low latency. In case you need a refresh, latency refers to the time it takes from receiving a query to returning the result. Throughput refers to how many queries are processed within a specific period of time.

Terminology Clash

Some books make the distinction between latency and response time. According to Martin Kleppmann in his book *Designing Data-Intensive Applications*, "The response time is what the client sees: besides the actual time to process the request (the service time), it includes network delays and queueing delays. Latency is the duration that a request is waiting to be handled—during which it is latent, awaiting service."[19]

In this book, to simplify the discussion and to be consistent with the terminology used in the ML community, we use latency to refer to the response time, so the latency of a request measures the time from when the request is sent to the time a response is received.

For example, the average latency of Google Translate is the average time it takes from when a user clicks Translate to when the translation is shown, and the throughput is how many queries it processes and serves a second.

If your system always processes one query at a time, higher latency means lower throughput. If the average latency is 10 ms, which means it takes 10 ms to process a query, the throughput is 100 queries/second. If the average latency is 100 ms, the throughput is 10 queries/second.

However, because most modern distributed systems batch queries to process them together, often concurrently, *higher latency might also mean higher throughput*. If you process 10 queries at a time and it takes 10 ms to run a batch, the average latency is still 10 ms but the throughput is now 10 times higher—1,000 queries/second. If you process 50 queries at a time and it takes 20 ms to run a batch, the average latency now is 20 ms and the throughput is 2,500 queries/second. Both latency and throughput have increased! The difference in latency and throughput trade-off for processing queries one at a time and processing queries in batches is illustrated in Figure 1-4.

19 Martin Kleppmann, *Designing Data-Intensive Applications* (Sebastopol, CA: O'Reilly, 2017).

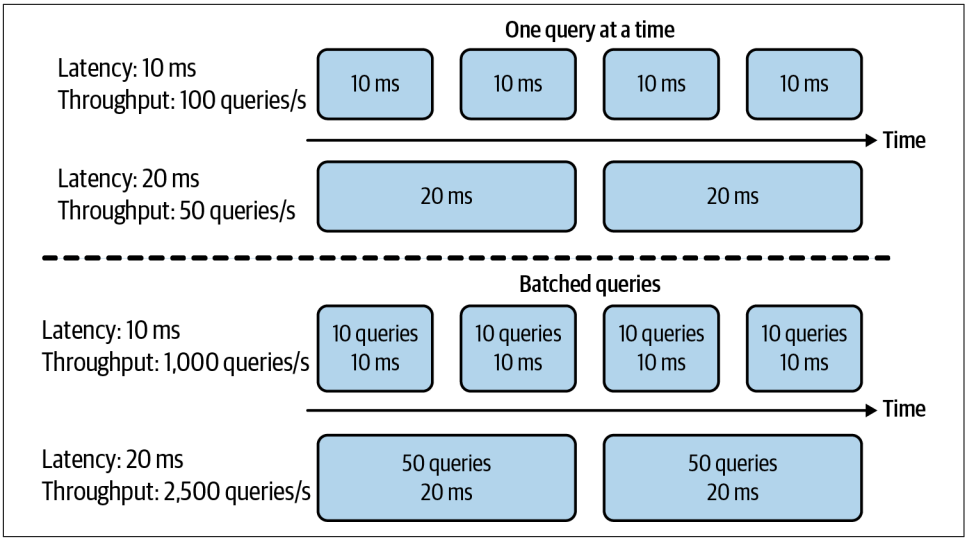

Figure 1-4. When processing queries one at a time, higher latency means lower throughput. When processing queries in batches, however, higher latency might also mean higher throughput.

This is even more complicated if you want to batch online queries. Batching requires your system to wait for enough queries to arrive in a batch before processing them, which further increases latency.

In research, you care more about how many samples you can process in a second (throughput) and less about how long it takes for each sample to be processed (latency). You're willing to increase latency to increase throughput, for example, with aggressive batching.

However, once you deploy your model into the real world, latency matters a lot. In 2017, an Akamai study found that a 100 ms delay can hurt conversion rates by 7%.[20] In 2019, Booking.com found that an increase of about 30% in latency cost about 0.5% in conversion rates—"a relevant cost for our business."[21] In 2016, Google found that more than half of mobile users will leave a page if it takes more than three seconds to load.[22] Users today are even less patient.

20 Akamai Technologies, *Akamai Online Retail Performance Report: Milliseconds Are Critical*, April 19, 2017, *https://oreil.ly/bEtRu*.

21 Lucas Bernardi, Themis Mavridis, and Pablo Estevez, "150 Successful Machine Learning Models: 6 Lessons Learned at Booking.com," KDD '19, August 4–8, 2019, Anchorage, AK, *https://oreil.ly/G5QNA*.

22 "Consumer Insights," Think with Google, *https://oreil.ly/JCp6Z*.

To reduce latency in production, you might have to reduce the number of queries you can process on the same hardware at a time. If your hardware is capable of processing many more queries at a time, using it to process fewer queries means underutilizing your hardware, increasing the cost of processing each query.

When thinking about latency, it's important to keep in mind that latency is not an individual number but a distribution. It's tempting to simplify this distribution by using a single number like the average (arithmetic mean) latency of all the requests within a time window, but this number can be misleading. Imagine you have 10 requests whose latencies are 100 ms, 102 ms, 100 ms, 100 ms, 99 ms, 104 ms, 110 ms, 90 ms, 3,000 ms, 95 ms. The average latency is 390 ms, which makes your system seem slower than it actually is. What might have happened is that there was a network error that made one request much slower than others, and you should investigate that troublesome request.

It's usually better to think in percentiles, as they tell you something about a certain percentage of your requests. The most common percentile is the 50th percentile, abbreviated as p50. It's also known as the median. If the median is 100 ms, half of the requests take longer than 100 ms, and half of the requests take less than 100 ms.

Higher percentiles also help you discover outliers, which might be symptoms of something wrong. Typically, the percentiles you'll want to look at are p90, p95, and p99. The 90th percentile (p90) for the 10 requests above is 3,000 ms, which is an outlier.

Higher percentiles are important to look at because even though they account for a small percentage of your users, sometimes they can be the most important users. For example, on the Amazon website, the customers with the slowest requests are often those who have the most data on their accounts because they have made many purchases—that is, they're the most valuable customers.[23]

It's a common practice to use high percentiles to specify the performance requirements for your system; for example, a product manager might specify that the 90th percentile or 99.9th percentile latency of a system must be below a certain number.

