数据科学与大数据技术

Python 和 PySpark 数据分析

[加] 乔纳森·里乌(Jonathan Rioux) 著

殷海英 译

清華大學出版社

北 京

北京市版权局著作权合同登记号 图字：01-2023-4142

Jonathan Rioux
Data Analysis with Python and PySpark
EISBN: 9781617297205
Original English language edition published by Manning Publications, USA © 2022 by Manning Publications. Simplified Chinese-language edition copyright © 2023 by Tsinghua University Press Limited. All rights reserved.

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。举报：010-62782989，beiqinquan@tup.tsinghua.edu.cn。

图书在版编目(CIP)数据

Python 和 PySpark 数据分析 / (加) 乔纳森 • 里乌(Jonathan Rioux) 著；殷海英译. —北京：清华大学出版社，2023.10
(数据科学与大数据技术)
书名原文：Data Analysis with Python and PySpark
ISBN 978-7-302-64536-8

I. ①P… II. ①乔… ②殷… III. ①软件工具—程序设计 ②数据处理 IV. ①TP311.561 ②TP274

中国国家版本馆 CIP 数据核字(2023)第 169057 号

责任编辑：王 军
装帧设计：孔祥峰
责任校对：成凤进
责任印制：宋 林

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-83470000 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：大厂回族自治县彩虹印刷有限公司
经 销：全国新华书店
开 本：170mm×240mm 印 张：26.25 字 数：606 千字
版 次：2023 年 10 月第 1 版 印 次：2023 年 10 月第 1 次印刷
定 价：118.00 元

产品编号：097314-01

译 者 序

在为学生讲授《数据科学 101》课程时，同学们常常向我提出以下问题：“为什么我们配置了拥有 16 颗计算核心的服务器，但在处理数据时，CPU 占用率仅为不到 10%，导致程序运行缓慢？” 我向他们解释道：“这是 Python 解释器中的全局解释器锁(GIL)导致的问题”。GIL 确保在任何给定时间只有一个线程可以在解释器中执行字节码。这意味着 Python 的多线程并不能真正实现并行执行，尤其是在 CPU 密集型任务中。为了解决这一问题，我鼓励大家注册参加我的《数据科学 102》课程，我将在那里介绍如何利用 PySpark 解决并行处理问题。于是，在《数据科学 102》的课堂上，我看到了许多熟悉的面孔。

近年来，“数据科学”这个词变得非常流行。市面上的书籍和教程(包括我翻译并出版的几本关于数据科学的书籍在内)，通常都侧重于某种特定的实现技术。然而，在实际工作中，我们首要面临的问题是如何快速完成数据处理和计算。因为在工作环境中，我们通常使用的是大规模计算集群，它由许多带有 128 颗计算核心的服务器组成，而不是像上课时使用的笔记本电脑那样只有几个计算核心。如果你按照那些“教程”编写程序，在程序运行时，单台服务器的 CPU 使用率可能不到 1%，而程序仍然无法及时完成任务。你的客户或内部经理肯定对这样的运行状况感到不满意。那么，如何使用简单的技术解决这个问题呢？使用本书介绍的 PySpark 绝对是一个很好的选择。

为什么选择 PySpark？PySpark 是 Python 编程语言与 Apache Spark 分布式计算框架的完美结合。Apache Spark 作为一种快速、通用且可扩展的大数据处理框架，通过其强大的分布式计算能力，为数据科学家、工程师和研究人员提供了处理大规模数据集的解决方案。而 PySpark 作为 Python 开发者的首选工具，为他们提供了使用 Python 语言轻松处理大数据的能力。PySpark 的魅力在于其简洁而强大的 API。通过 PySpark，开发者可以利用 Python 的直观和简洁语法，编写并行、可扩展的数据处理和分析应用程序。PySpark 提供了一系列高级抽象，如 RDDs、DataFrames 和 Spark SQL 等，使得处理结构化和非结构化数据变得更加容易。通过 PySpark，你可以对数据进行各种转换和操作，如筛选、映射、聚合、连接和排序等。无论是简单的数据处理任务还是复杂的机器学习和深度学习应用，PySpark 都能够以高效的方式处理。其内置的 MLlib 机器学习库提供了丰富的算法和工具，使得训练和评估机器学习模型变得更加简单和可扩展。在 PySpark 中，分布式计算是关键。它利用 Spark 的任务调度和执行引擎，将任务分解为多个阶段，并在集群中的多个计算节点上并行执行这些阶段。这种分布式计算的优势使得 PySpark 能够处理大规模数据集，并以极快的速度完成任务。通过利用分布式计算，PySpark 能够提供卓越的性能和可扩展性，使得数据处理和分析的工作变得更加高效和可靠。不仅如此，PySpark

还具备与 Python 生态系统的良好兼容性。开发者可以轻松地整合其他 Python 库和工具，如 NumPy、Pandas 和 Matplotlib 等，以便进行更丰富的数据处理、可视化和分析。

本书有 14 章和 3 个附录，详细介绍了如何将 PySpark 应用到日常的数据科学工作中。通过通俗易懂的示例，介绍了 PySpark 中的实用知识点和语法，使你能够轻松掌握 PySpark 的核心概念，并将其应用于实际工作中。

在本书中，你将学习如何使用 PySpark 进行数据预处理、模型训练和评估等常见的数据科学任务。每个章节都以具体的示例和案例展示了 PySpark 在不同情景下的应用方法和技巧。通过实际操作，你将了解如何使用 PySpark 处理大规模数据集、构建复杂的数据处理流程以及应对常见的数据质量和性能挑战。

最后，我要由衷感谢清华大学出版社的王军老师。感谢他给我提供机会翻译并出版了多本关于机器学习、人工智能、云计算和高性能计算的书籍。他提供了一种新的方式，让我能够与大家分享我的知识和经验。他的支持和帮助使我能够将这些书籍带给读者，并希望这些书籍能对读者在相关领域的学习和工作有所帮助。

殷海英
埃尔赛贡多，加利福尼亚州

作 者 简 介

Jonathan Rioux 每天都在使用 PySpark。他还向数据科学家、数据工程师和精通数据的业务分析师讲授大规模数据分析。

在进入咨询行业成为机器学习和数据分析专家之前，Jonathan 在保险行业的各种分析岗位上工作了 10 年。他目前担任 Laivly 公司的机器学习总监，该公司为客户提供友好的人工智能和自动化机器学习解决方案，以创造最好的客户体验。

致　谢

虽然封面上有我的名字，但本书是团队努力的成果，我想花点时间感谢一路帮助我的人。

首先，我要感谢我的家人。写一本书是一项艰巨的工作，同时伴随着许多抱怨。Simon、Catherine、Véronique、Jean，谢谢你们。

Regina，在某种程度上，你是我的第一个 PySpark 学生。通过与你合作，真的改变了我事业上的一切。我将永远珍惜我们一起工作的时光，遇到你是我的幸运。

我要感谢 Renata Pompas，她允许我使用在她的监督下制作的调色板来制作书中的图表。我是色盲，所以找到一套让我满意且一致的安全颜色，对本书的编写很有帮助。如果你觉得这些图表不错，那就感谢她(以及优秀的平面设计师)。如果它们看起来很糟糕，就怪我。

感谢我在 EPAM 的团队，特别感谢 Zac、James、Nasim、Vahid、Dmitrii、Yuriy、Val、Robert、Aliaksandra、Ihor、Pooyan、Artem、Volha、Ekaterina、Sergey、Sergei、Siarhei、Kseniya、Artemii、Anatoly、Yuliya、Nadzeya、Artsiom、Denis、Yevhen、sofiia、Roman、Mykola、Lisa、Gaurav、Megan。从我宣布要写书的那天起，到我写下这段话的时候，我都感到了团队的支持和鼓励。感谢 Laivly 团队，Jeff、Rod、Craig、Jordan、Abu、Brendan、Daniel、Guy 和 Reid，感谢他们与我一起冒险的经历。我向你们保证未来是光明的。

