Book Reviews

ELIZABETH ZWICKY, WITH PETER GUTMANN, SAM STOVER, AND MARK LAMOURINE

Machine Learning for Hackers

Drew Conway and John Myles White

O'Reilly, 2012. 293 pp.
ISBN 978-1-449-30371-6

Suppose you have a pile of data, and you have a sense that what the fancy guys do with this stuff involves machine learning, but you have neither the time nor the inclination to go out and learn a whole bunch of new theory. This is the book for you, although you should note that for many, many people there is simply no practical approach to machine learning without picking up a language or two along the way, so you should be prepared to learn more than straightforward techniques. This book uses R for its examples in order to get something with a lot of power and less overhead than installing an entire Hadoop cluster. This is a tradeoff; if you already have Big Data around, somebody probably has already installed a Hadoop cluster or so for you, so that using R may actually be more work, both in installation and in trimming data sets to fit on a single machine. Still, it gives a consistent base to work from.

If you don't already have machine-learning people doing specialized stuff, you may be surprised at the amount of progress that can be made with algorithms that are not particularly smart. A few tricks will go a long way. If you do have such people, a little learning will help you talk to them. I found this an approachable introduction to the base techniques, although not without quirks. Some sections are made much more difficult to follow by having lovely big graphs. That sounds like a plus, but when a page introduces 5 graphs, which then show up on the 5 following pages with supporting text across the next 10, the text and the picture get punishingly out of sync, with text about illustration 6 showing up on the same page as illustration 2. Admittedly, this problem is definitely worse in the electronic editions I prefer to review, where flipping back and forth 2 or 3 pages is not easy.

The Scrum Field Guide: Practical Advice for Your First Year

Mitch Lacey

Addison-Wesley, 2012. 365 pp.
ISBN 978-0-321-55415-4

As the author points out, Scrum (which is one of the extended and incestuous family of Agile programming techniques) is simple but not easy. All you need to do is change everything about how your programmers work and how that work is specified and directed; what could possibly go wrong? For most organizations, converting to Scrum is like moving to another hemisphere. This book is aimed at providing useful advice for this transition, and the author speaks with the practical voice of experience, providing effective ways of making the transition.

As a "first year" book, it's not just about techniques; it's also an introduction to the common problems, inevitable miseries, and eventual joys of the transition. This kind of roadmap is a great thing to have when you are wondering whether you screwed things up, your team is hopeless, or this is just one of those bad patches that happen in every transition. It is inspirational; it did a good job of making me feel that this is a plausible transition. Furthermore, it covers a relatively wide range of situations, including outsourcing, operational/support groups, and at least one way of handling common functions like security and user interface design that span multiple teams. Daily team meetings are one thing if you're on a single team. What if you're on three or more? It is more applicable to isolated teams than to entire organizations going to Scrum, which brings up new issues. Getting buy-in for a single team is a different deal from getting buy-in when it's an edict from above.

User Stories Applied for Agile Software Development

Mike Cohn

Addison-Wesley, 2004. 262 pp.
ISBN 0-321-20568-5

User Stories makes an interesting pair to *The Scrum Field Guide*. It has a reasonably similar loose, practical tone, and a noticeably different take on some issues. Which issues? It's right up there with league vs. union in rugby, which is to say, I could tell you, but if you're not an aficionado, you're just going to be baffled. *User Stories* believes in half-point stories! What kind of madness is this? Everybody knows you only assign points from the Fibonacci series. Everybody who does Scrum, that is. The rest of you are wondering if I am actually making this up.

User stories are one of the key parts of Agile; they replace requirements documents. It is entirely possible that Agile is in fact entirely a plot to keep people from ever having to write 400-page documents with seven levels of nesting. I would consider restructuring the entire programming culture as a reasonable response to this experience. However, getting user stories to actually be more useful than requirements documents and getting people who are used to requirements documents to accept them are both tricky.

This is one of those books with a summary and quiz questions at the end of every chapter, which always annoys me, but it's a quick, straightforward guide to how you would get user stories, including advice on what generally goes wrong and what to do when, for instance, you don't get to speak to actual users to get user stories. I love the advice on what to do with stories when you're finished; tear them up if they're written on paper, but keep them if they're electronic. They're not very useful—but destroying bits isn't very satisfying, either. This kind of advice is also evidence of solid experience.