Data

During the research phase, the datasets you work with are often clean and well-formatted, freeing you to focus on developing models. They are static by nature so that the community can use them to benchmark new architectures and techniques. This means that many people might have used and discussed the same datasets, and quirks of the dataset are known. You might even find open source scripts to process and feed the data directly into your models.

23 Kleppmann, *Designing Data-Intensive Applications*.

In production, data, if available, is a lot more messy. It's noisy, possibly unstructured, constantly shifting. It's likely biased, and you likely don't know how it's biased. Labels, if there are any, might be sparse, imbalanced, or incorrect. Changing project or business requirements might require updating some or all of your existing labels. If you work with users' data, you'll also have to worry about privacy and regulatory concerns. We'll discuss a case study where users' data is inadequately handled in the section "Case study II: The danger of "anonymized" data" on page 344.

In research, you mostly work with historical data, e.g., data that already exists and is stored somewhere. In production, most likely you'll also have to work with data that is being constantly generated by users, systems, and third-party data.

Figure 1-5 has been adapted from a great graphic by Andrej Karpathy, director of AI at Tesla, that illustrates the data problems he encountered during his PhD compared to his time at Tesla.

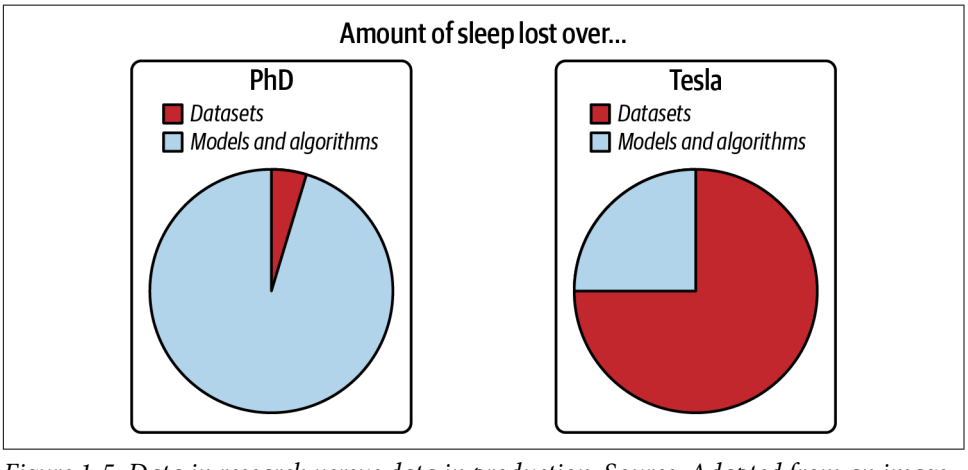

Figure 1-5. Data in research versus data in production. Source: Adapted from an image by Andrej Karpathy[24]

Fairness

During the research phase, a model is not yet used on people, so it's easy for researchers to put off fairness as an afterthought: "Let's try to get state of the art first and worry about fairness when we get to production." When it gets to production, it's too late. If you optimize your models for better accuracy or lower latency, you can show that your models beat state of the art. But, as of writing this book, there's no equivalent state of the art for fairness metrics.

24 Andrej Karpathy, "Building the Software 2.0 Stack," Spark+AI Summit 2018, video, 17:54, *https://oreil.ly/Z21Oz*.

You or someone in your life might already be a victim of biased mathematical algorithms without knowing it. Your loan application might be rejected because the ML algorithm picks on your zip code, which embodies biases about one's socioeconomic background. Your resume might be ranked lower because the ranking system employers use picks on the spelling of your name. Your mortgage might get a higher interest rate because it relies partially on credit scores, which favor the rich and punish the poor. Other examples of ML biases in the real world are in predictive policing algorithms, personality tests administered by potential employers, and college rankings.

In 2019, "Berkeley researchers found that both face-to-face and online lenders rejected a total of 1.3 million creditworthy Black and Latino applicants between 2008 and 2015." When the researchers "used the income and credit scores of the rejected applications but deleted the race identifiers, the mortgage application was accepted."[25] For even more galling examples, I recommend Cathy O'Neil's *Weapons of Math Destruction*.[26]

ML algorithms don't predict the future, but encode the past, thus perpetuating the biases in the data and more. When ML algorithms are deployed at scale, they can discriminate against people at scale. If a human operator might only make sweeping judgments about a few individuals at a time, an ML algorithm can make sweeping judgments about millions in split seconds. This can especially hurt members of minority groups because misclassification on them could only have a minor effect on models' overall performance metrics.

If an algorithm can already make correct predictions on 98% of the population, and improving the predictions on the other 2% would incur multiples of cost, some companies might, unfortunately, choose not to do it. During a McKinsey & Company research study in 2019, only 13% of the large companies surveyed said they are taking steps to mitigate risks to equity and fairness, such as algorithmic bias and discrimination.[27] However, this is changing rapidly. We'll cover fairness and other aspects of responsible AI in Chapter 11.

Interpretability

In early 2020, the Turing Award winner Professor Geoffrey Hinton proposed a heatedly debated question about the importance of interpretability in ML systems. "Suppose you have cancer and you have to choose between a black box AI surgeon

25 Khristopher J. Brooks, "Disparity in Home Lending Costs Minorities Millions, Researchers Find," *CBS News*, November 15, 2019, *https://oreil.ly/UiHUB*.

26 Cathy O'Neil, *Weapons of Math Destruction* (New York: Crown Books, 2016).

27 Stanford University Human-Centered Artificial Intelligence (HAI), *The 2019 AI Index Report*, 2019, *https://oreil.ly/xs8mG*.

that cannot explain how it works but has a 90% cure rate and a human surgeon with an 80% cure rate. Do you want the AI surgeon to be illegal?"[28]

A couple of weeks later, when I asked this question to a group of 30 technology executives at public nontech companies, only half of them would want the highly effective but unable-to-explain AI surgeon to operate on them. The other half wanted the human surgeon.

While most of us are comfortable with using a microwave without understanding how it works, many don't feel the same way about AI yet, especially if that AI makes important decisions about their lives.

Since most ML research is still evaluated on a single objective, model performance, researchers aren't incentivized to work on model interpretability. However, interpretability isn't just optional for most ML use cases in the industry, but a requirement.