在写本书的过程中，我有幸与一些优秀的播客制作人在 PySpark 上交流：Test & Code 的 Brian，WHAT the Data?!的 Lior 和 Michael，以及 Profitable Python 的 Ben。非常感谢你们邀请我参加交流。

我想感谢在编写过程中对手稿提供意见的读者，以及提供出色反馈的审稿人：Alex Lucas、David Cronkite、Dianshuang Wu、Gary Bake、Geoff Clark、Gustavo Patino、Igor Vieira、Javier Collado Cabeza、Jeremy Loscheider、Josh Cohen、Kay Engelhardt、Kim Falk、Michael Kareev、Mike Jensen、Patrick A. Mol、Paul Fornia、Peter Hampton、Philippe Van Bergen、Rambabu Posa、Raushan Jha、Sergio Govoni、Sriram Macharla、Stephen Oates 以及 Werner Nindl。

最后，也是最重要的，我要感谢 Manning 出版社的梦之队，是他们让本书得以出版。有许多人为我提供了不可思议的神奇经历：Marjan Bace、Michael Stephens、Rebecca Rinehart、Bert Bates、Candace Gillhoolley、Radmila Ercegovac、Aleks Dragosavljevic、Matko Hrvatin、Christopher Kaufmann、Ana Romac、Branko Latincic、Lucas Weber、Stjepan Jurekovic、Goran Ore、Keri Hales、Michele Mitchell、Melody Dolab 以及 Manning 制作团队的其他成员。

说到 Manning，我要感谢两本书的作者：*wxPython in Action* 的作者 Noel Rappin 和 Robin Dunn(Manning, 2016)，以及 *The Joy of Clojure* 的作者 Michael Fogus 和 Chris Houser(Manning, 2014)。这些书激发了我的灵感，让我一头扎进了编程(然后是数据科学)。在某种程度上，他们是本书诞生的最初火花。

我想强调 Manning 团队，他们帮助我在日常工作中保持责任感，也让我为本书感到自豪。 Arthur Zubarev，真不敢相信我们住在同一个城市却还没见过面！感谢你的反馈，感谢你回答我的许多问题。Alex Ott，我觉得不可能找到比你更好的技术顾问了。Databricks，能拥有你真是太幸运了。最后，同样重要的是，我要感谢 Marina Michaels，从我有写本书的想法起，她就一直支持我。写一本书比我最初想象的要困难得多，但你让整个经历充满乐趣。我从心底感谢你。

关 于 本 书

本书将教你如何使用 PySpark 执行自己的大数据分析程序，以实际的场景讲授如何使用 PySpark 以及为什么使用 PySpark。你将学习如何有效地采集和处理大规模数据，以及如何编写自己的数据转换程序。读完本书后，你应该能够熟练地使用 PySpark 编写自己的数据分析程序。

本书目标读者

本书使用由浅入深的用例展开，从简单的数据转换一直到机器学习管道。本书涵盖了数据分析的整个生命周期，从数据采集到结果使用，添加了更多关于数据源使用和转换的实用技术。

本书主要面向数据分析师、数据工程师和数据科学家，他们希望将 Python 代码扩展到更大的数据集。理想情况下，你应该在工作中或学习编程时编写过一些与数据相关的程序。如果已经熟练使用 Python 编程语言及其生态系统，你会从本书中学到更多的实用内容。

Spark(当然还有 PySpark)从面向对象和函数式编程中借鉴了很多内容。我认为，仅仅为了有效地使用大数据，就要求完全掌握两种编程范式是不合理的。如果你理解 Python 类、装饰器和高级函数，就能熟练使用书中一些更高级的结构，让 PySpark 按照你的意愿运行。如果你对这些概念不熟悉，我会在本书(如果适用)正文和附录中讨论 PySpark。

本书组织结构：路线图

本书分为 3 部分。第 I 部分介绍 PySpark 及其计算模型。本部分还会介绍如何构建和提交一个简单的数据程序，重点介绍在每个 PySpark 程序中都会用到的核心操作，如数据帧中的数据选择、筛选、连接和分组。

第 II 部分通过引入分层数据进一步深入介绍数据转换，分层数据是 PySpark 中可扩展数据程序的关键元素。我们还通过明智地引入 SQL 代码，探索弹性分布式数据集/用户自定义函数，在 PySpark 中高效地使用 pandas 及窗口函数，使程序更具表现力、更加灵活并具有更好的性能。我们还会探讨 Spark 的报表功能和资源管理，以找出潜在的性能问题。

最后，第 III 部分在第 I 部分和第 II 部分的基础上介绍了如何在 PySpark 中构建机器学习程序。在构建和评估机器学习管道之前，使用数据转换工具包创建和选择特征。最后，创建自己的机器学习管道组件，确保我们的机器学习程序具有最大的可用性和可读性。

本书的大部分章节都有练习。你应该能够用学到的知识回答这些问题。

我建议你从头到尾按顺序阅读本书，并根据需要使用附录。如果你想直接深入某个主题，我仍然建议你在深入某一章节之前先学习第 I 部分。下面是一些硬依赖和软依赖，有助于你更高效地阅读本书。

- 第 3 章是第 2 章的直接延续。
- 第 5 章是第 4 章的直接延续。
- 第 9 章使用了第 8 章介绍的一些概念，但资深读者可以自己阅读。
- 第 12 章、第 13 章和第 14 章最好依次阅读。

关于代码

本书最适合使用 Spark 3.1 或 Spark 3.2：Spark 3 中引入了许多新功能，目前大多数商业版本都默认使用该版本。在适当的时候，我会为 Spark 2.3/2.4 提供向后兼容的指令。不推荐 Spark 2.2 或更低版本。我还推荐使用 Python 3.6 及以上版本(本书中我使用的是 Python 3.8.8)。安装说明见附录 B。

可以在 https://github.com/jonesberg/DataAnalysisWithPythonAndPySpark 上找到本书的配套代码库，其中包含数据和代码。有些部分还包含本书所开发程序的可运行版本，以及一些可选的练习。可扫描封底二维码下载本书源代码。

本书包含了许多源代码的示例，有带编号的代码清单，也有正常的文本。在这两种情况下，源代码的格式都是固定等宽字体，以将其与普通文本区分开。有时，代码也会以粗体字显示，以突出显示与本章前面步骤不同的代码，例如在现有代码中添加了新功能。

在许多情况下，原始源代码被重新格式化；添加了换行符，并修改了缩进，以方便排版和印刷。在极少数情况下，即使这样也不能满足排版需求，代码清单还会包括行连续标记(➥)。此外，正文中源代码的注释通常会被删除。代码清单中大都包含注释，以突出重要的概念。

前　言

虽然计算机的功能越来越强大，能够处理更大的数据集，但我们对数据的需求增长得更快。因此，我们构建了新的工具，以便在多台计算机上扩展大数据任务。这是有代价的，早期的工具很复杂，不仅需要用户管理数据程序，还需要用户管理计算机集群本身的运行状况和性能。我记得我曾尝试扩展自己的程序，但却得到“只需要对数据集进行采样，然后继续工作”的建议。

PySpark 改变了游戏规则。从流行的 Python 编程语言开始，它提供了一个清晰可读的 API 来操作非常大的数据集。尽管如此，当你坐下来编写代码时，就像在操作一台复杂且功能强大的机器。PySpark 集功能强大、表现力强、用途广泛于一体。通过强大的多维数据模型，无论多大的数据规模，都可以通过清晰的路径构建数据程序，以实现可伸缩性。