The Art of Community, Second Edition

Jono Bacon

O'Reilly, 2012. 526 pp.
ISBN 978-1-449-31206-0

This book's central audience is people who are trying to build free software communities, although it has wider applicability, particularly to communities at the tricky interface of the commercial and the volunteer. It does a good job of covering this tricky territory, with advice that blends the practical and the theoretical. (I am amused to note that it has the best burn-down graphs of the lot, better than either of the books that are actually about programming models.) In this second edition, the author brings in some interviews to help flesh out the picture of other kinds of communities, but the bias is still pretty strong. If you want to build a group of system administrators, or a coalition of commercial companies (to name the communities I'm most familiar with), you will be doing some translating and some skipping, but you will still find useful information.

I have two mild regrets about the book. First, it does not at any point talk about how you deal with community members who are simply incapable of participating in the community as part of the consensus reality. It strongly implies that all community members are willing and able, if approached correctly, to come into harmony; in my experience, almost every community has members who are not willing and able to do so, for one reason or another. Managing these people is its own art. The author does encourage readers to try to bring people, even difficult people, into the community, which I think is a good choice. However, I think there are situations where removing people is also a valid if difficult choice, and it is useful to community managers to have some idea how to negotiate these incredibly difficult waters.

Second, the author uses technical terms of the online community overly broadly. This may be intentional, as a way to speak to people who come from other backgrounds, but it clashes on my sensitive ears and in one case obscures an important distinction. "Trolling" is not any type of disruptive online behavior; it's online behavior motivated by a desire to create controversy and dissent. It's important to know this and to discuss it, for a couple of reasons. People who don't understand that trolls exist can easily be sucked into dealing with fictional people, whereas people who do understand it can leap to deciding that real, if deluded, people are trolls. Either way, you get bad effects on the community. There is no point trying to help a troll, and there is no point ignoring real people. Lumping all of this together does a disservice to people who're trying to deal with online communities. The other case that bothered me was "bikeshedding" being described as all forms of discussion rather than getting something done; I think this is wrong, but probably not importantly so.

Domain-Driven Design

Eric Evans

Addison-Wesley, 2004. 519 pp.
ISBN 0-321-12521-5

This is a book about domain modeling for people who model domains for a living. It is not a book for people who might want to bring a bit of domain modeling into their existing process, although I suppose if you're already doing architecture for large development teams that are strongly separated

from the domain they're building for, it might be a valuable tool for you. But for the most part, it's about adding depth and finesse to your domain modeling.

Because of this, I'm not a great reviewer for it. I'm nearer to being a domain expert than a domain modeler, and I found it hard to sympathize with. Plus, the author is fond of patterns, and carefully puts the name of every pattern in small caps. This leads to entire sentences in which all the noun phrases are in small caps, which rapidly became annoying and added to the sensation that I was reading a careful exegesis of somebody else's religion. I'm pretty sure that this is dry going for all but a small audience, for whom it is probably stuffy but enlightening.

Electronic Signatures in Law

Stephen Mason

Cambridge University Press, Third Edition, 2007. 728 pp.
ISBN 978-1-84592-425-6

As any IT person knows, in order to create an electronic signature you need (depending on your particular fashion tastes) public and private keys, digital signatures, certificates, smart cards, and all manner of other paraphernalia. Lawyers—the people who actually work with signatures and signature law on a daily basis—don't see it quite that way, and this book explains why. For example, the function of a signature isn't necessarily to form a contract but may be merely to provide a cautionary function, encouraging the person affixing the signature to a document to take extra care in reading it, or a channeling function in which the signature records the time at which a document was accepted by the signer. In extreme cases it serves purposes quite unrelated to what we'd normally associate with signatures, such as allowing documents to be taxed in the form of a stamp tax or stamp duty.

