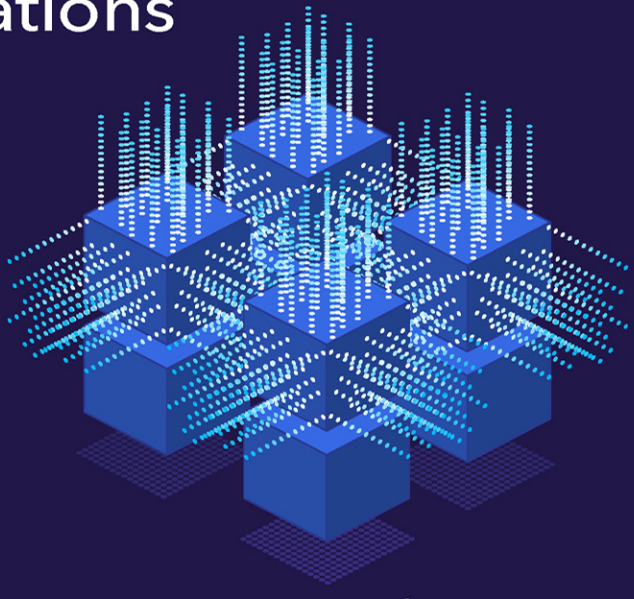
ADDISON WESLEY DATA & ANALYTICS SERIES

MACHINE LEARNING IN PRODUCTION

Developing and Optimizing Data Science Workflows and Applications



ANDREW KELLEHER | ADAM KELLEHER

Machine Learning in Production

Machine Learning in Production

Developing and Optimizing Data Science Workflows and Applications

Andrew Kelleher
Adam Kelleher

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

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2018954331

ISBN-13: 978-0-13-411654-9
ISBN-10: 0-13-411654-2

1 19

❖

This book is dedicated to our lifelong mentor, William F. Walsh III. We could never thank you enough for all the years of support and encouragement.

❖

This page intentionally left blank

Contents

Foreword

This pragmatic book introduces both machine learning and data science, bridging gaps between data scientist and engineer, and helping you bring these techniques into production. It helps ensure that your efforts actually solve your problem, and offers unique coverage of real-world optimization in production settings. This book is filled with code examples in Python and visualizations to illustrate concepts in algorithms. Validation, hypothesis testing, and visualization are introduced early on as these are all key to ensuring that your efforts in data science are actually solving your problem. Part III of the book is unique among data science and machine learning books because of its focus on real-world concerns in optimization. Thinking about hardware, infrastructure, and distributed systems are all steps to bringing machine learning and data science techniques into a production setting.

Andrew and Adam Kelleher bring their experience in engineering and data science, respectively, from their work at BuzzFeed. The topics covered and where to provide breadth versus depth are informed by their real-world experience solving problems in a large production environment. Algorithms for comparison, classification, clustering, and dimensionality reduction are all presented with examples of specific problems that can be solved with each. Explorations into more advanced topics like Bayesian networks or deep learning are provided after the framework for basic machine learning tasks is laid.

This book is a great addition to the Data & Analytics Series. It provides a well-grounded introduction to data science and machine learning with a focus on problem-solving. It should serve as a great resource to any engineer or "accidental programmer" with a more traditional math or science background looking to apply machine learning to their production applications and environment.

—Paul Dix, series editor

This page intentionally left blank

Preface

Most of this book was written while Andrew and Adam were working together at BuzzFeed. Adam was a data scientist, Andrew was an engineer, and they spent a good deal of time working together on the same team! Given that they're identical twins of triplets, it was confusing and amusing for everyone involved.

The idea for this book came after PyGotham in New York City in August 2014. There were several talks relating to the relatively broadly defined field of "data science." What we noticed was that many data scientists start their careers driven by the curiosity and excitement of learning new things. They discover new tools and often have a favorite technique or algorithm. They'll apply that tool to the problem they're working on. When you have a hammer, every problem looks like a nail. Often, as with neural networks (discussed in Chapter 14), it's more like a pile driver. We wanted to push past the hype of data science by giving data scientists, especially at the time they're starting their careers, a whole tool box. One could argue the context and error analysis tools of Part I are actually more important than the advanced techniques discussed in Part III. In fact, they're a major motivator in writing this book. It's very unlikely a choice of algorithm will be successful if its signal is trumped by its noise, or if there is a high amount of systematic error. We hope this book provides the right tools to take on the projects our readers encounter, and to be successful in their careers.

There's no lack of texts in machine learning or computer science. There are even some decent texts in the field of data science. What we hope to offer with this book is a comprehensive and rigorous entry point to the field of data science. This tool box is slim and driven by our own experience of what is useful in practice. We try to avoid opening up paths that lead to research-level problems. If you're solving research-level problems as a junior data scientist, you've probably gone out of scope.

There's a critical side of data science that is separate from machine learning: engineering. In Part III of this text we get into the engineering side. We discuss the problems you're likely to encounter and give you the fundamentals you'll need to overcome them. Part III is essentially a Computer Science 201-202 crash course. Once you know what you're building, you still have to address many considerations on the path to production. This means understanding your toolbox from the perspective of the tools.

Who This Book Is For

For the last several years there has been a serious demand for good engineers. During the Interactive session of SXSW in 2008 we heard the phrase "accidental developer" coined for the first time. It was used to describe people playing the role of engineer without having had formal training. They simply happened into that position and began filling it out of necessity. More than a decade later we still see this demand for developers, but it's also begun to extend to data scientists. Who fills the role of the "accidental data scientist"? Well, it's usually developers. Or physics undergraduates. Or math majors. People who haven't had much if any formal training in all the disciplines required of a data scientist. People who don't lack for technical training, and have all the prerequisite curiosity and ambition to succeed. People in need of a tool box.

This book is intended to be a crash course for those people. We run through a basic procedure for taking on most data science tasks, encouraging data scientists to use their data set, rather than the tools of the day, as the starting point. Data-driven data science is key to success. The big open secret

of data science is that while modeling is important, the bread and butter of data science is simple queries, aggregations, and visualizations. Many industries are in a place where they're accumulating and seeing data for the very first time. There is value to be delivered quickly and with minimal complexity.

Modeling is important, but hard. We believe in applying the principles of agile development to data science. We talk about this a lot in Chapter 2. Start with a minimal solution: a simple heuristic based on a data aggregation, for example. Improve the heuristic with a simple model when your data pipeline is mature and stable. Improve the model when you don't have anything more important to do with your time. We'll provide realistic case studies where this approach is applied.

What This Book Covers

We start this text by providing you with some background on the field of data science. Part I, "Principles of Framing," includes Chapter 1, "The Role of the Data Scientist," which serves as a starting point for your understanding of the data industry.

Chapter 2, "Project Workflow," sets the context for data science by describing agile development. It's a philosophy that helps keep scope small, and development efficient. It can be hard to keep yourself from trying out the latest machine learning framework or tools offered by cloud platforms, but it pays off in the long run.

Next, in Chapter 3, "Quantifying Error," we provide you with a basic introduction to error analysis. Much of data science is reporting simple statistics. Without understanding the error in those statistics, you're likely to come to invalid conclusions. Error analysis is a foundational skill and important enough to be the first item in your tool kit.

We continue in Chapter 4, "Data Encoding and Preprocessing," by discovering a few of the many ways of encoding the real world in the form of data. Naturally this leads us to ask data-driven questions about the real world. The framework for answering these questions is hypothesis testing, which we provide a foundation for in Chapter 5, "Hypothesis Testing."

