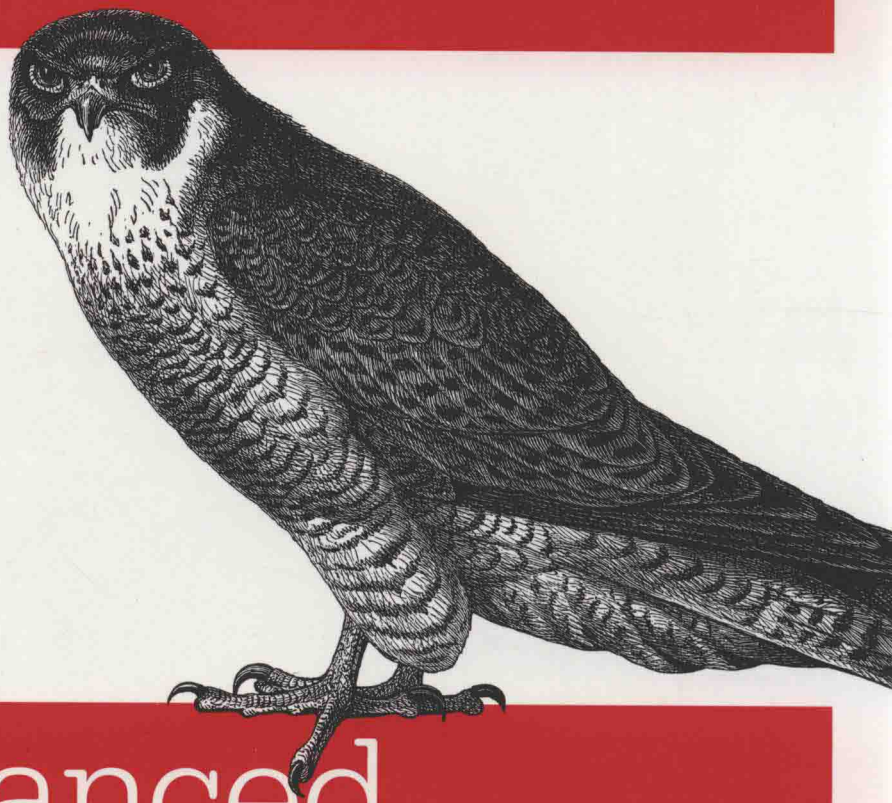
O'REILLY®



Advanced Analytics with Spark

Spark高级数据分析（影印版）

Sandy Ryza, Uri Laserson,
Sean Owen, Josh Wills 著

東南大學出版社

Spark高级数据分析（影印版）

Advanced Analytics with Spark

Sandy Ryza, Uri Laserson,
Sean Owen, Josh Wills 著

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo O'REILLY®

O'Reilly Media, Inc.授权东南大学出版社出版

南京 东南大学出版社

图书在版编目(CIP)数据

Spark高级数据分析:英文/(美)里扎(Ryza, S.)等著. —影印本. —南京:东南大学出版社,2015.9

书名原文:Advanced Analytics with Spark

ISBN 978-7-5641-5910-8

Ⅰ.①S… Ⅱ.①里… Ⅲ.①数据处理软件—英文 Ⅳ.①TP274

中国版本图书馆CIP数据核字(2015)第165527号

江苏省版权局著作权合同登记

图字:10-2015-242号

Spark高级数据分析(影印版)

出版发行:东南大学出版社

地　　址:南京四牌楼2号　　邮编:210096

出 版 人:江建中

网　　址:http://www.seupress.com

电子邮件:press@seupress.com

印　　刷:常州市武进第三印刷有限公司

开　　本:787毫米×980毫米　　16开本

印　　张:17.25

字　　数:338千字

版　　次:2015年9月第1版

印　　次:2015年9月第1次印刷

书　　号:ISBN 978-7-5641-5910-8

定　　价:56.00元

本社图书若有印装质量问题,请直接与营销部联系。电话(传真):025-83791830

Foreword

Ever since we started the Spark project at Berkeley, I've been excited about not just building fast parallel systems, but helping more and more people make use of large-scale computing. This is why I'm very happy to see this book, written by four experts in data science, on advanced analytics with Spark. Sandy, Uri, Sean, and Josh have been working with Spark for a while, and have put together a great collection of content with equal parts explanations and examples.

The thing I like most about this book is its focus on examples, which are all drawn from real applications on real-world data sets. It's hard to find one, let alone ten examples that cover big data and that you can run on your laptop, but the authors have managed to create such a collection and set everything up so you can run them in Spark. Moreover, the authors cover not just the core algorithms, but the intricacies of data preparation and model tuning that are needed to really get good results. You should be able to take the concepts in these examples and directly apply them to your own problems.

