O'REILLY®

Building Machine Learning Powered Applications

Going from Idea to Product

Emmanuel Ameisen

O'REILLY®

Building Machine Learning Powered Applications

Learn the skills necessary to design, build, and deploy applications powered by machine learning (ML). Through the course of this hands-on book, you'll build an example ML-driven application from initial idea to deployed product. Data scientists, software engineers, and product managers—including experienced practitioners and novices alike—will learn the tools, best practices, and challenges involved in building a real-world ML application step by step.

Author Emmanuel Ameisen, an experienced data scientist who led an AI education program, demonstrates practical ML concepts using code snippets, illustrations, screenshots, and interviews with industry leaders. Part I teaches you how to plan an ML application and measure success. Part II explains how to build a working ML model. Part III demonstrates ways to improve the model until it fulfills your original vision. Part IV covers deployment and monitoring strategies.

This book will help you:

- Define your product goal and set up a machine learning problem
- Build your first end-to-end pipeline quickly and acquire an initial dataset
- Train and evaluate your ML models and address performance bottlenecks
- Deploy and monitor your models in a production environment

"So many books about machine learning skip the hardest parts: refining the problem, debugging models, and deploying to customers. But this book focuses on them so you can move your projects from an idea to making an impact."

—Alexander Gude
Data Scientist at Intuit

Emmanuel Ameisen, a machine learning engineer at Stripe, implemented and deployed predictive analytics and machine learning solutions for Local Motion and Zipcar. Recently, he led Insight Data Science's AI program, directing more than a hundred machine learning projects. Emmanuel holds graduate degrees in artificial intelligence, computer engineering, and management from three of France's top schools.

AI & SEMANTICS

US $59.99 CAN $79.99

ISBN: 978-1-492-04511-3

Twitter: @oreillymedia
facebook.com/oreilly

Building Machine Learning Powered Applications

Going from Idea to Product

Emmanuel Ameisen

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Building Machine Learning Powered Applications
by Emmanuel Ameisen

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: Jonathan Hassell
Development Editor: Melissa Potter
Production Editor: Deborah Baker
Copyeditor: Kim Wimpsett
Proofreader: Christina Edwards

Indexer: Judith McConville
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

February 2020: First Edition

Revision History for the First Edition
2020-01-17: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492045113* for release details.

978-1-492-04511-3

[LSI]

Table of Contents

Part III. Iterate on Models

Preface

The Goal of Using Machine Learning Powered Applications

Over the past decade, machine learning (ML) has increasingly been used to power a variety of products such as automated support systems, translation services, recommendation engines, fraud detection models, and many, many more.

Surprisingly, there aren't many resources available to teach engineers and scientists how to build such products. Many books and classes will teach how to train ML models or how to build software projects, but few blend both worlds to teach how to build practical applications that are powered by ML.

Deploying ML as part of an application requires a blend of creativity, strong engineering practices, and an analytical mindset. ML products are notoriously challenging to build because they require much more than simply training a model on a dataset. Choosing the right ML approach for a given feature, analyzing model errors and data quality issues, and validating model results to guarantee product quality are all challenging problems that are at the core of the ML building process.

This book goes through every step of this process and aims to help you accomplish each of them by sharing a mix of methods, code examples, and advice from me and other experienced practitioners. We'll cover the practical skills required to design, build, and deploy ML–powered applications. The goal of this book is to help you succeed at every part of the ML process.

Use ML to Build Practical Applications

If you regularly read ML papers and corporate engineering blogs, you may feel overwhelmed by the combination of linear algebra equations and engineering terms. The hybrid nature of the field leads many engineers and scientists who could contribute their diverse expertise to feel intimidated by the field of ML. Similarly, entrepreneurs

and product leaders often struggle to tie together their ideas for a business with what is possible with ML today (and what may be possible tomorrow).

This book covers the lessons I have learned working on data teams at multiple companies and helping hundreds of data scientists, software engineers, and product managers build applied ML projects through my work leading the artificial intelligence program at Insight Data Science.

The goal of this book is to share a step-by-step practical guide to building ML–powered applications. It is practical and focuses on concrete tips and methods to help you prototype, iterate, and deploy models. Because it spans a wide range of topics, we will go into only as much detail as is needed at each step. Whenever possible, I will provide resources to help you dive deeper into the topics covered if you so desire.

Important concepts are illustrated with practical examples, including a case study that will go from idea to deployed model by the end of the book. Most examples will be accompanied by illustrations, and many will contain code. All of the code used in this book can be found in the book's companion GitHub repository.

Because this book focuses on describing the process of ML, each chapter builds upon concepts defined in earlier ones. For this reason, I recommend reading it in order so that you can understand how each successive step fits into the entire process. If you are looking to explore a subset of the process of ML, you might be better served with a more specialized book. If that is the case, I've shared a few recommendations.

Additional Resources

- If you'd like to know ML well enough to write your own algorithms from scratch, I recommend *Data Science from Scratch*, by Joel Grus. If the theory of deep learning is what you are after, the textbook *Deep Learning* (MIT Press), by Ian Goodfellow, Yoshua Bengio, and Aaron Courville, is a comprehensive resource.
- If you are wondering how to train models efficiently and accurately on specific datasets, Kaggle and fast.ai are great places to look.
- If you'd like to learn how to build scalable applications that need to process a lot of data, I recommend looking at *Designing Data-Intensive Applications* (O'Reilly), by Martin Kleppmann.

If you have coding experience and some basic ML knowledge and want to build ML–driven products, this book will guide you through the entire process from product idea to shipped prototype. If you already work as a data scientist or ML engineer, this book will add new techniques to your ML development tool. If you do not know how to code but collaborate with data scientists, this book can help you understand the process of ML, as long as you are willing to skip some of the in-depth code examples.

Let's start by diving deeper into the meaning of practical ML.

Practical ML

For the purpose of this introduction, think of ML as the process of leveraging patterns in data to automatically tune algorithms. This is a general definition, so you will not be surprised to hear that many applications, tools, and services are starting to integrate ML at the core of the way they function.

Some of these tasks are user-facing, such as search engines, recommendations on social platforms, translation services, or systems that automatically detect familiar faces in photographs, follow instructions from voice commands, or attempt to provide useful suggestions to finish a sentence in an email.

Some work in less visible ways, silently filtering spam emails and fraudulent accounts, serving ads, predicting future usage patterns to efficiently allocate resources, or experimenting with personalizing website experiences for each user.

Many products currently leverage ML, and even more could do so. Practical ML refers to the task of identifying practical problems that could benefit from ML and delivering a successful solution to these problems. Going from a high-level product goal to ML–powered results is a challenging task that this book tries to help you to accomplish.

Some ML courses will teach students about ML methods by providing a dataset and having them train a model on them, but training an algorithm on a dataset is a small part of the ML process. Compelling ML–powered products rely on more than an aggregate accuracy score and are the results of a long process. This book will start from ideation and continue all the way through to production, illustrating every step on an example application. We will share tools, best practices, and common pitfalls learned from working with applied teams that are deploying these kinds of systems every day.

What This Book Covers

To cover the topic of building applications powered by ML, the focus of this book is concrete and practical. In particular, this book aims to illustrate the whole process of building ML–powered applications.

To do so, I will first describe methods to tackle each step in the process. Then, I will illustrate these methods using an example project as a case study. The book also contains many practical examples of ML in industry and features interviews with professionals who have built and maintained production ML models.

The entire process of ML

To successfully serve an ML product to users, you need to do more than simply train a model. You need to thoughtfully *translate* your product need to an ML problem,

gather adequate data, efficiently *iterate* in between models, *validate* your results, and *deploy* them in a robust manner.

Building a model often represents only a tenth of the total workload of an ML project. Mastering the entire ML pipeline is crucial to successfully build projects, succeed at ML interviews, and be a top contributor on ML teams.