At this point, we haven't seen many graphs, and our tool kit is lacking in communicating our results to the outside (nontechnical) world. We aim to resolve this in Chapter 6, "Data Visualization," where we learn many approaches to it. We keep the scope small and aim to mostly either make plots of quantities we know how to calculate errors for, or plots that resolve some of the tricky nuances of data visualization. While these tools aren't as flashy as interactive visualizations in d3 (which are worth learning!), they serve as a solid foundational skill set for communicating results to nontechnical audiences.

Having provided the basic tools for working with data, we move on to more advanced concepts in Part II, "Algorithms and Architecture." We start with a brief introduction to data architectures in Chapter 7, "Data Architectures," and an introduction to basic concepts in machine learning in Chapter 8, "Comparison." You now have some very handy methods for measuring the similarities of objects.

From there, we have some tools to do basic machine learning. In Chapter 9, "Regression," we introduce regression and start with one of the most important tools: linear regression. It's odd to start with such a simple tool in the age of neural networks and nonlinear machine learning, but

linear regression is outstanding for several reasons. As we'll detail later, it's interpretable, stable, and often provides an excellent baseline. It can describe nonlinearities with some simple tricks, and recent results have shown that polynomial regression (a simple modification of linear regression) can outperform deep feedforward networks on typical applications!

From there, we describe one more basic workhorse of regression: the random forest. These are nonlinear algorithms that rely on a statistical trick, called "bagging," to provide excellent baseline performance for a wide range of tasks. If you want a simple model to start a task with and linear regression doesn't quite work for you, random forest is a nice candidate.

Having introduced regression and provided some basic examples of the machine learning workflow, we move on to Chapter 10, "Classification and Clustering." We see a variety of methods that work on both vector and graph data. We use this section to provide some basic background on graphs and an abbreviated introduction to Bayesian inference. We dive into Bayesian inference and causality in the next chapter.

Our Chapter 11, "Bayesian Networks," is both unconventional and difficult. We take the view that Bayesian networks are most intuitive (though not necessarily easiest) from the viewpoint of causal graphs. We lay this intuition as the foundation for our introduction of Bayesian networks and come back to it in later sections as the foundation for understanding causal inference. In the Chapter 12, "Dimensional Reduction and Latent Variable Models," we build off of the foundation of Bayesian networks to understand PCA and other variants of latent factor models. Topic modeling is an important example of a latent variable model, and we provide a detailed example on the newgroups data set.

As the next to last data-focused chapter, we focus on the problem of causal inference in Chapter 13, "Causal Inference." It's hard to understate the importance of this skill. Data science typically aims to inform how businesses act. The assumption is that the data tells you something about the outcomes of your actions. That can only be true if your analysis has captured causal relationships and not just correlative ones. In that sense, understanding causation underlies much of what we do as data scientists. Unfortunately, with a view toward minimizing scope, it's also too often the first thing to cut. It's important to balance stakeholder expectations when you scope a project, and good causal inference can take time. We hope to empower data scientists to make informed decisions and not to accept purely correlative results lightly.

Finally, in the last data-focused chapter we provide a section to introduce some of the nuances of more advanced machine learning techniques in Chapter 14, "Advanced Machine Learning." We use neural networks as a tool to discuss overfitting and model capacity. The focus should be on using as simple a solution as is available. Resist the urge to start with neural networks as a first model. Simple regression techniques almost always provide a good enough baseline for a first solution.

Up to this point, the platform on which all of the data science happens has been in the background. It's where you do the data science and is not the primary focus. Not anymore. In the last part of this book, Part III, "Bottlenecks and Optimizations," we go in depth on hardware, software, and the systems they make up.

We start with a comprehensive look at hardware in Chapter 15, "Hardware Fundamentals." This provides a tool box of basic resources we have to work with and also provides a framework to discuss

the constraints under which we must operate. These constraints are physical limitations on what is possible, and those limitations are realized in the hardware.

Chapter 16, "Software Fundamentals," provides the fundamentals of software and a basic description of data logistics with a section on extract-transfer/transform-load, commonly known as ETL.

Next, we give an overview of design considerations for architecture in Chapter 17, "Architecture Fundamentals." Architecture is the design for how your whole system fits together. It includes the components for data storage, data transfer, and computation, as well as how they all communicate with one another. Some architectures are more efficient than others and objectively do their jobs better than others. Still, a less efficient solution might be more practical, given constraints on time and resources. We hope to provide enough context so you can make informed decisions. Even if you're a data scientist and not an engineer, we hope to provide enough knowledge so you can at least understand what's happening with your data platform.

We then move on to some more advanced topics in engineering. Chapter 18, "The CAP Theorem," covers some fundamental bounds on database performance. Finally, we discuss how it all fits together in the last chapter, which is on network topology: Chapter 19, "Logical Network Topological Nodes."

Going Forward

We hope that not only can you do the machine learning side of data science, but you can also understand what's possible in your own data platform. From there, you can understand what you might need to build and find an efficient path for building out your infrastructure as you need to. We hope that with a complete toolbox, you're free to realize that the tools are only a part of the solution. They're a means to solve real problems, and real problems always have resource constraints.

If there's one lesson to take away from this book, it's that you should always direct your resources toward solving the problems with the highest return on investment. Solving your problem is a real constraint. Occasionally, it might be true that nothing but the best machine learning models can solve it. The question to ask, then, is whether that's the best problem to solve or if there's a simpler one that presents a lower-risk value proposition.

Finally, while we would have liked to have addressed all aspects of production machine learning in this book, it currently exists more as a production data science text. In subsequent editions, we intend to cover omissions, especially in the area of machine learning infrastructure. This new material will include methods to parallelize model training and prediction; the basics of Tensorflow, Apache Airflow, Spark, and other frameworks and tools; the details of several real machine learning platforms, including Uber's Michelangelo, Google's TFX, and our own work on similar systems; and avoiding and managing coupling in machine learning systems. We encourage the reader to seek out the many books, papers, and blog posts covering these topics in the meantime, and to check for updates on the book's website at adamkelleher.com/ml_book.

We hope you'll enjoy learning these tools as much as we did, and we hope this book will save you time and effort in the long run.

About the Authors

Andrew Kelleher is a staff software engineer and distributed systems architect at Venmo. He was previously a staff software engineer at BuzzFeed and has worked on data pipelines and algorithm implementations for modern optimization. He graduated with a BS in physics from Clemson University. He runs a meetup in New York City that studies the fundamentals behind distributed systems in the context of production applications, and was ranked one of FastCompany's most creative people two years in a row.

Adam Kelleher wrote this book while working as principal data scientist at BuzzFeed and adjunct professor at Columbia University in the City of New York. As of May 2018, he is chief data scientist for research at Barclays and teaches causal inference and machine learning products at Columbia. He graduated from Clemson University with a BS in physics, and has a PhD in cosmology from University of North Carolina at Chapel Hill.

This page intentionally left blank

13

Causal Inference

13.1 Introduction

We've introduced a couple of machine-learning algorithms and suggested that they can be used to produce clear, interpretable results. You've seen that logistic regression coefficients can be used to say how much more likely an outcome will occur in conjunection with a feature (for binary features) or how much more likely an outcome is to occur per unit increase in a variable (for real-valued features). We'd like to make stronger statements. We'd like to say "If you increase a variable by a unit, then it will have the effect of making an outcome more likely."

These two interpretations of a regression coefficient are so similar on the surface that you may have to read them a few times to take away the meaning. The key is that in the first case, we're describing what usually happens in a system that we observe. In the second case, we're saying what will happen if we intervene in that system and disrupt it from its normal operation.