First, interpretability is important for users, both business leaders and end users, to understand why a decision is made so that they can trust a model and detect potential biases mentioned previously.[29] Second, it's important for developers to be able to debug and improve a model.

Just because interpretability is a requirement doesn't mean everyone is doing it. As of 2019, only 19% of large companies are working to improve the explainability of their algorithms.[30]

Discussion

Some might argue that it's OK to know only the academic side of ML because there are plenty of jobs in research. The first part—it's OK to know only the academic side of ML—is true. The second part is false.

While it's important to pursue pure research, most companies can't afford it unless it leads to short-term business applications. This is especially true now that the research community took the "bigger, better" approach. Oftentimes, new models require a massive amount of data and tens of millions of dollars in compute alone.

As ML research and off-the-shelf models become more accessible, more people and organizations would want to find applications for them, which increases the demand for ML in production.

The vast majority of ML-related jobs will be, and already are, in productionizing ML.

---

28 Tweet by Geoffrey Hinton (@geoffreyhinton), February 20, 2020, *https://oreil.ly/KdfD8*.

29 For certain use cases in certain countries, users have a "right to explanation": a right to be given an explanation for an output of the algorithm.

30 Stanford HAI, *The 2019 AI Index Report*.

Machine Learning Systems Versus Traditional Software

Since ML is part of software engineering (SWE), and software has been successfully used in production for more than half a century, some might wonder why we don't just take tried-and-true best practices in software engineering and apply them to ML.

That's an excellent idea. In fact, ML production would be a much better place if ML experts were better software engineers. Many traditional SWE tools can be used to develop and deploy ML applications.

However, many challenges are unique to ML applications and require their own tools. In SWE, there's an underlying assumption that code and data are separated. In fact, in SWE, we want to keep things as modular and separate as possible (see the Wikipedia page on separation of concerns).

On the contrary, ML systems are part code, part data, and part artifacts created from the two. The trend in the last decade shows that applications developed with the most/best data win. Instead of focusing on improving ML algorithms, most companies will focus on improving their data. Because data can change quickly, ML applications need to be adaptive to the changing environment, which might require faster development and deployment cycles.

In traditional SWE, you only need to focus on testing and versioning your code. With ML, we have to test and version our data too, and that's the hard part. How to version large datasets? How to know if a data sample is good or bad for your system? Not all data samples are equal—some are more valuable to your model than others. For example, if your model has already trained on one million scans of normal lungs and only one thousand scans of cancerous lungs, a scan of a cancerous lung is much more valuable than a scan of a normal lung. Indiscriminately accepting all available data might hurt your model's performance and even make it susceptible to data poisoning attacks.[31]

The size of ML models is another challenge. As of 2022, it's common for ML models to have hundreds of millions, if not billions, of parameters, which requires gigabytes of random-access memory (RAM) to load them into memory. A few years from now, a billion parameters might seem quaint—like, "Can you believe the computer that sent men to the moon only had 32 MB of RAM?"

However, for now, getting these large models into production, especially on edge devices,[32] is a massive engineering challenge. Then there is the question of how to get these models to run fast enough to be useful. An autocompletion model is useless if

31 Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song, "Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning," *arXiv*, December 15, 2017, *https://oreil.ly/OkAjb*.

32 We'll cover edge devices in Chapter 7.

the time it takes to suggest the next character is longer than the time it takes for you to type.

Monitoring and debugging these models in production is also nontrivial. As ML models get more complex, coupled with the lack of visibility into their work, it's hard to figure out what went wrong or be alerted quickly enough when things go wrong.

The good news is that these engineering challenges are being tackled at a breakneck pace. Back in 2018, when the Bidirectional Encoder Representations from Transformers (BERT) paper first came out, people were talking about how BERT was too big, too complex, and too slow to be practical. The pretrained large BERT model has 340 million parameters and is 1.35 GB.[33] Fast-forward two years later, BERT and its variants were already used in almost every English search on Google.[34]

Summary

This opening chapter aimed to give readers an understanding of what it takes to bring ML into the real world. We started with a tour of the wide range of use cases of ML in production today. While most people are familiar with ML in consumer-facing applications, the majority of ML use cases are for enterprise. We also discussed when ML solutions would be appropriate. Even though ML can solve many problems very well, it can't solve all the problems and it's certainly not appropriate for all the problems. However, for problems that ML can't solve, it's possible that ML can be one part of the solution.

This chapter also highlighted the differences between ML in research and ML in production. The differences include the stakeholder involvement, computational priority, the properties of data used, the gravity of fairness issues, and the requirements for interpretability. This section is the most helpful to those coming to ML production from academia. We also discussed how ML systems differ from traditional software systems, which motivated the need for this book.

ML systems are complex, consisting of many different components. Data scientists and ML engineers working with ML systems in production will likely find that focusing only on the ML algorithms part is far from enough. It's important to know about other aspects of the system, including the data stack, deployment, monitoring, maintenance, infrastructure, etc. This book takes a system approach to developing ML systems, which means that we'll consider all components of a system holistically instead of just looking at ML algorithms. We'll provide detail on what this holistic approach means in the next chapter.

---

33 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv*, October 11, 2018, *https://oreil.ly/TG3ZW*.

34 Google Search On, 2020, *https://oreil.ly/M7YjM*.

CHAPTER 2

Introduction to Machine Learning Systems Design

Now that we've walked through an overview of ML systems in the real world, we can get to the fun part of actually designing an ML system. To reiterate from the first chapter, ML systems design takes a system approach to MLOps, which means that we'll consider an ML system holistically to ensure that all the components—the business requirements, the data stack, infrastructure, deployment, monitoring, etc.—and their stakeholders can work together to satisfy the specified objectives and requirements.

We'll start the chapter with a discussion on objectives. Before we develop an ML system, we must understand why this system is needed. If this system is built for a business, it must be driven by business objectives, which will need to be translated into ML objectives to guide the development of ML models.

Once everyone is on board with the objectives for our ML system, we'll need to set out some requirements to guide the development of this system. In this book, we'll consider the four requirements: reliability, scalability, maintainability, and adaptability. We will then introduce the iterative process for designing systems to meet those requirements.

You might wonder: with all these objectives, requirements, and processes in place, can I finally start building my ML model yet? Not so soon! Before using ML algorithms to solve your problem, you first need to frame your problem into a task that ML can solve. We'll continue this chapter with how to frame your ML problems. The difficulty of your job can change significantly depending on how you frame your problem.