A technical, practical case study

While we won't be re-implementing algorithms from scratch in C, we will stay practical and technical by using libraries and tools providing higher-level abstractions. We will go through this book building an example ML application together, from the initial idea to the deployed product.

I will illustrate key concepts with code snippets when applicable, as well as figures describing our application. The best way to learn ML is by practicing it, so I encourage you to go through the book reproducing the examples and adapting them to build your own ML–powered application.

Real business applications

Throughout this book, I will include conversations and advice from ML leaders who have worked on data teams at tech companies such as StitchFix, Jawbone, and FigureEight. These discussions will cover practical advice garnered after building ML applications with millions of users and will correct some popular misconceptions about what makes data scientists and data science teams successful.

Prerequisites

This book assumes some familiarity with programming. I will mainly be using Python for technical examples and assume that the reader is familiar with the syntax. If you'd like to refresh your Python knowledge, I recommend *The Hitchhiker's Guide to Python* (O'Reilly), by Kenneth Reitz and Tanya Schlusser.

In addition, while I will define most ML concepts referred to in the book, I will not cover the inner workings of all ML algorithms used. Most of these algorithms are standard ML methods that are covered in introductory-level ML resources, such as the ones mentioned in "Additional Resources" on page x.

Our Case Study: ML–Assisted Writing

To concretely illustrate this idea, we will build an ML application together as we go through this book.

As a case study, I chose an application that can accurately illustrate the complexity of iterating and deploying ML models. I also wanted to cover a product that could pro-

duce value. This is why we will be implementing a *machine learning–powered writing assistant*.

Our goal is to build a system that will help users write better. In particular, we will aim to help people write better questions. This may seem like a very vague objective, and I will define it more clearly as we scope out the project, but it is a good example for a few key reasons.

Text data is everywhere
: Text data is abundantly available for most use cases you can think of and is core to many practical ML applications. Whether we are trying to better understand the reviews of our product, accurately categorize incoming support requests, or tailor our promotional messages to potential audiences, we will consume and produce text data.

Writing assistants are useful
: From Gmail's text prediction feature to Grammarly's smart spellchecker, ML–powered editors have proven that they can deliver value to users in a variety of ways. This makes it particularly interesting for us to explore how to build them from scratch.

ML–assisted writing is self-standing
: Many ML applications can function only when tightly integrated into a broader ecosystem, such as ETA prediction for ride-hailing companies, search and recommendation systems for online retailers, and ad bidding models. A text editor, however, even though it could benefit from being integrated into a document editing ecosystem, can prove valuable on its own and be exposed through a simple website.

Throughout the book, this project will allow us to highlight the challenges and associated solutions we suggest to build ML–powered applications.

The ML Process

The road from an idea to a deployed ML application is long and winding. After seeing many companies and individuals build such projects, I've identified four key successive stages, which will each be covered in a section of this book.

1. *Identifying the right ML approach:* The field of ML is broad and often proposes a multitude of ways to tackle a given product goal. The best approach for a given problem will depend on many factors such as success criteria, data availability, and task complexity. The goals of this stage are to set the right success criteria and to identify an adequate initial dataset and model choice.
2. *Building an initial prototype:* Start by building an end-to-end prototype before working on a model. This prototype should aim to tackle the product goal with

no ML involved and will allow you to determine how to best apply ML. Once a prototype is built, you should have an idea of whether you need ML, and you should be able to start gathering a dataset to train a model.

3. *Iterating on models:* Now that you have a dataset, you can train a model and evaluate its shortcomings. The goal of this stage is to repeatedly alternate between error analysis and implementation. Increasing the speed at which this iteration loop happens is the best way to increase ML development speed.
4. *Deployment and monitoring:* Once a model shows good performance, you should pick an adequate deployment option. Once deployed, models often fail in unexpected ways. The last two chapters of this book will cover methods to mitigate and monitor model errors.

There is a lot of ground to cover, so let's dive right in and start with Chapter 1!

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.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This element signifies a tip or suggestion.

This element signifies a general note.

This element indicates a warning or caution.

Using Code Examples

Supplemental code examples for this book are available for download at *https://oreil.ly/ml-powered-applications*.

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: *Building Machine Learning Powered Applications* by Emmanuel Ameisen (O'Reilly). Copyright 2020 Emmanuel Ameisen, 978-1-492-04511-3."

If you feel your use of code examples falls outside fair use or the permission given here, 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, conferences, 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, please visit *http://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)

You can access the web page for this book, where we list errata, examples, and any additional information, at *https://oreil.ly/Building_ML_Powered_Applications*.

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

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

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

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

The project of writing this book started as a consequence of my work mentoring Fellows and overseeing ML projects at Insight Data Science. For giving me the opportunity to lead this program and for encouraging me to write about the lessons learned doing so, I'd like to thank Jake Klamka and Jeremy Karnowski, respectively. I'd also like to thank the hundreds of Fellows I've worked with at Insight for allowing me to help them push the limits of what an ML project can look like.

Writing a book is a daunting task, and the O'Reilly staff helped make it more manageable every step of the way. In particular, I would like to thank my editor, Melissa Potter, who tirelessly provided guidance, suggestions, and moral support throughout the journey that is writing a book. Thank you to Mike Loukides for somehow convincing me that writing a book was a reasonable endeavor.

Thank you to the tech reviewers who combed through early drafts of this book, pointing out errors and offering suggestions for improvement. Thank you Alex Gude, Jon Krohn, Kristen McIntyre, and Douwe Osinga for taking the time out of your busy schedules to help make this book the best version of itself that it could be. To data practitioners whom I asked about the challenges of practical ML they felt needed the

most attention, thank you for your time and insights, and I hope you'll find that this book covers them adequately.

Finally, for their unwavering support during the series of busy weekends and late nights that came with writing this book, I'd like to thank my unwavering partner Mari, my sarcastic sidekick Eliott, my wise and patient family, and my friends who refrained from reporting me as missing. You made this book a reality.

PART I

Find the Correct ML Approach

Most individuals or companies have a good grasp of which problems they are interested in solving—for example, predicting which customers will leave an online platform or building a drone that will follow a user as they ski down a mountain. Similarly, most people can quickly learn how to train a model to classify customers or detect objects to reasonable accuracy given a dataset.

What is much rarer, however, is the ability to take a problem, estimate how best to solve it, build a plan to tackle it with ML, and confidently execute on said plan. This is often a skill that has to be learned through experience, after multiple overly ambitious projects and missed deadlines.

For a given product, there are many potential ML solutions. In Figure I-1, you can see a mock-up of a potential writing assistant tool on the left, which includes a suggestion and an opportunity for the user to provide feedback. On the right of the image is a diagram of a potential ML approach to provide such recommendations.

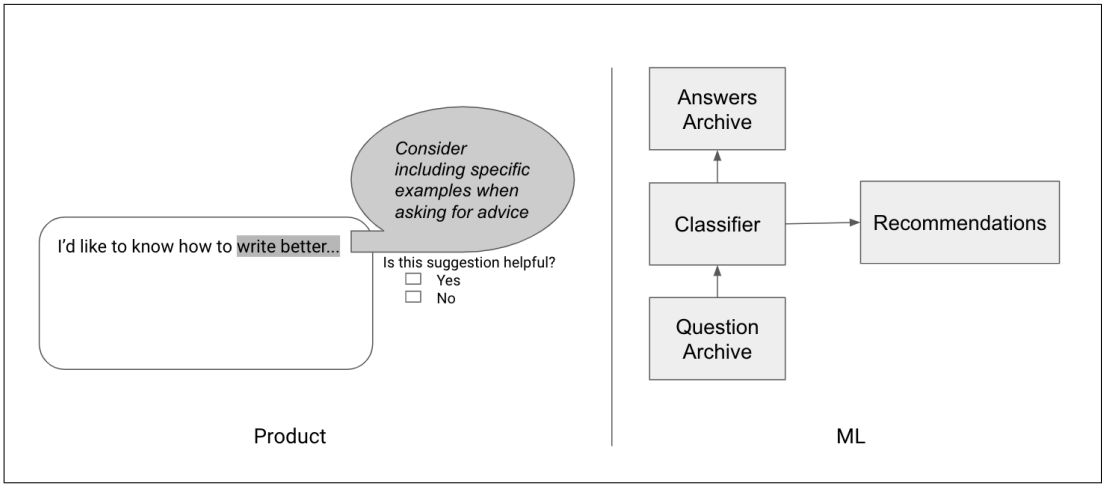

Figure I-1. From product to ML

This section starts by covering these different potential approaches, as well as methods to choose one over the others. It then dives into methods to reconcile a model's performance metrics with product requirements.

