Building Intelligent Systems

A Guide to Machine Learning Engineering

—

Geoff Hulten

Apress®

Building Intelligent Systems: A Guide to Machine Learning Engineering

Geoff Hulten
Lynnwood, Washington, USA

ISBN-13 (pbk): 978-1-4842-3431-0
ISBN-13 (electronic): 978-1-4842-3432-7
https://doi.org/10.1007/978-1-4842-3432-7

Library of Congress Control Number: 2018934680

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Susan McDermott
Development Editor: Laura Berendson
Coordinating Editor: Rita Fernando

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484234310. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author

Geoff Hulten is a machine learning scientist and PhD in machine learning. He has managed applied machine learning teams for over a decade, building dozens of Internet-scale Intelligent Systems that have hundreds of millions of interactions with users every day. His research has appeared in top international conferences, received thousands of citations, and won a SIGKDD Test of Time award for influential contributions to the data mining research community that have stood the test of time.

About the Technical Reviewer

Jeb Haber has a BS in Computer Science from Willamette University. He spent nearly two decades at Microsoft working on a variety of projects across Windows, Internet Explorer, Office, and MSN. For the last decade-plus of his Microsoft career, Jeb led the program management team responsible for the safety and security services provided by Microsoft SmartScreen (anti-phishing, anti-malware, and so on.) Jeb's team developed and managed global-scale Intelligent Systems with hundreds of millions of users. His role included product vision/planning/strategy, project management, metrics definition and people/team development. Jeb helped organize a culture along with the systems and processes required to repeatedly build and run global scale, 24×7 intelligence and reputation systems. Jeb is currently serving as the president of two non-profit boards for organizations dedicated to individuals and families dealing with the rare genetic disorder phenylketonuria (PKU).

Acknowledgments

There are so many people who were part of the Intelligent Systems I worked on over the years. These people helped me learn, helped me understand. In particular, I'd like to thank:

Jeb Haber and John Scarrow for being two of the key minds in developing the concepts described in this book and for being great collaborators over the years. None of this would have happened without their leadership and dedication.

Also: Anthony P., Tomasz K., Rob S., Rob M., Dave D., Kyle K., Eric R., Ameya B., Kris I., Jeff M., Mike C., Shankar S., Robert R., Chris J., Susan H., Ivan O., Chad M. and many others…

Introduction

Building Intelligent Systems is a book about leveraging machine learning in practice.

It covers everything you need to produce a fully functioning Intelligent System, one that leverages machine learning and data from user interactions to improve over time and achieve success.

After reading this book you'll be able to design an Intelligent System end-to-end. You'll know:

- When to use an Intelligent System and how to make it achieve your goals.
- How to design effective interactions between users and Intelligent Systems.
- How to implement an Intelligent System across client, service, and back end.
- How to build the intelligence that powers an Intelligent System and grow it over time.
- How to orchestrate an Intelligent System over its life-cycle.

You'll also understand how to apply your existing skills, whether in software engineering, data science, machine learning, management or program management to the effort.

There are many great books that teach data and machine-learning skills. Those books are similar to books on programming languages; they teach valuable skills in great detail. This book is more like a book on software engineering; it teaches how to take those base skills and produce working systems.

This book is based on more than a decade of experience building Internet-scale Intelligent Systems that have hundreds of millions of user interactions per day in some of the largest and most important software systems in the world. I hope this book helps accelerate the proliferation of systems that turn data into impact and helps readers develop practical skills in this important area.

Who This Book Is For

This book is for anyone with a computer science degree who wants to understand what it takes to build effective Intelligent Systems.

Imagine a typical software engineer who is assigned to a machine learning project. They want to learn more about it so they pick up a book, and it is technical, full of statistics and math and modeling methods. These are important skills, but they are the wrong information to help the software engineer contribute to the effort. *Building Intelligent Systems* is the right book for them.

Imagine a machine learning practitioner who needs to understand how the end-to-end system will interact with the models they produce, what they can count on, and what they need to look out for in practice. *Building Intelligent Systems* is the right book for them.

Imagine a technical manager who wants to begin benefiting from machine learning. Maybe they hire a machine learning PhD and let them work for a while. The machine learning practitioner comes back with charts, precision/recall curves, and training data requests, but no framework for how they should be applied. *Building Intelligent Systems* is the right book for that manager.

Data and Machine Learning Practitioners

Data and machine learning are at the core of many Intelligent Systems, but there is an incredible amount of work to be done between the development of a working model (created with machine learning) and the eventual sustainable customer impact. Understanding this supporting work will help you be better at modeling in a number of ways.

First, it's important to **understand the constraints** these systems put on your modeling. For example, where will the model run? What data will it have access to? How fast does it need to be? What is the business impact of a false positive? A false negative? How should the model be tuned to maximize business results?

Second, it's important to be able to **influence the other participants**. Understanding the pressures on the engineers and business owners will help you come to good solutions and maximize your chance for success. For example, you may not be getting all the training data you'd like because of telemetry sampling. Should you double down on modeling around the problem, or would an engineering solution make more sense? Or maybe you are being pushed to optimize for a difficult extremely-high precision, when your models are already performing at a very good (but slightly lower) precision. Should you keep chasing that super-high precision or should you work to influence the user experience in ways that reduce the customer impact of mistakes?

Third, it's important to understand how the **supporting systems can benefit you**. The escalation paths, the manual over-rides, the telemetry, the guardrails that prevent against major mistakes—these are all tools you can leverage. You need to understand when to use them and how to integrate them with your modeling process. Should you discard a model that works acceptably for 99% of users but really, really badly for 1% of users? Or maybe you can count on other parts of the system to address the problem.

Software Engineers

Building software that delights customers is a lot of work. No way around it, behind every successful software product and service there is some serious engineering. Intelligent Systems have some unique properties which present interesting challenges. This book describes the associated concepts so you can design and build Intelligent Systems that are efficient, reliable, and that best-unlock the power of machine learning and data science.

First, this book will identify the **entities and abstractions** that need to exist within a successful Intelligent System. You will learn the concepts behind the intelligence runtime, context and features, models, telemetry, training data, intelligence management, orchestration, and more.

Second, the book will give you a **conceptual understanding of machine learning and data sciences**. These will prepare you to have good discussions about tradeoffs between engineering investments and modeling investments. Where can a little bit of your work really enable a solution? And where are you being asked to boil the ocean to save a little bit of modeling time?

Third, the book will explore **patterns for Intelligent Systems** that my colleagues and I have developed over a decade and through implementing many working systems. What are the pros and cons or running intelligence in a client or in a service? How do you bound and verify components that are probabilistic? What do you need to include in telemetry so the system can evolve?

Program Managers

Machine learning and Data Sciences are hot topics. They are fantastic tools, but they are tools; they are not solutions. This book will give you enough conceptual understanding so you know what these tools are good at and how to deploy them to solve your business problems.

The first thing you'll learn is to develop an **intuition for when machine learning and data science are appropriate**. There is nothing worse than trying to hammer a square peg into a round hole. You need to understand what types of problems can be solved by machine learning. But just as importantly, you need to understand what types of problems can't be—or at least not easily. There are so many participants in a successful endeavor, and they speak such different, highly-technical, languages, that this is particularly difficult. This book will help you understand enough so you can ask the right questions and understand what you need from the answers.

The second is to get an intuition on return on investment so you can determine **how much Intelligent System to use**. By understanding the real costs of building and maintaining a system that turns data into impact you can make better choices about when to do it. You can also go into it with open eyes, and have the investment level scoped for success. Sometimes you need all the elements described in this book, but sometimes the right choice for your business is something simpler. This book will help you make good decisions and communicate them with confidence and credibility.