Because ML is a data-driven approach, a book on ML systems design will be amiss if it fails to discuss the importance of data in ML systems. The last part of this chapter touches on a debate that has consumed much of the ML literature in recent years: which is more important—data or intelligent algorithms?

Let's get started!

Business and ML Objectives

We first need to consider the objectives of the proposed ML projects. When working on an ML project, data scientists tend to care about the ML objectives: the metrics they can measure about the performance of their ML models such as accuracy, F1 score, inference latency, etc. They get excited about improving their model's accuracy from 94% to 94.2% and might spend a ton of resources—data, compute, and engineering time—to achieve that.

But the truth is: most companies don't care about the fancy ML metrics. They don't care about increasing a model's accuracy from 94% to 94.2% unless it moves some business metrics. A pattern I see in many short-lived ML projects is that the data scientists become too focused on hacking ML metrics without paying attention to business metrics. Their managers, however, only care about business metrics and, after failing to see how an ML project can help push their business metrics, kill the projects prematurely (and possibly let go of the data science team involved).[1]

So what metrics do companies care about? While most companies want to convince you otherwise, the sole purpose of businesses, according to the Nobel-winning economist Milton Friedman, is to maximize profits for shareholders.[2]

The ultimate goal of any project within a business is, therefore, to increase profits, either directly or indirectly: directly such as increasing sales (conversion rates) and cutting costs; indirectly such as higher customer satisfaction and increasing time spent on a website.

For an ML project to succeed within a business organization, it's crucial to tie the performance of an ML system to the overall business performance. What business performance metrics is the new ML system supposed to influence, e.g., the amount of ads revenue, the number of monthly active users?

1 Eugene Yan has a great post on how data scientists can understand the business intent and context of the projects they work on.

2 Milton Friedman, "A Friedman Doctrine—The Social Responsibility of Business Is to Increase Its Profits," *New York Times Magazine,* September 13, 1970, *https://oreil.ly/Fmbem*.

Imagine that you work for an ecommerce site that cares about purchase-through rate and you want to move your recommender system from batch prediction to online prediction.[3] You might reason that online prediction will enable recommendations more relevant to users right now, which can lead to a higher purchase-through rate. You can even do an experiment to show that online prediction can improve your recommender system's predictive accuracy by *X*% and, historically on your site, each percent increase in the recommender system's predictive accuracy led to a certain increase in purchase-through rate.

One of the reasons why predicting ad click-through rates and fraud detection are among the most popular use cases for ML today is that it's easy to map ML models' performance to business metrics: every increase in click-through rate results in actual ad revenue, and every fraudulent transaction stopped results in actual money saved.

Many companies create their own metrics to map business metrics to ML metrics. For example, Netflix measures the performance of their recommender system using *take-rate*: the number of quality plays divided by the number of recommendations a user sees.[4] The higher the take-rate, the better the recommender system. Netflix also put a recommender system's take-rate in the context of their other business metrics like total streaming hours and subscription cancellation rate. They found that a higher take-rate also results in higher total streaming hours and lower subscription cancellation rates.[5]

The effect of an ML project on business objectives can be hard to reason about. For example, an ML model that gives customers more personalized solutions can make them happier, which makes them spend more money on your services. The same ML model can also solve their problems faster, which makes them spend less money on your services.

To gain a definite answer on the question of how ML metrics influence business metrics, experiments are often needed. Many companies do that with experiments like A/B testing and choose the model that leads to better business metrics, regardless of whether this model has better ML metrics.

3 We'll cover batch prediction and online prediction in Chapter 7.

4 Ashok Chandrashekar, Fernando Amat, Justin Basilico, and Tony Jebara, "Artwork Personalization at Netflix," *Netflix Technology Blog*, December 7, 2017, *https://oreil.ly/UEDmw*.

5 Carlos A. Gomez-Uribe and Neil Hunt, "The Netflix Recommender System: Algorithms, Business Value, and Innovation," *ACM Transactions on Management Information Systems* 6, no. 4 (January 2016): 13, *https://oreil.ly/JkEPB*.

Yet, even rigorous experiments might not be sufficient to understand the relationship between an ML model's outputs and business metrics. Imagine you work for a cybersecurity company that detects and stops security threats, and ML is just a component in their complex process. An ML model is used to detect anomalies in the traffic pattern. These anomalies then go through a logic set (e.g., a series of if-else statements) that categorizes whether they constitute potential threats. These potential threats are then reviewed by security experts to determine whether they are actual threats. Actual threats will then go through another, different process aimed at stopping them. When this process fails to stop a threat, it might be impossible to figure out whether the ML component has anything to do with it.

Many companies like to say that they use ML in their systems because "being AI-powered" alone already helps them attract customers, regardless of whether the AI part actually does anything useful.[6]

When evaluating ML solutions through the business lens, it's important to be realistic about the expected returns. Due to all the hype surrounding ML, generated both by the media and by practitioners with a vested interest in ML adoption, some companies might have the notion that ML can magically transform their businesses overnight.

Magically: possible. Overnight: no.

There are many companies that have seen payoffs from ML. For example, ML has helped Google search better, sell more ads at higher prices, improve translation quality, and build better Android applications. But this gain hardly happened overnight. Google has been investing in ML for decades.

Returns on investment in ML depend a lot on the maturity stage of adoption. The longer you've adopted ML, the more efficient your pipeline will run, the faster your development cycle will be, the less engineering time you'll need, and the lower your cloud bills will be, which all lead to higher returns. According to a 2020 survey by Algorithmia, among companies that are more sophisticated in their ML adoption (having had models in production for over five years), almost 75% can deploy a model in under 30 days. Among those just getting started with their ML pipeline, 60% take over 30 days to deploy a model (see Figure 2-1).[7]

6 Parmy Olson, "Nearly Half of All 'AI Startups' Are Cashing In on Hype," *Forbes*, March 4, 2019, *https://oreil.ly/w5kOr*.

7 "2020 State of Enterprise Machine Learning," Algorithmia, 2020, *https://oreil.ly/FlIV1*.