作为一名构建信用风险模型的数据科学家，我深深喜欢上 PySpark。在即将把模型迁移到新的大数据环境时，我们需要设计一个计划：在保持业务连续运行的同时智能地转换我们的数据产品。作为一个自封的 Python 人，我的任务是帮助团队熟悉 PySpark 并加速过渡。当我有机会在不同的用例上与无数客户合作时，我对 PySpark 的喜爱程度呈指数级增长。这些项目的共同点是什么？大数据和大问题都可以通过强大的数据模型解决。需要注意的是，大多数学习 Spark 的材料都是针对 Scala 和 Java 的，Python 开发人员只能将代码转换成他们喜欢的编程语言。我写本书是为了将 PySpark 作为数据分析师的优秀工具进行推广。幸运的是，Spark 项目真的让 Python 成为主要参与者。现在，你拥有了扩展数据程序的强大工具，这是史无前例的。

而一旦大数据被驯服，就可以发挥数据的无限可能。

目　录

第 1 章　介绍……1
1.1　什么是 PySpark……1
1.1.1　从头开始：什么是 Spark……2
1.1.2　PySpark = Spark + Python……3
1.1.3　为什么选择 PySpark……3
1.2　PySpark 的工作原理……5
1.2.1　使用集群管理器进行物理规划……6
1.2.2　懒惰的主管成就工厂的高效……8
1.3　你将从本书学到什么……11
1.4　我们将如何开始……12
1.5　本章小结……13

第 I 部分　介绍：PySpark 的第一步

第 2 章　使用 PySpark 编写的第一个数据处理程序……17
2.1　设置 pyspark shell……18
2.1.1　SparkSession 入口点……20
2.1.2　配置 PySpark 的日志级别……21
2.2　映射程序……22
2.3　采集和探索：为数据转换奠定基础……23
2.3.1　用 spark.read 将数据读入数据帧……24
2.3.2　从结构到内容：使用 show() 探索数据帧……27
2.4　简单的列转换：将句子拆解为单词列表……29
2.4.1　使用 select()选择特定的列……30
2.4.2　转换列：将字符串拆分为单词列表……31
2.4.3　重命名列：alias 和 withColumnRenamed……33
2.4.4　重塑数据：将 list 分解成行……34
2.4.5　处理单词：更改大小写并删除标点符号……36
2.5　筛选记录……38
2.6　本章小结……40
2.7　扩展练习……40

第 3 章　提交并扩展你的第一个 PySpark 程序……43
3.1　对记录进行分组：计算词频…43
3.2　使用 orderBy 对结果排序……46
3.3　保存数据帧中的数据……48
3.4　整合所有内容：计数……49
3.4.1　使用 PySpark 的导入约定简化依赖……50
3.4.2　通过方法链简化程序……51
3.5　使用 spark-submit 以批处理模式启动程序……53
3.6　本章未涉及的内容……54
3.7　扩展词频程序……55
3.8　本章小结……56
3.9　扩展练习……57

第 4 章　使用 pyspark.sql 分析表格数据……59
4.1　什么是表格数据……60
4.2　使用 PySpark 分析和处理表格数据……62
4.3　在 PySpark 中读取和评估带分隔符的数据……63

4.3.1 第一次使用专门处理 CSV 文件的 SparkReader …… 63
4.3.2 自定义 SparkReader 对象来读取 CSV 数据文件 …… 65
4.3.3 探索数据世界的轮廓 …… 67
4.4 数据操作基础：选择、删除、重命名、排序及诊断 …… 68
4.4.1 了解我们想要什么：选择列 …… 69
4.4.2 只保留我们想要的：删除列 …… 71
4.4.3 创建新列：使用 withColumn() 创建新列 …… 73
4.4.4 整理数据帧：对列进行重命名和重排序 …… 76
4.4.5 用 describe()和 summary()分析数据帧 …… 78
4.5 本章小结 …… 80
4.6 扩展练习 …… 80

第 5 章 数据帧操作：连接和分组 ……83
5.1 连接数据 …… 83
5.1.1 探索连接的世界 …… 84
5.1.2 了解连接的两边 …… 85
5.1.3 成功连接的规则：谓词 …… 85
5.1.4 连接方法 …… 87
5.1.5 连接的命名约定 …… 91
5.2 通过 groupby 和 GroupedData 汇总数据 …… 94
5.2.1 一个简单的分组蓝图 …… 95
5.2.2 对自定义列使用 agg() …… 98
5.3 处理 null 值：删除或填充 …… 100
5.3.1 立即删除：使用 dropna() 删除具有 null 值的记录 …… 100
5.3.2 使用 fillna()替换 null 值 …… 101
5.4 问题回顾：端到端程序 …… 102
5.5 本章小结 …… 105
5.6 扩展练习 …… 105

第 II 部分 进级：将你的想法转化为代码

第 6 章 多维数据帧：使用 PySpark 处理 JSON 数据 ……109
6.1 读取 JSON 数据：为 schemapocalypse 做好准备 …… 109
6.1.1 从小处开始：将 JSON 数据作为受限的 Python 字典 …… 111
6.1.2 更进一步：在 PySpark 中读取 JSON 数据 …… 113
6.2 用复杂的数据类型突破二维数据 …… 114
6.2.1 当有多个值时：使用数组 …… 116
6.2.2 映射类型：同一列中的键和值 …… 119
6.3 结构体：列中的嵌套列 …… 121
6.4 构建和使用数据帧模式 …… 125
6.4.1 使用 Spark 类型作为 schema 的基本代码块 …… 125
6.4.2 使用结构体模式原地读取 JSON 文档 …… 127
6.4.3 循环往复：以 JSON 格式指定模式 …… 131
6.5 进行整合：使用复杂数据类型减少重复数据 …… 133
6.5.1 获取“刚刚好”的数据帧：explode 和 collect …… 135
6.5.2 创建自己的层次结构：作为函数的结构体 …… 138
6.6 本章小结 …… 139
6.7 扩展练习 …… 139

第 7 章 双语 PySpark：混合 Python 和 SQL ……141
7.1 根据我们的了解：pyspark.sql 与普通 SQL …… 142
7.2 为 SQL 准备数据帧 …… 143

7.2.1 将数据帧转换为 Spark 表……144
7.2.2 使用 Spark 目录……145
7.3 SQL 和 PySpark……146
7.4 在数据帧方法中使用类似 SQL 的语法……148
7.4.1 获取所需的行和列：select 和 where……148
7.4.2 将相似的记录分组在一起：group by 和 order by……149
7.4.3 在分组后使用 having 进行筛选……150
7.4.4 使用 CREATE 关键字创建新表/视图……151
7.4.5 使用 UNION 和 JOIN 向表中添加数据……153
7.4.6 通过子查询和公共表表达式更好地组织 SQL 代码……154
7.4.7 PySpark 与 SQL 语法的快速总结……157
7.5 简化代码：混合 SQL 和 Python……157
7.5.1 使用 Python 提高弹性并简化数据读取阶段……158
7.5.2 在 PySpark 中使用 SQL 风格的表达式……159
7.6 结论……161
7.7 本章小结……161
7.8 扩展练习……162

第 8 章 使用 Python 扩展 PySpark: RDD 和 UDF……165
8.1 PySpark 中的 RDD……166
8.2 使用 Python 通过 UDF 扩展 PySpark……173
8.2.1 这一切都从简单的 Python 开始：使用类型化的 Python 函数……174
8.2.2 使用 udf()将 Python 函数转换为 UDF……176
8.3 本章小结……179
8.4 扩展练习……179

第 9 章 大数据就是大量的小数据：使用 pandas UDF……181
9.1 通过 pandas 进行列转换：使用 Series UDF……182
9.1.1 将 Spark 连接到谷歌的 BigQuery……183
9.1.2 Series to Series UDF：使用 pandas 的列函数……188
9.1.3 标量 UDF+冷启动=Series UDF 的迭代器……190
9.2 分组数据上的 UDF：聚合与应用……193
9.2.1 分组聚合 UDF……194
9.2.2 分组映射 UDF……196
9.3 何时用，怎么用……198
9.4 本章小结……200
9.5 扩展练习……200