Finally, the third thing a program manager will learn here is to understand how to **plan, staff, and manage an Intelligent System project**. You will get the benefit of our experience building many large-scale Intelligent Systems: the life cycle of an Intelligent System; the day-to-day process of running it; the team and skills you need to succeed.

PART I

Approaching an Intelligent Systems Project

Chapters 1-4 set the groundwork for a successful Intelligent Systems project. This part describes what Intelligent Systems are and what they are good for. It refreshes some important background. It explains how to ensure that an Intelligent System has a useful, achievable goal. And it gives an overview of what to expect when taking on an Intelligent Systems project.

CHAPTER 1

Introducing Intelligent Systems

Intelligent Systems are all around us. In our light bulbs. In our cars. In our watches. In our thermostats. In our computers. How do we make them do the things that make our lives better? That delight us?

When should a light bulb turn on? When should an e-commerce site show us a particular product? When should a search engine take us to a particular site? When should a speaker play some music?

Answering questions like these, and answering them extremely well, is fundamental to unlocking the value of Intelligent Systems. And it is really hard.

Some of the biggest, most valuable companies in the world have their core business built around answering simple questions, like these:

- What web page should I display based on a short query?
- What ad should I present in a particular context?
- What product should I show to this shopper?
- What movie would this user enjoy right now?
- What book would this person like to read?
- Which news stories will generate the most interest?
- Which programs should I block from running to keep a machine safe?

Answering each of these questions "well enough" has made companies worth billions—in a few cases hundreds of billions—of dollars. And it has done so by making lots of people smarter, more productive, happier, and safer. But this is just the tip of the iceberg.

© Geoff Hulten 2018
G. Hulten, *Building Intelligent Systems*, https://doi.org/10.1007/978-1-4842-3432-7_1

There are tens of thousands of similar questions we could try to answer: When should my front door unlock? What exercise should a fitness app suggest next? What type of song should an artist write next? How should a game evolve to maximize player engagement?

This book is about reliably and efficiently unlocking the potential of Intelligent Systems.

Elements of an Intelligent System

Intelligent Systems connect users to artificial intelligence (machine learning) to achieve meaningful objectives. An Intelligent System is one in which the intelligence evolves and improves over time, particularly when the intelligence improves by watching how users interact with the system.

Successful Intelligent Systems have all of the following:

- **A meaningful objective**. An Intelligent System must have a reason for being, one that is meaningful to users and accomplishes your goals, and one that is achievable by the Intelligent System you will be able to build and run. Selecting an objective is a critical part of achieving success, but it isn't easy to do. The first part of this book will help you understand what Intelligent Systems do, so you'll know when you should use one and what kinds of objectives you should set for it.
- **The intelligent experience**. An intelligent experience must take the output of the system's intelligence (such as the predictions its machine learning makes) and present it to users to achieve the desired outcomes.

  To do this it must have a user interface that adapts based on the predictions and that puts the intelligence in a position to shine when it is right—while minimizing the cost of mistakes it makes when it is wrong. The intelligent experience must also elicit both implicit and explicit feedback from users to help the system improve its intelligence over time. The second part of this book will explore intelligent experiences, the options and the pitfalls for connecting users with intelligence.

- **The implementation of the intelligence**. The Intelligent System implementation includes everything it takes to execute intelligence, to move the intelligence where it needs to be, to manage it, to light up the intelligent experiences based on it, to collect telemetry to verify the system is functioning, and to gather the user feedback that will improve the intelligence over time. The third part of this book describes all the components of an intelligence implementation. It will prepare you to design and implement an Intelligent System of your own.
- **Intelligence creation**. Intelligent Systems are about setting intelligence up for success. This intelligence can come from many different places, ranging from simple heuristics to complex machine learning. Intelligence must be organized so that the right types of intelligence address the right parts of the problem, and so it can be effectively created by a team of people over an extended time. The fourth part of this book discusses the act of creating and growing intelligence for Internet-scale Intelligent Systems. It will prepare you to leverage all available approaches and achieve success.
- **The orchestration**. An Intelligent System lives over time, and all its elements must be kept in balance to achieve its objectives. This orchestration includes controlling how the system changes, keeping the experience in sync with the quality of the intelligence, deciding what telemetry to gather to track down and eliminate problems, and how much money to spend building and deploying new intelligence. It also involves dealing with mistakes, controlling risk, and defusing abuse. The fifth part of this book explains everything it takes to orchestrate an Intelligent System and achieve its goals through all phases of its life-cycle.

An Intelligent System is one way to apply machine learning in practice. An Intelligent System takes *intelligence* (produced via machine learning and other approaches) and leverages and supports it to achieve your objectives and to improve over time.

An Example Intelligent System

Intelligent Systems might be used to implement search engines, e-commerce sites, self-driving cars, and computer vision systems that track the human body and know who a person is and when they are smiling. But these are big and complicated systems.

Let's look at a much simpler example to see how a solution might evolve from a traditional system into an Intelligent System.

The Internet Toaster

Let's consider an Internet-connected smart-toaster. A good idea? Maybe, maybe not. But let's consider it. Our toaster has two controls: a slider that controls the intensity of the toasting and a lever that starts the toast.

It seems simple enough. The toaster's intelligence simply needs to map settings of the intensity slider to toast times. At a low setting, the toaster runs for, say, 30 seconds. At high settings the toaster runs for two minutes. That kind of thing.

So sit down, think for a while, come up with some toast-time settings and send the toaster to customers. What could go wrong?

Well, if you choose a maximum intensity that toasts for too long it could burn the things it toasts. Most customers who use that setting will be unhappy, throw away their cinder-toast, and start again.

You can imagine other failure cases, all the little irritations of toast-making that result in customers standing over their toasters, hands on the levers, ready to cut the toast short. Or customers repeatedly toasting the same piece of bread, a bit at a time, to get it the way they like it.

That's not good. If we are going to build a toaster, we want to build a really good one.

So maybe we do some testing, tweaking the toaster until both the high and low settings make toast that we think is desirable. Not too crispy, not too cool.

Great.

Did we get it right? Will this toaster do what our customers want?

It's hard to know. No matter how much toast we've eaten in our lives, it's really impossible to prove we've gotten the settings right for all the types of toast all our customers might want to make.

And so we realize we need to incorporate the opinions and experiences of others into our toaster-building process. But how?

Maybe we start with a focus group. Bring in dozens of members of the toast-making public, put them in a toasting lab, and take notes as they toast.

Then we tweak the toast-time settings again to reflect the way these volunteers toast. Now do we have the perfect toaster? Will this focus-group–tuned toaster make all the right toast that hundreds of thousands of people all around the world will want?

What if someone puts something frozen into the toaster? Or something from the fridge? Or what if someone has a tradition of making toaster s'mores? Or what if someone invents a new toaster-product, unlike anything humanity has ever toasted before? Is our toaster right for all of these situations? Probably not.

Using Data to Toast

So maybe making a perfect toasting machine is a bit harder than just asking a few people what they like.

There are just too many use cases to optimize by hand if we want to get the perfect toast in every conceivable situation. We could run focus groups every day for the rest of our lives and we still wouldn't see all the types of toast a toaster might make.

We need to do better. It's time for some serious data science.

The toaster is Internet connected, so we can program it to send telemetry back to our service. Every time someone toasts something we can know what setting they used and how long the toasting went before it stopped.

We ship version 1 of the toaster (perhaps to a controlled set of users), and the toast telemetry starts flooding in to our servers.

Now we know exactly which intensity settings people are using in their real lives (not in some contrived lab setting). We know how many times people push the lever down to start a toast job and how many times they pop the lever up to stop one early.

Can we use this data to make a better toaster?

Of course!

We could set the maximum intensity to something that at least a few users are actually using. Then we could set up metrics to make sure we don't have the toaster biased to over-toasting. For example, we could monitor the percentage of toasts that are early-stopped by their user (presumably because they were about to burn something) and tweak and tweak till we get those under control.