To do this, we will tackle two successive topics:

Chapter 1

By the end of this chapter, you will be able to take an idea for an application, estimate whether it is possible to solve, determine whether you would need ML to do so, and figure out which kind of model would make the most sense to start with.

Chapter 2

In this chapter, we will cover how to accurately evaluate your model's performance within the context of your application's goals and how to use this measure to make regular progress.

CHAPTER 1

From Product Goal to ML Framing

ML allows machines to learn from data and behave in a probabilistic way to solve problems by optimizing for a given objective. This stands in opposition to traditional programming, in which a programmer writes step-by-step instructions describing *how* to solve a problem. This makes ML particularly useful to *build systems for which we are unable to define a heuristic solution*.

Figure 1-1 describes two ways to write a system to detect cats. On the left, a program consists of a procedure that has been manually written out. On the right, an ML approach leverages a dataset of photos of cats and dogs labeled with the corresponding animal to allow a model to learn the mapping from image to category. In the ML approach, there is no specification of how the result should be achieved, only a set of example inputs and outputs.

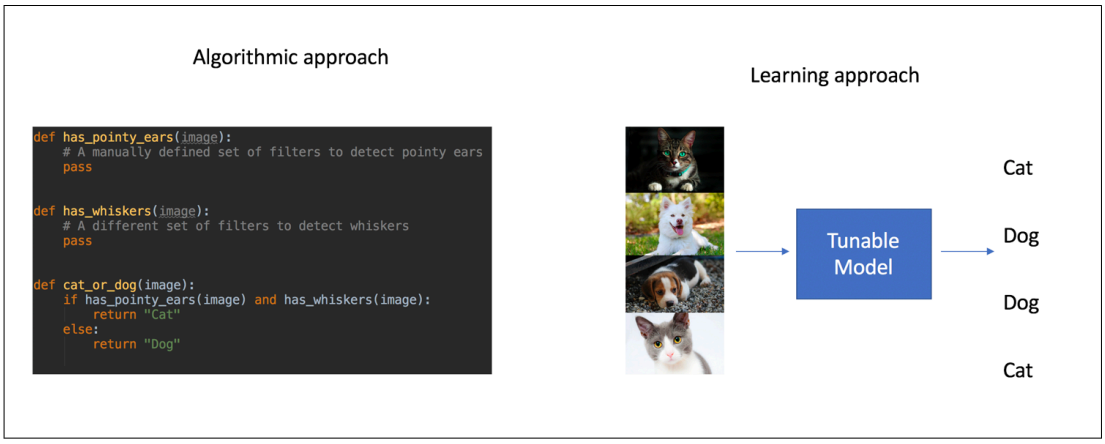

Figure 1-1. From defining procedures to showing examples

ML is powerful and can unlock entirely new products, but since it is based on pattern recognition, it introduces a level of uncertainty. It is important to identify which parts

of a product would benefit from ML and how to frame a learning goal in a way that minimizes the risks of users having a poor experience.

For example, it is close to impossible (and extremely time-consuming to attempt) for humans to write step-by-step instructions to automatically detect which animal is in an image based on pixel values. By feeding thousands of images of different animals to a convolutional neural network (CNN), however, we can build a model that performs this classification more accurately than a human. This makes it an attractive task to tackle with ML.

On the other side, an application that calculates your taxes automatically should rely on guidelines provided by the government. As you may have heard, having errors on your tax return is generally frowned upon. This makes the use of ML for automatically generating tax returns a dubious proposition.

You never want to use ML when you can solve your problem with a manageable set of deterministic rules. By manageable, I mean a set of rules that you could confidently write and that would not be too complex to maintain.

So while ML opens up a world of different applications, it is important to think about which tasks *can* and *should* be solved by ML. When building products, you should start from a concrete business problem, determine whether it requires ML, and then work on finding the ML approach that will allow you to iterate as rapidly as possible.

We will cover this process in this chapter, starting with methods to estimate what tasks are able to be solved by ML, which ML approaches are appropriate for which product goals, and how to approach data requirements. I will illustrate these methods with the ML Editor case study that we mentioned in "Our Case Study: ML–Assisted Writing" on page xii, and an interview with Monica Rogati.

Estimate What Is Possible

Since ML models can tackle tasks without humans needing to give them step-by-step instructions, that means they are able to perform some tasks better than human experts (such as detecting tumors from radiology images or playing Go) and some that are entirely inaccessible to humans (such as recommending articles out of a pool of millions or changing the voice of a speaker to sound like someone else).

The ability of ML to learn directly from data makes it useful in a broad range of applications but makes it harder for humans to accurately distinguish which problems are solvable by ML. For each successful result published in a research paper or a corporate blog, there are hundreds of reasonable-sounding ideas that have entirely failed.

While there is currently no surefire way to predict ML success, there are guidelines that can help you reduce the risk associated with tackling an ML project. Most importantly, you should always start with a product goal to then decide how best to solve it.

At this stage, be open to any approach whether it requires ML or not. When considering ML approaches, make sure to evaluate those approaches based on how appropriate they are for the product, not simply on how interesting the methods are in a vacuum.

The best way to do this is by following two successive steps: (1) framing your product goal in an ML paradigm, and (2) evaluating the feasibility of that ML task. Depending on your evaluation, you can readjust your framing until we are satisfied. Let's explore what these steps really mean.

1. *Framing a product goal in an ML paradigm:* When we build a product, we start by thinking of what service we want to deliver to users. As we mentioned in the introduction, we'll illustrate concepts in this book using the case study of an editor that helps users write better questions. The goal of this product is clear: we want users to receive actionable and useful advice on the content they write. ML problems, however, are framed in an entirely different way. An ML problem concerns itself with *learning a function from data*. An example is learning to take in a sentence in one language and output it in another. For one product goal, there are usually many different ML formulations, with varying levels of implementation difficulty.
2. *Evaluating ML feasibility:* All ML problems are not created equal! As our understanding of ML has evolved, problems such as building a model to correctly classify photos of cats and dogs have become solvable in a matter of hours, while others, such as creating a system capable of carrying out a conversation, remain open research problems. To efficiently build ML applications, it is important to consider multiple potential ML framings and start with the ones we judge as the simplest. One of the best ways to evaluate the difficulty of an ML problem is by looking at both the kind of data it requires and at the existing models that could leverage said data.

To suggest different framings and evaluate their feasibility, we should examine two core aspects of an ML problem: data and models.

We will start with models.

Models

There are many commonly used models in ML, and we will abstain from giving an overview of all of them here. Feel free to refer to the books listed in "Additional Resources" on page x for a more thorough overview. In addition to common models, many model variations, novel architectures, and optimization strategies are published on a weekly basis. In May 2019 alone, more than 13,000 papers were submitted to ArXiv, a popular electronic archive of research where papers about new models are frequently submitted.

It is useful, however, to share an overview of different categories of models and how they can be applied to different problems. To this end, I propose here a simple taxonomy of models based on how they approach a problem. You can use it as a guide for selecting an approach to tackle a particular ML problem. Because models and data are closely coupled in ML, you will notice some overlap between this section and "Data types" on page 13.

