O'REILLY®

Data Science from Scratch

FIRST PRINCIPLES WITH PYTHON

Joel Grus

O'REILLY®

Data Science from Scratch

Data science libraries, frameworks, modules, and toolkits are great for doing data science, but they're also a good way to dive into the discipline without actually understanding data science. In this book, you'll learn how many of the most fundamental data science tools and algorithms work by implementing them from scratch.

If you have an aptitude for mathematics and some programming skills, author Joel Grus will help you get comfortable with the math and statistics at the core of data science, and with hacking skills you need to get started as a data scientist. Today's messy glut of data holds answers to questions no one's even thought to ask. This book provides you with the know-how to dig those answers out.

- Get a crash course in Python
- Learn the basics of linear algebra, statistics, and probability—and understand how and when they're used in data science
- Collect, explore, clean, munge, and manipulate data
- Dive into the fundamentals of machine learning
- Implement models such as k-nearest neighbors, Naive Bayes, linear and logistic regression, decision trees, neural networks, and clustering
- Explore recommender systems, natural language processing, network analysis, MapReduce, and databases

Joel Grus is a software engineer at Google. Before that, he worked as a data scientist at multiple startups. He lives in Seattle, where he regularly attends data science happy hours. He blogs infrequently at *joelgrus.com* and tweets all day long at @joelgrus.

"Joel takes you on a journey from being data-curious to getting a thorough understanding of the bread-and-butter algorithms that every data scientist should know."

—Rohit Sivaprasad
Data Science, Soylent
datatau.com

DATA/DATA SCIENCE

US $39.99 **CAN $45.99**
ISBN: 978-1-491-90142-7

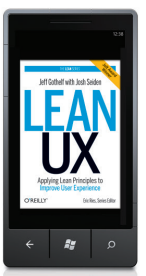
Twitter: @oreillymedia
facebook.com/oreilly

O'Reilly ebooks.

Your bookshelf on your devices.

PDF

Mobi



ePub

DAISY

When you buy an ebook through oreilly.com you get lifetime access to the book, and whenever possible we provide it to you in four DRM-free file formats—PDF, .epub, Kindle-compatible .mobi, and DAISY—that you can use on the devices of your choice. Our ebook files are fully searchable, and you can cut-and-paste and print them. We also alert you when we've updated the files with corrections and additions.

Learn more at ebooks.oreilly.com

You can also purchase O'Reilly ebooks through the iBookstore, the Android Marketplace, and Amazon.com.

O'REILLY®

©2015 O'Reilly Media, Inc. The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. 15055

Data Science from Scratch
by Joel Grus

Copyright © 2015 O'Reilly Media. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Marie Beaugureau
Production Editor: Melanie Yarbrough
Copyeditor: Nan Reinhardt
Proofreader: Eileen Cohen

Indexer: Ellen Troutman-Zaig
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

April 2015: First Edition

Revision History for the First Edition
2015-04-10: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491901427* for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Data Science from Scratch,* the cover image of a Rock Ptarmigan, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-90142-7

[LSI]

Table of Contents

CHAPTER 1

Introduction

"Data! Data! Data!" he cried impatiently. "I can't make bricks without clay."
—Arthur Conan Doyle

The Ascendance of Data

We live in a world that's drowning in data. Websites track every user's every click. Your smartphone is building up a record of your location and speed every second of every day. "Quantified selfers" wear pedometers-on-steroids that are ever recording their heart rates, movement habits, diet, and sleep patterns. Smart cars collect driving habits, smart homes collect living habits, and smart marketers collect purchasing habits. The Internet itself represents a huge graph of knowledge that contains (among other things) an enormous cross-referenced encyclopedia; domain-specific databases about movies, music, sports results, pinball machines, memes, and cocktails; and too many government statistics (some of them nearly true!) from too many governments to wrap your head around.

Buried in these data are answers to countless questions that no one's ever thought to ask. In this book, we'll learn how to find them.

What Is Data Science?

There's a joke that says a data scientist is someone who knows more statistics than a computer scientist and more computer science than a statistician. (I didn't say it was a good joke.) In fact, some data scientists are—for all practical purposes—statisticians, while others are pretty much indistinguishable from software engineers. Some are machine-learning experts, while others couldn't machine-learn their way out of kindergarten. Some are PhDs with impressive publication records, while others have never read an academic paper (shame on them, though). In short, pretty much no

matter how you define data science, you'll find practitioners for whom the definition is totally, absolutely wrong.