We could set the minimum intensity to something reasonable too. Something that users seem to use. We could track the double-toast rate (where someone toasts something and immediately re-toasts it) and tweak to make sure the toaster isn't biased to under-toasting.

Heck, we can even set the default intensity, the one in the middle of the range, to the most commonly used toast time.

Since our toasters are Internet-connected, we can update them with the new settings by having them pull the data from our servers. Heck, we could tweak the toaster's settings every single day, twice on Sunday—this is the age of miracles and wonders!

There are some seams with this approach, some things we had to assume. For example, we had to assume that toasting multiple times in quick succession is a sign of failure, that the customer is re-toasting the same bread instead of making several pieces of toast rapid-fire.

We had to assume that an early-stop is a sign the bread was starting to burn and not a sign that the customer was late for work and rushing out the door.

Also when we deploy the new settings to toasters, how do we ensure that users are going to like them? We are pretty sure (based on data science) that the new settings better-match what our overall user population is doing, so that's good.

But what about the user who was getting their own brand of perfectly toasted bagel yesterday and today they get… something different?

Despite these problems, we've got a pretty decent toaster. We've got telemetry to know the toaster is roughly doing its job. We've got a way to service it and improve it over time. Now let's really blow the doors off this thing.

Sensors and Heuristic Intelligence

If we want to make the best toaster, we're going to need more than a single slider and a toast lever. Let's add some sensors:

- A weight sensor to know how much toast is in the toaster and to determine when a customer places something in the toaster and when they take something out of it.
- A temperature sensor to know if the item placed in the toaster is chilled, frozen, or room temperature.
- A location sensor to know what region of the world the toaster is in so it can adapt to different tastes in different locales.

- A proximity sensor to know if someone is near the toaster and a camera to identify who it is.
- A clock to know if a toast is a breakfast toast or a dinner one.
- A little memory to know what's been toasted recently and to monitor the pattern of setting changes and toastings.
- A smoke sensor to know when the toaster has made a bad mistake and is about to burn something.

Now when a customer walks up to the toaster and puts something in it, the toaster can look at who the user is, try to guess what they are trying to toast, and automatically suggest a setting.

Heck, if the toaster is good enough there is no longer any need for the intensity setting or the toasting lever at all. We could update the toasting experience to be totally automatic. We could ship toasters with no buttons or knobs or anything. The customer drops something in, walks away, and comes back to delightfully toasted—anything!

If we could just figure out a way to turn all these sensor readings into the correct toast times for… anything.

To do that we need intelligence—the program or rules or machine-learned model that makes these types of decisions.

Let's start simply, by hand-crafting some intelligence.

Let's write a set of rules that consider the sensor readings and output intensity suggestions. For example: If something cold and heavy is put into the toaster, then toast for 5 minutes at high intensity. But for every degree above freezing, reduce the toast time by 2 seconds. But in Britain add 15 seconds to the toast time (because they like it that way). But if the weight and size match a known toaster product (some big brand toaster meal), then toast it the "right amount of time" based on the product's directions.

And that kind of stuff.

Every time a user complains, because the toaster toasted something wrong, you can add a new rule.

Whenever the telemetry shows a spike of double-toastings you can tweak the rules to improve.

Every time you add a new rule or update an old one you can update all the toasters you shipped all around the world by having them download the new settings from your server.

With this approach, you'll probably need to write and maintain a lot of rules to deal with all the possibilities. It's going to be a lot of work. You could employ a dozen people and give them months of rule-writing-time, and you might not be happy with the outcome.

You might never be.

Toasting with Machine Learning

In situations like this, where there is an optimization problem that is too hard or too expensive to do by hand, people turn to machine learning.

At a high level, machine learning can observe examples of users using their Internet toasters and automatically produce a set of rules to control the toaster just the way users would want it. Kind of like the hand-crafted heuristic ones we just talked about, only machine-crafted.

And machines can make lot more rules than humans can. Machines can balance inputs from dozens of sensors; they can optimize for thousands of different types of users simultaneously. They can incorporate new data and reoptimize everything every day, every hour—sometimes even faster. They can personalize rules to each user.

In order to work, machine learning needs data that shows the situation the user faced (the sensor readings) the action they took (how they set the toaster) and the outcome they got (if they liked the toast or had to tweak something and try again). Machine learning needs examples of when things are going right as well as examples of when things are going wrong. Lots and lots of examples of them.

To find examples of when things are going right, we can look through the telemetry for times when the user put something in the toaster, pressed the start lever, waited to completion, took the item out, and walked away. These are cases where the user probably got what they wanted. We have a record of all the sensor settings, the toasting time. We can use this as training for the machine learning system. Perfect.

And for examples of when things went wrong we can look through the telemetry again. This time we find all the places where users had to fiddle with the toaster, every time they stopped the toaster early or retoasted the same item multiple times. We have records of the sensor readings and intensity settings that gave users bad outcomes, too.

So combining the good outcomes and the bad outcomes and some machine learning, we can automatically train the perfect toaster-controlling algorithm.

Put in data, get out a program that looks at sensor readings and determines the best intensity and time settings to use.

We push this learned program to our customers' toasters.

We can do it every day, every minute.

We can learn from hundreds of thousands of customer interactions—from hundreds of millions of them.

Magic!

Making an Intelligent System

And this is what Intelligent Systems do. Building an effective Intelligent System requires balancing five major components: the objective, the experience, the implementation, the intelligence, and the orchestration.

We need a problem that is suitable to solve with an Intelligent System, and worth the effort. We need to control the toaster based on what the intelligence thinks. We need to do it in a way the user finds helpful, and we also need to give the user the right controls to interact with the toaster and to give feedback we can use to learn. We need to build all the services and tools and code that gathers telemetry, produces intelligence, moves it where it needs to be, and hooks it up with users.

We need to create the intelligence every day, over and over, in a way that is predictable. And we need to keep everything running over time as new toastable products come into the market and tastes change.

We need to decide how much telemetry we should collect to trade off operational costs with the potential value. We need to decide how much to change the intelligence on any particular day so that the toaster improves, but without confusing or upsetting users.

We need to monitor our key metrics and react if they start to degrade. Maybe tune the experience? Maybe call an emergency intelligence intervention? And we need to deal with mistakes. And mistakes happen.

Intelligence (particularly machine-learned intelligence) can make bad mistakes—spectacular, counter-intuitive, highly customer-damaging mistakes.

For example, our toaster might learn that people in a specific zip code love cinder-toast. No matter what you put into the thing, if you live in that zip code—cinder. So we need ways to identify and manage these mistakes. We need ways to place guardrails on machine learning.

And unfortunately… people are people. Any time we use human feedback to optimize something—like using machine learning to build a toaster—we have to think about all the ways a human might benefit from things going wrong.

Imagine a major bread-maker paying a sweatshop of people to toast their competitor's products using crazy settings. They could carry out millions of toasts per week, flooding our telemetry system with misleading data. Our machine learning might pick up on this and "learn" what this toast-hacker is trying to teach it—our toaster might start regularly undercooking the competitor's product, leading to food safety issues.

Not good.

A successful Intelligent System will have to consider all of these issues and more.

Summary

This chapter introduced Intelligent Systems along with five key conceptual challenges that every Intelligent System must address—the objective, the experience, the implementation, the intelligence, and the orchestration.

It should now be clear that there are many ways to approach these challenges and that they are highly interrelated. To have a successful Intelligent System you must balance them. If one of the conceptual challenges is difficult in your context, the others will need to work harder to make up for it.

For example, if you are trying to retrofit an Intelligent System into an existing system where the experience has already been defined (and can't be changed), the Intelligent System might need to accept a less aggressive objective, to invest more in intelligence, or to have a fuller mistake-mitigation strategy.