After we go through an example, we'll build up the mathematical and conceptual machinery to describe interventions. We'll cover how to go from a Bayesian network describing observational data to one that describes the effects of an intervention. We'll go through some classic approaches to estimating the effects of interventions, and finally we'll explain how to use machine-learning estimators to estimate the effects of interventions.

If you imagine a binary outcome, such as "I'm late for work," you can imagine some features that might vary with it. Bad weather can cause you to be late for work. Bad weather can also cause you to wear rain boots. Days when you're wearing rain boots, then, are days when you're more likely be late for work. If you look at the correlation between the binary feature "wearing rain boots" and the outcome "I'm late for work," you'll find a positive relationship. It's nonsense, of course, to say that wearing rain boots causes you to be late for work. It's just a proxy for bad weather. You'd never recommend a policy of "You shouldn't wear rain boots, so you'll be late for work less often." That would be reasonable only if "wearing rain boots" was *causally* related to "being late for work." As an intervention to prevent lateness, not wearing rain boots doesn't make any sense.

In this chapter, you'll learn the difference between correlative (rain boots and lateness) and causal (rain and lateness) relationships. We'll discuss the gold standard for establishing causality: an experiment. We'll also cover some methods to discover causal relationships in cases when you're not able to run an experiment, which happens often in realistic settings.

13.2 Experiments

The case that might be familiar to you is an AB test. You can make a change to a product and test it against the original version of the product. You do this by randomly splitting your users into two groups. The group membership is denoted by D, where $D = 1$ is the group that experiences the new change (the test group), and $D = 0$ is the group that experiences the original version of the product (the control group). For concreteness, let's say you're looking at the effect of a recommender system change that recommends articles on a website. The control group experiences the original algorithm, and the test group experiences the new version. You want to see the effect of this change on total pageviews, Y.

You'll measure this effect by looking at a quantity called the *average treatment effect* (ATE). The ATE is the average difference in the outcome between the test and control groups, $E_{test}[Y] - E_{control}[Y]$, or $\delta_{naive} = E[Y|D = 1] - E[Y|D = 0]$. This is the "naive" estimator for the ATE since here we're ignoring everything else in the world. For experiments, it's an unbiased estimate for the true effect.

A nice way to estimate this is to do a regression. That lets you also measure error bars at the same time and include other covariates that you think might reduce the noise in Y so you can get more precise results. Let's continue with this example.

```
import numpy as np
import pandas as pd

N = 1000

x = np.random.normal(size=N)
d = np.random.binomial(1., 0.5, size=N)
y = 3. * d + x + np.random.normal()

X = pd.DataFrame({'X': x, 'D': d, 'Y': y})
```

Here, we've randomized D to get about half in the test group and half in the control. X is some other covariate that causes Y, and Y is the outcome variable. We've added a little extra noise to Y to just make the problem a little noisier.

You can use a regression model $Y = \beta_0 + \beta_1 D$ to estimate the expected value of Y, given the covariate D, as $E[Y|D] = \beta_0 + \beta_1 D$. The β_0 piece will be added to $E[Y|D]$ for all values of D (i.e., 0 or 1). The β_1 part is added only when $D = 1$ because when $D = 0$, it's multiplied by zero. That means $E[Y|D = 0] = \beta_0$ when $D = 0$ and $E[Y|D = 1] = \beta_0 + \beta_1$ when $D = 1$. Thus, the β_1 coefficient is going to be the difference in average Y values between the $D = 1$ group and the $D = 0$ group, $E[Y|D = 1] - E[Y|D = 0] = \beta_1$! You can use that coefficient to estimate the effect of this experiment.

When you do the regression of Y against D, you get the result in Figure 13.1.

```
from statsmodels.api import OLS
X['intercept'] = 1.
model = OLS(X['Y'], X[['D', 'intercept']])
result = model.fit()
result.summary()
```

OLS Regression Results

Dep. Variable:	Y	**R-squared:**	0.560
Model:	OLS	**Adj. R-squared:**	0.555
Method:	Least Squares	**F-statistic:**	124.5
Date:	Sun, 08 Apr 2018	**Prob (F-statistic):**	3.79e-19
Time:	22:28:01	**Log-Likelihood:**	-180.93
No. Observations:	100	**AIC:**	365.9
Df Residuals:	98	**BIC:**	371.1
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
D	3.3551	0.301	11.158	0.000	2.758	3.952
intercept	-0.1640	0.225	-0.729	0.468	-0.611	0.283

Omnibus:	0.225	**Durbin-Watson:**	1.866
Prob(Omnibus):	0.894	**Jarque-Bera (JB):**	0.360
Skew:	0.098	**Prob(JB):**	0.835
Kurtosis:	2.780	**Cond.No.**	2.78

Figure 13.1 The regression for $Y = \beta_0 + \beta_1 D$

Why did this work? Why is it okay to say the effect of the experiment is just the difference between the test and control group outcomes? It seems obvious, but that intuition will break down in the next section. Let's make sure you understand it deeply before moving on.

Each person can be assigned to the test group or the control group, but not both. For a person assigned to the test group, you can talk hypothetically about the value their outcome would have had, had they been assigned to the control group. You can call this value Y^0 because it's the value Y would take if D had been set to 0. Likewise, for control group members, you can talk about a hypothetical Y^1. What you really want to measure is the difference in outcomes $\delta = Y^1 - Y^0$ for each person. This is impossible since each person can be in only one group! For this reason, these Y^1 and Y^0 variables are called *potential outcomes*.

If a person is assigned to the test group, you measure the outcome $Y = Y^1$. If a person is assigned to the control group, you measure $Y = Y^0$. Since you can't measure the individual effects, maybe you can measure population level effects. We can try to talk instead about $E[Y^1]$ and $E[Y^0]$. We'd like $E[Y^1] = E[Y|D = 1]$ and $E[Y^0] = E[Y|D = 0]$, but we're not guaranteed that that's true. In the

recommender system test example, what would happen if you assigned people with higher Y^0 pageview counts to the test group? You might measure an effect that's larger than the true effect!

Fortunately, you randomize D to make sure it's independent of Y^0 and Y^1. That way, you're sure that $E[Y^1] = E[Y|D = 1]$ and $E[Y^0] = E[Y|D = 0]$, so you can say that $\delta = E[Y^1 - Y^0] = E[Y|D = 1] - E[Y|D = 0]$. When other factors can influence assignment, D, then you can no longer be sure you have correct estimates! This is true in general when you don't have control over a system, so you can't ensure D is independent of all other factors.

In the general case, D won't just be a binary variable. It can be ordered, discrete, or continuous. You might wonder about the effect of the length of an article on the share rate, about smoking on the probability of getting lung cancer, of the city you're born in on future earnings, and so on.

Just for fun before we go on, let's see something nice you can do in an experiment to get more precise results. Since we have a co-variate, X, that also causes Y, we can account for more of the variation in Y. That makes our predictions less noisy, so our estimates for the effect of D will be more precise! Let's see how this looks. We regress on both D and X now to get Figure 13.2.