ML algorithms can be categorized based on whether they require labels. Here, a label refers to the presence in the data of an ideal output that a model should produce for a given example. Supervised algorithms leverage datasets that contain labels for inputs, and they aim to learn a mapping from inputs to labels. Unsupervised algorithms, on the other hand, do not require labels. Finally, weakly supervised algorithms leverage labels that aren't exactly the desired output but that resemble it in some way.

Many product goals can be tackled by both supervised and unsupervised algorithms. A fraud detection system can be built by training a model to detect transactions that differ from the average one, requiring no labels. Such a system could also be built by manually labeling transactions as fraudulent or legitimate, and training a model to learn from said labels.

For most applications, supervised approaches are easier to validate since we have access to labels to assess the quality of a model's prediction. This also makes it easier to train models since we have access to desired outputs. While creating a labeled dataset can sometimes be time-consuming initially, it makes it much easier to build and validate models. For this reason, this book will mostly cover supervised approaches.

With that being said, determining which kind of inputs your model will take in and which outputs it will produce will help you narrow down potential approaches significantly. Based on these types, any of the following categories of ML approaches could be a good fit:

- Classification and regression
- Knowledge extraction
- Catalog organization
- Generative models

I'll expand on these further in the following section. As we explore these different modeling approaches, I recommend thinking about which kind of data you have available to you or could gather. Oftentimes, data availability ends up being the limiting factor in model selection.

Classification and regression

Some projects are focused on effectively classifying data points between two or more categories or attributing them a value on a continuous scale (referred to as *regression* instead of *classification*). Regression and classification are technically different, but oftentimes methods to tackle them have significant overlap, so we lump them together here.

One of the reasons classification and regression are similar is because most classification models output a probability score for a model to belong to a category. The classification aspect then boils down to deciding how to attribute an object to a category based on said scores. At a high level, a classification model can thus be seen as a regression on probability values.

Commonly, we classify or score individual examples, such as spam filters that classify each email as valid or junk, fraud detection systems that classify users as fraudulent or legitimate, or computer vision radiology models that classify bones as fractured or healthy.

In Figure 1-2, you can see an example of classifying a sentence according to its sentiment, and the topic it covers.

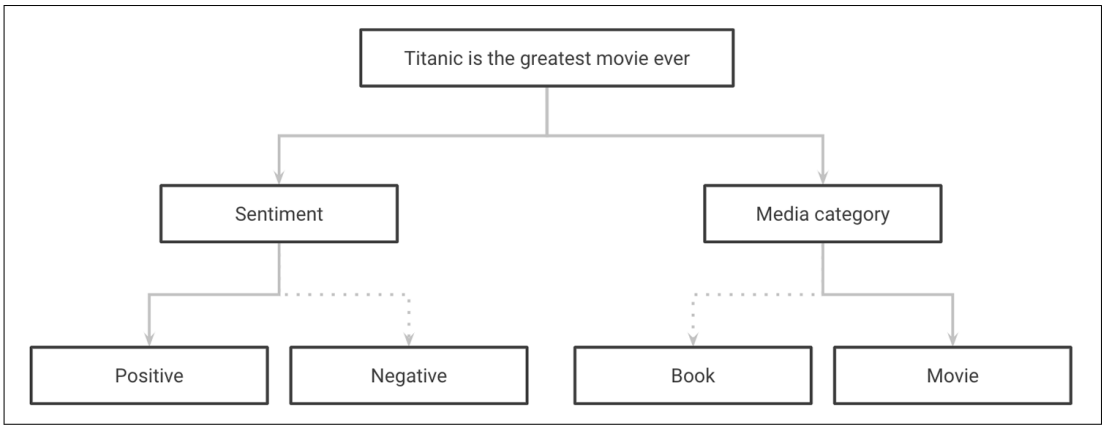

Figure 1-2. Classifying a sentence in multiple categories

In regression projects, instead of attributing a class to each example, we give them a value. Predicting the sale price of a home based on attributes such as how many rooms it has and where it is located is an example of a regression problem.

In some cases, we have access to a series of past data points (instead of one) to predict an event in the future. This type of data is often called a *time series*, and making predictions from a series of data points is referred to as *forecasting*. Time-series data could represent a patient's medical history or a series of attendance measurements from national parks. These projects often benefit from models and features that can leverage this added temporal dimension.

In other cases, we attempt to detect unusual events from a dataset. This is called *anomaly detection*. When a classification problem is trying to detect events that represent a small minority of the data and thus are hard to detect accurately, a different set of methods is often required. Picking a needle out of a haystack is a good analogy here.

Good classification and regression work most often requires significant feature selection and feature engineering work. Feature selection consists of identifying a subset of features that have the most predictive value. Feature generation is the task of identifying and generating good predictors of a target by modifying and combining existing features of a dataset. We will cover both of these topics in more depth in Part III.

Recently, deep learning has shown a promising ability to automatically generate useful features from images, text, and audio. In the future, it may play a larger part in simplifying feature generation and selection, but for now, they remain integral parts of the ML workflow.

Finally, we can often build on top of the classification or score described earlier to provide useful advice. This requires building an interpretable classification model and using its feature to generate actionable advice. More on this later!

Not all problems aim to attribute a set of categories or values to an example. In some cases, we'd like to operate at a more granular level and extract information from parts of an input, such as knowing where an object is in a picture, for example.

Knowledge extraction from unstructured data

Structured data is data that is stored in a tabular format. Database tables and Excel sheets are good examples of structured data. *Unstructured data* refers to datasets that are not in a tabular format. This includes text (from articles, reviews, Wikipedia, and so on), music, videos, and songs.

In Figure 1-3, you can see an example of structured data on the left and unstructured data on the right. Knowledge extraction models focus on taking a source of unstructured data and extracting structure out of it using ML.

In the case of text, knowledge extraction can be used to add structure to reviews, for example. A model can be trained to extract aspects such as cleanliness, service quality, and price from reviews. Users could then easily access reviews that mention topics they are interested in.

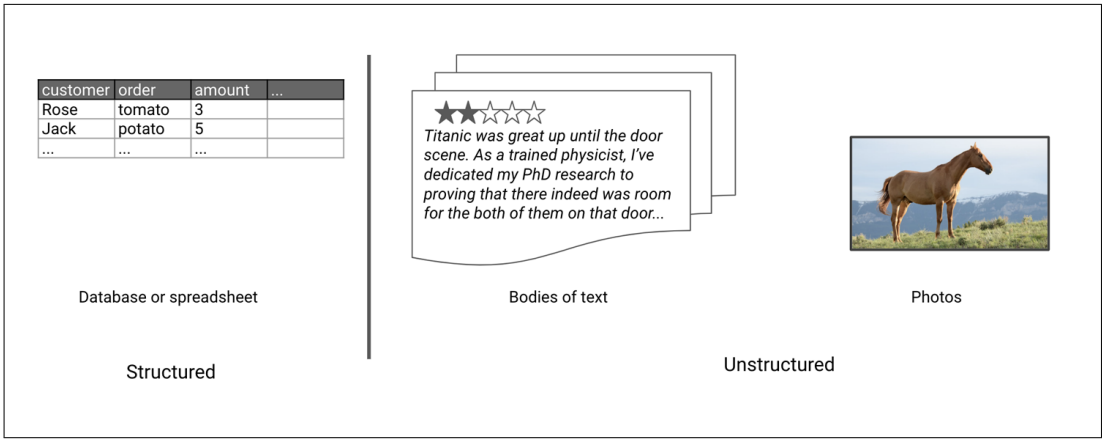

Figure 1-3. Example types of structured and unstructured data

In the medical domain, a knowledge extraction model could be built to take raw text from medical papers as input, and extract information such as the disease that is discussed in the paper, as well as the associated diagnosis and its performance. In Figure 1-4, a model takes a sentence as an input and extracts which words refer to a type of media and which words refer to the title of a media. Using such a model on comments in a fan forum, for example, would allow us to generate summaries of which movies frequently get discussed.

