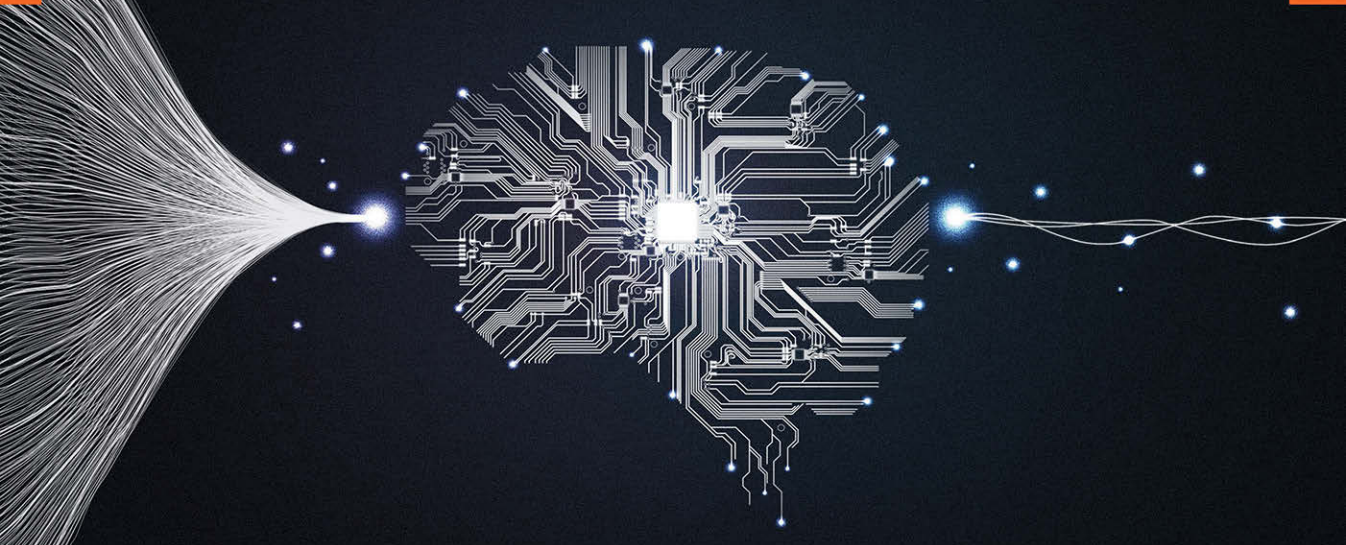
ADDISON WESLEY DATA & ANALYTICS SERIES

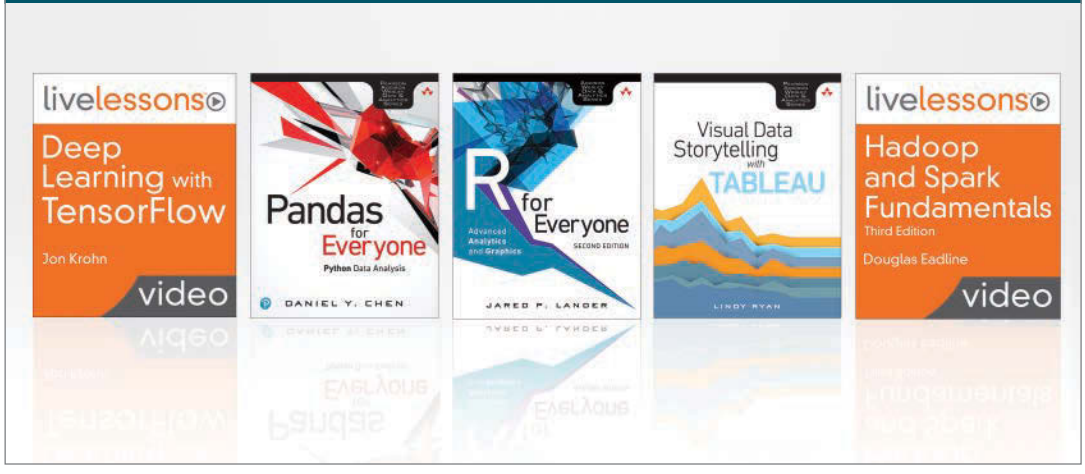
MACHINE LEARNING WITH PYTHON FOR EVERYONE



MARK E. FENNER

Machine Learning with Python for Everyone

The Pearson Addison-Wesley Data & Analytics Series



Visit **informit.com/awdataseries** for a complete list of available publications.