Dep. Variable:	Y	**R-squared:**	0.754
Model:	OLS	**Adj. R-squared:**	0.749
Method:	Least Squares	**F-statistic:**	148.8
Date:	Sun, 08 Apr 2018	**Prob (F-statistic):**	2.75e-30
Time:	22:59:08	**Log-Likelihood:**	-151.76
No. Observations:	100	**AIC:**	309.5
Df Residuals:	97	**BIC:**	317.3
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
D	3.2089	0.226	14.175	0.000	2.760	3.658
X	1.0237	0.117	8.766	0.000	0.792	1.256
intercept	0.0110	0.170	0.065	0.949	-0.327	0.349

Omnibus:	2.540	**Durbin-Watson:**	1.648
Prob(Omnibus):	0.281	**Jarque-Bera (JB):**	2.362
Skew:	-0.293	**Prob(JB):**	0.307
Kurtosis:	2.528	**Cond.No.**	2.81

Figure 13.2 The regression for $Y = \beta_0 + \beta_1 D + \beta_2 X$

Notice that the R^2 is much better. Also, notice that the confidence interval for D is much narrower! We went from a range of $3.95 - 2.51 = 1.2$ down to $3.65 - 2.76 = 0.89$. In short, finding covariates that account for the outcome can increase the precision of your experiments!

13.3 Observation: An Example

Let's look at an example of what happens when you don't make your cause independent of everything else. We'll use it to show how to build some intuition for how observation is different from intervention. Let's look at a simple model for the correlation between race, poverty, and crime in a neighborhood. Poverty reduces people's options in life and makes crime more likely. That makes poverty a cause of crime. Next, neighborhoods have a racial composition that can persist over time, so the neighborhood is a cause of racial composition. The neighborhood also determines some social factors, like culture and education, and so can be a cause of poverty. This gives us the causal diagram in Figure 13.3.

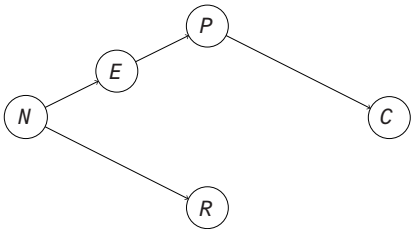


Figure 13.3 The neighborhood is a cause of its racial composition and poverty levels. The poverty level is a cause of crime.

Here, there is no causal relationship between race and crime, but you would find them to be correlated in observational data. Let's simulate some data to examine this.

```
N = 10000

neighborhood = np.array(range(N))

industry = neighborhood % 3

race = ((neighborhood % 3

         + np.random.binomial(3, p=0.2, size=N))) % 4

income = np.random.gamma(25, 1000*(industry + 1))

crime = np.random.gamma(100000. / income, 100, size=N)

X = pd.DataFrame({'$R$': race, '$I$': income, '$C$': crime,

                  '$E$': industry, '$N$': neighborhood})
```

Here, each data point will be a neighborhood. There are common historic reasons for the racial composition and the dominant industry in each neighborhood. The industry determines the income levels in the neighborhood, and the income level is inversely related with crime rates.

If you plot the correlation matrix for this data (Figure 13.4), you can see that race and crime are correlated, even though there is no causal relationship between them!

	C	*E*	*I*	*N*	*R*
C	1.000000	-0.542328	-0.567124	0.005518	-0.492169
E	-0.542328	1.000000	0.880411	0.000071	0.897789
I	-0.567124	0.880411	1.000000	-0.005650	0.793993
N	0.005518	0.000071	-0.005650	1.000000	-0.003666
R	-0.492169	0.897789	0.793993	-0.003666	1.000000

Figure 13.4 Raw data showing correlations between crime (C), industry (E), income (I), neighborhood (N), and race (R)

You can take a regression approach and see how you can interpret the regression coefficients. Since we know the right model to use, we can just do the right regression, which gives the results in Figure 13.5.

Generalized Linear Model Regression Results

Dep. Variable:	*C*	**No. Observations:**	10000
Model:	GLM	**Df Residuals:**	9999
Model Family:	Gamma	**Df Model:**	0
Link Function:	inverse_power	**Scale:**	1.53451766278
Method:	IRLS	**Long-Likelihood:**	-68812.
Date:	Sun, 06 Aug 2017	**Deviance:**	15138.
Time:	22:43:02	**Pearson chi2:**	1.53e+04
No. Iterations:	7		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
1/*I*	123.0380	1.524	80.726	0.000	120.051	126.025

Figure 13.5 Regression for crime against the inverse of income

```
from statsmodels.api import GLM
import statsmodels.api as sm

X['$1/I$'] = 1. / X['$I$']
model = GLM(X['$C$'], X[['$1/I$']], family=sm.families.Gamma())
result = model.fit()
result.summary()
```

From this you can see that when $1/I$ increases by a unit, the number of crimes increases by 123 units. If the crime units are in crimes per 10,000 people, this means 123 more crimes per 10,000 people.

This is a nice result, but you'd really like to know whether the result is causal. If it is causal, that means you can design a policy intervention to exploit the relationship. That is, you'd like to know if people earned more income, everything else held fixed, would there be less crime? If this were a causal result, you could say that if you make incomes higher (independent of everything else), then you can expect that for each unit decrease in $1/I$, you'll see 123 fewer crimes. What is keeping us from making those claims now?

You'll see that regression results aren't necessarily causal; let's look at the relationship between race and crime. We'll do another regression as shown here:

```
from statsmodels.api import GLM
import statsmodels.api as sm

races = {0: 'african-american', 1: 'hispanic',
         2: 'asian', 3: 'white'}
X['race'] = X['$R$'].apply(lambda x: races[x])
race_dummies = pd.get_dummies(X['race'])
X[race_dummies.columns] = race_dummies
model = OLS(X['$C$'], race_dummies)
result = model.fit()
result.summary()
```

Figure 13.6 show the result.

Here, you find a strong correlative relationship between race and crime, even though there's no causal relationship. You know that if we moved a lot of white people into a black neighborhood (holding income level constant), you should have no effect on crime. If this regression were causal, then you would. Why do you find a significant regression coefficient even when there's no causal relationship?

In this example, you went wrong because racial composition and income level were both caused by the history of each neighborhood. This is a case where two variables share a common cause. If you don't control for that history, then you'll find a spurious association between the two variables. What you're seeing is a general rule: when two variables share a common cause, they will be correlated (or, more generally, statistically dependent) even when there's no causal relationship between them.

Another nice example of this common cause problem is that when lemonade sales are high, crime rates are also high. If you regress crime on lemonade sales, you'd find a significant increase in crimes per unit increase in lemonade sales! Clearly the solution isn't to crack down on lemonade stands. As it happens, more lemonade is sold on hot days. Crime is also higher on hot days. The weather is a common cause of crime and lemonade sales. We find that the two are correlated even though there is no causal relationship between them.

The solution in the lemonade example is to control for the weather. If you look at all days where it is sunny and 95 degrees Fahrenheit, the effect of the weather on lemonade sales is constant. The

Dep. Variable:	*C*	**R-squared:**	0.262
Model:	OLS	**Adj. R-squared:**	0.262
Method:	Least Squares	**F-statistic:**	1184.
Date:	Sun, 06 Aug 2017	**Prob (F-statistic):**	0.00
Time:	22:59:47	**Log-Likelihood:**	-65878.
No. Observations:	10000	**AIC:**	1.318e+05
Df Residuals:	9996	**BIC:**	1.318e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
african-american	411.9718	3.395	121.351	0.000	405.317	418.626
asian	155.0682	3.020	51.350	0.000	149.149	160.988
hispanic	248.8263	3.066	81.159	0.000	242.817	254.836
white	132.0232	6.909	19.108	0.000	118.479	145.567

Omnibus:	2545.693	**Durbin-Watson:**	1.985
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	7281.773
Skew:	1.335	**Prob(JB):**	0.00
Kurtosis:	6.217	**Cond. No.**	2.29

Figure 13.6 Statistics highlighting relationships between race and crime

effect of weather and crime is also constant in the restricted data set. Any variance in the two must be because of other factors. You'll find that lemonade sales and crime no longer have a significant correlation in this restricted data set. This problem is usually called *confounding*, and the way to break confounding is to control for the confounder.