The book starts with precedents going back to the Magna Carta, at which time "signatures" consisted of drawing a cross as a sign of Christian truth because writing your name on a piece of paper would have meant nothing, so that only non-Christian could sign their names on a contract. The book traces the history of what in legal terminology is called a manuscript signature, traditionally a pen-and-paper process, through to more modern forms such as telegrams and telexes. In none of these cases was special legislation necessary, since courts interpreted existing signature law and practice to cover newer technology that was introduced after the laws were written. This part of the book is a fascinating look at just how flexible the interpretation of what constitutes a signature actually is, with courts finding that "signatures" can include things like signing as "Mum" or saying "yes" (verbally), and more recently typing a name in an email message, sending an SMS, or clicking "Send" on email that has your name in the "From:" field. In fact, provided that the identities of the parties to the agreement are fairly obvious, even a document containing no conventional signature at all may be enough to meet the legal requirements for a contract, if the intent of the participants is manifest and the method of conveying this is appropriate to the particular transaction. The extreme flexibility of existing contract law is demonstrated by the fact that a court case over the validity of email that's been "signed" by having the sender's name on it was supported by a quote from the Statute of Frauds Act of 1677, predating the existence of email by several centuries. The later parts of the book cover electronic and digital signature laws in great detail, including the twisty maze of signature laws, all different, that were passed in various countries around the time of the dot-com boom. This portion is probably of interest only to lawyers (or someone having to deal with the mess of subtly incompatible laws), and it appears to be an absolute minefield compared to the relative simplicity of the earlier case-law-based portions. In any case, much of what's contained in the laws seems to present little more than unnecessary complications. As the book points out when discussing the calisthenics required by electronic signature laws (p. 101), "contracts conducted by post . . . were commonplace two hundred years before the Internet, and it is to be wondered why businesses need such guidance when they have been dealing with such issues for such a long period of time."

One thing to be aware of is that this book will take a bit of getting used to for someone who's not familiar with the format of legal documents. The narrative progresses in two parallel channels: the main body text at the top of the page and extensive references, footnotes, comments, and annotations at the bottom, and because of the emphasis on case law there are plenty of those. The first half of the book, which covers existing case law and precedents for allowing various forms of non-manuscript signature, will be a real eye-opener for anyone who has grown up expecting to have to use certificates and smart cards and assorted other paraphernalia in order to form an electronic legally binding agreement. The second half, covering the ins and outs of electronic and digital signature legislation, will probably be of interest mostly to lawyers.

One downside to the book is that it's quite pricey, around $200 US. On the other hand, if you can find it in a library or get your employer to buy it for you, it's definitely worth a read if you're in a position where you have to deal with electronic/digital signatures.

—Peter Gutmann

Metasploit

David Kennedy, Jim O'Gorman, Devon Kearns, and Mati Aharoni

No Starch Press, 2011. 299 pp.
ISBN 978-1-59327-288-3

This book was designed as a "useful tutorial for the beginner and a reference for practitioners" for the Metasploit penetration testing framework. I think the authors definitely achieved this goal: the book is well written, contains tons of useful information, and is extremely thorough. If you haven't already picked up a book on Metasploit, this is the one to buy, and even if you already have a decent book on Metasploit, you probably should give this one a hard look. The first couple of chapters deal with the basics of penetration testing and where Metasploit fits into the process. Metasploit certainly handles the exploitation aspect of a pen-test, but there are lots of other things, such as gathering intel on your targets, that require time and other tools. While the book definitely focuses on Metasploit, a fair bit of time and effort was put into making you a better overall pen-tester, not just a Metasploit expert.

Chapter 5, "The Joy of Exploitation," is where you really start digging into the awesomeness of Metasploit. By this point, you should be comfortable with the various Metasploit Framework (MSF) interfaces, as well as several other standard tools/utilities such as nmap, whois, Nessus, and NeXpose. The ability to take Nessus/NeXpose/nmap output and import it into Metasploit makes it a lot easier to keep track of your targets and their likely vulnerabilities. This book doesn't just walk you through using Metasploit, it walks you through how to use other tools that make the Metasploit experience even more rewarding. Good stuff. Chapter 6 focuses on using the Meterpreter, which is a mini-shell delivered as a payload to an exploit. Once you've owned a box, you can use Meterpreter to interact with the system instead of being limited to the victim command prompt. In other words, just owning the box is only half the battle—once you're there, you want to perform local attacks on the system (e.g., grab password hashes), and Meterpreter makes your job that much easier.

Chapters 7–9 go a little deeper into the more advanced features of the MSF. Detection avoidance using MSFencode to create custom binaries, client-side attacks, and auxiliary modules, such as SSH brute-forcing and protocol (FTP, HTTP, SMTP, SSH) fuzzers, are all tools that advanced pen-testers can rely on. You may not need them the way you need the exploits, but when you do, Metasploit has them.

Another vector that can't be ignored is social engineering. One of the authors of this book developed the Social-Engineer Toolkit (SET), which is a separate tool that relies on the Metasploit Framework. Each example comes with a little story, and I think this is a great way to teach/demonstrate the capability, because we can all imagine day-to-day scenarios like these. Lots of great stuff in this chapter, from creating your own malicious PDF to cloning a target's Web site and harvesting credentials. Fantastic.