Big data processing is undoubtedly one of the most exciting areas in computing today, and remains an area of fast evolution and introduction of new ideas. I hope that this book helps you get started in this exciting new field.

—Matei Zaharia, CTO at Databricks and Vice President, Apache Spark

Preface

Sandy Ryza

I don't like to think I have many regrets, but it's hard to believe anything good came out of a particular lazy moment in 2011 when I was looking into how to best distribute tough discrete optimization problems over clusters of computers. My advisor explained this newfangled Spark thing he had heard of, and I basically wrote off the concept as too good to be true and promptly got back to writing my undergrad thesis in MapReduce. Since then, Spark and I have both matured a bit, but one of us has seen a meteoric rise that's nearly impossible to avoid making "ignite" puns about. Cut to two years later, and it has become crystal clear that Spark is something worth paying attention to.

Spark's long lineage of predecessors, running from MPI to MapReduce, makes it possible to write programs that take advantage of massive resources while abstracting away the nitty-gritty details of distributed systems. As much as data processing needs have motivated the development of these frameworks, in a way the field of big data has become so related to these frameworks that its scope is defined by what these frameworks can handle. Spark's promise is to take this a little further—to make writing distributed programs feel like writing regular programs.

Spark will be great at giving ETL pipelines huge boosts in performance and easing some of the pain that feeds the MapReduce programmer's daily chant of despair ("why? whyyyyy?") to the Hadoop gods. But the exciting thing for me about it has always been what it opens up for complex analytics. With a paradigm that supports iterative algorithms and interactive exploration, Spark is finally an open source framework that allows a data scientist to be productive with large data sets.

I think the best way to teach data science is by example. To that end, my colleagues and I have put together a book of applications, trying to touch on the interactions between the most common algorithms, data sets, and design patterns in large-scale analytics. This book isn't meant to be read cover to cover. Page to a chapter that looks like something you're trying to accomplish, or that simply ignites your interest.

What's in This Book

The first chapter will place Spark within the wider context of data science and big data analytics. After that, each chapter will comprise a self-contained analysis using Spark. The second chapter will introduce the basics of data processing in Spark and Scala through a use case in data cleansing. The next few chapters will delve into the meat and potatoes of machine learning with Spark, applying some of the most common algorithms in canonical applications. The remaining chapters are a bit more of a grab bag and apply Spark in slightly more exotic applications—for example, querying Wikipedia through latent semantic relationships in the text or analyzing genomics data.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at *https://github.com/sryza/aas*.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Advanced Analytics with Spark* by Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills (O'Reilly). Copyright 2015 Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills, 978-1-491-91276-8."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/advanced-spark*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

It goes without saying that you wouldn't be reading this book if it were not for the existence of Apache Spark and MLlib. We all owe thanks to the team that has built and open sourced it, and the hundreds of contributors who have added to it.

We would like to thank everyone who spent a great deal of time reviewing the content of the book with expert eyes: Michael Bernico, Ian Buss, Jeremy Freeman, Chris Fregly, Debashish Ghosh, Juliet Hougland, Jonathan Keebler, Frank Nothaft, Nick Pentreath, Kostas Sakellis, Marcelo Vanzin, and Juliet Hougland again. Thanks all! We owe you one. This has greatly improved the structure and quality of the result.

I (Sandy) also would like to thank Jordan Pinkus and Richard Wang for helping me with some of the theory behind the risk chapter.

Thanks to Marie Beaugureau and O'Reilly, for the experience and great support in getting this book published and into your hands.

Table of Contents

CHAPTER 1

Analyzing Big Data

Sandy Ryza

[Data applications] are like sausages. It is better not to see them being made.
—Otto von Bismarck

- Build a model to detect credit card fraud using thousands of features and billions of transactions.
- Intelligently recommend millions of products to millions of users.
- Estimate financial risk through simulations of portfolios including millions of instruments.
- Easily manipulate data from thousands of human genomes to detect genetic associations with disease.

These are tasks that simply could not be accomplished 5 or 10 years ago. When people say that we live in an age of "big data," they mean that we have tools for collecting, storing, and processing information at a scale previously unheard of. Sitting behind these capabilities is an ecosystem of open source software that can leverage clusters of commodity computers to chug through massive amounts of data. Distributed systems like Apache Hadoop have found their way into the mainstream and have seen widespread deployment at organizations in nearly every field.

But just as a chisel and a block of stone do not make a statue, there is a gap between having access to these tools and all this data, and doing something useful with it. This is where "data science" comes in. As sculpture is the practice of turning tools and raw material into something relevant to nonsculptors, data science is the practice of turning tools and raw data into something that nondata scientists might care about.