Nonetheless, we won't let that stop us from trying. We'll say that a data scientist is someone who extracts insights from messy data. Today's world is full of people trying to turn data into insight.

For instance, the dating site OkCupid asks its members to answer thousands of questions in order to find the most appropriate matches for them. But it also analyzes these results to figure out innocuous-sounding questions you can ask someone to find out how likely someone is to sleep with you on the first date.

Facebook asks you to list your hometown and your current location, ostensibly to make it easier for your friends to find and connect with you. But it also analyzes these locations to identify global migration patterns and where the fanbases of different football teams live.

As a large retailer, Target tracks your purchases and interactions, both online and in-store. And it uses the data to predictively model which of its customers are pregnant, to better market baby-related purchases to them.

In 2012, the Obama campaign employed dozens of data scientists who data-mined and experimented their way to identifying voters who needed extra attention, choosing optimal donor-specific fundraising appeals and programs, and focusing get-out-the-vote efforts where they were most likely to be useful. It is generally agreed that these efforts played an important role in the president's re-election, which means it is a safe bet that political campaigns of the future will become more and more data-driven, resulting in a never-ending arms race of data science and data collection.

Now, before you start feeling too jaded: some data scientists also occasionally use their skills for good—using data to make government more effective, to help the homeless, and to improve public health. But it certainly won't hurt your career if you like figuring out the best way to get people to click on advertisements.

Motivating Hypothetical: DataSciencester

Congratulations! You've just been hired to lead the data science efforts at DataSciencester, *the* social network for data scientists.

Despite being *for* data scientists, DataSciencester has never actually invested in building its own data science practice. (In fairness, DataSciencester has never really invested in building its product either.) That will be your job! Throughout the book, we'll be learning about data science concepts by solving problems that you encounter at work. Sometimes we'll look at data explicitly supplied by users, sometimes we'll look at data generated through their interactions with the site, and sometimes we'll even look at data from experiments that we'll design.

And because DataSciencester has a strong "not-invented-here" mentality, we'll be building our own tools from scratch. At the end, you'll have a pretty solid understanding of the fundamentals of data science. And you'll be ready to apply your skills at a company with a less shaky premise, or to any other problems that happen to interest you.

Welcome aboard, and good luck! (You're allowed to wear jeans on Fridays, and the bathroom is down the hall on the right.)

Finding Key Connectors

It's your first day on the job at DataSciencester, and the VP of Networking is full of questions about your users. Until now he's had no one to ask, so he's very excited to have you aboard.

In particular, he wants you to identify who the "key connectors" are among data scientists. To this end, he gives you a dump of the entire DataSciencester network. (In real life, people don't typically hand you the data you need. Chapter 9 is devoted to getting data.)

What does this data dump look like? It consists of a list of users, each represented by a `dict` that contains for each user his or her `id` (which is a number) and `name` (which, in one of the great cosmic coincidences, rhymes with the user's `id`):

```
users = [
    { "id": 0, "name": "Hero" },
    { "id": 1, "name": "Dunn" },
    { "id": 2, "name": "Sue" },
    { "id": 3, "name": "Chi" },
    { "id": 4, "name": "Thor" },
    { "id": 5, "name": "Clive" },
    { "id": 6, "name": "Hicks" },
    { "id": 7, "name": "Devin" },
    { "id": 8, "name": "Kate" },
    { "id": 9, "name": "Klein" }
]
```

He also gives you the "friendship" data, represented as a list of pairs of IDs:

```
friendships = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
               (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]
```

For example, the tuple `(0, 1)` indicates that the data scientist with `id` 0 (Hero) and the data scientist with `id` 1 (Dunn) are friends. The network is illustrated in Figure 1-1.