The **Pearson Addison-Wesley Data & Analytics Series** provides readers with practical knowledge for solving problems and answering questions with data. Titles in this series primarily focus on three areas:

1. **Infrastructure:** how to store, move, and manage data
2. **Algorithms:** how to mine intelligence or make predictions based on data
3. **Visualizations:** how to represent data and insights in a meaningful and compelling way

The series aims to tie all three of these areas together to help the reader build end-to-end systems for fighting spam; making recommendations; building personalization; detecting trends, patterns, or problems; and gaining insight from the data exhaust of systems and user interactions.

Make sure to connect with us!
informit.com/socialconnect

Pearson Addison-Wesley

informIT.com
the trusted technology learning source

O'REILLY Safari

Machine Learning with Python for Everyone

Mark E. Fenner

Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2019938761

Copyright © 2020 Pearson Education, Inc.

Cover image: cono0430/Shutterstock

Pages 58, 87: Screenshot of seaborn © 2012–2018 Michael Waskom.
Pages 167, 177, 192, 201, 278, 284, 479, 493: Screenshot of seaborn heatmap © 2012–2018 Michael Waskom.
Pages 178, 185, 196, 197, 327, 328: Screenshot of seaborn swarmplot © 2012–2018 Michael Waskom.
Page 222: Screenshot of seaborn stripplot © 2012–2018 Michael Waskom.
Pages 351, 354: Screenshot of seaborn implot © 2012–2018 Michael Waskom.
Pages 352, 353, 355: Screenshot of seaborn distplot © 2012–2018 Michael Waskom.
Pages 460, 461: Screenshot of Manifold © 2007–2018, scikit-learn developers.
Page 480: Screenshot of cluster © 2007–2018, scikit-learn developers.
Pages 483, 484, 485: Image of accordion, Vereshchagin Dmitry/Shutterstock.
Page 485: Image of fighter jet, 3dgenerator/123RF.
Page 525: Screenshot of seaborn jointplot © 2012–2018 Michael Waskom.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

ISBN-13: 978-0-13-484562-3
ISBN-10: 0-13-484562-5

ScoutAutomatedPrintCode

To my son, Ethan—
with the eternal hope of a better tomorrow

This page intentionally left blank

Contents

This page intentionally left blank

Foreword

Whether it is called statistics, data science, machine learning, or artificial intelligence, learning patterns from data is transforming the world. Nearly every industry imaginable has been touched (or soon will be) by machine learning. The combined progress of both hardware and software improvements are driving rapid advancements in the field, though it is upon software that most people focus their attention.

While many languages are used for machine learning, including R, C/C++, Fortran, and Go, Python has proven remarkably popular. This is in large part thanks to scikit-learn, which makes it easy to not only train a host of different models but to also engineer features, evaluate the model quality, and score new data. The scikit-learn project has quickly become one of Python's most important and powerful software libraries.

While advanced mathematical concepts underpin machine learning, it is entirely possible to train complex models without a thorough background in calculus and matrix algebra. For many people, getting into machine learning through programming, rather than math, is a more attainable goal. That is precisely the goal of this book: to use Python as a hook into machine learning and then add in some math as needed. Following in the footsteps of *R for Everyone* and *Pandas for Everyone*, *Machine Learning with Python for Everyone* strives to be open and accessible to anyone looking to learn about this exciting area of math and computation.

Mark Fenner has spent years practicing the communication of science and machine learning concepts to people of varying backgrounds, honing his ability to break down complex ideas into simple components. That experience results in a form of storytelling that explains concepts while minimizing jargon and providing concrete examples. The book is easy to read, with many code samples so the reader can follow along on their computer.

With more people than ever eager to understand and implement machine learning, it is essential to have practical resources to guide them, both quickly and thoughtfully. Mark fills that need with this insightful and engaging text. *Machine Learning with Python for Everyone* lives up to its name, allowing people with all manner of previous training to quickly improve their machine learning knowledge and skills, greatly increasing access to this important field.