But here's another way to look at it: there are a lot of ways to succeed.

Deploy an Intelligent System and discover that the intelligence problem is harder than you thought? No need to panic. There are lots of ways to compensate and make a system that delights customers and helps your business while the intelligence takes time to grow.

The rest of this book will give you the tools to approach Intelligent Systems projects with confidence.

For Thought…

After reading this chapter you should be able to:

- Identify Intelligent Systems in the world around you.
- See the potential that Intelligent Systems can unlock.
- Understand the difference between intelligence (which makes predictions about the world) and an Intelligent System (which combines objective, experience, implementation, intelligence, and orchestration to achieve results).
- Articulate all the conceptually hard things you will need to address to build a successful Intelligent System.
- Understand how these difficult things interact, including some of the tradeoffs and ways they can support one another.

You should be able to answer questions like these:

- What services do you use that (you suspect) are built by turning customer data into intelligence?
- What is the most magical experience you've had with one of these services?
- What is the worst experience you've had?
- Can you identify how the user experience supports the intelligence?
- Can you find any information on how its intelligence is produced? Maybe in a news story or publication?
- Can you determine any of the ways it detects and mitigates intelligence mistakes?

CHAPTER 2

Knowing When to Use Intelligent Systems

So you have a problem you want to solve. An existing system you need to optimize. A new idea to create a business. Something you think your customers will love. A machine-learning–based solution that isn't producing the value you'd hoped. Is an Intelligent System right for you? Sometimes yes. Sometimes no.

This chapter discusses when Intelligent Systems might be the right approach, and provides guidance on when other approaches might be better. It begins by describing the types of problems that can benefit from Intelligent Systems. It then discusses some of the properties required to make Intelligent Systems work.

Types of Problems That Need Intelligent Systems

It is always best to do things the easy way. If you can solve your problem without an Intelligent System, maybe you should. One key factor in knowing whether you'll need an Intelligent System is how often you think you'll need to update the system before you have it right. If the number is small, then an Intelligent System is probably not right.

For example, imagine implementing intelligence for part of a banking system. Your "intelligence" needs to update account balances as people withdraw money. The solution is pretty simple:

```
NewBalance = OldBalance - WithdrawalAmount
```

Something like that (and a bunch of error checking and logging and transaction management), not rocket science. Maybe you'll have a bug. Maybe you'll miss an error case (for example, if the balance goes negative). Then you might need to update the "intelligence" to get it right. Maybe a couple times? Or maybe you're super sloppy and you need to update it a dozen times before you're done.

© Geoff Hulten 2018
G. Hulten, *Building Intelligent Systems*, https://doi.org/10.1007/978-1-4842-3432-7_2

Intelligent Systems are not for problems like this. They are for problems where you think you're going to have to update the system much, much more. Thousands of times, tens of thousands of times, every hour for as long as the system exists. That kind of thing.

There are four situations that clearly require that level of iteration:

- Big problems, that require a lot of work to solve.
- Open-ended problems, which continue to grow over time.
- Time-changing problems, where the right answer changes over time.
- Intrinsically hard problems, which push the boundaries of what we think is possible.

The rest of this section will explore these each in turn.

Big Problems

Some problems are big. They have so many variables and conditions that need to be addressed that they can't really be completed in a single shot.

For example, there are more web pages than a single person could read in their lifetime—more than a hundred people could read. There are so many books, television programs, songs, video games, live event streams, tweets, news stories, and e-commerce products that it would take thousands of person-years just to experience them all.

These problems and others like them require massive scale. If you wanted to build a system to reason about one of these, and wanted to completely finish it before deploying a first version… Well, you'd probably go broke trying.

When you have a big problem that you don't think you can finish in one go, an Intelligent System might be a great way to get started, and an efficient way to make progress on achieving your vision, by giving users something they find valuable and something they are willing to help you improve.

Open-Ended Problems

Some problems are more than big. Some problems are open-ended. That is, they don't have a single fixed solution at all. They go on and on, requiring more work, without end. Web pages, books, television programs, songs, video games, live event streams—more and more of them are being created every day.

Trying to build a system to reason about and organize things that haven't even been created yet is hard.

In these cases, a static solution—one where you build it, deploy it, and walk away—is unlikely to work. Instead, these situations require services that live over long periods of time and grow throughout their lifetimes.

If your problem has a bounded solution, an Intelligent System might not be right. But if your problem is big and on-going, an Intelligent System might be the right solution.

Time-Changing Problems

Things change. Sometimes the right answer today is wrong tomorrow. For example:

- Imagine a system for identifying human faces—and then facial tattoos become super popular.
- Imagine a system for predicting stock prices—and then an airplane crashes into a building.
- Imagine a system for moving spam email to a junk folder—and then a new genius-savant decides to get in the spam business and changes the game.
- Or Imagine a UX that users struggle to use—and then they begin to learn how to work with it.

One thing's for certain—things are going to change.

Change means that the intelligence you implemented yesterday—which was totally right for what was happening, which was making a lot of users happy, maybe even making your business a lot of money—might be totally wrong for what is going to happen tomorrow.

Addressing problems that change over time requires the ability to detect that something has changed and to adapt quickly enough to be meaningful.

If your domain changes slowly or in predictable ways, an Intelligent System might not be needed. On the other hand, if change in your domain is unpredictable, drastic, or frequent, an Intelligent System might be the right solution for you.

Intrinsically Hard Problems

Some problems are just hard. So hard that humans can't quite figure out how to solve them. At least not all at once, not perfectly. Here are some examples of hard problems:

- Understanding human speech.
- Identifying objects in pictures.
- Predicting the weather more than a few minutes in the future (apparently).
- Competing with humans in complex, open-ended games.
- Understanding human expressions of emotion in text and video.

In these situations, machine learning has had great success, but this success has come on the back of years (or decades) of effort, gathering training data, understanding the problems, and developing intelligence. These types of systems are still improving and will continue to improve for the foreseeable future.

There are many ways to make progress on such hard problems. One way is to close the loop between users and intelligence creation in a meaningful application using an Intelligent System as described in this book.

Situations When Intelligent Systems Work

In addition to having a problem that is difficult enough to need an Intelligent System—one that is big, open-ended, time-changing, intrinsically hard, or some combination—an Intelligent System needs a few more things to succeed:

- A problem where a partial solution is viable and interesting.
- A way to get data from usage of the system to improve intelligence.
- An ability to influence a meaningful objective.
- A problem that justifies the effort of building an Intelligent System.

The rest of this section explores these requirements in turn.

When a Partial System Is Viable and Interesting

Intelligent Systems are essentially always incomplete, incorrect in important ways, and likely to make all sorts of mistakes. It isn't important that an Intelligent System is perfect. What matters is that an Intelligent System must be good enough to be interesting to users (and valuable to you).

An Intelligent System is viable when the value of the things it does right is (much) higher than the cost of the things it does wrong.

Consider an Intelligent System that makes "cheap" mistakes. Maybe the system is supposed to show a shopping list in an order that optimizes the user's time in the grocery store. Head to the right, pick up eggs, then milk, then loop to the back of the store for the steak, then into the next aisle for pickles… and on and on. If this system gets a few things backwards it might waste time by sending users in the wrong direction, say 15–20 seconds per mistake. Irritating, but not the end of the world. If the mistakes aren't too frequent—say just one every other shopping trip—it's easy to imagine someone using the system while it is learning the layouts of all the grocery stores in the world and adapting to changes.

Consider another Intelligent System, one that makes "expensive" mistakes. Maybe it is controlling surgical robots, and mistakes put human life at risk. This system would have to meet a much higher quality bar before it is viable. That is, you'd need to produce a much better partial system before deploying it to users. But even this system doesn't need to be perfect to be viable. Remember—surgeons make mistakes, too. An Intelligent System doesn't have to be perfect (and most never will be), it just has to be better than the other guy.