Chapter 11 introduces Fast-Track, which is another standalone tool that uses the MSF. Unlike the SET, this is an advanced pen-testing tool which offers exploitation methods that complement the MSF. This tool provides some MS-SQL attacks, different exploits from those bundled with Metasploit, and some browser attack vectors. While Fast-Track is quite advanced, its interface is pretty intuitive, and this chapter walks you through it.

Chapter 12 is dedicated to Karmetasploit, which is "Metasploit's implementation of the KARMA attack." This involves setting up a fake access point and enticing users to connect to it instead of to the "real" access point. Successfully executing this attack allows access to all network traffic (e.g., passwords) as well as the ability to launch client-side attacks on unsuspecting users. Karmetasploit is fairly simple to set up and execute, and this chapter provides all the necessary bits.

Chapters 13–16 are for the hard-core Metasploit user. If you want to write your own module, create your own exploits, port your exploits into the Framework, or write your own Meterpreter API scripts, these are the chapters for you. Each chapter deals with one of the topics and presents them in a very accessible manner. If you want to write or import your own exploits, you'll need a firm grasp of assembler, so this is not for the faint of heart. If you're up for a challenge, though, look no further.

The final chapter walks you through a simulated pen-test from soup to nuts. Complete and succinct, the chapter encapsulates everything you've learned in this book. Two appendices give some help on configuring your target systems and a "cheat sheet" of MSF commands. No stone is left unturned in this book, I'll tell ya.

This is a book about how to become a (better) pen-tester, written by people who know what they are talking about and focusing on one of the best tools available, open source or otherwise. You simply cannot go wrong with this book if you want to learn more about Metasploit, or even if you want to learn more about penetration testing. Hands down, my new favorite Metasploit book.

—Sam Stover

Python for Software Design: How to Think Like a Computer Scientist

Allen B. Downey

Cambridge University Press, 2009. 270 pp.
ISBN 978-0-521-72596-5

Think Python: How to Think Like a Computer Scientist

Allen Downey

Green Tea Press, 2012. 212 pp.
http://www.greenteapress.com/thinkpython/thinkpython.html

It has always seemed to me that there were three types of technical books: tutorials, user guides, and references. Allan Downey's "Think" series has reminded me that there is a fourth type: teaching texts.

A typical tutorial tries to guide the reader down a specific path, controlling and limiting the reader's experience. It assumes that the user has little or no prior experience with the subject. A user guide is similar but might be more comprehensive and more useful once the user has some experience. A reference provides the most detail, but generally no learning narrative. All of these are focused on the single topic at hand and try to minimize diversions.

A good teaching text can be special in several ways (Downey's books being special in all of them):

- Promotes classroom discussion
- Provides a learn-by-experience framework
- Encourages exploration

At first glance, "Think Python" is just another beginner's guide to a popular scripting language. The subtitle, while less catchy, describes what's really going on. The first two paragraphs of the first chapter make it explicit: "The goal of this book is to teach you to think like a computer scientist" and "The single most important skill for a computer scientist is *problem solving*."

Most chapters introduce a single programming language concept. Sprinkled in among these are case study chapters that expose and discuss concepts such as interface and data structure design. Each chapter concludes with a glossary, a set of exercises, and a section on debugging specifically addressing the new material.

While the book begins with such basic concepts as variables, expressions, and statements, by the end the student will be working with techniques of classic object-oriented programming. Along the way the reader is introduced to the concepts and practice of lists (including map and reduce) and hashes (dictionaries in Python).

The real point of the book is to teach the underlying concepts of software development. The examples in the book are all written in Python, but Python is only the framework on which the real lessons are hung. The book was originally written and taught using Java.

There is very little exposition of Python syntax. The examples are clear and brief, and the text explains the intent and meaning, but almost no time is spent on particular language elements. A true beginner without the benefit of a classroom and teacher will probably want to have a traditional language manual as well while working through this book. A professional developer who is not familiar with Python will want access to the Net or at least a pocket reference.

Think Stats: Probability and Statistics for Programmers

Allen B. Downey

O'Reilly Media, 2011. 120 pp.
ISBN 978-1-449-30711-0

Again, the subtitle of *Think Stats* gets to the heart of why Downey's second book is different from most ordinary textbooks on probability and statistics. While most programmers do not need to be statisticians, the wide variety of fields that are using mathematical methods makes it unlikely that most will be able to escape without some exposure.