Similarly, if you look only at neighborhoods with a specific history (in this case the relevant variable is the dominant industry), then you'll break the relationship between race and income and so also the relationship between race and crime.

To reason about this more rigorously, let's look at Figure 13.3. We can see the source of dependence, where there's a path from *N* to *R* and a path from *N* through *E* and *P* to *C*. If you were able to break this path by holding a variable fixed, you could disrupt the dependence that flows along it. The result will be different from the usual observational result. You will have changed the dependencies in the graph, so you will have changed the joint distribution of all these variables.

If you intervene to set the income level in an area in a way that is independent of the dominant industry, you'll break the causal link between the industry and the income, resulting in the graph in Figure 13.7. In this system, you should find that the path that produces dependence between race and crime is broken. The two should be independent.

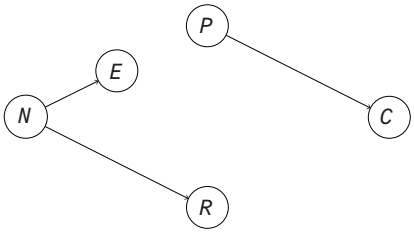


Figure 13.7 The result of an intervention, where you set the income level by direct intervention in a way that is independent of the dominant industry in the neighborhood

How can you do this controlling using only observational data? One way is just to restrict to subsets of the data. You can, for example, look only at industry 0 and see how this last regression looks.

```
X_restricted = X[X['$E$'] == 0]

races = {0: 'african-american', 1: 'hispanic',
         2: 'asian', 3: 'white'}
X_restricted['race'] = X_restricted['$R$'].apply(lambda x: races[x])
race_dummies = pd.get_dummies(X_restricted['race'])
X_restricted[race_dummies.columns] = race_dummies
model = OLS(X_restricted['$C$'], race_dummies)
result = model.fit()
result.summary()
```

This produces the result in Figure 13.8.

Now you can see that all of the results are within confidence of each other! The dependence between race and crime is fully explained by the industry in the area. In other words, in this hypothetical data set, crime is independent of race when you know what the dominant industry is in the area. What you have done is the same as the conditioning you did before.

Notice that the confidence intervals on the new coefficients are fairly wide compared to what they were before. This is because you've restricted to a small subset of your data. Can you do better, maybe by using more of the data? It turns out there's a better way to control for something than restricting the data set. You can just regress on the variables you'd like to control for!

```
from statsmodels.api import GLM
import statsmodels.api as sm

races = {0: 'african-american', 1: 'hispanic',
         2: 'asian', 3: 'white'}
X['race'] = X['$R$'].apply(lambda x: races[x])
race_dummies = pd.get_dummies(X['race'])
X[race_dummies.columns] = race_dummies

industries = {i: 'industry_{}'.format(i) for i in range(3)}
X['industry'] = X['$E$'].apply(lambda x: industries[x])
```

Dep. Variable:	*C*	**R-squared:**	0.001
Model:	OLS	**Adj. R-squared:**	0.000
Method:	Least Squares	**F-statistic:**	1.109
Date:	Sun, 06 Aug 2017	**Prob (F-statistic):**	0.344
Time:	23:19:14	**Log-Likelihood:**	-22708.
No. Observations:	3334	**AIC:**	4.542e+04
Df Residuals:	3330	**BIC:**	4.545e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
african-american	415.1116	5.260	78.914	0.000	404.798	425.425
asian	421.3615	12.326	34.185	0.000	397.194	445.529
hispanic	421.3907	6.239	67.536	0.000	409.157	433.624
white	484.8838	40.816	11.880	0.000	404.856	564.911

Omnibus:	538.823	**Durbin-Watson:**	1.947
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	943.390
Skew:	1.038	**Prob(JB):**	1.40e-205
Kurtosis:	4.575	**Cond. No.**	7.76

Figure 13.8 A hypothetical regression on race indicator variables predicting crime rates, but controlling for local industry using stratification of the data. There are no differences in expected crimes, controlling for industry.

```
industry_dummies = pd.get_dummies(X['industry'])
X[industry_dummies.columns] = industry_dummies

x = list(industry_dummies.columns)[1:] + list(race_dummies.columns)

model = OLS(X['$C$'], X[x])
result = model.fit()
result.summary()
```

Then, you get Figure 13.9 shows the result.

Here, the confidence intervals are much narrower, and you see there's still no significant association between race and income level: the coefficients are roughly equal. This is a causal regression result: you can now see that there would be no effect of an intervention to change the racial composition of neighborhoods. This simple example is nice because you can see what to control for, and you've measured the things you need to control for. How do you know what to

Dep. Variable:	*C*	**R-squared:**	0.331
Model:	OLS	**Adj. R-squared:**	0.331
Method:	Least Squares	**F-statistic:**	988.5
Date:	Sun, 06 Aug 2017	**Prob (F-statistic):**	0.00
Time:	23:29:24	**Log-Likelihood:**	-65483.
No. Observations:	10000	**AIC:**	1.310e+05
Df Residuals:	9994	**BIC:**	1.310e+05
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
industry_1	-215.1618	4.931	-43.638	0.000	-224.827	-205.497
industry_2	-278.9783	5.581	-49.984	0.000	-289.919	-268.038
african-american	415.2042	3.799	109.306	0.000	407.758	422.650
asian	418.0980	5.203	80.361	0.000	407.900	428.296
hispanic	423.5622	4.216	100.464	0.000	415.298	431.827
white	422.1700	6.530	64.647	0.000	409.369	434.971

Omnibus:	2493.579	**Durbin-Watson:**	1.991
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	7156.219
Skew:	1.306	**Prob(JB):**	0.00
Kurtosis:	6.218	**Cond. No.**	4.56

Figure 13.9 Statistics highlighting the relationship between race and industry from an OLS fit

control for in general? Will you always be able to do it successfully? It turns out it's very hard in practice, but sometimes it's the best you can do.

13.4 Controlling to Block Non-causal Paths

You just saw that you can take a correlative result and make it a causal result by controlling for the right variables. How do you know what variables to control for? How do you know that regression analysis will control for them? This section relies heavily on d-separation from Chapter 11. If that material isn't fresh, you might want to review it now.

You saw in the previous chapter that conditioning can break statistical dependence. If you condition on the middle variable of a path $X \rightarrow Y \rightarrow Z$, you'll break the dependence between X and Z that the path produces. If you condition on a confounding variable $X \leftarrow Z \rightarrow Y$, you can break the dependence between X and Y induced by the confounder as well. It's important to note that statistical dependence induced by other paths between X and Y is left unharmed by this

conditioning. If, for example, you condition on Z in the system in Figure 13.10, you'll get rid of the confounding but leave the causal dependence.

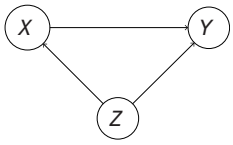


Figure 13.10 Conditioning on Z disrupts the confounding but leaves the causal statistical dependence between X and Y intact

If you had a general rule to choose which paths to block, you could eliminate all noncausal dependence between variables but save the causal dependence. The "back-door" criterion is the rule you're looking for. It tells you what set of variables, Z, you should control for to eliminate any noncausal statistical dependence between X_i and X_j. You should note a final nuance before introducing the criterion. If you want to know if the correlation between X_i and X_j is "causal," you have to worry about the direction of the effect. It's great to know, for example, that the correlation "being on vacation" and "being relaxed" is not confounded, but you'd really like to know whether "being on vacation" causes you to "be relaxed." That will inform a policy of going on vacation in order to be relaxed. If the causation were reversed, you couldn't take that policy.