Jared Lander,
Series Editor

This page intentionally left blank

Preface

In 1983, the movie *WarGames* came out. I was a preteen and I was absolutely engrossed: by the possibility of a nuclear apocalypse, by the almost magical way the lead character interacted with computer systems, but mostly by the potential of machines that could *learn*. I spent years studying the strategic nuclear arsenals of the East and the West—fortunately with a naivete of a tweener—but it was almost ten years before I took my first serious steps in computer programming. Teaching a computer to do a set process was amazing. Learning the intricacies of complex systems and bending them around my curiosity was a great experience. Still, I had a large step forward to take. A few short years later, I worked with my first program that was explicitly designed to *learn*. I was blown away and I knew I found my intellectual home. I want to share the world of *computer programs that learn* with you.

Audience

Who do I think *you* are? I've written *Machine Learning with Python for Everyone* for the absolute beginner to machine learning. Even more so, you may well have very little college-level mathematics in your toolbox *and I'm not going to try to change that*. While many machine learning books are very heavy on mathematical concepts and equations, I've done my best to *minimize* the amount of mathematical luggage you'll have to carry. I do expect, given the book's title, that you'll have some basic proficiency in Python. If you can *read* Python, you'll be able to get a lot more out of our discussions. While many books on machine learning rely on mathematics, I'm relying on stories, pictures, and Python code to communicate with you. There *will* be the occasional equation. Largely, these can be skipped if you are so inclined. But, if I've done my job well, I'll have given you enough context around the equation to maybe—just *maybe*—understand what it is trying to say.

Why might you have this book in your hand? The least common denominator is that all of my readers want to *learn* about machine learning. Now, you might be coming from very different backgrounds: a student in an introductory computing class focused on machine learning, a mid-career business analyst who all of sudden has been thrust beyond the limits of spreadsheet analysis, a tech hobbyist looking to expand her interests, or a scientist needing to analyze data in a new way. Machine learning is permeating society. Depending on your background, *Machine Learning with Python for Everyone* has different things to offer you. Even a mathematically sophisticated reader who is looking to do a break-in to machine learning using Python can get a lot out of this book.

So, my goal is to take someone with an interest or need to do some machine learning and teach them the *process* and the most important *concepts* of machine learning in a concrete way using the Python scikit-learn library and some of its friends. You'll come

away with overall patterns, strategies, pitfalls, and gotchas that will be applicable in every learning system you ever study, build, or use.

Approach

Many books that try to explain mathematical topics, such as machine learning, do so by presenting equations as if they tell a story to the uninitiated. I think that leaves many of us—even those of us who like mathematics!—stuck. Personally, I build a far better mental picture of the process of machine learning by combining visual and verbal descriptions with *running code*. I'm a computer scientist at heart and by training. I love building things. Building things is how I know that I've reached a level where I *really* understand them. You might be familiar with the phrase, "If you really want to know something, teach it to someone." Well, there's a follow-on. "If you really want to know something, teach a computer to do it!" That's my take on how I'm going to teach you machine learning. With minimal mathematics, I want to give you the concepts behind the most important and frequently used machine learning tools and techniques. Then, I want you to immediately see how to make a computer do it. One note: we won't be programming these methods from scratch. We'll be standing on the shoulders of giants and using some very powerful, time-saving, prebuilt software libraries (more on that shortly).

We won't be covering all of these libraries in great detail—there is simply too much material to do that. Instead, we are going to be practical. We are going to use the best tool for the job. I'll explain enough to orient you in the concept we're using—and then we'll get to using it. For our mathematically inclined colleagues, I'll give pointers to more in-depth references they can pursue. I'll save most of this for end-of-the-chapter notes so the rest of us can skip it easily.

If you are flipping through this introduction, deciding if you want to invest time in this book, I want to give you some insight into things that are out-of-scope for us. We aren't going to dive into mathematical proofs or rely on mathematics to explain things. There are many books out there that follow that path and I'll give pointers to my favorites at the ends of the chapters. Likewise, I'm going to assume that you are fluent in basic- to intermediate-level Python programming. However, for more advanced Python topics—and things that show up from third-party packages like NumPy or Pandas—I'll explain enough of what's going on so that you can understand each technique and its context.