第 10 章 不同视角下的数据：窗口函数……203
10.1 学习并使用简单的窗口函数……204
10.1.1 确定每年最冷的一天……204
10.1.2 创建并使用一个简单的窗口函数来获取最冷的日子……206
10.1.3 比较两种方法……210
10.2 除了汇总：使用排名和分析功能……211
10.2.1 排名函数：看看谁是第一……212
10.2.2 分析函数：回顾过去和展望未来……216
10.3 弹性窗口！使用行和范围的边界……218
10.3.1 计数，窗口样式：静态、增长和无边界……219
10.3.2 范围和行……221

10.4 综合实践：在窗口中使用 UDF……224
10.5 查看窗口：成功的窗口函数的主要步骤……226
10.6 本章小结……226
10.7 扩展练习……226

第 11 章 加速 PySpark：理解 Spark 的查询计划……229
11.1 芝麻开门：通过 Spark UI 了解 Spark 环境……229
11.1.1 查看配置：Environment 选项卡……231
11.1.2 Executors 选项卡和 resource management 选项卡……233
11.1.3 了解任务的运行：通过 Spark UI 判断任务是否完成……237
11.1.4 通过 Spark 查询计划映射操作：SQL 选项卡……240
11.1.5 Spark 的核心：解析、分析、优化和物理计划……242
11.2 关于性能：操作和内存……246
11.2.1 宽操作与窄操作……246
11.2.2 缓存数据帧：功能强大，但往往致命(对于性能而言)…251
11.3 本章小结……254

第Ⅲ部分 使用 PySpark 进行机器学习

第 12 章 准备工作：为机器学习准备特征……259
12.1 阅读、探索和准备机器学习数据集……260
12.1.1 使用 toDF()对列名进行标准化……261
12.1.2 探索数据并获取第一个特征列……262
12.1.3 解决数据错误并构建第一个特征集……264
12.1.4 删除无用记录并估算二分类特征……266
12.1.5 处理极值：清洗带有连续值的列……268
12.1.6 删除不常见的二分类列……271
12.2 特征创建和细化……272
12.2.1 创建自定义特征……272
12.2.2 去除高度相关的特征……273
12.3 基于转换器和估计器的特征准备……276
12.3.1 使用 Imputer 估计器填充连续特征……277
12.3.2 使用 MinMaxScaler 估计器对特征进行缩放……279
12.4 本章小结……281

第 13 章 通过机器学习管道增强机器学习……283
13.1 转换器和估计器： Spark 中机器学习的构建块……284
13.1.1 数据进出：转换器……285
13.1.2 估计器……289
13.2 构建(完整的)机器学习管道……290
13.2.1 使用向量列类型组装最终数据集……292
13.2.2 使用 LogisticRegression 分类器训练机器学习模型……295
13.3 评估和优化模型……297
13.3.1 评估模型准确率：混淆矩阵和评估器对象……298
13.3.2 真阳性与假阳性：ROC 曲线……300
13.3.3 使用交叉验证优化超参数…302
13.4 从模型中获取最大的驱动力：提取系数……305
13.5 本章小结……307

第 14 章　构建自定义机器学习转换器和估计器 ……309
14.1　创建自己的转换器……309
14.1.1　设计转换器：从 Param 和转换的角度思考……310
14.1.2　创建转换器的 Param……312
14.1.3　getter 和 setter：成为 PySpark 中优秀的一员……314
14.1.4　创建自定义转换器的初始化函数……316
14.1.5　创建转换函数……317
14.1.6　使用转换器……319
14.2　创建自己的估计器……321
14.2.1　设计估计器：从模型到 Param……321
14.2.2　实现伴生模型：创建自己的 Mixin……324
14.2.3　创建 ExtremeValueCapper 估计器……326
14.2.4　使用自定义估计器……328
14.3　在机器学习管道中使用转换器和估计器……328
14.3.1　处理多个输入……329
14.3.2　将自定义组件应用于机器学习管道……331
14.4　本章小结……334
14.5　结论：有数据，我就开心……334
附录 A　习题答案……337
附录 B　安装 PySpark……369
附录 C　一些有用的 Python 内容……389

第1章 介　绍

本章主要内容

- 什么是 PySpark
- 为什么 PySpark 是一个有用的分析工具
- Spark 平台的多功能性及其局限性
- PySpark 处理数据的方式

根据几乎所有新闻媒体的说法，数据就是一切，无处不在。新石油、新电力、新黄金、钚，甚至培根！数据是强大的、无形的、珍贵的，也是危险的。与此同时，只有数据本身是不够的，重要的是你用它做什么。毕竟，对于计算机来说，任何数据都是 0 和 1 的集合。作为用户，我们有责任理解如何将其转换为有用的东西。

就像石油、电力、黄金、钚和培根(尤其是培根)一样，我们对数据的需求也在不断增长。事实上，如此之多的数据，以至于计算机都跟不上了。数据的规模和复杂性都在增长，但消费硬件却停滞不前。大多数笔记本电脑的内存在 8GB～16 GB 徘徊，而 SSD 超过几 TB 就会变得非常昂贵。对于当今的数据分析师来说，用 3 倍抵押贷款支付顶级硬件来解决大数据问题的解决方案是什么？

这里介绍了 Apache Spark(在本书中称为 Spark)及其配套工具 PySpark。它们可以利用超级计算机(功能强大但可管理的计算单元组成一个机器网络)的强大功能服务大众。再加上一组强大的数据结构，你可以随时使用它们进行任何工作，这样你就有了一个可以随你一起成长的工具。

本书的目标是提供使用 PySpark 分析数据的工具，无论是需要回答一个快速的数据驱动问题还是构建一个机器学习模型。它涵盖了足够的理论知识，同时提供足够的实践机会。大多数章节都包含一些练习来巩固你刚刚学到的内容。这些练习都在附录 A 中进行了解答和说明。

1.1 什么是 PySpark

名字有什么含义？事实上，有很多含义。只要把 PySpark 分成两部分，就可以推断

出它与 Spark 和 Python 相关。

就其核心而言，PySpark 可以概括为 Spark 的 Python API。虽然这是一个准确的定义，但除非你了解 Python 和 Spark 的含义，否则它并不能提供太多信息。不过，让我们先回答“什么是 Spark?”。有了这些，再看看为什么 Spark 在与 Python 令人难以置信的分析(和机器学习)库结合时变得如此强大。

1.1.1 从头开始：什么是 Spark

据该软件的作者介绍，Apache Spark 是一个“用于大规模数据处理的统一分析引擎”，本书将始终将其称为 Spark。这是一个非常准确的定义，虽然有点枯燥。可以简单地把 Spark 想象为一个分析工厂。在这里，原始材料——数据——被输入，数据、见解、可视化、模型及你能想到的一切——被输出。

就像其他应用场景中通常会通过增加内存来获得更大的容量一样，Spark 也可以通过横向扩展(跨多台较小的计算机)来处理越来越大的数据量，而不是通过纵向扩展(向单台计算机添加更多的资源，如 CPU、内存和磁盘空间)来处理。内存与世界上的大多数东西不一样，它的数量和价格不成正比(例如，一根 128 GB 的内存比两根 64 GB 的内存贵)。这意味着，你将依赖多台计算机，在它们之间分配任务，而不是购买数千美元的内存来容纳数据集。在两台普通计算机比一台大型计算机成本更低的世界中，横向扩展比纵向扩展的成本更低。