The point is that to be viable, an Intelligent System must provide a good deal for users (and for you) by producing more value with positive outcomes than it produces irritation (and cost) with negative outcomes.

When You Can Use Data from the System to Improve

Intelligent Systems work by closing the loop between usage and intelligence. When users use the system, the intelligence gets better; and when the intelligence gets better, the system produces more value for users.

To make intelligence better with usage, you need to record the interactions between users and the systems in a careful way (which we will discuss in detail later). At a high level, this involves capturing what the user saw, what they did, and whether the outcome was positive or negative. And you need to observe many, many user interactions (the harder the problem, the more you'll need). This type of data can be used to improve intelligence in many ways, including with machine learning.

When the System Can Interface with the Objective

The Intelligent System must be able to change things that influence the objective. This could be through automating a part of the system, or presenting information or options to a user that help them achieve their goals. Intelligent Systems are most effective when the following conditions are met:

- The actions the Intelligent System can take *directly* affect the objective. When there is a positive outcome, it should be largely because the Intelligent System did well; and when there is a negative outcome, it should be largely because the Intelligent System did poorly. The more external factors that affect the outcome, the harder it is to make an effective Intelligent System.
- The actions the Intelligent System can take *quickly* affect the objective. Because the more time between the action and the outcome, the more chance there is for external factors to interfere.
- The actions the Intelligent System can take are in balance with the objectives. That is, the actions the Intelligent System can take are meaningful enough to the outcome that they are usually measurable.

In practice, figuring out where to put Intelligent Systems and how much of your problem to try to solve with them will be key challenges. Too big an objective and you won't be able to connect the Intelligent System to it; too small and it won't be worth all the work.

In fact, many large, complex systems have space for more than one Intelligent System inside them. For example, an operating system might have an Intelligent System to predict what to put into a RAM cache, one to predict what files to show the user when they search, one to recommend apps to users based on usage, and another to manage the tradeoffs between power and performance when the computer is not plugged in.

You can imagine using a single Intelligent System to manage an entire OS. But it probably wouldn't work, because:

- There are too many things to control.
- The controls are too abstract from the goal.
- The types of feedback the system gets would not relate directly enough to the decisions the Intelligent System needs to make.

When it is Cost Effective

Intelligent Systems have different costs than other approaches to building computer systems and services. There are three main components of cost in most similar systems: The Intelligence, the Implementation, and the Orchestration. We will discuss these concepts in Parts 3, 4, and 5 of this book. But roughly *Intelligence* is the ways the system makes decisions about what is right to do and when, the "logic" of the system; *Implementation* is all the services, back-end systems, and client code that implements the system and interfaces intelligence outcomes with users; and *Orchestration* is the managing of the system throughout its life-cycle, such as making sure it is achieving its objectives, and dealing with mistakes.

The intelligence is usually cheaper in Intelligent Systems than other approaches, for a couple of reasons:

1. In general, Intelligent Systems produce the complex logic of a system automatically (instead of using humans). When addressing a big, hard, or open-ended task, Intelligent Systems can lead to substantial savings on intelligence production.
2. With the proper intelligent experience, Intelligent Systems produce the training data (which machine learning requires) automatically as users interact with it. Gathering training data can be a major cost driver for intelligence creation, often requiring substantial human effort, and so getting data from users can drastically shift the economics of a problem.

The implementation of an Intelligent System is similar to or a bit more expensive than the implementation other service-based systems (such as a web service or a chat service). Most of the components are similar between an Intelligent System and a nonintelligent system, but Intelligent Systems have additional requirements related to orchestrating them over their lifecycles, including creating intelligence, changing intelligent experiences, managing mistakes, organizing intelligence, and so on. We will discuss all of these in detail later in the book.

The orchestration costs of an Intelligent System are similar but a bit more expensive compared to running a non-intelligent service (like a web service or a chat service). Intelligent Systems require ongoing work, including tuning the way the intelligence is exposed to users, growing the intelligence to be better and deal with new situations, and identifying and mitigating mistakes.

By the time you've finished this book you'll be able to prepare a detailed analysis of the efficiency of using an Intelligent System for your application.

When You Aren't Sure You Need an Intelligent System

Sometimes you have a place where users, data, and intelligence interact, but you aren't sure you need all the elements of an Intelligent System. This might be for any of these reasons:

- You aren't sure your problem is actually hard, so you don't want to build a whole bunch of fancy intelligence you might not need.
- You realize the problem is hard, but aren't sure the extra work of building an Intelligent System will justify the effort for you.
- You want to take an incremental approach and solve problems as you encounter them, rather than stopping to build an Intelligent System.

That's fine. Intelligent Systems (as described in this book) aren't right for everyone—there are plenty of ways to build systems that solve hard problems. But if you do try to solve a big, open-ended, time-changing, or hard problem, you'll eventually run into many of the challenges described in this book. Even if you choose not to build an Intelligent System as described here, this book will arm you to identify common issues quickly, understand what is going on, and will prepare you to respond with proven approaches.

Summary

Intelligent Systems are most useful when solving big, open-ended, time-changing, or intrinsically hard problems. These are problems where a single solution won't work, and where the system needs to improve over time—months or years.

Intelligent Systems work well when:

- A partial solution is interesting to users and will support a viable business, so that you can start simply and grow the system's capability over time.
- You can get data to improve the system from users as they use the system. Data is usually one of the main costs in building intelligence, so crafting a system that produces it automatically is very valuable.
- When the things the system can control affect the objectives: directly, quickly, and meaningfully. That is, the outcomes the system is getting can be tied to things the Intelligent System is doing.
- When other approaches are too expensive to scale.

When looking for places to use Intelligent Systems, break down your problem into pieces. Each piece should have a clear objective that is critical to overall success, feedback that is quick and direct, and controls (or user experiences your system can control) that directly impact the objective. Each such piece is a candidate for an Intelligent System.

And even if you choose not to use an Intelligent System at all, the patterns in this book about practical machine learning, interfacing intelligence with user experience, and more will benefit just about any machine learning endeavor.

For Thought

After reading this chapter, you should know:

- Whether your system could benefit from an Intelligent System.
- How to find potential Intelligent Systems in the world around you.

You should be able to answer questions like these:

- What is your favorite hobby that would benefit from an Intelligent System?
- What is an example of a common activity you do every day where an Intelligent System might not be right? Why not?

Consider your favorite piece of software:

- Identify three places the software could use an Intelligent System.
- Which one is the most likely to succeed? Which the least? Why?

CHAPTER 3

A Brief Refresher on Working with Data

Data is central to every Intelligent System. This chapter gives a conceptual overview of working with data, introducing key concepts from data science, statistics, and machine learning. The goal is to establish a baseline of understanding to facilitate discussions and decision making between all participants of an Intelligent System project.

This chapter will cover:

- Structured data.
- Asking questions of data.
- Data models.
- Conceptual machine learning.
- Some common pitfalls of working with data.

Structured Data

Data is changing the world, that's for sure.

But what is it? A bunch of pictures on a disk? All the term papers you wrote in high school? The batting averages of every baseball player who ever played?