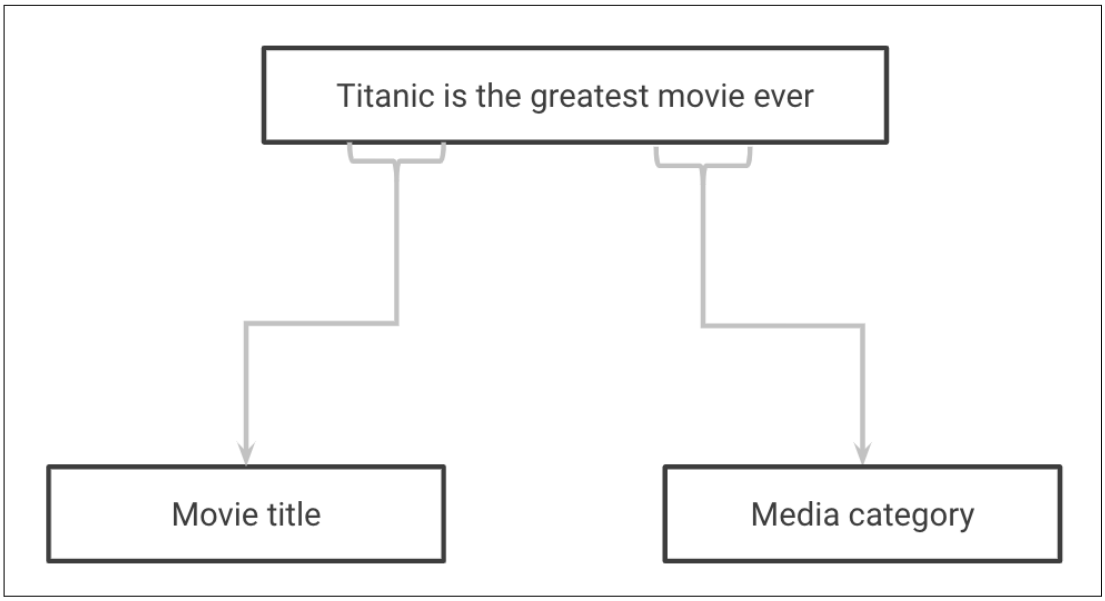

Figure 1-4. Extracting media type and title from a sentence

For images, knowledge extraction tasks often consist of finding areas of interest in an image and categorizing them. Two common approaches are depicted in Figure 1-5: object detection is a coarser approach that consists of drawing rectangles (referred to

as *bounding boxes*) around areas of interest, while segmentation precisely attributes each pixel of an image to a given category.

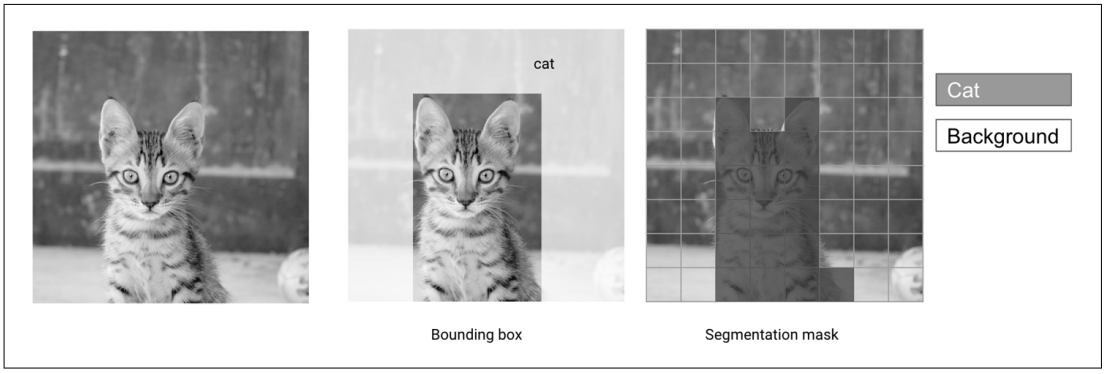

Figure 1-5. Bounding boxes and segmentation masks

Sometimes, this extracted information can be used as an input to another model. An example is using a pose detection model to extract key points from a video of a yogi, and feeding those key points to a second model that classifies the pose as correct or not based on labeled data. Figure 1-6 shows an example of a series of two models that could do just this. The first model extracts structured information (the coordinates of joints) from unstructured data (a photo), and the second one takes these coordinates and classifies them as a yoga pose.

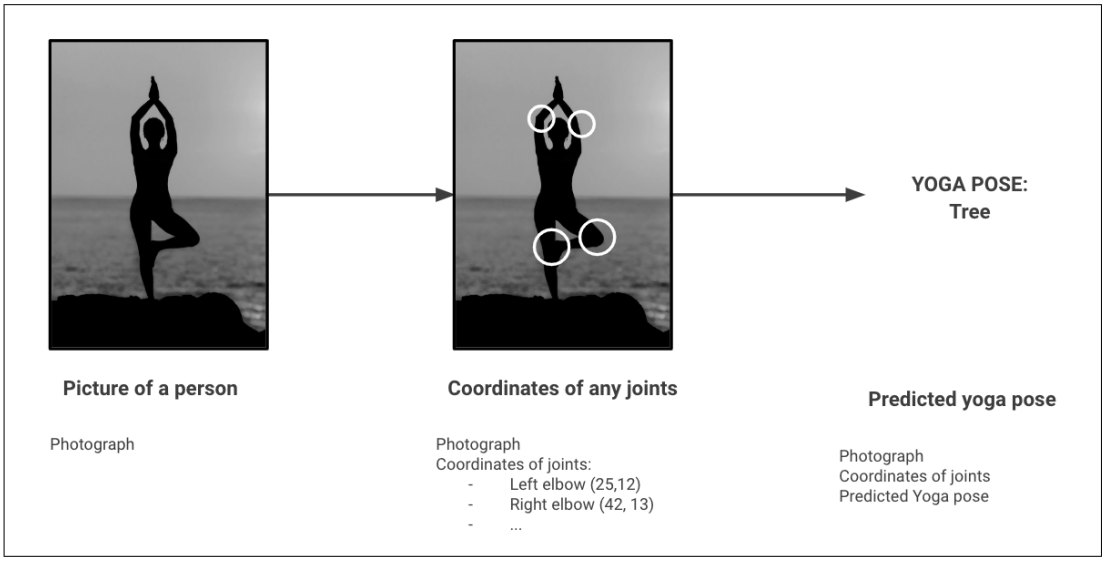

Figure 1-6. Yoga pose detection

The models we've seen so far focus on generating outputs conditioned on a given input. In some cases such as search engines or recommendation systems, the product goal is about surfacing relevant items. This is what we will cover in the following category.

Catalog organization

Catalog organization models most often produce a set of results to present to users. These results can be conditioned on an input string typed into a search bar, an uploaded image, or a phrase spoken to a home assistant. In many cases such as streaming services, this set of results can also be proactively presented to the user as content they may like without them making a request at all.

Figure 1-7 shows an example of such a system that volunteers potential candidate movies to watch based on a movie the user just viewed, but without having the user perform any form of search.