Think Stats introduces statistics as a means to answer a certain class of questions involving populations and data sets. In the first chapter, Downey asks if there is a way to confirm the common belief that first children are born later in a pregnancy than subsequent ones. He addresses that question in new ways as the book progresses, adding nuance to the answers each time.

Throughout this book Downey uses references to online resources for additional reading and for data sets to use for exploring. He introduces topics with appropriate mathematical notation, but, rather than taking up space with formal derivations or proofs, he refers to online articles, usually from Wikipedia, for students to explore on their own (or as directed by the instructor). Downey focuses, instead, on illustrating each of the concepts in the context of some problem or question in need of a solution.

Downey covers the traditional topics for an introductory stats course: cumulative distribution functions, population distribution curves and characteristics, and probability. And, unusually for an introductory text, he also discusses Bayesian statistics. The last half of the book covers hypothesis testing, estimation, and correlation, again standard topics, but again including Bayesian Estimation.

The book illustrates examples and provides exercises in Python. It uses pyplot for graphing and provides other statistical demonstration code as downloads from the book's Web site. It follows *Think Python*'s pattern of ending each chapter with a glossary and exercises.

This book is aimed at a guided introductory college-level classroom, but the frequent external references make it suitable for a motivated self-learner with working knowledge of Python and enough calculus to be able to interpret the few formal mathematical expressions: simple summation and integration. The presentation is also ideal for a knowledgeable programmer to explore and understand the basis of statistical methods.

Think Complexity: Exploring Complexity Science with Python

Allen B. Downey

O'Reilly Media, 2012. 142 pp.
ISBN 978-1-449-31463-7

Think Complexity is Downey's most recent book and the least conventional. As before, the subtitle gives a better feel for the content, but this time it still needs some clarification. The book really explores the techniques and applications of computational modeling. "Complexity Science" is the name given to the study of the set of problems where traditional analysis is ineffective and to which computational modeling is a better approach.

Downey devotes the first chapter to explaining what he means by "Complexity Science," although in the end you've only really learned what it is not. He defines it largely in contrast to traditional deterministic reductionist science and engineering. A deep understanding of what that means has to wait until the reader has gotten some experience with modeling complex systems. Downey also promises to address questions of the philosophy of science that are raised by the development and use of computational modeling.

Chapters 2 through 5 are used to build up the base of tools and concepts that will be needed for application to real problems. These include graphs and some analysis of algorithms and statistics (there is some overlap from *Think Stats* here).

Chapters 6 through 10 are the meat of the book. Downey proceeds to introduce and explore the characteristics and uses of cellular automata, self-organizing critical systems, and, finally, agent-based models.

Each of the sections contains some code, but more is provided as downloadable modules, which are then used as a base for the student's work. The samples are implemented in Python and use the NumPy and SciPy modules, but almost no text is devoted to them. It is assumed either that the teacher will provide what is needed or that the reader has the ability to find what they need online.

These first two sections of the book are liberally laced with references to classic papers and studies which have informed the development of computational modeling as a discipline. These often include links to Wikipedia articles and, occasionally, to the original source papers. Downey invites the reader to question and understand the limits of classical scientific methods and the utility of computational methods for understanding systems that resist traditional analysis.

The last few chapters of the book contain a set of case studies presented by his students and evaluated by members of the faculty. Each uses some form of non-deterministic modeling to explore the characteristics of some dynamic system. Downey invites instructors and readers to submit additional case studies for inclusion in the evolving text for the course.

In this latest book, Downey makes explicit something that has been integral to the first two as well, but without recognition. The preface includes a short section of guidance for teachers who intend to use the book as the basis for a course, and then a section dedicated to self-learners. He addresses the fact that many popular books on these kinds of topics are aimed at a casual audience and tend to avoid details and depth that would scare off the lay reader (or a publisher's concept of a lay reader). These books are addressed to the dedicated learner who is interested in the details and is motivated to follow leads to the source materials.

As teaching texts these books are special. At about $30 US they are each much less expensive than comparable textbooks. They are smaller, with many external references; rather than including redundant and static expositions of concepts, they take advantage of existing online resources.

All three of these books have one other characteristic that sets them apart: they are all freely available under the Creative Commons Attribution-Noncommercial-ShareAlike 3.0 license. All three are available as HTML and PDF online from Downey's own Web site at http://www.greenteapress.com.

—Mark Lamourine