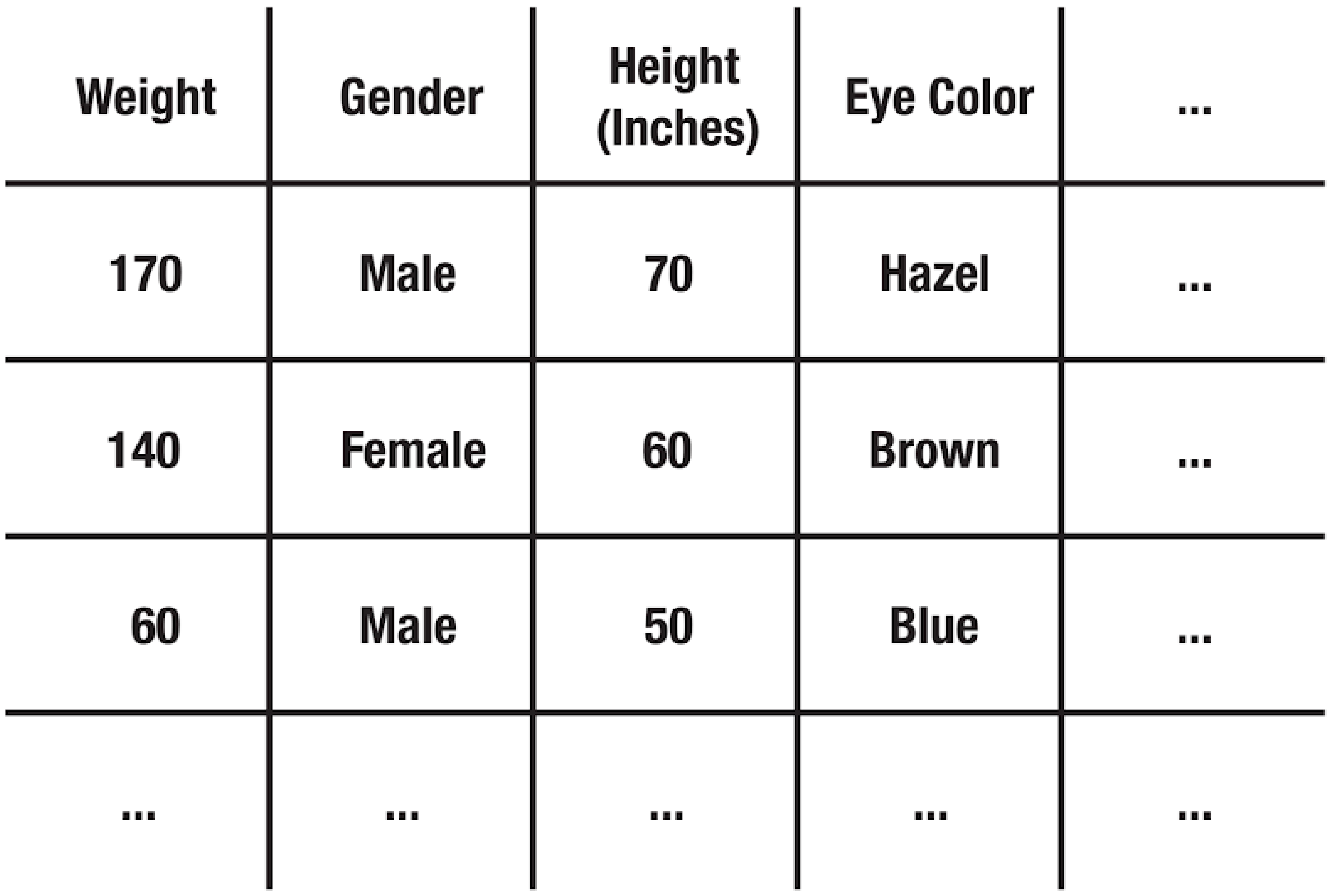
Yeah, all of that is data. There is so much data—oceans of numbers, filling hard disk drives all over the planet with ones and zeros. In raw form, data is often referred to as unstructured, and it can be quite difficult to work with.

When we turn data into intelligence, we usually work with data that has some structure to it. That is, data that is broken up into units that describe entities or events.

© Geoff Hulten 2018
G. Hulten, *Building Intelligent Systems*, https://doi.org/10.1007/978-1-4842-3432-7_3

For example, imagine working with data about people. Each unit of the data describes a single person by: their weight, their height, their gender, their eye color, that sort of stuff.

One convenient way to think about it is as a table in a spreadsheet (Figure 3-1). There is one row for each person and one column for each property of the person. Maybe the first column of each row contains a number, which is the corresponding person's weight. And maybe the third column of each row contains a number, which is the person's height. On and on.



Weight	Gender	Height (Inches)	Eye Color	...
170	Male	70	Hazel	...
140	Female	60	Brown	...
60	Male	50	Blue	...
...	...	...	...	...

Figure 3-1. *An example of structured data*

The columns will usually contain numbers (like height and weight), or be chosen from a small number of categories (like brown or blue for eye color), or be short text strings like names. There are more advanced options, of course, but this simple scheme is quite powerful and can be used to represent all sorts of things (and unlock our ability to do statistics and machine learning on them).

For example:

- You can represent a web page by: the number of words it contains, the number of images it has in it, the number of links it contains, and the number of times it contains each of 1,000 common keywords.

Working with Data Models

Models capture the answers from data, converting them into *more convenient* or *more useful* formats.

Technically, the projection we discussed previously was a model. Recall that the data set on call center volumes was converted into an average weekly volume of 75 with a 95% confidence interval of 40 - 110. The raw data might have been huge. It might have contained ten terabytes of telemetry going back a decade. But the model contained just four numbers: 75, 95%, 40, and 110.

And this simple model was more useful for making decisions about capacity planning, because it contained exactly the relevant information, and captured it in an intuitive way for the task at hand.

Models can also *fill in gaps in data* and estimate answers to questions that the data doesn't contain.

Consider the data set of people and their heights and weights. Imagine the data set contains a person who is 5'8" and a person who is 5'10", but no one who is 5'9".

What if you need to predict how much a 5'9" person will weigh?

Well, you could build a simple model. Looking into the data, you might notice that the people in the data set tend to weigh about 2.5 pounds per inch of height. Using that model, a 5'9" person would weigh 172.5 pounds. Great. Or is it?

One common way to evaluate the quality of models is called *generalization error*. This is done by reserving a part of the data set, hiding it from the person doing the modeling, and then testing the model on the holdout data to see how well the model does—how well it generalizes to data it has not seen.

For example, maybe there really was a 5'9" person in the data set, but someone hid them from the modeling process. Maybe the hidden 5'9"er weighed 150 pounds. But the model had predicted 172.5 pounds. That's 22.5 pounds off.

Good? Bad? Depends on what you need the model for.

If you sum up these types of errors over dozens or hundreds of people who were held out from the modeling process, you can get some sense of how good the model is. There are lots of technical ways to sum up errors and communicate the quality of a model, two important ones are:

- **Regression Errors** are the difference between a numeric value and the predicted value, as in the weight example. In the previous example, the model's regression error was 22.5.

- **Classification Errors** are for models that predict categories, for example one that tries to predict if a person is a male or female based on height and weight (and, yeah, I agree that model wouldn't have many friends). One way to measure classification errors involves accuracy: the model is 85% accurate at telling males from females.

Conceptual Machine Learning

Simple models can be useful. The weight-at-height model is very simple, and it's easy to find flaws with it. For example, it doesn't take gender into account, or body fat percent, or waist circumference, or where the person is from. Statistics are useful for making projections. Create a simple model, use it to answer questions. But models can be complex too—very, very, very complex.

Machine learning uses computers to improve the process of producing (complex) models from data. And these models can make projections from the data, sometimes surprisingly accurate—and sometimes surprisingly inaccurate.

A machine-learning algorithm explores the data to determine the best way to combine the information contained in the representation (the columns in the data set) into a model that generalizes accurately to data you haven't already seen.

For example, a machine-learning algorithm might predict any of the following:

- Gender by combining height, weight, age, and name.
- Weight using gender, height, age, and name.
- Height from the gender and weight.
- And so on…

And the way the model works can be very complicated, multiplying and scaling the various inputs in ways that make no sense to humans but produce accurate results.