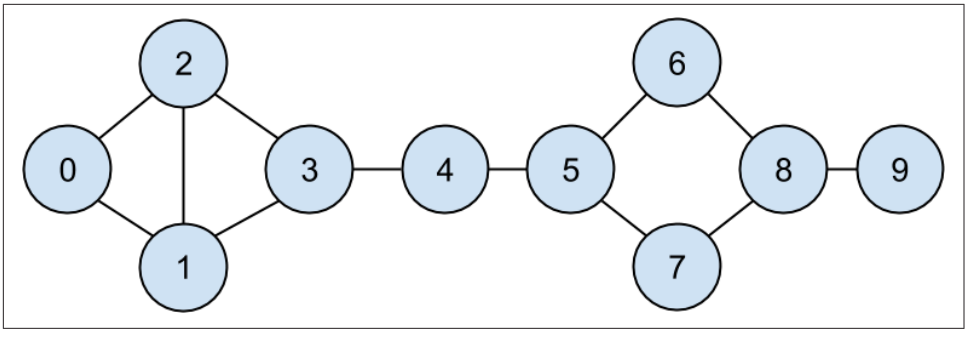


Figure 1-1. The DataSciencester network

Since we represented our users as `dicts`, it's easy to augment them with extra data.

Don't get too hung up on the details of the code right now. In Chapter 2, we'll take you through a crash course in Python. For now just try to get the general flavor of what we're doing.

For example, we might want to add a list of friends to each user. First we set each user's `friends` property to an empty list:

```
for user in users:
    user["friends"] = []
```

And then we populate the lists using the `friendships` data:

```
for i, j in friendships:
    # this works because users[i] is the user whose id is i
    users[i]["friends"].append(users[j]) # add i as a friend of j
    users[j]["friends"].append(users[i]) # add j as a friend of i
```

Once each `user` `dict` contains a list of friends, we can easily ask questions of our graph, like "what's the average number of connections?"

First we find the *total* number of connections, by summing up the lengths of all the `friends` lists:

```
def number_of_friends(user):
    """how many friends does _user_ have?"""
    return len(user["friends"])                 # length of friend_ids list

total_connections = sum(number_of_friends(user)
                        for user in users)      # 24
```

And then we just divide by the number of users:

```
from __future__ import division                 # integer division is lame
num_users = len(users)                          # length of the users list
avg_connections = total_connections / num_users # 2.4
```

It's also easy to find the most connected people—they're the people who have the largest number of friends.

Since there aren't very many users, we can sort them from "most friends" to "least friends":

```
# create a list (user_id, number_of_friends)
num_friends_by_id = [(user["id"], number_of_friends(user))
                     for user in users]

sorted(num_friends_by_id,                                   # get it sorted
       key=lambda (user_id, num_friends): num_friends,      # by num_friends
       reverse=True)                                        # largest to smallest

# each pair is (user_id, num_friends)
# [(1, 3), (2, 3), (3, 3), (5, 3), (8, 3),
#  (0, 2), (4, 2), (6, 2), (7, 2), (9, 1)]
```

One way to think of what we've done is as a way of identifying people who are somehow central to the network. In fact, what we've just computed is the network metric *degree centrality* (Figure 1-2).

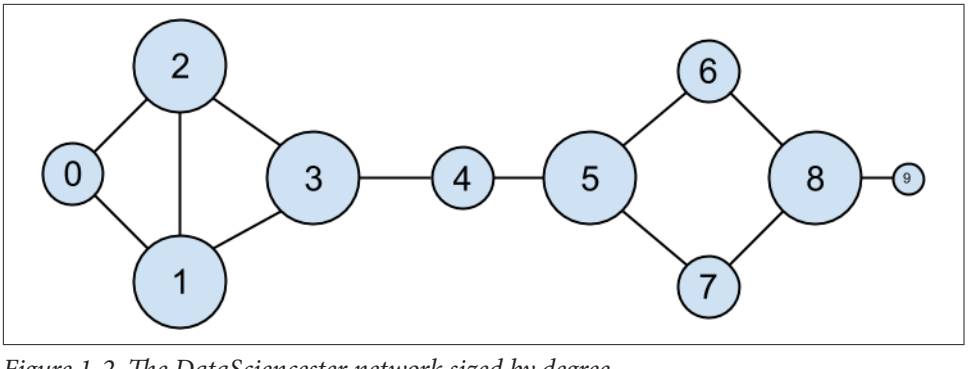


Figure 1-2. The DataSciencester network sized by degree

This has the virtue of being pretty easy to calculate, but it doesn't always give the results you'd want or expect. For example, in the DataSciencester network Thor (`id` 4) only has two connections while Dunn (`id` 1) has three. Yet looking at the network it intuitively seems like Thor should be more central. In Chapter 21, we'll investigate networks in more detail, and we'll look at more complex notions of centrality that may or may not accord better with our intuition.