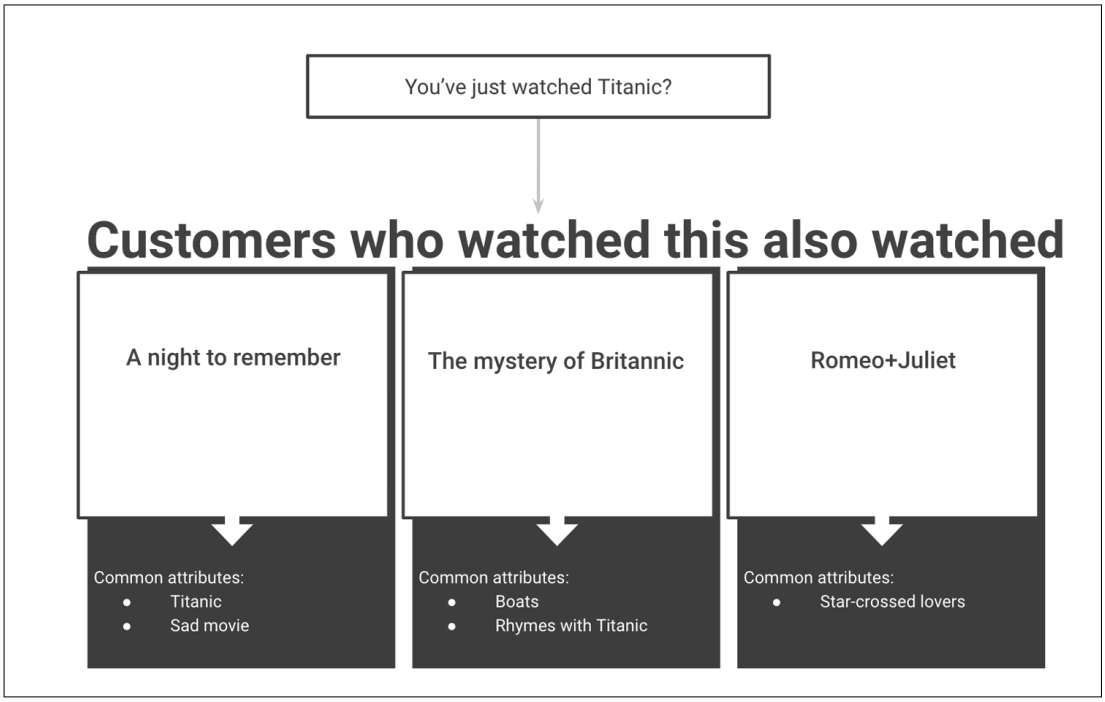

Figure 1-7. Movie recommendations

These models thus either *recommend* items that are related to an item the user already expressed interest in (similar Medium articles or Amazon products) or provide a useful way to *search* through a catalog (allowing users to search for items by typing text or submitting their own photos).

These recommendations are most often based on learning from previous user patterns, in which case they are called *collaborative* recommendation systems. Sometimes, they are based on particular attributes of items, in which case they are called *content-based* recommendation systems. Some systems leverage both collaborative and content-based approaches.

Finally, ML can also be used for creative purposes. Models can learn to generate aesthetically pleasing images, audio, and even amusing text. Such models are referred to as generative models.

Generative models

Generative models focus on generating data, potentially dependent on user input. Because these models focus on generating data rather than classifying it in categories, scoring it, extracting information from it, or organizing it, they usually have a wide range of outputs. This means that generative models are uniquely fit for tasks such as translation, where outputs are immensely varied.

On the other side, generative models are often used to train and have outputs that are less constrained, making them a riskier choice for production. For that reason, unless they are necessary to attain your goal, I recommend starting with other models first. For readers who would like to dive deeper into generative models, however, I recommend the book *Generative Deep Learning*, by David Foster.

Practical examples include translation, which maps sentences in one language to another; summarization; subtitle generation, which maps videos and audio tracks to transcripts; and neural style transfer (see Gatys et al., "A Neural Algorithm of Artistic Style"), which maps images to stylized renditions.

Figure 1-8 shows an example of a generative model transforming a photograph on the left by giving it a style similar to a painting shown in the vignette on the right side.

Figure 1-8. Style transfer example from Gatys et al., "A Neural Algorithm of Artistic Style"

As you can tell by now, each type of model requires a different type of data to be trained on. Commonly, a choice of a model will strongly depend on the data you are able to obtain—data availability often drives model selection.

Let's cover a few common data scenarios and associated models.

Data

Supervised ML models leverage patterns in data to learn useful mappings between inputs and outputs. If a dataset contains features that are predictive of the target output, it should be possible for an appropriate model to learn from it. Most often, however, we do not initially have the right data to train a model to solve a product use case from end-to-end.

For example, say we are training a *speech recognition* system that will listen for requests from customers, understand their intent, and perform actions depending on said intent. When we start working on this project, we may define a set of intents we would want to understand, such as "playing a movie on the television."

To train an ML model to accomplish this task, we would need to have a dataset containing audio clips of users of diverse backgrounds asking in their own terms for the system to play a movie. Having a representative set of inputs is crucial, as any model will only be able to learn from the data that we present to it. If a dataset contains examples from only a subset of the population, a product will be useful to only that subset. With that in mind, because of the specialized domain we have selected, it is extremely unlikely that a dataset of such examples already exists.

For most applications we would want to tackle, we will need to search for, curate, and collect additional data. The data acquisition process can vary widely in scope and complexity depending on the specifics of a project, and estimating the challenge ahead of time is crucial in order to succeed.

To start, let's define a few different situations you can find yourself in when searching for a dataset. This initial situation should be a key factor in deciding how to proceed.

Data types

Once we've defined a problem as *mapping inputs to outputs*, we can search for data sources that follow this mapping.

For fraud detection, these could be examples of fraudulent and innocent users, along with features of their account that we could use to predict their behavior. For translation, this would be a corpus of sentence pairs in the source and target domains. For content organization and search, this could be a history of past searches and clicks.

We will rarely be able to find the exact mapping we are looking for. For this reason, it is useful to consider a few different cases. Think of this as a hierarchy of needs for data.

Data availability

There are roughly three levels of data availability, from best-case scenario to most challenging. Unfortunately, as with most other tasks, you can generally assume that the most useful type of data will be the hardest to find. Let's go through them.

Labeled data exists
: This is the leftmost category in Figure 1-9. When working on a supervised model, finding a *labeled dataset* is every practitioner's dream. Labeled here means that many data points contain the target value that the model is trying to predict. This makes training and judging model quality much easier, as labels provide ground truth answers. Finding a labeled dataset that fits your needs and is freely available on the web is rare in practice. It is common, however, to mistake the dataset that you find for the dataset that you need.

Weakly labeled data exists
: This is the middle category in Figure 1-9. Some datasets contain labels that are not exactly a modeling target, but somewhat correlated with it. Playback and skip history for a music streaming service are examples of a weakly labeled dataset for predicting whether a user dislikes a song. While a listener may have not marked a song as disliked, if they skipped it as it was playing, it is an indication that they may have not been fond of it. Weak labels are less precise by definition but often easier to find than perfect labels.

Unlabeled data exists
: This category is on the right side of Figure 1-9. In some cases, while we do not have a labeled dataset mapping desired inputs to outputs, we at least have access to a dataset containing relevant examples. For the text translation example, we might have access to large collections of text in both languages, but with no direct mapping between them. This means we need to label the dataset, find a model that can learn from unlabeled data, or do a little bit of both.

We need to acquire data
: In some cases, we are one step away from unlabeled data, as we need to first acquire it. In many cases, we do not have a dataset for what we need and thus will need to find a way to acquire such data. This is often seen as an insurmountable task, but many methods now exist to rapidly gather and label data. This will be the focus of Chapter 4.

For our case study, an ideal dataset would be a set of user-typed questions, along with a set of better worded questions. A *weakly labeled* dataset would be a dataset of many questions with some weak labels indicative of their quality such as "likes" or "upvotes." This would help a model learn what makes for good and bad questions but would not provide side-by-side examples for the same question. You can see both of these examples in Figure 1-9.