There are probably thousands of machine-learning algorithms that can produce models of data. Some are fast; others run for days or weeks before producing a model. Some produce very simple models; some produce models that take megabytes or gigabytes of disk space. Some produce models that can make predictions very quickly on new data, a millisecond or less; some produce models that are computationally intensive to execute. And some do well on certain types of data, but fail at other types of problems.

There doesn't seem to be one universally best machine-learning algorithm to use for all situations—most work well in some situations, but poorly in others. Every few years a new technique emerges that does surprisingly well at solving important problems, and gets a lot of attention.

And some machine-learning people really like the algorithm they are most familiar with and will fight to the death to defend it despite any evidence to the contrary. You have been warned.

The machine-learning process generally breaks down into the following phases, all of which are difficult and require considerable effort from experts:

- **Getting the data to model.** This involves dealing with noise, and figuring out exactly what will be predicted and how to get the training examples for the modeling algorithm.
- **Feature engineering.** This involves processing the data into a data set with the right representation (columns in the spreadsheet) to expose to machine learning algorithms.
- **The modeling.** This involves running one or more machine-learning algorithms on the data set, looking at the mistakes they make, tweaking and tuning things, and repeating until the model is accurate enough.
- **The deployment.** This involves choosing which model to run, how to connect it into the system to create positive impact and minimal damage from mistakes.
- **The maintenance.** This involves monitoring that the model is behaving as expected, and fixing it if it starts to go out of control. For example, by rerunning the training algorithm on new data.

This book is about how to do these steps for large, complex systems.

Common Pitfalls of Working with Data

Data can be complicated, and there are a lot of things that can go wrong. Here are a few common pitfalls to keep in mind when working with data intensive systems, like Intelligent Systems.

Confidence intervals can be broken. Confidence intervals are very useful. Knowing that something is 95% likely to be between two values is great. But a 95% chance of being within an interval means there is a 5% chance of being outside the interval, too. And if it's out, it can be way out. What does that mean? 5% is one in twenty. So if you are estimating something weekly, for example to adjust capacity of a call center, one out of every twenty weeks will be out of interval—that's 2.6 times per year that you'll have too much capacity or not enough. 5% sounds small, like something that might never happen, but keep in mind that even low probability outcomes will happen eventually, if you push your luck long enough.

There is noise in your data. Noise is another word for errors. Like maybe there is a person in the dataset who has a height of 1", or 15'7". Is this real? Probably not. Where does the noise come from? All sorts of places. For example, software has bugs; data has bugs too. Telemetry systems log incorrect values. Data gets corrupted while being processed. The code that implements statistical queries has mistakes. Users turn off their computers at odd times, resulting in odd client states and crazy telemetry. Sometimes when a client has an error, it won't generate a data element. Sometimes it will generate a partial one. Sometimes this noise is caused by computers behaving in ways they shouldn't. Sometimes it is caused by miscommunication between people. Every large data set will have noise, which will inject some error into the things created from the data.

Your data has bias. Bias happens when data is collected in ways that are systematically different from the way the data is used. For example, maybe the people dataset we used to estimate height was created by interviewing random people on the streets of New York. Would a model of this data be accurate for estimating the heights of people in Japan? In Guatemala? In Timbuktu?

Bias can make data less useful and bias can come from all sorts of innocent places, like simple oversights about where the data is collected, or the context of when the data was collected. For example, users who "sorta-liked" something are less likely to respond to a survey than users who loved it or who hated it. Make sure data is being used to do the thing it was meant to do, or make sure you spend time to understand the implications of the bias your data has.

Your data is out of date. Most (simple) statistical and machine learning techniques have a big underlying assumption—that is, that things don't change. But things do change. Imagine building a model of support call volume, then using it to try to estimate support call volume for the week after a major new feature is released. Or the week after a major storm. One of the main reasons to implement all the parts of an Intelligent System is to make the system robust to change.

You want the data to say things it doesn't. Sometimes data is inconclusive, but people like to have answers. It's human nature. People might downplay the degree of uncertainty by saying things like "the answer is 42" instead of saying "we can't answer that question" or "the answer is between 12 and 72." It makes people feel smarter to be precise. It can almost seem more polite to give people answers they can work with (instead of giving them a long, partial answer). It is fun to find stories in the data, like "our best product is the toothpaste, because we redid the display-case last month." These stories are so seductive that people will find them even where they don't exist.

Tip When working with data, always ask a few questions: Is this right? How sure are we? Is there another interpretation? How can we know which is correct?

Your models of data will make mistakes. Intelligent Systems, especially ones built by modeling data, are wrong. A lot. Sometimes these mistakes make sense. But sometimes they are totally counter-intuitive gibberish. And sometimes it is very hard to fix one mistake without introducing new ones. But this is fine. Don't be afraid of mistakes. Just keep in mind: any system based on models is going to need a plan for dealing with mistakes.

Summary

Data is changing the world. Data is most useful when it is structured in a way that lines up with what it needs to be used for. Structured data is like a spreadsheet, with one row per thing (person, event, web page, and so on), and one column per property of that thing (height, weight, gender).

Data can be used to answer questions and make projections. Statistics can answer simple questions and give you guidance on how accurate the answer is. Machine learning can create very complicated models to make very sophisticated projections. Machine learning has many, many useful tools that can help make accurate models of data. More are being developed all the time.

And data can be misused badly—you have to be careful.

Criteria for a Good Goal

A successful goal will do all of the following:

1. Clearly ***communicate the desired outcome*** to all participants. Everyone should be able to understand what success looks like and why it is important, no matter what their background or technical experience.

2. ***Be achievable***. Everyone on the team should believe they are set up to succeed. The goal can be difficult, but team members should be able to explain roughly how they are going to approach success and why there is a good chance it will work.

3. ***Be measurable***. Intelligent Systems are about optimizing, and so Intelligent Systems are about measuring. Because if you can't measure something you really aren't going to be able to optimize it.

It isn't easy to know if a goal is correct. In fact, bridging the gap between high-level objectives and detailed properties of the implementation is often the key challenge to creating a successful Intelligent System. Some goals will seem perfect to some participants but make no sense to others. Some will clearly align with positive impact but be impossible to measure or achieve. There will always be trade-offs, and it is common to spend a great deal of time refining the definition of success.

But that's OK. It's absolutely worth the effort because failing to define success before starting a project is the easiest, most certain way to waste time and money.

An Example of Why Choosing Goals Is Hard

Consider an anti-phishing feature backed by an Intelligent System.

One form of phishing involves web sites that look like legitimate banking sites but are actually fake sites, controlled by abusers. Users are lured to these phishing sites and tricked into giving their banking passwords to criminals. Not good.

So what should an Intelligent System do?

Talk to a machine-learning person and it won't take long to get them excited. They'll quickly see how to build a model that examines web pages and predicts whether they are phishing sites or not. These models will consider things like the text and images on the web pages to make their predictions. If the model thinks a page is a phish, block it. If a page is blocked, a user won't browse to it, won't type their banking password into it. No more problem. Easy. Everyone knows what to do.

So the number of blocks seems like a great thing to measure—block more sites and the system is doing a better job.

Or is it?

What if the system is so effective that phishers quit? Every single phisher in the world gives up and finds something better to do with their time?

Perfect!

But then there wouldn't be any more phishing sites and the number of blocks would drop to zero. The system has achieved total success, but the metric indicates total failure.

Not great.

Or what if the system blocks one million phishing sites per day, every day, but the phishers just don't care? Every time the system blocks a site, the phishers simply make another site. The Intelligent System is blocking millions of things, everyone on the team is happy, and everyone feels like they are helping people—but the same number of users are losing their credentials to abusers after the system was built as were losing their credentials before it was built.