With that in mind, the back-door criterion is defined relative to an ordered pair of variables, (X_i, X_j), where X_i will be the cause, and X_j will be the effect.

Definition 13.1. Back-Door Conditioning

It's sufficient to control for a set of variables, Z, to eliminate noncausal dependence for the effect of X_i on X_j in a causal graph, G, if

- No variable in Z is a descendant of X_i, and
- Z blocks every path between X_i and X_j that contains an arrow into X_i.

We won't prove this theorem, but let's build some intuition for it. First, let's examine the condition "no variable in Z is a descendant of X_i." You learned earlier that if you condition on a common effect of X_i and X_j, then the two variables will be conditionally dependent, even if they're normally independent. This remains true if you condition on any effect of the common effect (and so on down the paths). Thus, you can see that the first part of the back-door criterion prevents you from introducing extra dependence where there is none.

There is something more to this condition, too. If you have a chain like $X_i \to X_k \to X_j$, you see that X_k is a descendant of X_i. It's not allowed in Z. This is because if you condition on X_k, you'd block a causal path between X_i and X_j. Thus, you see that the first condition also prevents you from conditioning on variables that fall along causal paths.

The second condition says "Z blocks every path between X_i and X_j that contains an arrow into X_i." This part will tell us to control for confounders. How can you see this? Let's consider some cases where there is one or more node along the path between X_i and X_j and the path contains an arrow into X_i. If there is a collider along the path between X_i and X_j, then the path is already blocked, so

you just condition on the empty set to block that path. Next, if there is a fork along the path, like the path $X_i \leftarrow X_k \rightarrow X_j$, and no colliders, then you have typical confounding. You can condition on any node along the path that will block it. In this case, you add X_k to the set Z. Note that there can be no causal path from X_i to X_j with an arrow pointing into X_i because of the arrow pointing into X_i.

Thus, you can see that you're blocking all noncausal paths from X_i to X_j, and the remaining statistical dependence will be showing the causal dependence of X_j on X_i. Is there a way you can use this dependence to estimate the effects of interventions?

13.4.1 The G-formula

Let's look at what it really means to make an intervention. What it means is that you have a graph like in Figure 13.11.

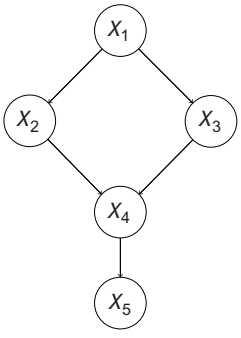


Figure 13.11 A pre-intervention causal graph. Data collected from this system reflects the way the world works when we just observe it.

You want to estimate the effect of X_2 on X_5. That is, you want to say "If I intervene in this system to set the value of X_2 to x_2, what will happen to X_5? To quantify the effect, you have to realize that all of these variables are taking on values that depend not only on their predecessors but also on noise in the system. Thus, even if there's a deterministic effect of X_2 on X_5 (say, raising the value of X_5 by exactly one unit), you can only really describe the value X_5 will take with a distribution of values. Thus, when you're estimating the effect of X_2 on X_5, what you really want is the distribution of X_5 when you intervene to set the value of X_2.

Let's look at what we mean by *intervene*. We're saying we want to ignore the usual effect of X_1 on X_2 and set the value of X_2 to x_2 by applying some external force (our action) to X_2. This removes the usual dependence between X_2 and X_1 and disrupts the downstream effect of X_1 on X_4 by breaking the path that passes through X_2. Thus, we'll also expect the marginal distribution between X_1 and X_4, $P(X_1, X_4)$ to change, as well as the distribution of X_1 and X_5! Our intervention can affect every variable downstream from it in ways that don't just depend on the value x_2. We actually disrupt other dependences.

You can draw a new graph that represents this intervention. At this point, you're seeing that the operation is very different from observing the value of $X_2 = x_2$, i.e., simply conditioning on

$X_2 = x_2$. This is because you're disrupting other dependences in the graph. You're actually talking about a new system described by the graph in Figure 13.12.

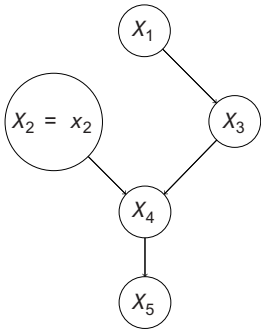


Figure 13.12 The graph representing the intervention $do(X_2 = x_2)$. The statistics of this data will be different from that in from the system in Figure 13.11

You need some new notation to talk about an intervention like this, so you'll denote $do(X_2 = x_2)$ the intervention where you perform this operation. This gives you the definition of the intervention, or *do-operation*.

Definition 13.2. Do-operation

We describe an intervention called the *do() operation* in a system described by a DAG, G as an operation where we do X_i by

- Delete all edges in G that point into X_i, and
- Set the value of X_i to x_i.

What does the joint distribution look like for this new graph? Let's use the usual factorization, and write the following:

$$P_{do(X_2=x_2)}(X_1, X_2, X_3, X_4, X_5) = P(X_5|X_4)P(X_4|X_2, X_3)P(X_3|X_1)\delta(X_2, x_2)P(X_1) \tag{13.1}$$

Here we've just indicated $P(X_2)$ by the δ-function, so $P(X_2) = 0$ if $X_2 \neq x_2$, and $P(X_2) = 1$ when $X_2 = x_2$. We're basically saying that when we intervene to set $X_2 = x_2$, we're sure that it worked. We can carry through that $X_2 = x_2$ elsewhere, like in the distribution for $P(X_4|X_2, X_3)$, but just replacing X_2 with $X_2 = x_2$, since the whole right-hand side is zero if $X_2 \neq x_2$.

Finally, let's just condition on the X_2 distribution to get rid of the weirdness on the right-hand side of this formula. We can write the following:

$$P_{do(X_2=x2)}(X_1, X_2, X_3, X_4, X_5|X_2) = P(X_5|X_4)P(X_4|X_2 = x_2, X_3)P(X_3|X_1)P(X_1) \tag{13.2}$$

However, this is the same as the original formula, divided by $P(X_2|X_1)$! To be precise,

$$P_{do(X_2=x2)}(X_1, X_2, X_3, X_4, X_5|X_2 = x_2) = \frac{P(X_1, X_2 = x_2, X_3, X_4, X_5)}{P(X_2 = x_2|X_1)} \tag{13.3}$$

Incredibly, this formula works in general. We can write the following:

$$P(X_1, ..., X_n|do(X_i = x_i)) = \frac{P(X_1, ..., X_n)}{P(X_i|Pa(X_i))} \tag{13.4}$$

This leads us to a nice general rule: the parents of a variable will always satisfy the back-door criterion! It turns out we can be more general than this even. If we marginalize out everything except X_i and X_j, we see the parents are the set of variables that control confounders.

$$P(X_j, Pa(X_i)|do(X_i = xi)) = \frac{P(X_j, X_i, Pa(X_i))}{P(X_i|Pa(X_i))} \tag{13.5}$$

It turns out (we'll state without proof) that you can generalize the parents to any set, Z, that satisfies the back door criterion.

$$P(X_j, Z|do(X_i = xi)) = \frac{P(X_j, X_i, Z)}{P(X_i|Z)} \tag{13.6}$$

You can marginalize Z out of this and use the definition of conditional probability to write an important formula, shown in Definition 13.3.