Overview

In **Part I**, we establish a foundation. I'll give you some verbal and conceptual introductions to machine learning in Chapter 1. In Chapter 2 we introduce and take a slightly different approach to some mathematical and computational topics that show up repeatedly in machine learning. Chapters 3 and 4 walk you through your first steps in building, training, and evaluating learning systems that classify examples (classifiers) and quantify examples (regressors).

Part II shifts our focus to the most important aspect of applied machine learning systems: evaluating the success of our system in a realistic way. Chapter 5 talks about general

evaluation techniques that will apply to all of our learning systems. Chapters 6 and 7 take those general techniques and add evaluation capabilities for classifiers and regressors.

Part III broadens our toolbox of learning techniques and fills out the components of a practical learning system. Chapters 8 and 9 give us additional classification and regression techniques. Chapter 10 describes *feature engineering*: how we smooth the edges of rough data into forms that we can use for learning. Chapter 11 shows how to chain multiple steps together as a single learner and how to tune a learner's inner workings for better performance.

Part IV takes us beyond the basics and discusses more recent techniques that are driving machine learning forward. We look at learners that are made up of multiple little learners in Chapter 12. Chapter 13 discusses learning techniques that incorporate automated feature engineering. Chapter 14 is a wonderful capstone because it takes the techniques we describe throughout the book and applies them to two particularly interesting types of data: images and text. Chapter 15 both reviews many of the techniques we discuss and shows how they relate to more advanced learning architectures—neural networks and graphical models.

Our main focus is on the techniques of machine learning. We will investigate a number of learning algorithms and other processing methods along the way. However, completeness is not our goal. We'll discuss the most common techniques and only glance briefly at the two large subareas of machine learning: graphical models and neural, or deep, networks. However, we will see how the techniques we focus on relate to these more advanced methods.

Another topic we won't cover is implementing specific learning algorithms. We'll build on top of the algorithms that are already available in scikit-learn and friends; we'll create larger solutions using them as components. Still, someone has to implement the gears and cogs inside the black-box we funnel data into. If you are really interested in implementation aspects, you are in good company: I love them! Have all your friends buy a copy of this book, so I can argue I need to write a follow-up that dives into these lower-level details.

Acknowledgments

I must take a few moments to thank several people that have contributed greatly to this book. My editor at Pearson, Debra Williams Cauley, has been instrumental in every phase of this book's development. From our initial meetings, to her probing for a topic that might meet both our needs, to gently shepherding me through many (many!) early drafts, to constantly giving me just enough of a push to keep going, and finally climbing the steepest parts of the mountain at its peak . . . through all of these phases, Debra has shown the highest degrees of professionalism. I can only respond with a heartfelt *thank you*.

My wife, Dr. Barbara Fenner, also deserves more praise and thanks than I can give her in this short space. In addition to the burdens that any partner of an author must bear, she *also* served as my primary draft reader *and* our intrepid illustrator. She did the hard work of drafting all of the non-computer-generated diagrams in this book. While this is not our first joint academic project, it has been turned into the longest. Her patience is, by all appearances, never ending. Barbara, *I thank you!*

My primary technical reader was Marilyn Roth. Marilyn was unfailingly positive towards even my most egregious errors. *Machine Learning with Python for Everyone* is immeasurably better for her input. *Thank you*.

I would also like to thank several members of Pearson's editorial staff: Alina Kirsanova and Dmitry Kirsanov, Julie Nahil, and many other behind-the-scenes folks that I didn't have the pleasure of meeting. This book would not exist without you and your hardworking professionalism. *Thank you*.

Publisher's Note

The text contains unavoidable references to color in figures. To assist readers of the print edition, color PDFs of figures are available for download at http://informit.com/title/9780134845623.

For formatting purposes, decimal values in many tables have been manually rounded to two place values. In several instances, Python code and comments have been slightly modified—all such modifications should result in valid programs.

Online resources for this book are available at https://github.com/mfenner1.

Register your copy of *Machine Learning with Python for Everyone* on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780134845623) and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

About the Author

Mark Fenner, PhD, has been teaching computing and mathematics to adult audiences—from first-year college students to grizzled veterans of industry—since 1999. In that time, he has also done research in machine learning, bioinformatics, and computer security. His projects have addressed design, implementation, and performance of machine learning and numerical algorithms; security analysis of software repositories; learning systems for user anomaly detection; probabilistic modeling of protein function; and analysis and visualization of ecological and microscopy data. He has a deep love of computing and mathematics, history, and adventure sports. When he is not actively engaged in writing, teaching, or coding, he can be found launching himself, with abandon, through the woods on his mountain bike or sipping a post-ride beer at a swimming hole. Mark holds a *nidan* rank in judo and is a certified Wilderness First Responder. He and his wife are graduates of Allegheny College and the University of Pittsburgh. Mark holds a PhD in computer science. He lives in northeastern Pennsylvania with his family and works through his company, Fenner Training and Consulting, LLC.