云成本和内存

在云计算中，价格往往更为重要。例如，2022 年 1 月，16 核/128 GB 内存的云主机的成本可能是 8 核/64 GB 内存云主机的两倍左右。随着数据量的增长，Spark 可以通过调整给定任务的工作进程和执行进程的数量控制成本。例如，如果有一个数据转换任务，数据集规模不大(只有几 TB)，你可以限制自己使用更小的计算规模，如 5 台计算机。当你想进行机器学习时，可以扩展到 60 台。Databricks(见附录 B)等供应商提供自动扩展功能。也就是说，它们会根据集群的压力，在执行一个任务时增加或减少计算机的数量。自动扩展/成本控制的实现完全取决于供应商(第 11 章介绍了组成 Spark 集群的资源，以及这些资源的用途)。

一台计算机有时会崩溃或表现出不可预测的行为。如果不是 1 台，而是 100 台，那么它们中至少有 1 台出现问题的可能性就要大得多[1]。因此，Spark 有很多需要管理、扩展和监控的环节，让你可以专注于想做的事情，也就是处理数据。

事实上，这正是 Spark 的关键之处：它是一个很好的工具，因为它可以帮助你完成很多事情，之前的许多繁重任务都可以交给它去做。Spark 提供了强大的 API(Application Programming Interface，应用编程接口，提供交互式的函数、类和变量)，让你看起来像在处理一个内聚的数据源，同时又在后台努力优化程序，利用所有可用的功能。你不必是分布式计算方面的专家，只需要熟悉用于构建程序的语言。

1 这可能是一个有趣的概率计算练习，但我会尽量减少数学计算。

1.1.2　PySpark = Spark + Python

PySpark 在 Spark 的计算模型中提供了一个访问 Python 的入口。Spark 本身是用 Scala[1] 编写的。作者在提供不同语言之间的一致接口方面做得很好，同时适当保留了每种语言的特性。因此，无论是 Scala/Spark 程序员还是 Python 程序员，都会非常容易读懂你的 PySpark 程序。

Python 是一种动态的通用语言，可用于许多平台和任务。它的多功能性和表达能力使其特别适合 PySpark。该语言是许多领域最流行的语言之一，目前是数据分析和数据科学的主要力量。它的语法易于学习和阅读，而且可用的软件库数量意味着你经常会找到一个(或多个)适合你的问题的软件库。

PySpark 不仅提供了 Spark 核心 API，还提供了一组定制的扩展常规 Python 代码的功能，以及 pandas 的转换操作。在 Python 的数据分析生态系统中，pandas 实际上是用于内存绑定数据帧的数据帧库(整个数据帧需要驻留在单个计算机的内存中)。现在这不是 PySpark 或 pandas 的问题，而是 PySpark 与 pandas 的问题。第 8 章和第 9 章专门介绍如何将 Python、pandas 和 PySpark 组合成一个愉快的程序。对于那些真正致力于 pandas 语法的人(或者如果你有一个大型的 pandas 程序，想扩展到 PySpark)，Koalas(现在称为 pyspark.pandas 并且是 Spark 3.2.0 版本的一部分。在 PySpark 之上提供了一个类似 pandas 的解决方案。如果你要在 Python 中创建一个新的 Spark 程序，建议使用 PySpark 语法——本书将会详细介绍——当你想更轻松地从 pandas 过渡到 PySpark 时，建议使用 Koalas 语法。程序会运行得更快，可读性也会更好。

1.1.3　为什么选择 PySpark

有很多处理数据的软件库和框架，为什么要花时间专门学习 PySpark？

PySpark 对于现代数据工作负载有很多优势。它集快速、富有表现力和多功能于一身。本节将介绍 PySpark 的许多优点，为什么它的价值主张不仅仅是“Spark, with Python”，以及何时使用其他工具更好。

1. PySpark 速度很快

如果你在搜索引擎中搜索“大数据”，Hadoop 很有可能出现在前几个结果中。这有一个很好的理由：Hadoop 普及了谷歌在 2004 年首创的著名的 MapReduce 框架，并启发大家如何大规模处理数据(第 8 章讨论 PySpark 的底层数据结构时会涉及 MapReduce，即弹性分布式数据集)。

Spark 是几年后创建的，它继承了 Hadoop 令人难以置信的优势。使用积极的查询优化器，明智地使用内存(减少磁盘 I/O，参见第 11 章)，以及其他改进，Spark 的运行速度可以比普通 Hadoop 快 100 倍。由于这两个框架之间的集成，你可以轻松地将 Hadoop 工

1　开发 Spark 的 Databricks 公司有一个名为 Photon 的项目，它是用 C++重写的 Spark 执行引擎。

作流切换到 Spark，从而在不改变硬件的情况下获得一些性能提升。[1]

2. PySpark 简单明了

Python 是最流行且最容易学习的语言之一，PySpark 的 API 从头开始设计，易于理解。PySpark 借用并扩展了 SQL 中的数据操作功能。它以一种流畅的方式做到这一点：对数据帧的每个操作都返回一个“新”数据帧，因此可以将操作一个接一个地连接起来。虽然我们还处在学习 PySpark 的早期阶段，但代码清单 1-1 显示了 PySpark 的可读性及其优越设计。即使没有先验知识，词汇的选择和语法的一致性也使它读起来更加容易。读取一个 CSV 文件，创建一个包含旧列条件值的新列，筛选(使用 where)，根据列值进行分组，生成每个组的计数，最后将结果写回 CSV 文件。所有这些方法在本书的第 I 部分中都有介绍，但现在已经可以推断出这段代码的功能了。

代码清单 1-1　简单的 ETL 管道显示 PySpark 语言简单明了

```
(
    spark.read.csv("./data/list_of_numbers/sample.csv", header=True)
    .withColumn(
       "new_column", F.when(F.col("old_column") > 10, 10).otherwise(0)
    )
    .where("old_column > 8")
    .groupby("new_column")
    .count()
    .write.csv("updated_frequencies.csv", mode="overwrite")
)
```

实际上，Spark 优化了这些操作，这样就不必在每个方法之后获得中间数据帧。因此，可以以一种非常简洁和自描述的方式编写数据转换代码，依靠 Spark 优化最终结果——这是程序员最舒适的状态。

你将在本书中看到许多(更复杂的)示例。我在编写示例时，会很高兴代码最终看起来与最初的(纸笔)推理非常接近。在理解了框架的基本原理之后，我相信你的情况会和我一样。

3. PySpark 是万能的

PySpark 的一个关键优势是多功能性：学习一种工具，然后在各种设置中使用它。这种多功能性有两个组成部分。首先，是框架的可用性；其次，Spark 具有多样化的生态系统。

PySpark 无处不在。三大云提供商[Amazon Web Services (AWS)、Google Cloud Platform (GCP)、Microsoft Azure]都有一个受管理的 Spark 集群作为其产品的一部分，这意味着你只需要单击几个按钮就可以拥有一个功能完整的集群。你还可以轻松地在计算机上安装 Spark，以便在扩展到更强大的集群之前验证你的程序。附录 B 介绍了如何在本地运行 Spark，并简要介绍了当前主要的云产品。

---

1 一如既往，标准的免责声明：不是每个 Hadoop 任务都能在 Spark 中得到加速。你的情况可能会有所不同。在进行大型的架构更改之前，始终要进行充分的测试。

PySpark 是开源的。不像其他分析软件，你不会被某家公司绑住。如果你感到好奇，可以查看 PySpark 的源代码；如果你对新功能有想法或发现 bug，甚至可以贡献代码。它还降低了使用的门槛：只需要下载、学习，就能带来收益。

最后，Spark 的生态系统并不仅限于 PySpark。它还支持 Scala、Java 和 R 的 API，以及最先进的 SQL 层。这使在 Spark 中编写多语言程序变得很容易。Java 软件工程师可以使用 Java 处理 Spark 中的数据转换管道，而数据科学家可以使用 PySpark 构建模型。

4. PySpark 的不足之处

如果 PySpark 能解决所有数据问题，那就太棒了。遗憾的是，没有一种技术是完美的，PySpark 也不例外。因此，当你为下一个项目选择工具时，必须考虑以下因素。