Often, "doing something useful" means placing a schema over it and using SQL to answer questions like "of the gazillion users who made it to the third page in our

registration process, how many are over 25?" The field of how to structure a data warehouse and organize information to make answering these kinds of questions easy is a rich one, but we will mostly avoid its intricacies in this book.

Sometimes, "doing something useful" takes a little extra. SQL still may be core to the approach, but to work around idiosyncrasies in the data or perform complex analysis, we need a programming paradigm that's a little bit more flexible and a little closer to the ground, and with richer functionality in areas like machine learning and statistics. These are the kinds of analyses we are going to talk about in this book.

For a long time, open source frameworks like R, the PyData stack, and Octave have made rapid analysis and model building viable over small data sets. With fewer than 10 lines of code, we can throw together a machine learning model on half a data set and use it to predict labels on the other half. With a little more effort, we can impute missing data, experiment with a few models to find the best one, or use the results of a model as inputs to fit another. What should an equivalent process look like that can leverage clusters of computers to achieve the same outcomes on huge data sets?

The right approach might be to simply extend these frameworks to run on multiple machines, to retain their programming models and rewrite their guts to play well in distributed settings. However, the challenges of distributed computing require us to rethink many of the basic assumptions that we rely on in single-node systems. For example, because data must be partitioned across many nodes on a cluster, algorithms that have wide data dependencies will suffer from the fact that network transfer rates are orders of magnitude slower than memory accesses. As the number of machines working on a problem increases, the probability of a failure increases. These facts require a programming paradigm that is sensitive to the characteristics of the underlying system: one that discourages poor choices and makes it easy to write code that will execute in a highly parallel manner.

Of course, single-machine tools like PyData and R that have come to recent prominence in the software community are not the only tools used for data analysis. Scientific fields like genomics that deal with large data sets have been leveraging parallel computing frameworks for decades. Most people processing data in these fields today are familiar with a cluster-computing environment called HPC (high-performance computing). Where the difficulties with PyData and R lie in their inability to scale, the difficulties with HPC lie in its relatively low level of abstraction and difficulty of use. For example, to process a large file full of DNA sequencing reads in parallel, we must manually split it up into smaller files and submit a job for each of those files to the cluster scheduler. If some of these fail, the user must detect the failure and take care of manually resubmitting them. If the analysis requires all-to-all operations like sorting the entire data set, the large data set must be streamed through a single node, or the scientist must resort to lower-level distributed frameworks like MPI, which are difficult to program without extensive knowledge of C and distributed/networked

systems. Tools written for HPC environments often fail to decouple the in-memory data models from the lower-level storage models. For example, many tools only know how to read data from a POSIX filesystem in a single stream, making it difficult to make tools naturally parallelize, or to use other storage backends, like databases. Recent systems in the Hadoop ecosystem provide abstractions that allow users to treat a cluster of computers more like a single computer—to automatically split up files and distribute storage over many machines, to automatically divide work into smaller tasks and execute them in a distributed manner, and to automatically recover from failures. The Hadoop ecosystem can automate a lot of the hassle of working with large data sets, and is far cheaper than HPC.

The Challenges of Data Science

A few hard truths come up so often in the practice of data science that evangelizing these truths has become a large role of the data science team at Cloudera. For a system that seeks to enable complex analytics on huge data to be successful, it needs to be informed by, or at least not conflict with, these truths.

First, the vast majority of work that goes into conducting successful analyses lies in preprocessing data. Data is messy, and cleansing, munging, fusing, mushing, and many other verbs are prerequisites to doing anything useful with it. Large data sets in particular, because they are not amenable to direct examination by humans, can require computational methods to even discover what preprocessing steps are required. Even when it comes time to optimize model performance, a typical data pipeline requires spending far more time in feature engineering and selection than in choosing and writing algorithms.

For example, when building a model that attempts to detect fraudulent purchases on a website, the data scientist must choose from a wide variety of potential features: any fields that users are required to fill out, IP location info, login times, and click logs as users navigate the site. Each of these comes with its own challenges in converting to vectors fit for machine learning algorithms. A system needs to support more flexible transformations than turning a 2D array of doubles into a mathematical model.

Second, *iteration* is a fundamental part of the data science. Modeling and analysis typically require multiple passes over the same data. One aspect of this lies *within* machine learning algorithms and statistical procedures. Popular optimization procedures like stochastic gradient descent and expectation maximization involve repeated scans over their inputs to reach convergence. Iteration also matters within the data scientist's own workflow. When data scientists are initially investigating and trying to get a feel for a data set, usually the results of a query inform the next query that should run. When building models, data scientists do not try to get it right in one try. Choosing the right features, picking the right algorithms, running the right significance tests, and finding the right hyperparameters all require experimentation. A

framework that requires reading the same data set from disk each time it is accessed adds delay that can slow down the process of exploration and limit the number of things we get to try.