Definition 13.3. Robins G-Formula

$$P(X_j|do(X_i = x_i)) = \sum_z P(X_j|X_i, Z)P(Z)$$

This is a general formula for estimating the distribution of X_j under the intervention X_i. Notice that all of these distributions are from the *pre*-intervention system. This means you can use observational data to estimate the distribution of X_j under some hypothetical intervention!

There are a few critical caveats here. First, the term in the denominator of Equation 13.4, $P(X_i|Pa(X_i))$, must be nonzero for the quantity on the left side to be defined. This means you would have to have observed X_i taking on the value you'd like to set it to with your intervention. If you've never seen it, you can't say how the system might behave in response to it!

Next, you're assuming that you have a set Z that you can control for. Practically, it's hard to know if you've found a good set of variables. There can always be a confounder you have never thought to measure. Likewise, your way of controlling for known confounders might not do a very good job. You'll understand this second caveat more as you go into some machine learning estimators.

With these caveats, it can be hard to estimate causal effects from observational data. You should consider the results of a conditioning approach to be a provisional estimate of a causal effect. If you're sure you're not violating the first condition of the back-door criterion, then you can expect that you've removed some spurious dependence. You can't say for sure that you've reduced bias.

Imagine, for example, two sources of bias for the effect of X_i on X_j. Suppose you're interested in measuring an average value of X_j, $E_{do(X_i=x_i)}[X_j] = \mu_j$. Path A introduces a bias of $-\delta$, and path B introduces a bias of 2δ. If you estimate the mean without controlling for either path, you'll find $\mu_j^{(biased)} = \mu_j + 2\delta - \delta = \mu_j + \delta$. If you control for a confounder along path A, then you remove its contribution to the bias, which leaves $\mu_j^{(biased,A)} = \mu_j + 2\delta$. Now the bias is twice as large! The

problem, of course, is that the bias you corrected was actually pushing our estimate back toward its correct value. In practice, more controlling usually helps, but you can't be guaranteed that you won't find an effect like this.

Now that you have a good background in observational causal inference, let's see how machine-learning estimators can help in practice!

13.5 Machine-Learning Estimators

In general, you won't want to estimate the full joint distribution under an intervention. You may not even be interested in marginals. Usually, you're just interested in a difference in average effects.

In the simplest case, you'd like to estimate the expected difference in some outcome, X_j, per unit change in a variable you have control over, X_i. For example, you'd might like to measure $E[X_j|do(X_i = 1)] - E[X_j|do(X_i = 0)]$. This tells you the change in X_j you can expect on average when you set X_i to 1 from what it would be if X_i were set to 0.

Let's revisit the g-formula to see how can measure these kinds of quantities.

13.5.1 The G-formula Revisited

The g-formula tells you how to estimate a causal effect, $P(X_j|do(X_i = xi))$, using observational data and a set of variables to control for (based on our knowledge of the causal structure). It says this:

$$P(X_j|do(X_i = x_i)) = \sum_Z P(X_j|X_i, Z)P(Z) \tag{13.7}$$

If you take expectation values on each side (by multiplying by X_j and summing over X_j), then you find this:

$$E(X_j|do(X_i = x_i)) = \sum_Z E(X_j|X_i, Z)P(Z) \tag{13.8}$$

In practice, it's easy to estimate the first factor on the right side of this formula. If you fit a regression estimator using mean-squared error loss, then the best fit is just the expected value of X_j at each point (X_i, Z). As long as the model has enough freedom to accurately describe the expected value, you can estimate this first factor by using standard machine-learning approaches.

To estimate the whole left side, you need to deal with the $P(Z)$ term, as well as the sum. It turns out there's a simple trick for doing this. If your data was generated by drawing from the observational joint distribution, then your samples of Z are actually just draws from $P(Z)$. Then, if you replace the $P(Z)$ term by $1/N$ (for N samples) and sum over data points, you're left with an estimator for this sum. That is, you can make the substitution as follows:

$$E_N(X_j|do(X_i = x_i)) = \frac{1}{N}\sum_{k=1}^{N} E(X_j|X_i^{(k)}, Z^{(k)}), \tag{13.9}$$

where the (k) index runs over our data points, from 1 to N. Let's see how all of this works in an example.

13.5.2 An Example

Let's go back to the graph in Figure 13.11. We'll use an example from Judea Pearl's book. We're concerned with the sidewalk being slippery, so we're investigating its causes. X_5 can be 1 or 0, for slippery or not, respectively. You've found that the sidewalk is slippery when it's wet, and you'll use X_4 to indicate whether the sidewalk is wet. Next, you need to know the causes of the sidewalk being wet. You see that a sprinkler is near the sidewalk, and if the sprinkler is on, it makes the sidewalk wet. X_2 will indicate whether the sprinkler is on. You'll notice the sidewalk is also wet after it rains, which you'll indicate with X_3 being 1 after rain, 0 otherwise. Finally, you note that on sunny days you turn the sprinkler on. You'll indicate the weather with X_1, where X_1 is 1 if it is sunny, and 0 otherwise.

In this picture, rain and the sprinkler being on are negatively related to each other. This statistical dependence happens because of their mutual dependence on the weather. Let's simulate some data to explore this system. You'll use a lot of data, so the random error will be small, and you can focus your attention on the bias.

```
import numpy as np
import pandas as pd
from scipy.special import expit

N = 100000
inv_logit = expit
x1 = np.random.binomial(1, p=0.5, size=N)
x2 = np.random.binomial(1, p=inv_logit(-3.*x1))
x3 = np.random.binomial(1, p=inv_logit(3.*x1))
x4 = np.bitwise_or(x2, x3)
x5 = np.random.binomial(1, p=inv_logit(3.*x4))

X = pd.DataFrame({'$x_1$': x1, '$x_2$': x2, '$x_3$': x3,
                  '$x_4$': x4, '$x_5$': x5})
```

Every variable here is binary. You use a logistic link function to make logistic regression appropriate. When you don't know the data-generating process, you might get a little more creative. You'll come to this point in a moment!

Let's look at the correlation matrix, shown in Figure 13.13. When the weather is good, the sprinkler is turned on. When it rains, the sprinkler is turned off. You can see there's a negative relationship between the sprinkler being on and the rain due to this relationship.

There are a few ways you can get an estimate for the effect of X_2 on X_5. The first is simply by finding the probability that $X_5 = 1$ given that $X_2 = 1$ or $X_2 = 0$. The difference in these probabilities tells you how much more likely it is that the sidewalk is slippery given that the sprinkler was on. A simple way to calculate these probabilities is simply to average the X_5 variable in each subset of the data (where $X_2 = 0$ and $X_2 = 1$). You can run the following, which produces the table in Figure 13.14.

```
X.groupby('$x_2$').mean()[['$x_5$']]
```

	x_1	x_2	x_3	x_4	x_5
x_1	1.000000	-0.405063	0.420876	0.200738	0.068276
x_2	-0.405063	1.000000	-0.172920	0.313897	0.102955
x_3	0.420876	-0.172920	1.000000	0.693363	0.255352
x_4	0.200738	0.313897	0.693363	1.000000	0.362034
x_5	0.068276	0.102955	0.255352	0.362034	1.000000

Figure 13.13 The correlation matrix for the simulated data set. Notice that X_2 and X_3 are negatively related because of their common cause, X_1.

	x_5
x_2	
0	0.861767
1	0.951492

Figure 13.14 The naive conditional expectation values for whether the grass is wet given that the sprinkler is on, $E[X_5|X_2 = x_2]$. This is not a causal result because you haven't adjusted for confounders.