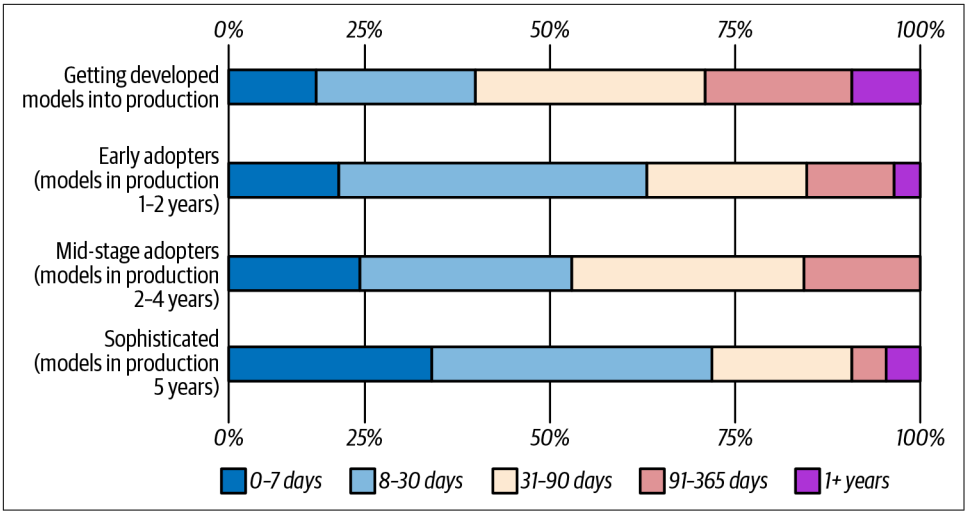

Figure 2-1. How long it takes for a company to bring a model to production is proportional to how long it has used ML. Source: Adapted from an image by Algorithmia

Requirements for ML Systems

We can't say that we've successfully built an ML system without knowing what requirements the system has to satisfy. The specified requirements for an ML system vary from use case to use case. However, most systems should have these four characteristics: reliability, scalability, maintainability, and adaptability. We'll walk through each of these concepts in detail. Let's take a closer look at reliability first.

Reliability

The system should continue to perform the correct function at the desired level of performance even in the face of adversity (hardware or software faults, and even human error).

"Correctness" might be difficult to determine for ML systems. For example, your system might call the predict function—e.g., `model.predict()`—correctly, but the predictions are wrong. How do we know if a prediction is wrong if we don't have ground truth labels to compare it with?

With traditional software systems, you often get a warning, such as a system crash or runtime error or 404. However, ML systems can fail silently. End users don't even know that the system has failed and might have kept on using it as if it were working. For example, if you use Google Translate to translate a sentence into a language you don't know, it might be very hard for you to tell even if the translation is wrong. We'll discuss how ML systems fail in production in Chapter 8.

Scalability

There are multiple ways an ML system can grow. It can grow in complexity. Last year you used a logistic regression model that fit into an Amazon Web Services (AWS) free tier instance with 1 GB of RAM, but this year, you switched to a 100-million-parameter neural network that requires 16 GB of RAM to generate predictions.

Your ML system can grow in traffic volume. When you started deploying an ML system, you only served 10,000 prediction requests daily. However, as your company's user base grows, the number of prediction requests your ML system serves daily fluctuates between 1 million and 10 million.

An ML system might grow in ML model count. Initially, you might have only one model for one use case, such as detecting the trending hashtags on a social network site like Twitter. However, over time, you want to add more features to this use case, so you'll add one more to filter out NSFW (not safe for work) content and another model to filter out tweets generated by bots. This growth pattern is especially common in ML systems that target enterprise use cases. Initially, a startup might serve only one enterprise customer, which means this startup only has one model. However, as this startup gains more customers, they might have one model for each customer. A startup I worked with had 8,000 models in production for their 8,000 enterprise customers.

Whichever way your system grows, there should be reasonable ways of dealing with that growth. When talking about scalability most people think of resource scaling, which consists of up-scaling (expanding the resources to handle growth) and down-scaling (reducing the resources when not needed).[8]

For example, at peak, your system might require 100 GPUs (graphics processing units). However, most of the time, it needs only 10 GPUs. Keeping 100 GPUs up all the time can be costly, so your system should be able to scale down to 10 GPUs.

An indispensable feature in many cloud services is autoscaling: automatically scaling up and down the number of machines depending on usage. This feature can be tricky to implement. Even Amazon fell victim to this when their autoscaling feature failed on Prime Day, causing their system to crash. An hour of downtime was estimated to cost Amazon between $72 million and $99 million.[9]

8 Up-scaling and down-scaling are two aspects of "scaling out," which is different from "scaling up." Scaling out is adding more equivalently functional components in parallel to spread out a load. Scaling up is making a component larger or faster to handle a greater load (Leah Schoeb, "Cloud Scalability: Scale Up vs Scale Out," *Turbonomic Blog*, March 15, 2018, *https://oreil.ly/CFPtb*).

9 Sean Wolfe, "Amazon's One Hour of Downtime on Prime Day May Have Cost It up to $100 Million in Lost Sales," *Business Insider*, July 19, 2018, *https://oreil.ly/VBezI*.

However, handling growth isn't just resource scaling, but also artifact management. Managing one hundred models is very different from managing one model. With one model, you can, perhaps, manually monitor this model's performance and manually update the model with new data. Since there's only one model, you can just have a file that helps you reproduce this model whenever needed. However, with one hundred models, both the monitoring and retraining aspect will need to be automated. You'll need a way to manage the code generation so that you can adequately reproduce a model when you need to.

Because scalability is such an important topic throughout the ML project workflow, we'll discuss it in different parts of the book. Specifically, we'll touch on the resource scaling aspect in the section "Distributed Training" on page 168, the section "Model optimization" on page 216, and the section "Resource Management" on page 311. We'll discuss the artifact management aspect in the section "Experiment Tracking and Versioning" on page 162 and the section "Development Environment" on page 302.

Maintainability

There are many people who will work on an ML system. They are ML engineers, DevOps engineers, and subject matter experts (SMEs). They might come from very different backgrounds, with very different programming languages and tools, and might own different parts of the process.

It's important to structure your workloads and set up your infrastructure in such a way that different contributors can work using tools that they are comfortable with, instead of one group of contributors forcing their tools onto other groups. Code should be documented. Code, data, and artifacts should be versioned. Models should be sufficiently reproducible so that even when the original authors are not around, other contributors can have sufficient contexts to build on their work. When a problem occurs, different contributors should be able to work together to identify the problem and implement a solution without finger-pointing.