Not great.

One pitfall with defining success in an Intelligent System is that there are so many things that can be measured and optimized. It's very easy to find something that is familiar to work with, choose it as an objective, and get distracted from true success.

Recall the three properties of a good success criterion:

1. Communicate the desired outcome
2. Be achievable
3. Be measurable

Using the number of blocked phishing pages as a success metric hits #2 and #3 out of the park, but fails on #1.

The desired outcome of this system isn't to block phishing sites—it is to stop abusers from getting users' banking passwords.

Types of Goals

There are many types of things a system can try to optimize, ranging from very concrete to very abstract.

A system's true objective tends to be very abstract (like making money next quarter), but the things it can directly affect tend to be very concrete (like deciding whether a toaster should run for 45 or 55 seconds). Finding a clear connection between the abstract and concrete is a key source of tension in setting effective goals. And it is really, really hard.

One reason it is hard is that different participants will care about different types of goals. For example:

- Some participants will care about making money and attracting and engaging customers.
- Some participants will care about helping users achieve what they are trying to do.
- Some participants will care that the intelligence of the system is accurate.

These are all important goals, and they are related, but the connection between them is indirect. For example, you won't make much money if the system is always doing the wrong thing; but making the intelligence 1% better will not translate into 1% more profit.

This section discusses different ways to consider the success of an Intelligent System, including:

- Organizational objectives
- Leading indicators
- User outcomes
- Model properties

Most Intelligent Systems use several of these on a regular basis but focus primarily on user outcomes and model properties for day-to-day optimization.

Organizational Objectives

Organizational objectives are the real reason for the Intelligent System. In a business these might be things like revenue, profit, or number of units sold. In a nonprofit organization these might be trees saved, lives improved, or other benefits to society.

Organizational objectives are clearly important to optimize. But they are problematic as direct objectives for Intelligent Systems for at least three reasons:

1. They are very distant from what the technology can affect. For example, a person working on an Internet toaster can change the amount of time a cold piece of bread is toasted—how does that relate to number of units sold?
2. They are affected by many things out of the system's control. For example, market conditions, marketing strategies, competitive forces, changes to user behavior over time, and so on.
3. They are very slow indicators. It may take weeks or months to know if any particular action has impacted an organizational objective. This makes them difficult to optimize directly.

Every Intelligent System should contribute to an organizational objective, but the day-to-day orchestration of an Intelligent System will usually focus on more direct measures—like the ones we'll discuss in the next few sections (particularly user outcomes and model properties).

Leading Indicators

Leading indicators are measures that correlate with future success. For example:

- You are more likely to make a profit when your customers like your product than when they hate it.
- You are more likely to grow your customer base when your customers are recommending your product to their friends than when your customers are telling their friends to stay away.
- You are more likely to retain customers when they use your product every day than when they use your product once every couple of months.

Leading indicators are a way to bridge between organizational objectives and the more concrete properties of an Intelligent System (like user outcomes and model properties). If an Intelligent System gets better, customers will probably like it more. That may lead to more sales or it might not, because other factors—like competitors, marketing activities, trends, and so on—can affect sales. Leading indicators factor some of these external forces out and can help you get quicker feedback as you change your Intelligent System.

There are two main types of leading indicators: customer sentiment and customer engagement.

Customer sentiment is a measure of how your customers feel about your product. Do they like using it? Does it make them happy? Would they recommend it to a friend (or would they rather recommend it to an enemy)?

If everyone who uses your product loves it, it is a sign that you are on the right track. Keep going, keep expanding your user base, and eventually you will have business success (make revenue, sell a lot of units, and so on).

On the other hand, if everyone who uses your product hates it you might be in for some trouble. You might have some customers, they might use your product, but they aren't happy with it. They are looking for a way out. If you get a strong competitor your customers are ready to jump ship.

Sentiment is a fuzzy measure, because users' feelings can be fickle. It can also be very hard to measure sentiment accurately—users don't always want to tell you exactly what you ask them to tell you. Still, swings in sentiment can be useful indicators of future business outcomes, and Intelligent Systems can certainly affect the sentiment of users who encounter them.

Customer engagement is a measure of how much your customers use your product. This could mean the frequency of usage. It could also mean the depth of usage, as in using all the various features your product has to offer.

Customers with high engagement are demonstrating that they find value in your product. They've made a habit of your product, and they come back again and again. They will be valuable to you and your business over time.

Customers with low engagement use the product infrequently. These customers may be getting value from your offering, but they have other things on their minds. They might drift away and never think about you or your product again.

Leading indicators have some disadvantages as goals for Intelligent Systems, similar to those that organizational outcomes suffer from:

- They are indirect.
- They are affected by factors out of control of the Intelligent System.
- They aren't good at detecting small changes.

minor that no one cares. Or maybe there is a really good way for the user to give feedback, which quickly corrects the mistake—way more cheaply than investing in optimizing the last few points of the model's performance.

3. *They are too familiar to machine-learning people.* It is easy to build Intelligent Systems to optimize model properties—it is precisely what machine-learning people spend their lives doing, so it will naturally come up in any conversation about objectives. Be careful with them. They are so powerful and familiar that they may stifle and hijack the system's actual objective.

Optimizing model properties is what intelligence is about, but it is seldom the goal. A good goal will show how improving model properties contributes to having the desired impact on users and the business. A good goal will give guidance on how much model property optimization is worth.

Layering Goals

Success in an Intelligent System project is hard to define with a single metric, and the metrics that define it are often hard to measure. One good practice is to define success on different levels of abstraction and have some story about how success at one layer contributes to the others. This doesn't have to be a precise technical endeavor, like a mathematical equation, but it should be an honest attempt at telling a story that all participants can get behind.

For example, participants in an Intelligent System might:

- On an hourly or daily basis optimize model properties.
- On a weekly basis review the user outcomes and make sure changes in model properties are affecting user outcomes as expected.
- On a monthly basis review the leading indicators and make sure nothing has gone off the rails.
- On a quarterly basis look at the organizational objectives and make sure the Intelligent System is moving in the right direction to affect them.

Revisit the goals of the Intelligent System often during the course of the project.

Because things change.

Ways to Measure Goals

One reason defining success is so hard is that measuring success is harder still.

How the heck are we supposed to know how many passwords abusers got with their phishing pages?

When we discuss intelligent experiences in Part II of this book we will discuss ways to design intelligent experiences to help measure goals and get data to make the intelligence better. This section introduces some basic approaches. Using techniques like these should allow more flexibility in defining success.

Waiting for More Information

Sometimes it is impossible to tell if an action is right or wrong at the time it happens, but a few hours or days or weeks later it becomes much easier. As time passes you'll usually have more information to interpret the interaction. Here are some examples of how waiting might help:

- The system recommends content to the user, and the user consumes it completely—by waiting to see if the user consumes the content, you can get some evidence of whether the recommendation was good or bad.
- The system allows a user to type their password into a web page—by waiting to see if the user logs in from eastern Europe and tries to get all their friends to install malware, you can get some evidence if the password was stolen or not.

Waiting can be a very cheap and effective way to make a success criterion easier to measure, particularly when the user's behavior implicitly indicates success or failure.

There are a couple of downsides.

First, waiting adds latency. This means that waiting might not help with optimizing, or making fine-grained measurements.

Second, waiting adds uncertainty. There are lots of reasons a user might change their behavior. Waiting gives more time for other factors to affect the measurement.

PART II

Intelligent Experiences

Chapters 5-10 explain how to connect intelligence with users to achieve an Intelligent System's objectives. This part discusses the pitfalls and challenges of creating user experiences based on models and machine learning, along with the properties of a successful intelligent experience. It explains how to adapt experiences to get data to grow the system. And it gives a framework for verifying that an intelligent experience is behaving as intended.