Data Scientists You May Know

While you're still filling out new-hire paperwork, the VP of Fraternization comes by your desk. She wants to encourage more connections among your members, and she asks you to design a "Data Scientists You May Know" suggester.

Your first instinct is to suggest that a user might know the friends of friends. These are easy to compute: for each of a user's friends, iterate over that person's friends, and collect all the results:

```
def friends_of_friend_ids_bad(user):
    # "foaf" is short for "friend of a friend"
    return [foaf["id"]
            for friend in user["friends"]     # for each of user's friends
            for foaf in friend["friends"]]    # get each of _their_ friends
```

When we call this on `users[0]` (Hero), it produces:

```
[0, 2, 3, 0, 1, 3]
```

It includes user 0 (twice), since Hero is indeed friends with both of his friends. It includes users 1 and 2, although they are both friends with Hero already. And it includes user 3 twice, as Chi is reachable through two different friends:

```
print [friend["id"] for friend in users[0]["friends"]]  # [1, 2]
print [friend["id"] for friend in users[1]["friends"]]  # [0, 2, 3]
print [friend["id"] for friend in users[2]["friends"]]  # [0, 1, 3]
```

Knowing that people are friends-of-friends in multiple ways seems like interesting information, so maybe instead we should produce a *count* of mutual friends. And we definitely should use a helper function to exclude people already known to the user:

```
from collections import Counter                    # not loaded by default

def not_the_same(user, other_user):
    """two users are not the same if they have different ids"""
    return user["id"] != other_user["id"]

def not_friends(user, other_user):
    """other_user is not a friend if he's not in user["friends"];
    that is, if he's not_the_same as all the people in user["friends"]"""
    return all(not_the_same(friend, other_user)
               for friend in user["friends"])

def friends_of_friend_ids(user):
    return Counter(foaf["id"]
                   for friend in user["friends"]    # for each of my friends
                   for foaf in friend["friends"]    # count *their* friends
                   if not_the_same(user, foaf)      # who aren't me
                   and not_friends(user, foaf))     # and aren't my friends

print friends_of_friend_ids(users[3])              # Counter({0: 2, 5: 1})
```

This correctly tells Chi (`id` 3) that she has two mutual friends with Hero (`id` 0) but only one mutual friend with Clive (`id` 5).

As a data scientist, you know that you also might enjoy meeting users with similar interests. (This is a good example of the "substantive expertise" aspect of data science.) After asking around, you manage to get your hands on this data, as a list of pairs `(user_id, interest)`:

```
interests = [
    (0, "Hadoop"), (0, "Big Data"), (0, "HBase"), (0, "Java"),
    (0, "Spark"), (0, "Storm"), (0, "Cassandra"),
    (1, "NoSQL"), (1, "MongoDB"), (1, "Cassandra"), (1, "HBase"),
    (1, "Postgres"), (2, "Python"), (2, "scikit-learn"), (2, "scipy"),
    (2, "numpy"), (2, "statsmodels"), (2, "pandas"), (3, "R"), (3, "Python"),
    (3, "statistics"), (3, "regression"), (3, "probability"),
    (4, "machine learning"), (4, "regression"), (4, "decision trees"),
    (4, "libsvm"), (5, "Python"), (5, "R"), (5, "Java"), (5, "C++"),
    (5, "Haskell"), (5, "programming languages"), (6, "statistics"),
    (6, "probability"), (6, "mathematics"), (6, "theory"),
    (7, "machine learning"), (7, "scikit-learn"), (7, "Mahout"),
    (7, "neural networks"), (8, "neural networks"), (8, "deep learning"),
    (8, "Big Data"), (8, "artificial intelligence"), (9, "Hadoop"),
    (9, "Java"), (9, "MapReduce"), (9, "Big Data")
]
```

For example, Thor (`id` 4) has no friends in common with Devin (`id` 7), but they share an interest in machine learning.