We'll go more into this in the section "Team Structure" on page 334.

Adaptability

To adapt to shifting data distributions and business requirements, the system should have some capacity for both discovering aspects for performance improvement and allowing updates without service interruption.

Because ML systems are part code, part data, and data can change quickly, ML systems need to be able to evolve quickly. This is tightly linked to maintainability. We'll discuss changing data distributions in the section "Data Distribution Shifts" on page 237, and how to continually update your model with new data in the section "Continual Learning" on page 264.

Iterative Process

Developing an ML system is an iterative and, in most cases, never-ending process.[10] Once a system is put into production, it'll need to be continually monitored and updated.

Before deploying my first ML system, I thought the process would be linear and straightforward. I thought all I had to do was to collect data, train a model, deploy that model, and be done. However, I soon realized that the process looks more like a cycle with a lot of back and forth between different steps.

For example, here is one workflow that you might encounter when building an ML model to predict whether an ad should be shown when users enter a search query:[11]

1. Choose a metric to optimize. For example, you might want to optimize for impressions—the number of times an ad is shown.
2. Collect data and obtain labels.
3. Engineer features.
4. Train models.
5. During error analysis, you realize that errors are caused by the wrong labels, so you relabel the data.
6. Train the model again.
7. During error analysis, you realize that your model always predicts that an ad shouldn't be shown, and the reason is because 99.99% of the data you have have NEGATIVE labels (ads that shouldn't be shown). So you have to collect more data of ads that should be shown.

10 Which, as an early reviewer pointed out, is a property of traditional software.

11 Praying and crying not featured, but present through the entire process.

8. Train the model again.
9. The model performs well on your existing test data, which is by now two months old. However, it performs poorly on the data from yesterday. Your model is now stale, so you need to update it on more recent data.
10. Train the model again.
11. Deploy the model.
12. The model seems to be performing well, but then the businesspeople come knocking on your door asking why the revenue is decreasing. It turns out the ads are being shown, but few people click on them. So you want to change your model to optimize for ad click-through rate instead.
13. Go to step 1.

Figure 2-2 shows an oversimplified representation of what the iterative process for developing ML systems in production looks like from the perspective of a data scientist or an ML engineer. This process looks different from the perspective of an ML platform engineer or a DevOps engineer, as they might not have as much context into model development and might spend a lot more time on setting up infrastructure.

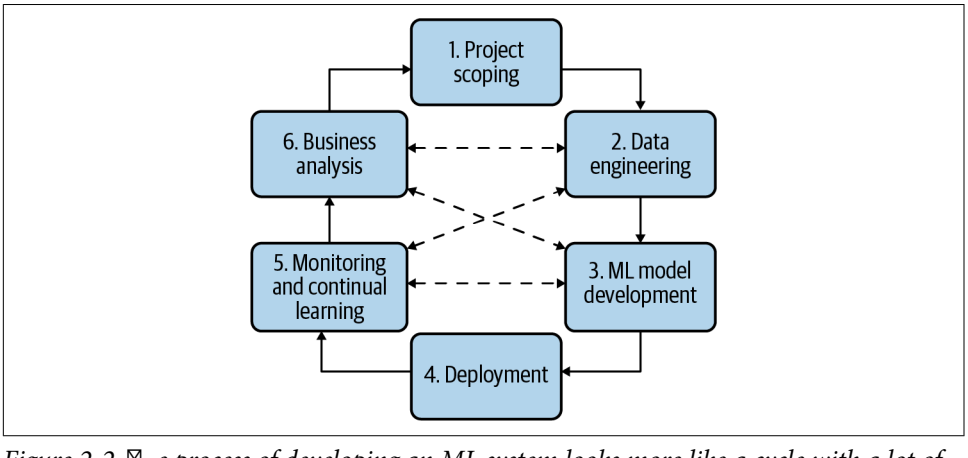

Figure 2-2. The process of developing an ML system looks more like a cycle with a lot of back and forth between steps

Later chapters will dive deeper into what each of these steps requires in practice. Here, let's take a brief look at what they mean:

Step 1. Project scoping
: A project starts with scoping the project, laying out goals, objectives, and constraints. Stakeholders should be identified and involved. Resources should be estimated and allocated. We already discussed different stakeholders and some of the foci for ML projects in production in Chapter 1. We also already discussed how to scope an ML project in the context of a business earlier in this chapter. We'll discuss how to organize teams to ensure the success of an ML project in Chapter 11.

Step 2. Data engineering
: A vast majority of ML models today learn from data, so developing ML models starts with engineering data. In Chapter 3, we'll discuss the fundamentals of data engineering, which covers handling data from different sources and formats. With access to raw data, we'll want to curate training data out of it by sampling and generating labels, which is discussed in Chapter 4.

Step 3. ML model development
: With the initial set of training data, we'll need to extract features and develop initial models leveraging these features. This is the stage that requires the most ML knowledge and is most often covered in ML courses. In Chapter 5, we'll discuss feature engineering. In Chapter 6, we'll discuss model selection, training, and evaluation.

Step 4. Deployment
: After a model is developed, it needs to be made accessible to users. Developing an ML system is like writing—you will never reach the point when your system is done. But you do reach the point when you have to put your system out there. We'll discuss different ways to deploy an ML model in Chapter 7.

Step 5. Monitoring and continual learning
: Once in production, models need to be monitored for performance decay and maintained to be adaptive to changing environments and changing requirements. This step will be discussed in Chapters 8 and 9.

Step 6. Business analysis
: Model performance needs to be evaluated against business goals and analyzed to generate business insights. These insights can then be used to eliminate unproductive projects or scope out new projects. This step is closely related to the first step.

Framing ML Problems

Imagine you're an ML engineering tech lead at a bank that targets millennial users. One day, your boss hears about a rival bank that uses ML to speed up their customer service support that supposedly helps the rival bank process their customer requests two times faster. He orders your team to look into using ML to speed up your customer service support too.

Slow customer support is a problem, but it's not an ML problem. An ML problem is defined by inputs, outputs, and the objective function that guides the learning process—none of these three components are obvious from your boss's request. It's your job, as a seasoned ML engineer, to use your knowledge of what problems ML can solve to frame this request as an ML problem.