This page intentionally left blank

Part I

First Steps

This page intentionally left blank

1

Let's Discuss Learning

1.1 Welcome

From time to time, people trot out a tired claim that computers can "only do what they are told to do." The claim is taken to mean that computers can only do what their programmers know how to do *and* can explain to the computer. This claim is *false*. Computers can perform tasks that their programmers cannot explain to them. Computers can solve tasks that their programmers do not understand. We will break down this paradox with an example of a computer program that *learns*.

I'll start by discussing one of the oldest—if not the oldest known—examples of a programmed machine-learning system. I've turned this into a story, but it is rooted in historical facts. Arthur Samuel was working for IBM in the 1950s and he had an interesting problem. He had to test the big computing machines that were coming off the assembly line to make sure transistors didn't blow up when you turned a machine on and ran a program—people don't like smoke in their workplace. Now, Samuel quickly got bored with running simple toy programs and, like many computing enthusiasts, he turned his attention towards *games*. He built a computer program that let him play checkers against himself. That was fun for a while: he tested IBM's computers by playing checkers. But, as is often the case, he got bored playing two-person games solo. His mind began to consider the possibility of getting a good game of checkers against a *computer opponent*. Problem was, he wasn't good enough at checkers to explain good checkers strategies to a computer!

Samuel came up with the idea of having the computer *learn* how to play checkers. He set up scenarios where the computer could make moves and evaluate the costs and benefits of those moves. At first, the computer was bad, very bad. But eventually, the program started making progress. It was slow going. Suddenly, Samuel had a great two-for-one idea: he decided to let one computer play another and take himself out of the loop. Because the computers could make moves much faster than Samuel could enter his moves—let alone think about them—the result was many more cycles of "make a move and evaluate the outcome" per minute and hour and day.

Here is the amazing part. It didn't take very long for the computer opponent to be able to consistently beat Samuel. *The computer became a better checkers player than its programmer!* How on earth could this happen, if "computers can only do what they are told to do"? The answer to this riddle comes when we analyze *what the computer was told to*

do. What Samuel told the computer to do was not the *play-checkers* task; it was the *learn-to-play-checkers* task. Yes, we just went all *meta* on you. *Meta* is what happens when you take a picture of someone taking a picture (of someone else). Meta is what happens when a sentence refers to itself; the next sentence is an example. *This sentence has five words*. When we access the meta level, we step outside the box we were playing in and we get an entirely new perspective on the world. *Learning to play checkers*—a task that develops skill at another task—is a meta task. It lets us move beyond a limiting interpretation of the statement, *computers can only do what they are told*. Computers do what they are told, but they can be told to *develop a capability*. Computers can be told to learn.

1.2 Scope, Terminology, Prediction, and Data

There are many kinds of computational learning systems out there. The academic field that studies these systems is called *machine learning*. Our journey will focus on the current *wunderkind* of learning systems that has risen to great prominence: *learning from examples*. Even more specifically, we will mostly be concerned with *supervised learning from examples*. What is that? Here's an example. I start by giving you several photos of two animals you've never seen before—with apologies to Dr. Seuss, they might be a Lorax or a Who—and then I tell you which animal is in which photo. If I give you a new, unseen photo you might be able to tell me the type of animal in it. Congratulations, *you're doing great*! You just performed supervised learning from examples. When a computer is coaxed to learn from examples, the examples are presented a certain way. Each example is measured on a common group of attributes and we record the values for each attribute on each example. Huh?

Imagine—or glance at Figure 1.1—a cartoon character running around with a basket of different measuring sticks which, when held up to an object, return some characteristic of that object, such as *this vehicle has four wheels*, *this person has brown hair*, *the temperature of that tea is 180° F*, and so on *ad nauseam* (that's an archaic way of saying *until you're sick of my examples*).