If you look at the difference here, you see that the sidewalk is $0.95 - 0.86 = 0.09$, or nine percentage points more likely to be slippery given that the sprinkler was on. You can compare this with the interventional graph to get the true estimate for the change. You can generate this data using the process shown here:

```
N = 100000
inv_logit = expit
x1 = np.random.binomial(1, p=0.5, size=N)
x2 = np.random.binomial(1, p=0.5, size=N)
x3 = np.random.binomial(1, p=inv_logit(3.*x1))
x4 = np.bitwise_or(x2, x3)
x5 = np.random.binomial(1, p=inv_logit(3.*x4))

X = pd.DataFrame({'$x_1$': x1, '$x_2$': x2, '$x_3$': x3,
                  '$x_4$': x4, '$x_5$': x5})
```

Now, X_2 is independent of X_1 and X_3. If you repeat the calculation from before (try it!), you get a difference of 0.12, or 12 percentage points. This is about 30 percent larger than the naive estimate!

Now, you'll use some machine learning approaches to try to get a better estimate of the true (0.12) effect strictly using the observational data. First, you'll try a logistic regression on the first data set. Let's re-create the naive estimate, just to make sure it's working properly.

```
from sklearn.linear_model import LogisticRegression

# build our model, predicting $x_5$ using $x_2$
model = LogisticRegression()
model = model.fit(X[['$x_2$']], X['$x_5$'])


# what would have happened if $x_2$ was always 0:
X0 = X.copy()
X0['$x_2$'] = 0
y_pred_0 = model.predict_proba(X0[['$x_2$']])

# what would have happened if $x_2$ was always 1:
X1 = X.copy()
X1['$x_2$'] = 1
y_pred_1 = model.predict_proba(X1[['$x_2$']])

# now, let's check the difference in probabilities
y_pred_1[:, 1].mean() - y_pred_0[:,1].mean()
```

You first build a logistic regression model using X_2 to predict X_5. You do the prediction and use it to get probabilities of X_5 under the $X_2 = 0$ and $X_2 = 1$ states. You did this over the whole data set. The reason for this is that you'll often have more interesting data sets, with many more variables changing, and you'll want to see the average effect of X_2 on X_5 over the whole data set. This procedure lets you do that. Finally, you find the average difference in probabilities between the two states, and you get the same 0.09 result as before!

Now, you'd like to do controlling on the same observational data to get the causal (0.12) result. You perform the same procedure as before, but this time you include X_1 in the regression.

```
model = LogisticRegression()
model = model.fit(X[['$x_2$', '$x_1$']], X['$x_5$'])

# what would have happened if $x_2$ was always 0:
X0 = X.copy()
X0['$x_2$'] = 0
y_pred_0 = model.predict_proba(X0[['$x_2$', '$x_1$']])

# what would have happened if $x_2$ was always 1:
X1 = X.copy()
X1['$x_2$'] = 1

# now, let's check the difference in probabilities
y_pred_1 = model.predict_proba(X1[['$x_2$', '$x_1$']])
y_pred_1[:, 1].mean() - y_pred_0[:,1].mean()
```

In this case, you find 0.14 for the result. You've over-estimated it! What went wrong? You didn't actually do anything wrong with the modeling procedure. The problem is simply that logistic

regression isn't the right model for this situation. It's the correct model for each variable's parents to predict its value but doesn't work properly for descendants that follow the parents. Can we do better, with a more general model?

This will be your first look at how powerful neural networks can be for general machine-learning tasks. You'll learn about building them in a little more detail in the next chapter. For now, let's try a deep feedforward neural network using keras. It's called *deep* because there are more than just the input and output layers. It's a *feedforward* network because you put some input data into the network and pass them forward through the layers to produce the output.

Deep feedforward networks have the property of being "universal function approximators," in the sense that they can approximate any function, given enough neurons and layers (although it's not always easy to learn, in practice). You'll construct the network like this:

```
from keras.layers import Dense, Input
from keras.models import Model

dense_size = 128
input_features = 2

x_in = Input(shape=(input_features,))
h1 = Dense(dense_size, activation='relu')(x_in)
h2 = Dense(dense_size, activation='relu')(h1)
h3 = Dense(dense_size, activation='relu')(h2)
y_out = Dense(1, activation='sigmoid')(h3)

model = Model(input=x_in, output=y_out)
model.compile(loss='binary_crossentropy', optimizer='adam')
model.fit(X[['$x_1$', '$x_2$']].values, X['$x_5$'])
```

Now do the same prediction procedure as before, which produces the result 0.129.

```
X_zero = X.copy()
X_zero['$x_2$'] = 0
x5_pred_0 = model.predict(X_zero[['$x_1$', '$x_2$']].values)

X_one = X.copy()
X_one['$x_2$'] = 1
x5_pred_1 = model.predict(X_one[['$x_1$', '$x_2$']].values)

x5_pred_1.mean() - x5_pred_0.mean()
```

You've done better than the logistic regression model! This was a tricky case. You're given binary data where it's easy to calculate probabilities, and you'd do the best by simply using the g-formula directly. When you do this (try it yourself!), you calculate the true result of 0.127 from this data. Your neural network model is very close!

Now, you'd like to enact a policy that would make the sidewalk less likely to be slippery. You know that if you turn the sprinkler on less often, that should do the trick. You see that enacting this policy (and so intervening to change the system), you can expect the slipperiness of the sidewalk to

decrease. How much? You want to compare the pre-intervention chance of slipperiness with the post-intervention chance, when you set sprinkler = off. You can simply calculate this with our neural network model like so:

```
X['$x_5$'].mean() - x5_pred_0.mean()
```

This gives the result 0.07. It will be 7 percent less likely that the sidewalk is slippery if you make a policy of keeping the sprinkler turned off!

13.6 Conclusion

In this chapter, you've developed the tools to do causal inference. You've learned that machine learning models can be useful to get more general model specifications, and you saw that the better you can predict an outcome using a machine learning model, the better you can remove bias from an observational causal effect estimate.

Observational causal effect estimates should always be used with care. Whenever possible, you should try to do a randomized controlled experiment instead of using the observational estimate. In this example, you should simply use randomized control: flip a coin each day to see whether the sprinkler gets turned on. This re-creates the post-intervention system and lets you measure how much less likely the sidewalk is to be slippery when the sprinkler is turned off versus turned on (or when the system isn't intervened upon). When you're trying to estimate the effect of a policy, it's hard to find a substitute for actually testing the policy through a controlled experiment.

It's especially useful to be able to think causally when designing machine-learning systems. If you'd simply like to say what outcome is most likely given what normally happens in a system, a standard machine learning algorithm is appropriate. You're not trying to predict the result of an intervention, and you're not trying to make a system that is robust to changes in how the system operates. You just want to describe the system's joint distribution (or expectation values under it).

If you would like to inform policy changes, predict the outcomes of intervention, or make the system robust to changes in variables upstream from it (i.e., external interventions), then you will want a causal machine learning system, where you control for the appropriate variables to measure causal effects.

An especially interesting application area is when you're estimating the coefficients in a logistic regression. Earlier, you saw that logistic regression coefficients had a particular interpretation in observational data: they describe how much more likely an outcome is per unit increase in some independent variable. If you control for the right variables to get a causal logistic regression estimate (or just do the regression on data generated by control), then you have a new, stronger interpretation: the coefficients tell you how much more likely an outcome is to occur when you intervene to increase the value of an independent variable by one unit. You can use these coefficients to inform policy!

Index

Numbers

A

B

C

D

E

F

G

H

I

N

O

P

Q

R

S

T

U

V

W

Z