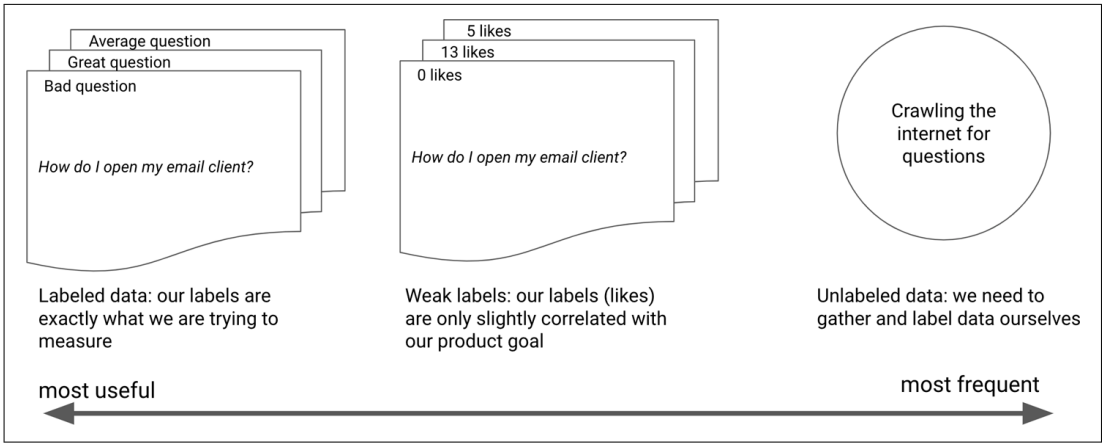

Figure 1-9. Data availability versus data usefulness

In general in ML, a weakly labeled dataset refers to a dataset that contains information that will help a model learn, but not the exact ground truth. In practice, most datasets that we can gather are weakly labeled.

Having an imperfect dataset is entirely fine and shouldn't stop you. The ML process is iterative in nature, so starting with a dataset and getting some initial results is the best way forward, regardless of the data quality.

Datasets are iterative

In many cases, since you will not be able to immediately find a dataset containing a direct mapping from inputs to your desired output, I suggest progressively iterating on the way you formulate the problem, making it easier to find an adequate dataset to start with. Each dataset you explore and use will provide you with valuable information that you can use to curate the next version of your dataset and generate useful features for your models.

Let's now dive into the case study and see how we can use what we've learned to identify different models and datasets we could use, and choose the most appropriate.

Framing the ML Editor

Let's see how we could iterate through a product use case to find the right ML framing. We'll get through this process by outlining a method to progress from a product goal (helping users write better questions) to an ML paradigm.

We would like to build an editor that accepts questions by users and improves them to be better written, but what does "better" mean in this case? Let's start by defining the writing assistant's product goal a little more clearly.

Many people use forums, social networks, and websites such as Stack Overflow to find answers to their questions. However, the way that people ask questions has a dramatic impact on whether they receive a useful answer. This is unfortunate both for the user looking to get their question answered and for future users that may have the same problem and could have found an existing answer useful. To that end, our goal will be to *build an assistant that can help users write better questions*.

We have a product goal and now need to decide which modeling approach to use. To make this decision, we will go through the iteration loop of model selection and data validation mentioned earlier.

Trying to Do It All with ML: An End-to-End Framework

In this context, *end-to-end* means using a single model to go from input to output with no intermediary steps. Since most product goals are very specific, attempting to solve an entire use case by learning it from end-to-end often requires custom-built cutting-edge ML models. This may be the right solution for teams that have the resources to develop and maintain such models, but it is often worth it to start with more well-understood models first.

In our case, we could attempt to gather a dataset of poorly formulated questions, as well as their professionally edited versions. We could then use a generative model to go straight from one text to the other.

Figure 1-10 depicts what this would look like in practice. It shows a simple diagram with user input on the left, the desired output on the right, and a model in between.

Figure 1-10. End-to-end approach

As you'll see, this approach comes with significant challenges:

Data
: To acquire such a dataset, we would need to find pairs of questions with the same intent but of different wording quality. This is quite a rare dataset to find as is. Building it ourselves would be costly as well, as we would need to be assisted by professional editors to generate this data.

Model
: Models going from one sequence of text to another, seen in the generative models category discussed earlier, have progressed tremendously in recent years. Sequence-to-sequence models (as described in the paper by I. Sutskever et al.,

"Sequence to Sequence Learning with Neural Networks") were originally proposed in 2014 for translation tasks and are closing the gap between machine and human translation. The success of these models, however, has mostly been on sentence-level tasks, and they have not been frequently used to process text longer than a paragraph. This is because so far, they have not been able to capture long-term context from one paragraph to another. Additionally, because they usually have a large number of parameters, they are some of the slowest models to train. If a model is trained only once, this is not necessarily an issue. If it needs to be retrained hourly or daily, training time can become an important factor.

Latency
: Sequence-to-sequence models are often *autoregressive models*, meaning they require the model's output of the previous word to start working on the next. This allows them to leverage information from neighboring words but causes them to be slower to train and slower to run inference on than the simpler models. Such models can take a few seconds to produce an answer at inference time, as opposed to subsecond latency for simpler models. While it is possible to optimize such a model to run quickly enough, it will require additional engineering work.

Ease of implementation
: Training complex end-to-end models is a very delicate and error-prone process, as they have many moving parts. This means that we need to consider the trade-off between a model's potential performance and the complexity it adds to a pipeline. This complexity will slow us down when building a pipeline, but it also introduces a maintenance burden. If we anticipate that other teammates may need to iterate on and improve on your model, it may be worthwhile to choose a set of simpler, more well-understood models.

This end-to-end approach could work, but it will require a lot of upfront data gathering and engineering effort, with no success guarantee, so it would be worthwhile to explore other alternatives, as we will cover next.

The Simplest Approach: Being the Algorithm

As you'll see in the interview at the end of this section, it is often a great idea for data scientists to *be the algorithm* before they implement it. In other words, to understand how to best automate a problem, start by attempting to solve it manually. So, if we were editing questions ourselves to improve readability and the odds of getting an answer, how would we go about it?

A first approach would be to not use data at all but leverage prior art to define what makes a question or a body of text well written. For general writing tips, we could reach out to a professional editor or research newspapers' style guides to learn more.

In addition, we should dive into a dataset to look at individual examples and trends and let those inform our modeling strategy. We will skip this for now as we will cover how to do this in more depth in Chapter 4.

To start, we could look at existing research to identify a few attributes we might use to help people write more clearly. These features could include factors such as:

Prose simplicity
: We often give new writers the advice to use simpler words and sentence structures. We could thus establish a set of criteria on the appropriate sentence and word length, and recommend changes as needed.

Tone
: We could measure the use of adverbs, superlatives, and punctuation to measure the polarity of the text. Depending on the context, more opinionated questions may receive fewer answers.

Structural features
: Finally, we could try to extract the presence of important structural attributes such as the use of greetings or question marks.

Once we have identified and generated useful features, we can build a simple solution that uses them to provide recommendations. There is no ML involved here, but this phase is crucial for two reasons: it provides a baseline that is very quick to implement and will serve as a yardstick to measure models against.

To validate our intuition about how to detect good writing, we can gather a dataset of "good" and "bad" text and see if we can tell the good from the bad using these features.

Middle Ground: Learning from Our Experience

Now that we have a baseline set of features, we can attempt to use them to *learn a model of style from a body of data*. To do this we can gather a dataset, extract the features we described earlier from it, and train a classifier on it to separate good and bad examples.

Once we have a model that can classify written text, we can inspect it to identify which features are highly predictive and use those as recommendations. We will see how to do this in practice in Chapter 7.

Figure 1-11 describes this approach. On the left side, a model is trained to classify a question as good or bad. On the right side, the trained model is given a question and scores candidate reformulations of this question that will lead to it receiving a better score. The reformulation with the highest score is recommended to the user.