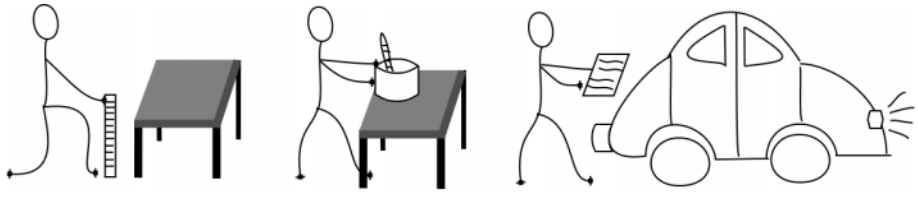


Figure 1.1 Humans have an insatiable desire to measure all sorts of things.

1.2.1 Features

Let's get a bit more concrete. For example—a meta-example, if you will—a dataset focused on human medical records might record several relevant values for each patient, such as height, weight, sex, age, smoking history, systolic and diastolic (that's the high and low numbers) blood pressures, and resting heart rate. The different people represented in the dataset are our examples. The biometric and demographic characteristics are our attributes.

We can capture this data very conveniently as in Table 1.1.

Table 1.1 A simple biomedical data table. Each row is an example. Each column contains values for a given attribute. Together, each attribute-value pair is a feature of an example.

patient id	height	weight	sex	age	smoker	hr	sys bp	dia bp
007	5'2"	120	M	11	no	75	120	80
2139	5'4"	140	F	41	no	65	115	75
1111	5'11"	185	M	41	no	52	125	75

Notice that each example—each row—is measured on the same attributes shown in the header row. The values of each attribute run down the respective columns.

We call the rows of the table the *examples* of the dataset and we refer to the columns as the *features*. Features are the measurements or values of our attributes. Often, people use "features" and "attributes" as synonyms describing the same thing; what they are referring to are the column of values. Still, some people like to distinguish among three concepts: *what-is-measured*, *what-the-value-is*, and *what-the-measured-value-is*. For those strict folks, the first is an attribute, the second is a value, and the last is a feature—an attribute and a value paired together. Again, we'll mostly follow the typical conversational usage and call the columns *features*. If we are specifically talking about *what-is-measured*, we'll stick with the term *attribute*. You will inevitably see both, used both ways, when you read about machine learning.

Let's take a moment and look at the types of values our attributes—what is measured—can take. One type of value distinguishes between different groups of people. We might see such groups in a census or an epidemiological medical study—for example, sex {*male*, *female*} or a broad record of ethnic-cultural-genetic heritage {*African*, *Asian*, *European*, *Native American*, *Polynesian*}. Attributes like these are called discrete, symbolic, categorical, or nominal attributes, but we are *not* going to stress about those names. If you struggled with those in a social science class, you are free to give a hearty huzzah.

Here are two important, or at least practical, points about categorical data. One point is that these values are discrete. They take a small, limited number of possibilities that typically represent one of several options. You're right that small and several are relative terms—just go with it. The second point is that the information in those attributes can be recorded in two distinct ways:

- As a single feature that takes one value for each option, *or*
- As several features, one per option, where one, and only one, of those features is marked as *yes* or *true* and the remainder are marked as *no* or *false*.

Here's an example. Consider

Name	Sex
Mark	Male
Barb	Female
Ethan	Male

versus:

Name	Sex is Female	Sex is Male
Mark	No	Yes
Barb	Yes	No
Ethan	No	Yes

If we had a column for community type in a census, the values might be Urban, Rural, and Suburban with three possible values. If we had the expanded, multicolumn form, it would take up three columns. Generally, we aren't motivated or worried about table size here. What matters is that some learning methods are, shall we say, particular in preferring one form or the other. There are other details to point out, but we'll save them for later.

Some feature values can be recorded and operated on as numbers. We may lump them together under the term *numerical* features. In other contexts, they are known as *continuous* or, depending on other details, *interval* or *ratio* values. Values for attributes like height and weight are typically recorded as decimal numbers. Values for attributes like age and blood pressure are often recorded as whole numbers. Values like counts—say, how many wheels are on a vehicle—are strictly whole numbers. Conveniently, we can perform arithmetic ($+, -, \times, /$) on these. While we *can* record categorical data as numbers, we can't necessarily perform *meaningful* numerical calculations directly on those values. If two states—say, Pennsylvania and Vermont—are coded as 2 and 14, it probably makes no sense to perform arithmetic on those values. There is an exception: if, by design, those values *mean* something beyond a unique identifier, we might be able to do some or all of the maths. For extra credit, you can find some meaning in the state values I used where mathematics would make sense.