Third, the task isn't over when a well-performing model has been built. If the point of data science is making data useful to nondata scientists, then a model stored as a list of regression weights in a text file on the data scientist's computer has not really accomplished this goal. Uses of data recommendation engines and real-time fraud detection systems culminate in data applications. In these, models become part of a production service and may need to be rebuilt periodically or even in real time.

For these situations, it is helpful to make a distinction between analytics in the *lab* and analytics in the *factory*. In the lab, data scientists engage in exploratory analytics. They try to understand the nature of the data they are working with. They visualize it and test wild theories. They experiment with different classes of features and auxiliary sources they can use to augment it. They cast a wide net of algorithms in the hopes that one or two will work. In the factory, in building a data application, data scientists engage in operational analytics. They package their models into services that can inform real-world decisions. They track their models' performance over time and obsess about how they can make small tweaks to squeeze out another percentage point of accuracy. They care about SLAs and uptime. Historically, exploratory analytics typically occurs in languages like R, and when it comes time to build production applications, the data pipelines are rewritten entirely in Java or C++.

Of course, everybody could save time if the original modeling code could be actually used in the app for which it is written, but languages like R are slow and lack integration with most planes of the production infrastructure stack, and languages like Java and C++ are just poor tools for exploratory analytics. They lack Read-Evaluate-Print Loop (REPL) environments for playing with data interactively and require large amounts of code to express simple transformations. A framework that makes modeling easy but is also a good fit for production systems is a huge win.

Introducing Apache Spark

Enter Apache Spark, an open source framework that combines an engine for distributing programs across clusters of machines with an elegant model for writing programs atop it. Spark, which originated at the UC Berkeley AMPLab and has since been contributed to the Apache Software Foundation, is arguably the first open source software that makes distributed programming truly accessible to data scientists.

One illuminating way to understand Spark is in terms of its advances over its predecessor, MapReduce. MapReduce revolutionized computation over huge data sets by offering a simple model for writing programs that could execute in parallel across

hundreds to thousands of machines. The MapReduce engine achieves near linear scalability—as the data size increases, we can throw more computers at it and see jobs complete in the same amount of time—and is resilient to the fact that failures that occur rarely on a single machine occur all the time on clusters of thousands. It breaks up work into small *tasks* and can gracefully accommodate task failures without compromising the job to which they belong.

Spark maintains MapReduce's linear scalability and fault tolerance, but extends it in three important ways. First, rather than relying on a rigid map-then-reduce format, its engine can execute a more general directed acyclic graph (DAG) of operators. This means that, in situations where MapReduce must write out intermediate results to the distributed filesystem, Spark can pass them directly to the next step in the pipeline. In this way, it is similar to *Dryad* (*http://research.microsoft.com/en-us/projects/dryad/*), a descendant of MapReduce that originated at Microsoft Research. Second, it complements this capability with a rich set of transformations that enable users to express computation more naturally. It has a strong developer focus and streamlined API that can represent complex pipelines in a few lines of code.

Third, Spark extends its predecessors with in-memory processing. Its Resilient Distributed Dataset (RDD) abstraction enables developers to materialize any point in a processing pipeline into memory across the cluster, meaning that future steps that want to deal with the same data set need not recompute it or reload it from disk. This capability opens up use cases that distributed processing engines could not previously approach. Spark is well suited for highly iterative algorithms that require multiple passes over a data set, as well as reactive applications that quickly respond to user queries by scanning large in-memory data sets.

Perhaps most importantly, Spark fits well with the aforementioned hard truths of data science, acknowledging that the biggest bottleneck in building data applications is not CPU, disk, or network, but analyst productivity. It perhaps cannot be overstated how much collapsing the full pipeline, from preprocessing to model evaluation, into a single programming environment can speed up development. By packaging an expressive programming model with a set of analytic libraries under a REPL, it avoids the round trips to IDEs required by frameworks like MapReduce and the challenges of subsampling and moving data back and forth from HDFS required by frameworks like R. The more quickly analysts can experiment with their data, the higher likelihood they have of doing something useful with it.

With respect to the pertinence of munging and ETL, Spark strives to be something closer to the Python of big data than the Matlab of big data. As a general-purpose computation engine, its core APIs provide a strong foundation for data transformation independent of any functionality in statistics, machine learning, or matrix algebra. Its Scala and Python APIs allow programming in expressive general-purpose languages, as well as access to existing libraries.