如果你正在处理(非常)小型数据集，并需要快速得到结果，PySpark 就不是最好的选择。在多台计算机上执行程序需要节点之间一定程度的协调工作，这带来了一些额外的开销。如果只使用单个节点，那么你付出了代价，但没有得到收益。例如，PySpark shell 需要几秒钟才能启动，这段时间足够完成内存中的数据处理(数据量较小时，可以将所有要处理的数据加载到内存中)。但是，随着新的 PySpark 版本的发布，有关小型数据集处理的性能差距会越来越小。

与 Java 和 Scala API 相比，PySpark 也存在一个小缺点。由于 Spark 是 Scala 程序的核心，纯 Python 代码必须在 JVM (Java 虚拟机，支持 Java 和 Scala 代码的运行时环境)指令之间进行转换。由于 PySpark 提供了 DataFrame API，不同语言之间的差异已经显著缩小：无论程序是用 Scala、Java 还是 Python 编写的，数据帧操作都将映射到高效的 Spark 操作上，并且运行速度相同。使用弹性分布式数据集(Resilient Distributed Data Set，RDD)数据结构或定义 Python 用户自定义函数时，仍然会发现操作速度较慢。这并不意味着要避免使用它们，这两个主题将在第 8 章讨论。

最后，虽然编写 PySpark 程序很简单，但管理集群可能有一些挑战。Spark 是一个相当复杂的软件。虽然代码库在过去几年已经非常成熟，但还不能像管理单个节点那样轻松地管理 100 台计算机的集群。第 11 章会介绍如何配置 Spark 和进行性能调优，而云计算的选项会让理解 Spark 比以往任何时候都容易(参见附录 B)。对于棘手的问题，可以像我一样做：和运维团队成为朋友。

本节介绍了为什么要使用 PySpark，以及为什么不使用 PySpark，因为知道在何时何地使用 PySpark 是拥有良好开发体验和处理性能的关键。1.2 节会更深入地探讨 Spark 如何处理数据，以及如何让分布式数据处理看起来像在管理一家工厂。

1.2　PySpark 的工作原理

本节介绍 Spark 如何处理程序。展示一个系统的工作原理和基础可能有点奇怪，我们刚刚声称，Spark 隐藏了复杂性。不过，了解 Spark 的设置方式、数据管理方式和查询优化方式仍然很重要。知道了这些，你将能够推理系统，改进代码，并在代码没有按照预期方式执行时及时找出原因并修改。

如果我们依旧以工厂为例进行说明，可以想象 Spark 所在的计算机集群是建筑物。如图 1-1 所示，有两种不同的解释数据工厂的方式。在左边，是从外部看到它的样子：一个整体的单元，项目进来，结果出去。这就是大多数时候看到的样子。内部结构看起来更像右边所示：有一些工作人员被分配到工作台。工作台类似于 Spark 集群中的计算机，数量是固定的。在一些现代的 Spark 实现中，如 Databricks(见附录 B)，支持在运行时自动扩展服务器的数量。有时候需要进行更多的提前计划，特别是在本地运行并且拥有自己的硬件时。在 Spark 的文献中，工作节点被称为执行器节点(executor)，它们在服务器或节点上执行具体的任务。

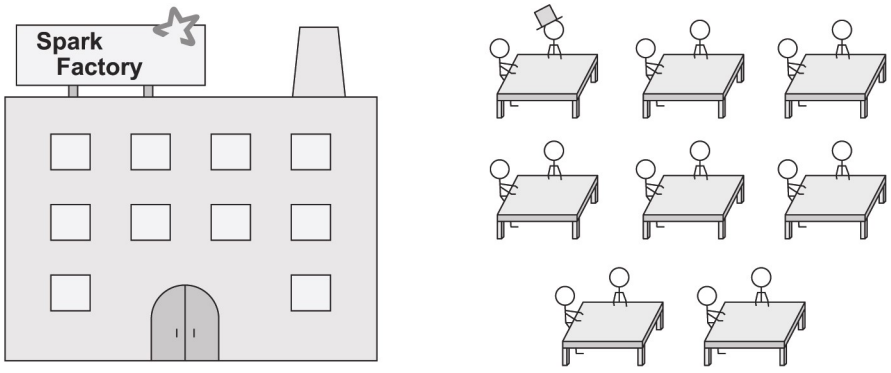

图 1-1　数据工厂内、外部概况图。90%的时间我们关心工厂的整体运行情况，但了解其布局有助于反思代码性能

其中一个工人看起来比其他工人更漂亮。那顶大礼帽绝对让他在人群中脱颖而出。在我们的数据工厂中，他是车间经理。在 Spark 中，称之为 master。这里的 master 位于一个工作台/服务器上，但它也可以位于不同的服务器上(甚至是你的计算机)，这取决于集群管理器和部署模式。master 的角色对程序的高效执行至关重要，因此 1.2.2 节将专门介绍。

提示

在云端，你可以拥有一个高可用性集群，这意味着你的主节点将被复制到多台计算机上。

1.2.1　使用集群管理器进行物理规划

接到任务(在 Spark 世界中称为驱动器程序)后，工厂程序就开始运行。这并不意味着要直接进行处理。在此之前，集群需要进行“容量规划”，也就是为程序分配计算能力。负责管理集群的实体或程序称为集群管理器(cluster manager)。在我们的工厂中，集群管理器将查看有可用空间的工作台，并确保有所需要的工作台数量，然后雇用工人开始生产。在 Spark 中，将查看具有可用计算资源的计算机，并在启动所需数量的执行程序之前保护必要的资源。

注意

Spark 提供了自己的集群管理器，称为独立集群管理器，但在与 Hadoop 或其他大数据平台协同工作时，它也可以与其他集群管理器很好地协同工作。如果你读过 YARN、Mesos 或 Kubernetes 的相关文章，就会知道它们被用作(就 Spark 而言)集群管理器。

任何关于容量的信息(计算机和 executor)都被编码在一个 SparkContext 中，表示到 Spark 集群的连接。如果指令中没有指定容量，集群管理器会按照 Spark 安装时指定的默认容量进行分配。

下面，使用代码清单 1-2 中的 sample.csv 文件(可在本书的配套程序中找到)计算这个程序的简化版本：返回 old_column 值的算术平均值。假设 Spark 实例有 4 个 executor，每个 executor 在各自的工作节点上工作。数据处理过程大致分配给 4 个 executor，每个 executor 处理数据帧中的一小部分。

代码清单 1-2　sample.csv 文件的内容

```
less data/list_of_numbers/sample.csv

old_column
1
4
4
5
7
7
7
10
14
1
4
8
```

图 1-2 描述了 PySpark 处理小数据帧中 old_column 的平均值的一种方式。这里选择平均值是因为它不像求和或计数那样是简单可分配的，而是要把每个 worker 的中间值相加。在计算平均值的情况下，每个 worker 独立计算它所负责的值的总和及其计数——而不是所有数据！最后，它们将结果交给一个 worker(或者当中间结果真的很小时，直接交给 master)，它将把这些值聚合成一个单一数值，即平均值。

对于这样一个简单的例子，映射 PySpark 的思维过程是一个简单而有趣的练习。数据的规模会越来越大，程序变得越来越复杂，我们将无法轻松地将代码映射到 Spark 实例执行的精确物理步骤。第 11 章会介绍 Spark 用来查看工作执行情况与工厂运行状况的机制。