It's easy to build a function that finds users with a certain interest:

```
def data_scientists_who_like(target_interest):
    return [user_id
            for user_id, user_interest in interests
            if user_interest == target_interest]
```

This works, but it has to examine the whole list of interests for every search. If we have a lot of users and interests (or if we just want to do a lot of searches), we're probably better off building an index from interests to users:

```
from collections import defaultdict

# keys are interests, values are lists of user_ids with that interest
user_ids_by_interest = defaultdict(list)

for user_id, interest in interests:
    user_ids_by_interest[interest].append(user_id)
```

And another from users to interests:

```
# keys are user_ids, values are lists of interests for that user_id
interests_by_user_id = defaultdict(list)
```

```
for user_id, interest in interests:
    interests_by_user_id[user_id].append(interest)
```

Now it's easy to find who has the most interests in common with a given user:

- Iterate over the user's interests.
- For each interest, iterate over the other users with that interest.
- Keep count of how many times we see each other user.

```
def most_common_interests_with(user):
    return Counter(interested_user_id
        for interest in interests_by_user_id[user["id"]]
        for interested_user_id in user_ids_by_interest[interest]
        if interested_user_id != user["id"])
```

We could then use this to build a richer "Data Scientists You Should Know" feature based on a combination of mutual friends and mutual interests. We'll explore these kinds of applications in Chapter 22.

Salaries and Experience

Right as you're about to head to lunch, the VP of Public Relations asks if you can provide some fun facts about how much data scientists earn. Salary data is of course sensitive, but he manages to provide you an anonymous data set containing each user's `salary` (in dollars) and `tenure` as a data scientist (in years):

```
salaries_and_tenures = [(83000, 8.7), (88000, 8.1),
                        (48000, 0.7), (76000, 6),
                        (69000, 6.5), (76000, 7.5),
                        (60000, 2.5), (83000, 10),
                        (48000, 1.9), (63000, 4.2)]
```

The natural first step is to plot the data (which we'll see how to do in Chapter 3). You can see the results in Figure 1-3.

Figure 1-3. Salary by years of experience

It seems pretty clear that people with more experience tend to earn more. How can you turn this into a fun fact? Your first idea is to look at the average salary for each tenure:

```
# keys are years, values are lists of the salaries for each tenure
salary_by_tenure = defaultdict(list)

for salary, tenure in salaries_and_tenures:
    salary_by_tenure[tenure].append(salary)

# keys are years, each value is average salary for that tenure
average_salary_by_tenure = {
    tenure : sum(salaries) / len(salaries)
    for tenure, salaries in salary_by_tenure.items()
}
```

This turns out to be not particularly useful, as none of the users have the same tenure, which means we're just reporting the individual users' salaries:

```
{0.7: 48000.0,
 1.9: 48000.0,
 2.5: 60000.0,
 4.2: 63000.0,
```

```
 6: 76000.0,
 6.5: 69000.0,
 7.5: 76000.0,
 8.1: 88000.0,
 8.7: 83000.0,
 10: 83000.0}
```

It might be more helpful to bucket the tenures:

```
def tenure_bucket(tenure):
    if tenure < 2:
        return "less than two"
    elif tenure < 5:
        return "between two and five"
    else:
        return "more than five"
```

Then group together the salaries corresponding to each bucket:

```
# keys are tenure buckets, values are lists of salaries for that bucket
salary_by_tenure_bucket = defaultdict(list)

for salary, tenure in salaries_and_tenures:
    bucket = tenure_bucket(tenure)
    salary_by_tenure_bucket[bucket].append(salary)
```

And finally compute the average salary for each group:

```
# keys are tenure buckets, values are average salary for that bucket
average_salary_by_bucket = {
  tenure_bucket : sum(salaries) / len(salaries)
  for tenure_bucket, salaries in salary_by_tenure_bucket.iteritems()
}
```

which is more interesting:

```
{'between two and five': 61500.0,
 'less than two': 48000.0,
 'more than five': 79166.66666666667}
```

And you have your soundbite: "Data scientists with more than five years experience earn 65% more than data scientists with little or no experience!"

But we chose the buckets in a pretty arbitrary way. What we'd really like is to make some sort of statement about the salary effect—on average—of having an additional year of experience. In addition to making for a snappier fun fact, this allows us to *make predictions* about salaries that we don't know. We'll explore this idea in Chapter 14.