1.2.2 Target Values and Predictions

Let's shift our focus back to the list of biomedical attributes we gathered. As a reminder, the column headings were height, weight, sex, age, smoker, heart rate, systolic blood pressure, and diastolic blood pressure. These attributes might be useful data for a health care provider trying to assess the likelihood of a patient developing cardiovascular heart. To do so, we would need another piece of information: did these folks develop heart disease?

If we have that information, we can add it to the list of attributes. We could capture and record the idea of "developing heart disease" in several different ways. Did the patient:

- Develop any heart disease within ten years: yes/no
- Develop X-level severity heart disease within ten years: None or Grade I, II, III
- Show some level of a specific indicator for heart disease within ten years: percent of coronary artery blockage

We could tinker with these questions based on resources at our disposal, medically relevant knowledge, and the medical or scientific puzzles we want to solve. Time is a precious resource; we might not have ten years to wait for an outcome. There might be medical knowledge about what percent of blockage is a critical amount. We could modify the time horizon or come up with different attributes to record.

In any case, we can pick a concrete, measurable target and ask, "Can we find a predictive relationship between the attributes we have *today* and the outcome that we will see *at some future time*?" We are literally trying to predict the future—maybe ten years from now—from things we know today. We call the concrete outcome our *target feature* or simply our *target*. If our target is a category like $\{sick, healthy\}$ or $\{None, I, II, III\}$, we call the process of learning the relationship *classification*. Here, we are using the term *classification* in the sense of finding the different classes, or categories, of a possible outcome. If the target is a smooth sweeping of numerical values, like the usual decimal numbers from elementary school $\{27.2, 42.0, 3.14159, -117.6\}$, we call the process *regression*. If you want to know why, go and google *Galton regression* for the history lesson.

We now have some handy terminology in our toolbox: most importantly *features*, both either *categorical* or *numerical*, and a *target*. If we want to emphasize the features being used to predict the future unknown outcome, we may call them *input features* or *predictive features*. There are a few issues I've swept under the carpet. In particular, we'll address some alternative terminology at the end of the chapter.

1.3 Putting the Machine in Machine Learning

I want you to create a mental image of a factory machine. If you need help, glance at Figure 1.2. On the left-hand side, there is a conveyor belt that feeds inputs into the machine. On the right-hand side, the machine spits out outputs which are words or numbers. The words might be *cat* or *dog*. The numbers might be $\{0, 1\}$ or $\{-2.1, 3.7\}$. The machine itself is a big hulking box of metal. We can't really see what happens on the inside. But we can see a control panel on the side of the machine, with an operator's seat in front of it. The control panel has some knobs we can set to numerical values and some switches we can flip on and off. By adjusting the knobs and switches, we can make different products appear on the right-hand side of the machine, depending on what came in the left-hand side. Lastly, there is a small side tray beside the operator's chair. The tray can be used to feed additional information, that is not easily captured by knobs and switches, into the machine. Two quick notes for the skeptical reader: our knobs *can* get us

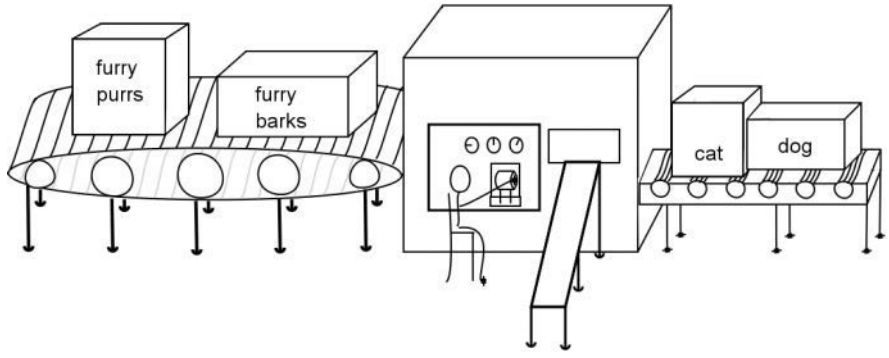


Figure 1.2 Descriptions go in. Categories or other values come out. We can adjust the machine to improve the relationship between the inputs and outputs.

arbitrarily small and large values ($-\infty$ to ∞, if you insist) and we don't *strictly* need on/off switches, since knobs set to precisely 0 or 1 could serve a similar purpose.