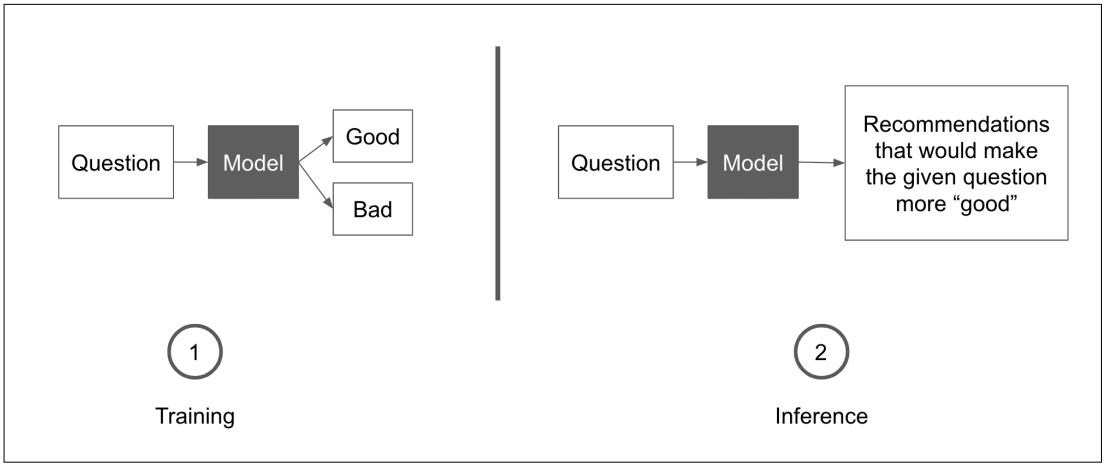

Figure 1-11. A middle ground between manual and end-to-end

Let's examine the challenges we outlined in "Trying to Do It All with ML: An End-to-End Framework" on page 16 and see whether the classifier approach makes them any easier:

Dataset
: We could obtain a dataset of good and bad examples by gathering questions from an online forum along with some measure of their quality, such as the number of views or upvotes. As opposed to the end-to-end approach, this does not require us to have access to revisions of the same questions. We simply need a set of good and bad examples we can hope to learn aggregate features from, which is an easier dataset to find.

Model
: We need to consider two things here: how predictive a model is (can it efficiently separate good and bad articles?) and how easily features can be extracted from it (can we see which attributes were used to classify an example?). There are many potential models that we could use here, along with different features we could extract from text to make it more explainable.

Latency
: Most text classifiers are quite quick. We could start with a simple model such as a random forest, which can return results in less than a tenth of a second on regular hardware, and move on to more complex architectures if needed.

Ease of implementation
: Compared to text generation, text classification is relatively well understood, meaning building such a model should be relatively quick. Many examples of working text classification pipelines exist online, and many such models have already been deployed to production.

If we start with a human heuristic and then build this simple model, we will quickly be able to have an initial baseline, and the first step toward a solution. Moreover, the initial model will be a great way to inform what to build next (more on this in Part III).

For more on the importance of starting with simple baselines, I sat down with Monica Rogati, who shares some of the lessons she has learned helping data teams deliver products.

Monica Rogati: How to Choose and Prioritize ML Projects

After getting her Ph.D. in computer science, Monica Rogati started her career at LinkedIn where she worked on core products such as integrating ML into the People You May Know algorithm and built the first version of job-to-candidate matching. She then became VP of data at Jawbone where she built and led the entire data team. Monica is now an adviser to dozens of companies whose number of employees ranges from 5 to 8,000. She has kindly agreed to share some of the advice she often gives to teams when it comes to designing and executing on ML products.

Q: *How do you scope out an ML product?*

A: You have to remember that you are trying to use the best tools to solve a problem, and only use ML if it makes sense.

Let's say you wanted to predict what a user of an application will do and show it to them as a suggestion. You should start by combining discussions on modeling and product. Among other things, this includes designing the product around handling ML failures gracefully.

You could start by taking into account the confidence our model has in its prediction. We could then formulate our suggestions differently based on the confidence score. If the confidence is above 90%, we present the suggestion prominently; if it is over 50%, we still display it but with less emphasis, and we do not display anything if the confidence is below this score.

Q: *How do you decide what to focus on in an ML project?*

A: You have to find the *impact bottleneck*, meaning the piece of your pipeline that could provide the most value if you improve on it. When working with companies, I often find that they may not be working on the right problem or not be at the right growth stage for this.

There are often problems around the model. The best way to find this out is to replace the model with something simple and debug the whole pipeline. Frequently, the issues will not be with the accuracy of your model. Frequently, your product is dead even if your model is successful.

Q: *Why do you usually recommend starting with a simple model?*

A: The goal of our plan should be to derisk our model somehow. The best way to do this is to start with a "strawman baseline" to evaluate worst-case performance. For our earlier example, this could be simply suggesting whichever action the user previously took.

If we did this, how often would our prediction be correct, and how annoying would our model be to the user if we were wrong? Assuming that our model was not much better than this baseline, would our product still be valuable?

This applies well to examples in natural language understanding and generation such as chatbots, translation, Q&A, and summarization. Oftentimes in summarization, for example, simply extracting the top keywords and categories covered by an article is enough to serve most users' needs.

Q: *Once you have your whole pipeline, how do you identify the impact bottleneck?*

A: You should start with imagining that the impact bottleneck is solved, and ask yourself whether it was worth the effort you estimated it would take. I encourage data scientists to compose a tweet and companies to write a press release before they even start on a project. That helps them avoid working on something just because they thought it was cool and puts the impact of the results into context based on the effort.

The ideal case is that you can pitch the results regardless of the outcome: if you do not get the best outcome, is this still impactful? Have you learned something or validated some assumptions? A way to help with this is to build infrastructure to help lower the required effort for deployment.

At LinkedIn, we had access to a very useful design element, a little window with a few rows of text and hyperlinks, that we could customize with our data. This made it easier to launch experiments for projects such as job recommendations, as the design was already approved. Because the resource investment was low, the impact did not have to be as large, which allowed for a faster iteration cycle. The barrier then becomes about nonengineering concerns, such as ethics, fairness, and branding.

Q: *How do you decide which modeling techniques to use?*

A: The first line of defense is looking at the data yourself. Let's say we want to build a model to recommend groups to LinkedIn users. A naive way would be to recommend the most popular group containing their company's name in the group title. After looking at a few examples, we found out one of the popular groups for the company Oracle was "Oracle sucks!" which would be a terrible group to recommend to Oracle employees.

It is always valuable to spend the manual effort to look at inputs and outputs of your model. Scroll past a bunch of examples to see if anything looks weird. The head of my

department at IBM had this mantra of doing something manually for an hour before putting in any work.

Looking at your data helps you think of good heuristics, models, and ways to reframe the product. If you rank examples in your dataset by frequency, you might even be able to quickly identify and label 80% of your use cases.

At Jawbone, for example, people entered "phrases" to log the content of their meals. By the time we labeled the top 100 by hand, we had covered 80% of phrases and had strong ideas of what the main problems we would have to handle, such as varieties of text encoding and languages.

The last line of defense is to have a diverse workforce that looks at the results. This will allow you to catch instances where a model is exhibiting discriminative behavior, such as tagging your friends as a gorilla, or is insensitive by surfacing painful past experiences with its smart "this time last year" retrospective.

Conclusion

As we've seen, building an ML-powered application starts with judging feasibility and picking an approach. Most often, picking a supervised approach is the simplest way to get started. Among those, classification, knowledge extraction, catalog organization, or generative models are the most common paradigms in practice.

As you are picking an approach, you should identify how easily you'll be able to access strongly or weakly labeled data, or any data at all. You should then compare potential models and datasets by defining a product goal and choosing the modeling approach that best allows you to accomplish this goal.

We illustrated these steps for the ML Editor, opting to start with simple heuristics and a classification-based approach. And finally, we covered how leaders such as Monica Rogati have been applying these practices to successfully ship ML models to users.

Now that we have chosen an initial approach, it is time to define success metrics and create an action plan to make regular progress. This will involve setting minimal performance requirements, doing a deep dive into available modeling and data resources, and building a simple prototype.

We will cover all of those in Chapter 2.