Upon investigation, you discover that the bottleneck in responding to customer requests lies in routing customer requests to the right department among four departments: accounting, inventory, HR (human resources), and IT. You can alleviate this bottleneck by developing an ML model to predict which of these four departments a request should go to. This makes it a classification problem. The input is the customer request. The output is the department the request should go to. The objective function is to minimize the difference between the predicted department and the actual department.

We'll discuss extensively how to extract features from raw data to input into your ML model in Chapter 5. In this section, we'll focus on two aspects: the output of your model and the objective function that guides the learning process.

Types of ML Tasks

The output of your model dictates the task type of your ML problem. The most general types of ML tasks are classification and regression. Within classification, there are more subtypes, as shown in Figure 2-3. We'll go over each of these task types.

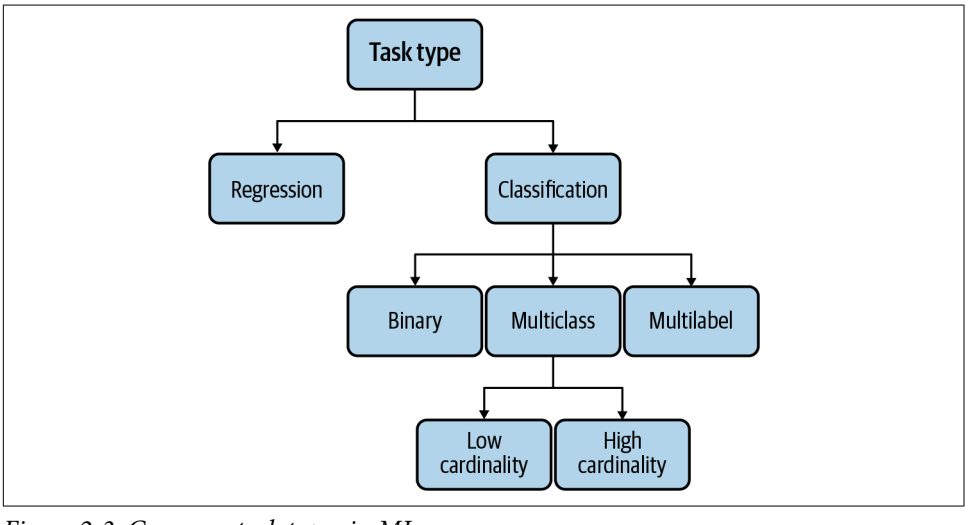

Figure 2-3. Common task types in ML

Classi□cation versus regression

Classification models classify inputs into different categories. For example, you want to classify each email to be either spam or not spam. Regression models output a continuous value. An example is a house prediction model that outputs the price of a given house.

A regression model can easily be framed as a classification model and vice versa. For example, house prediction can become a classification task if we quantize the house prices into buckets such as under $100,000, $100,000–$200,000, $200,000–$500,000, and so forth and predict the bucket the house should be in.

The email classification model can become a regression model if we make it output values between 0 and 1, and decide on a threshold to determine which values should be SPAM (for example, if the value is above 0.5, the email is spam), as shown in Figure 2-4.

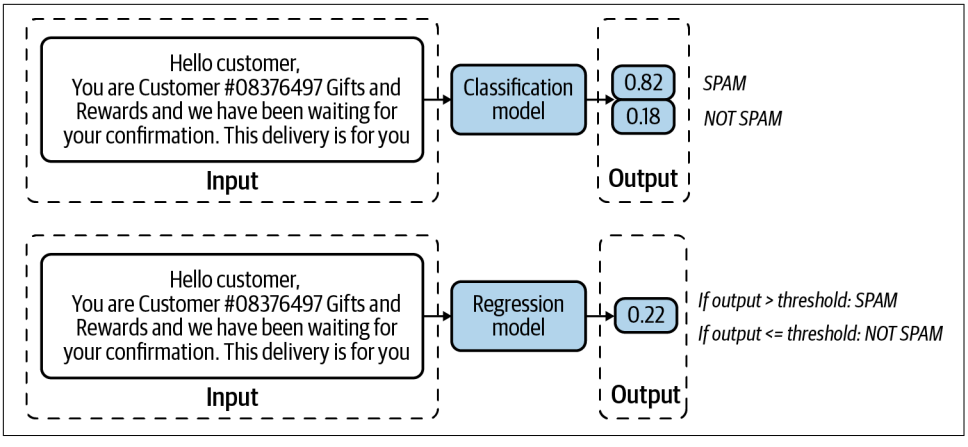

Figure 2-4. The email classification task can also be framed as a regression task

Binary versus multiclass classification

Within classification problems, the fewer classes there are to classify, the simpler the problem is. The simplest is *binary classification*, where there are only two possible classes. Examples of binary classification include classifying whether a comment is toxic, whether a lung scan shows signs of cancer, whether a transaction is fraudulent. It's unclear whether this type of problem is common in the industry because they are common in nature or simply because ML practitioners are most comfortable handling them.

When there are more than two classes, the problem becomes *multiclass classification*. Dealing with binary classification problems is much easier than dealing with multiclass problems. For example, calculating F1 and visualizing confusion matrices are a lot more intuitive when there are only two classes.

When the number of classes is high, such as disease diagnosis where the number of diseases can go up to thousands or product classifications where the number of products can go up to tens of thousands, we say the classification task has *high cardinality*. High cardinality problems can be very challenging. The first challenge is in data collection. In my experience, ML models typically need at least 100 examples for each class to learn to classify that class. So if you have 1,000 classes, you already need at least 100,000 examples. The data collection can be especially difficult for rare classes. When you have thousands of classes, it's likely that some of them are rare.

When the number of classes is large, hierarchical classification might be useful. In hierarchical classification, you have a classifier to first classify each example into one of the large groups. Then you have another classifier to classify this example into one of the subgroups. For example, for product classification, you can first classify each product into one of the four main categories: electronics, home and kitchen, fashion, or pet supplies. After a product has been classified into a category, say fashion, you can use another classifier to put this product into one of the subgroups: shoes, shirts, jeans, or accessories.

Multiclass versus multilabel classification

In both binary and multiclass classification, each example belongs to exactly one class. When an example can belong to multiple classes, we have a *multilabel classification* problem. For example, when building a model to classify articles into four topics—tech, entertainment, finance, and politics—an article can be in both tech and finance.

There are two major approaches to multilabel classification problems. The first is to treat it as you would a multiclass classification. In multiclass classification, if there are four possible classes [tech, entertainment, finance, politics] and the label for an example is entertainment, you represent this label with the vector [0, 1, 0, 0]. In multilabel classification, if an example has both labels entertainment and finance, its label will be represented as [0, 1, 1, 0].