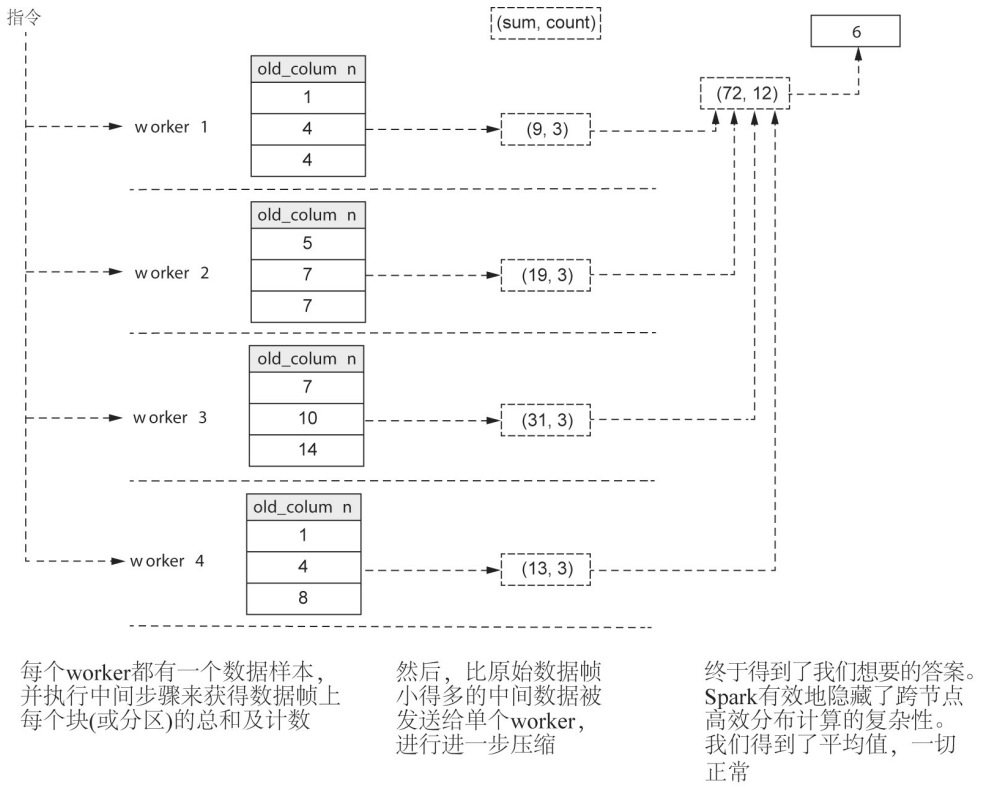

图 1-2　计算小数据帧的平均值。PySpark 风格：每个 worker 处理不同的数据块。必要时，会移动/混洗数据，以完成指令

本节举了一个简单的例子——计算一组数字组成的数据帧的平均值。为了得到正确的结果，我们绘制了 Spark 执行物理步骤的蓝图。1.2.2 节将介绍 Spark 最好但也最容易被误解的特性之一：惰性求值(laziness)。在进行大数据分析时，努力工作会有回报，但聪明的工作更好！

一些语言约定：data frame(数据帧)与 DataFrame

因为本书主要讨论的是数据帧，所以我更倾向于使用非大写的命名法(即 data frame)。我发现这比使用大写字母，甚至是没有空格的“dataframe”更具有可读性。

在直接引用 PySpark 对象时，将使用 DataFrame。这将有助于区分用于概念的“数据帧”和“DataFrame”对象。

1.2.2　懒惰的主管成就工厂的高效

本节介绍 Spark 最基本的特性之一：惰性计算能力。在讲授 PySpark 和对数据科学家的程序进行故障诊断时，我想说在 Spark 中惰性是最容易造成混淆的概念。这真是令人遗憾，因为惰性(在一定程度上)正是 Spark 获得惊人处理速度的原因。通过深入理解 Spark

如何实现惰性求值，将能够解释它的很多行为，并更好地调整性能。

就像在一个大型工厂里，你不会去每个员工那里给他们一个任务列表。不，在这里，master/manager 负责 worker。driver 是动作发生的地方。把 driver 想象成楼层主管：你给他们提供步骤列表，让他们处理。在 Spark 中，driver/楼层主管会接受你的指令(用 Python 代码仔细编写)，将它们转换为 Spark 步骤，然后在 worker 上处理。driver 还管理哪个 worker/table 拥有哪个数据切片，并确保你在这个过程中不会丢失数据。executor/factory worker 位于 worker /table 之上，在数据上执行具体的工作。

总结一下：

- master 就像工厂的主人一样，根据需要分配资源，以完成工作。
- driver 负责完成给定的工作。它根据需要向 master 请求资源。
- worker 是一组计算/内存资源，就像工厂中的工作台。
- executor 在 worker 上面，完成 driver 交代的工作，就像工作台上的员工一样。

第 11 章将回顾实践中使用的术语。

以代码清单 1-1 为例，将每条指令逐个打断，PySpark 直到 write 指令出现才会开始执行工作。如果使用常规的 Python 或 pandas 数据帧，它们不是惰性的(称为立即求值)，每条指令都是在读取时一个接一个执行的。

你的楼层主管/driver 具备一个好经理的所有品质：聪明、谨慎、懒惰。等等，什么？你没看错。在编程环境中(或者在现实世界中)，惰性求值可能是一件很好的事情。Spark 中提供的每条指令都可以分为两类：转换操作(transformation)和执行操作(action)，如图 1-3 所示。执行操作是许多编程语言中的 I/O 操作。最典型的执行操作如下：

- 在屏幕上打印信息。
- 向硬盘或云端存储桶写入数据。
- 计算记录的数量。

在 Spark 中，最常通过数据帧上的 show()、write()和 count()方法看到这些指令。

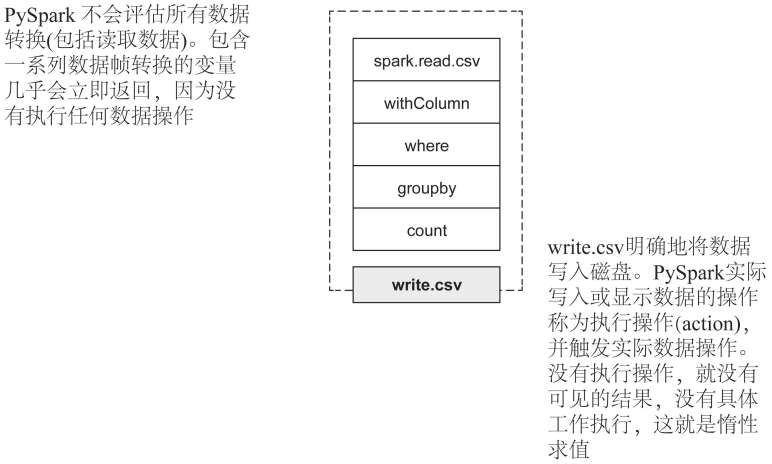

图 1-3　将数据帧指令分解为一系列转换操作和一个执行操作。Spark 执行的每个“任务”都由零个或多个转换操作和一个执行操作组成

转换操作几乎就是剩下的一切。下面是一些转换操作的例子。

- 向表中添加列。
- 根据某些键执行聚合。
- 计算汇总统计信息。
- 训练机器学习模型。

你可能会问，为什么要进行区分？当考虑对数据进行的计算时，作为开发人员，你只关心导致执行操作的计算。你将始终与执行操作的结果交互，因为这是你可以看到的东西。Spark 的惰性计算模型将其发挥到极致，避免执行数据工作，直到某个执行操作触发计算链。在此之前，driver 会存储指令。在处理大规模数据时，这种处理计算的方式有很多好处。

注意

正如将在第 5 章中看到的，当 count()作为聚合函数应用时，它是一个转换函数(计算每组记录的数量)。但应用于数据帧时，它是一个执行操作(计算数据帧中的记录数量)。

第一，在内存中存储指令比存储中间数据结果占用的空间要小得多。如果你在一个数据集上执行了很多操作，并且每一步都将数据物化，那么虽然不需要中间结果，但占用的存储空间会大大增加。我们都认为应该避免空间的浪费。