Paid Accounts

When you get back to your desk, the VP of Revenue is waiting for you. She wants to better understand which users pay for accounts and which don't. (She knows their names, but that's not particularly actionable information.)

You notice that there seems to be a correspondence between years of experience and paid accounts:

```
0.7 paid
1.9 unpaid
2.5 paid
4.2 unpaid
6   unpaid
6.5 unpaid
7.5 unpaid
8.1 unpaid
8.7 paid
10  paid
```

Users with very few and very many years of experience tend to pay; users with average amounts of experience don't.

Accordingly, if you wanted to create a model—though this is definitely not enough data to base a model on—you might try to predict "paid" for users with very few and very many years of experience, and "unpaid" for users with middling amounts of experience:

```
def predict_paid_or_unpaid(years_experience):
  if years_experience < 3.0:
    return "paid"
  elif years_experience < 8.5:
    return "unpaid"
  else:
    return "paid"
```

Of course, we totally eyeballed the cutoffs.

With more data (and more mathematics), we could build a model predicting the likelihood that a user would pay, based on his years of experience. We'll investigate this sort of problem in Chapter 16.

Topics of Interest

As you're wrapping up your first day, the VP of Content Strategy asks you for data about what topics users are most interested in, so that she can plan out her blog calendar accordingly. You already have the raw data from the friend-suggester project:

```
interests = [
    (0, "Hadoop"), (0, "Big Data"), (0, "HBase"), (0, "Java"),
    (0, "Spark"), (0, "Storm"), (0, "Cassandra"),
```

```
    (1, "NoSQL"), (1, "MongoDB"), (1, "Cassandra"), (1, "HBase"),
    (1, "Postgres"), (2, "Python"), (2, "scikit-learn"), (2, "scipy"),
    (2, "numpy"), (2, "statsmodels"), (2, "pandas"), (3, "R"), (3, "Python"),
    (3, "statistics"), (3, "regression"), (3, "probability"),
    (4, "machine learning"), (4, "regression"), (4, "decision trees"),
    (4, "libsvm"), (5, "Python"), (5, "R"), (5, "Java"), (5, "C++"),
    (5, "Haskell"), (5, "programming languages"), (6, "statistics"),
    (6, "probability"), (6, "mathematics"), (6, "theory"),
    (7, "machine learning"), (7, "scikit-learn"), (7, "Mahout"),
    (7, "neural networks"), (8, "neural networks"), (8, "deep learning"),
    (8, "Big Data"), (8, "artificial intelligence"), (9, "Hadoop"),
    (9, "Java"), (9, "MapReduce"), (9, "Big Data")
]
```

One simple (if not particularly exciting) way to find the most popular interests is simply to count the words:

1. Lowercase each interest (since different users may or may not capitalize their interests).
2. Split it into words.
3. Count the results.

In code:

```
words_and_counts = Counter(word
                           for user, interest in interests
                           for word in interest.lower().split())
```

This makes it easy to list out the words that occur more than once:

```
for word, count in words_and_counts.most_common():
    if count > 1:
        print word, count
```

which gives the results you'd expect (unless you expect "scikit-learn" to get split into two words, in which case it doesn't give the results you expect):

```
learning 3
java 3
python 3
big 3
data 3
hbase 2
regression 2
cassandra 2
statistics 2
probability 2
hadoop 2
networks 2
machine 2
neural 2
```

```
    scikit-learn 2
    r 2
```

We'll look at more sophisticated ways to extract topics from data in Chapter 20.

Onward

It's been a successful first day! Exhausted, you slip out of the building before anyone else can ask you for anything else. Get a good night's rest, because tomorrow is new employee orientation. (Yes, you went through a full day of work *before* new employee orientation. Take it up with HR.)

Want to read more?

You can buy this book at oreilly.com
in print and ebook format.

Buy 2 books, get the 3rd FREE!

Use discount code OPC10

All orders over $29.95 qualify for **free shipping** within the US.

It's also available at your favorite book retailer, including the iBookstore, the Android Marketplace, and Amazon.com.

O'REILLY®

©2015 O'Reilly Media, Inc. The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. 15055