Moving forward, this factory image is a great entry point to understand how *learning algorithms* figure out relationships between features and a target. We can sit down as the machine operator and press a magic—probably green—*go* button. Materials roll in the machine from the left and *something* pops out on the right. Our curiosity gets the best of us and we twiddle the dials and flip the switches. Then, *different* things pop out the right-hand side. We turn up KnobOne and the machine pays more attention to the sounds that the input object makes. We turn down KnobTwo and the machine pays less attention to the number of limbs on the input object. If we have a goal—if there is some *known* product we'd like to see the machine produce—hopefully our knob twiddling gets us closer to it.

Learning algorithms are formal rules for how we manipulate our controls. After seeing examples where the target is known, learning algorithms take a given big-black-box and use a well-defined method to set the dials and switches to *good* values. While *good* can be quite a debatable quality in an ethics class, here we have a gold standard: our known target values. If they don't match, we have a problem. The algorithm adjusts the control panel settings so our *predicted outs* match the *known outs*. Our name for the machine is a *learning model* or just a *model*.

An example goes into the machine and, based on the settings of the knobs and switches, a class or a numerical value pops out. Do you want a different output value from the same input ingredients? Turn the knobs to different settings or flip a switch. One machine has a *fixed* set of knobs and switches. The knobs can be turned, but we can't add new knobs. If we add a knob, we have a *different machine*. Amazingly, the differences between knob-based learning methods boil down to answering three questions:

1. What knobs and switches are there: what is on the control panel?
2. How do the knobs and switches interact with an input example: what are the inner workings of the machine?

3. How do we set the knobs from some *known* data: how do we align the inputs with the outputs we want to see?

Many learning models that we will discuss can be described as machines with knobs and switches—with no need for the additional side input tray. Other methods require the side tray. We'll hold off discussing that more thoroughly, but if your curiosity is getting the best of you, flip to the discussion of *nearest neighbors* in Section 3.5.

Each learning method—which we imagine as a black-box factory machine and a way to set knobs on that machine—is really an *implementation* of an *algorithm*. For our purposes, an algorithm is a finite, well-defined sequence of steps to solve a task. An implementation of an algorithm is the specification of those steps in a particular programming language. The algorithm is the abstract idea; the implementation is the concrete existence of that idea—at least, as concrete as a computer program can be! In reality, algorithms *can* also be implemented in hardware—just like our factory machines; it's far easier for us to work with software.

1.4 Examples of Learning Systems

Under the umbrella of supervised learning from examples, there is a major distinction between two things: predicting values and predicting categories. Are we trying (1) to relate the inputs to one of a few possible categories indicated by discrete symbols, or (2) to relate the inputs to a more-or-less continuous range of numerical values? In short, is the target categorical or numerical? As I mentioned, predicting a category is called *classification*. Predicting a numerical value is called *regression*. Let's explore examples of each.

1.4.1 Predicting Categories: Examples of Classifiers

Classifiers are models that take input examples and produce an output that is one of a small number of possible groups or classes:

1. **Image Classification.** From an input image, output the animal (e.g., cat, dog, zebra) that is in the image, or none if there is no animal present. Image analysis of this sort is at the intersection of machine learning and computer vision. Here, our inputs will be a large collection of image files. They might be in different formats (png, jpeg, etc.). There may be substantial differences between the images: (1) they might be at different scales, (2) the animals may be centered or cut-off on the edges of the frame, and (3) the animals might be blocked by other things (e.g., a tree). These all represent challenges for a learning system—and for learning researchers! But, there are some nice aspects to image recognition. Our concept of *cat* and what images constitute a cat is fairly fixed. Yes, there could be blurred boundaries with animated cats—Hobbes, Garfield, Heathcliff, I'm looking at you—but short of evolutionary time scales, cat is a pretty static concept. We don't have a moving target: the *relationship* between the images and our concept of *cat* is fixed over time.