第二，通过获取待执行任务的完整列表，driver 可以更高效地优化 executor 之间的工作。它可以使用运行时可用的信息，如特定部分数据所在的节点。如果有必要，它还可以重新排序、消除无用的转换、合并多个操作，以及更有效地重写程序的某些部分。

第三，如果一个节点在处理过程中失败——计算机失败！——Spark 能够重新创建缺失的数据块，因为它缓存了指令。它会读取相关的数据块，并在适当的位置重新计算，而不需要你做任何操作。这样，你就可以专注于代码中数据处理的部分，而把灾难和恢复的部分交给 Spark，如图 1-4 所示。第 11 章会详细介绍计算资源和内存资源，以及如何监控故障。

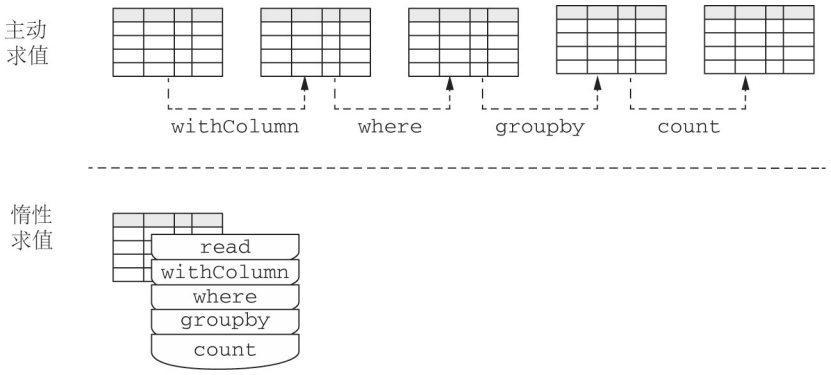

图 1-4 主动求值与惰性求值：存储(和动态计算)转换，通过减少对中间数据帧的需求来节省内存。如果其中一个节点发生故障，重新创建数据帧也更容易

最后，在交互式开发过程中，不必提交一大堆命令并等待计算发生。相反，你可以

迭代地构建转换链，一次一个。当准备好启动计算时，添加一个执行操作，让 Spark 发挥它的魔力。

惰性计算是 Spark 操作模型的一个基本方面，也是 Spark 速度如此之快的部分原因。大多数编程语言，包括 Python、R 和 Java，都是主动求值的。这意味着它们一收到指令就进行处理。使用 PySpark，可以使用一种积极的语言——Python，以及一个惰性的框架——Spark。这看起来可能有点陌生和吓人，但你不必担心。最好的学习方法是实践，本书提供了惰性的明确例子。你很快就会成为“惰性”的专业人士！

需要记住的一点是，Spark 不会为后续的计算保留操作的结果(或中间数据帧)。如果提交两次相同的程序，PySpark 将处理两次数据。我们使用缓存改变这种行为，并优化代码中的某些热点(最明显的是在训练机器学习模型时)，第 11 章将介绍如何以及何时进行缓存(剧透：并不像你想象的那么频繁)。

注意

虽然读取数据是 I/O 操作，但 Spark 认为这是一种转换操作。在大多数情况下，读取数据不会为用户执行任何可见的工作。因此，只有在需要对数据执行某些操作时才会读取数据(写入、读取、推断模式。更多信息参见第 6 章)。

没有有能力的员工，经理还有什么用？一旦接收到任务及其执行操作，driver 就会开始为被 Spark 称为 executor 的进程分配数据。executor 是指为应用运行计算和存储数据的进程。这些 executor 位于工作节点(worker node)上，即实际的计算机上。在我们的工厂类比中，executor 是执行工作的员工，而工作节点是工作台(table)，许多员工/执行器可以在其上工作。

我们的工厂之旅到此结束。总结一下典型的 PySpark 程序。

- 首先在 Python 代码中编写指令，形成一个 driver 程序。
- 在提交程序(或启动 PySpark shell)时，集群管理器会为我们分配资源。在程序运行期间，这些参数基本保持不变(自动扩展除外)。
- driver 程序采集你的代码并将其翻译成 Spark 指令。这些指令要么是转换操作，要么是执行操作。
- 一旦 driver 触发某个执行操作，它就会优化整个计算链，并在 executor 之间分配工作。executor 是执行实际数据工作的进程，它们位于被称为工作节点的计算机上。

就是这样！正如我们所看到的，整个过程非常简单，但很明显，Spark 隐藏了高效分布式处理带来的许多复杂性。对于开发人员来说，这意味着更短、更清晰的代码，以及更短的开发周期。

1.3　你将从本书学到什么

本书将使用 PySpark 解决数据分析师、数据工程师或数据科学家在日常生活中遇到的各种任务。因此将：

- 对于各种来源和格式，进行数据的读取和写入。
- 使用 PySpark 的数据操作功能处理混乱的数据。
- 发现新的数据集并执行探索性数据分析。
- 构建数据管道，以自动化的方式转换、总结和洞察数据。
- 排除常见的 PySpark 错误，以及从错误中恢复，并主动避免错误的发生。

在介绍了这些基础知识之后，还将处理不同的任务。这些任务也许并不常见，但它们很有趣，可以很好地展示 PySpark 的强大功能和多功能性。

- 建立机器学习模型，从简单的一次性实验到健壮的机器学习管道。
- 使用多种数据格式，从文本到表格再到 JSON。
- 无缝地融合 Python、pandas 和 PySpark 代码，利用它们各自的优势，最重要的是将 pandas 代码扩展到新的领域。

我们试图满足更多潜在读者，但主要关注那些与 Spark 和/或 PySpark 接触很少或没有接触过的人。更有经验的从业者可能会找到一些有用的知识，当他们需要解释困难的概念时，也许会从本书中受益！

1.4 我们将如何开始

本书主要介绍最新的 Spark 3.2 版本。数据帧是在 Spark 1.3 中出现的，所以有些代码可以在较早的 Spark 版本中使用。为了避免阅读本书时遇到麻烦，建议你使用 Spark 3.0 或更高版本，或者选择最新的版本。

我们假设你已经有一些基本的 Python 知识，附录 C 概述了一些有用的概念。如果你想更深入地了解 Python，我推荐 Naomi Ceder 撰写的 *Quick Python Book* (Manning，2018)，或 Reuven M. Lerner 撰写的 *Python Workout* (Manning，2020)。

要开始使用 Spark，只需要安装好 Spark。Spark 可以安装在计算机上，也可以安装在云端主机上(见附录 B)。本书中的大多数示例都可使用本地安装的 Spark，但也有一些可能需要更大的算力。

代码编辑器对于编写、阅读与编辑示例代码和编写程序也非常有用。有一个支持 Python 的编辑器很好，如 PyCharm、VS Code，甚至 Emacs/Vim，但并不是必要的。所有的示例也都适用于 Jupyter。请阅读附录 B 设置你的 Notebook 环境。

最后，我建议将你的想法用笔和纸记录下来。我就经常这样做，即使我在纸上记录的内容非常基本和粗糙，但我发现将想法在纸上写出来，有助于澄清这些想法。这意味着重写更少的代码，让程序员更快乐！不需要使用华丽的东西：一些废纸和一支铅笔就能创造奇迹。

1.5　本章小结

- PySpark 是用于大规模数据分析的分布式框架 Spark 的 Python API。Spark 继承了 Python 编程语言的表现力和动态性。
- Spark 之所以快，是因为它明智地使用了可用的内存，以及主动和惰性的查询优化器。
- 可在 Python、Scala、Java、R 等语言中使用 Spark，也可使用 SQL 进行数据操作。
- Spark 使用 driver 程序来处理指令并编排工作。executor 从 master 那里接受指令并执行工作。
- PySpark 中的所有指令要么是转换操作，要么是执行操作。因为 Spark 是惰性的，所以只有执行操作才会触发指令链的计算。