The second approach is to turn it into a set of binary classification problems. For the article classification problem, you can have four models corresponding to four topics, each model outputting whether an article is in that topic or not.

Out of all task types, multilabel classification is usually the one that I've seen companies having the most problems with. Multilabel means that the number of classes an example can have varies from example to example. First, this makes it difficult for label annotation since it increases the label multiplicity problem that we discuss in Chapter 4. For example, an annotator might believe an example belongs to two classes while another annotator might believe the same example to belong in only one class, and it might be difficult resolving their disagreements.

Second, this varying number of classes makes it hard to extract predictions from raw probability. Consider the same task of classifying articles into four topics. Imagine that, given an article, your model outputs this raw probability distribution: [0.45, 0.2, 0.02, 0.33]. In the multiclass setting, when you know that an example can belong to only one category, you simply pick the category with the highest probability, which is 0.45 in this case. In the multilabel setting, because you don't know how many categories an example can belong to, you might pick the two highest probability categories (corresponding to 0.45 and 0.33) or three highest probability categories (corresponding to 0.45, 0.2, and 0.33).

Multiple ways to frame a problem

Changing the way you frame your problem might make your problem significantly harder or easier. Consider the task of predicting what app a phone user wants to use next. A naive setup would be to frame this as a multiclass classification task—use the user's and environment's features (user demographic information, time, location, previous apps used) as input, and output a probability distribution for every single app on the user's phone. Let *N* be the number of apps you want to consider recommending to a user. In this framing, for a given user at a given time, there is only one prediction to make, and the prediction is a vector of the size *N*. This setup is visualized in Figure 2-5.

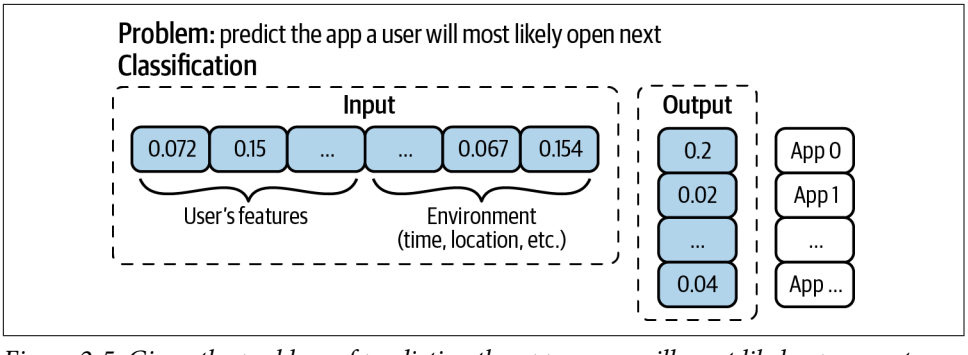

Figure 2-5. Given the problem of predicting the app a user will most likely open next, you can frame it as a classification problem. The input is the user's features and environment's features. The output is a distribution over all apps on the phone.

This is a bad approach because whenever a new app is added, you might have to retrain your model from scratch, or at least retrain all the components of your model whose number of parameters depends on *N*. A better approach is to frame this as a regression task. The input is the user's, the environment's, and the app's features. The output is a single value between 0 and 1; the higher the value, the more likely the user will open the app given the context. In this framing, for a given user at a given time, there are *N* predictions to make, one for each app, but each prediction is just a number. This improved setup is visualized in Figure 2-6.

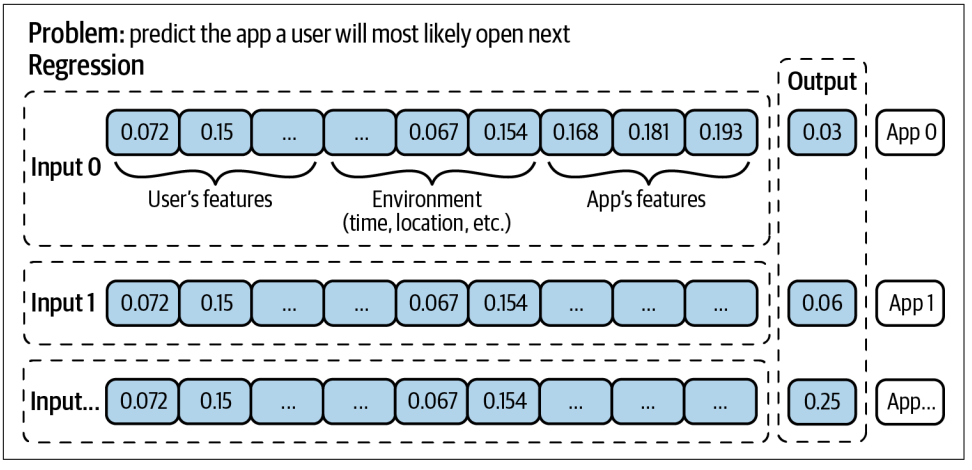

Figure 2-6. Given the problem of predicting the app a user will most likely open next, you can frame it as a regression problem. The input is the user's features, environment's features, and an app's features. The output is a single value between 0 and 1 denoting how likely the user will be to open the app given the context.

In this new framing, whenever there's a new app you want to consider recommending to a user, you simply need to use new inputs with this new app's feature instead of having to retrain your model or part of your model from scratch.

Objective Functions

To learn, an ML model needs an objective function to guide the learning process.[12] An objective function is also called a loss function, because the objective of the learning process is usually to minimize (or optimize) the loss caused by wrong predictions. For supervised ML, this loss can be computed by comparing the model's outputs with the ground truth labels using a measurement like root mean squared error (RMSE) or cross entropy.

To illustrate this point, let's again go back to the previous task of classifying articles into four topics [tech, entertainment, finance, politics]. Consider an article that belongs to the politics class, e.g., its ground truth label is [0, 0, 0, 1]. Imagine that, given this article, your model outputs this raw probability distribution: [0.45, 0.2, 0.02, 0.33]. The cross entropy loss of this model, given this example, is the cross entropy of [0.45, 0.2, 0.02, 0.33] relative to [0, 0, 0, 1]. In Python, you can calculate cross entropy with the following code:

12 Note that objective functions are mathematical functions, which are different from the business and ML objectives we discussed earlier in this chapter.