



Mahout in Action

Mahout 实战

[美] Sean Owen Robin Anil
Ted Dunning Ellen Friedman
王斌 韩冀中 万吉 译

Apache基金会官方推荐
Mahout核心团队权威力作
大数据时代机器学习的实战经典



人民邮电出版社
POSTS & TELECOM PRESS



Mahout in Action

Mahout 实战

“全面介绍Mahout机器学习实战的佳作。”

——Isabel Drost, Apache Mahout创始人

“深入浅出，复杂概念都讲解得透彻明白。”

——Rick Wagner, Red Hat

“出自核心开发团队之手，学习Mahout必读。”

——Philipp K. Janert, *Gnuplot in Action* 作者

通过收集数据来学习和演进的计算机系统威力无穷。Mahout作为Apache的开源机器学习项目，把推荐系统、分类和聚类等领域的核心算法浓缩到了可扩展的现成的库中。使用Mahout，你可以立即在自己的项目中应用亚马逊、Netflix及其他互联网公司所采用的机器学习技术。

本书出自Mahout核心成员之手，得到Apache官方推荐，权威性毋庸置疑。作者凭借多年实战经验，为读者展现了丰富的应用案例，并细致地介绍了Mahout的解决之道。本书还重点讨论了可扩展性问题，介绍了如何利用Apache Hadoop框架应对大数据的挑战。

本书内容：

- 利用分组数据实现个性化推荐；
- 寻找数据中的逻辑簇；
- 通过即时分类实现过滤与调优。



 MANNING

图灵社区：iTuring.cn

热线：(010)51095186转600

分类建议 计算机/程序设计/Mahout

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-34722-0



ISBN 978-7-115-34722-0

定价：79.00元

TURING

图灵程序设计丛书



Mahout in Action

Mahout 实战

[美] Sean Owen Robin Anil
Ted Dunning Ellen Friedman

王斌 韩冀中 万吉 译

著

人民邮电出版社
北京

图书在版编目 (C I P) 数据

Mahout 实战 / (美) 欧文 (Owen, S.) 等著 ; 王斌,
韩冀中, 万吉译. -- 北京 : 人民邮电出版社, 2014. 3

(图灵程序设计丛书)

书名原文: Mahout in action

ISBN 978-7-115-34722-0

I. ①M… II. ①欧… ②王… ③韩… ④万… III. ①
机器学习②电子计算机—算法理论 IV. ①TP181
②TP301. 6

中国版本图书馆CIP数据核字(2014)第031399号

内 容 提 要

本书是 Mahout 领域的权威著作，出自该项目核心成员之手，立足实践，全面介绍了基于 Apache Mahout 的机器学习技术。本书开篇从 Mahout 的故事讲起，接着分三部分探讨了推荐系统、聚类和分类，最后的附录涵盖 JVM 调优、Mahout 数学知识和相关资源。

本书适合所有数据分析和数据挖掘人员阅读，需要有 Java 语言基础。

◆ 著 [美] Sean Owen Robin Anil Ted Dunning
Ellen Friedman
译 王 斌 韩冀中 万 吉
责任编辑 毛倩倩
执行编辑 刘 帅
责任印制 焦志炜
◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
三河市海波印务有限公司印刷
◆ 开本：800×1000 1/16
印张：21.25
字数：502千字 2014年3月第1版
印数：1-4 000册 2014年3月河北第1次印刷
著作权合同登记号 图字：01-2011-7805号

定价：79.00元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

前　　言

追溯这本书的由来，就我个人而言，要从2005年说起。我的一个朋友当时正在创办一家公司，急需协同过滤技术。虽然当时可以找到成熟、开源的软件包，但是它们要么太过繁复，要么太学术化。所以，我决定从零开始，为这个朋友的创业公司开发了一个推荐系统的简单原型。遗憾的是，这家创业公司夭折了。然而，我却无法说服自己删除这个原型。它实在有趣，于是我对它进行整理并写了文档，用Taste这个名字将它发布为一个开源项目。

一年无声地过去了。我在业余时间为其增加了一些代码，并修复了一些问题。接着，有一两个用户出现，并提交了一些软件bug和补丁，然后又有了几个用户，再后来又增加了好一些。到了2008年，虽然小但却稳定的用户群形成了。后来，Apache Lucene的人把机器学习相关的部分剥离出来形成了Apache Mahout，他们建议把我们的两个项目进行合并。此后，本书在2009年晚些时候开始立项。而今，当看到这个项目滚雪球般地发展到2011年，并开始被大公司在生产系统中使用，我自己既惊讶又欣喜。

的确，我只是无心插柳。即便我已经是一个高级工程师，曾在谷歌工作过，也没有人会误认为我是这个领域的专家。我更像是一个博物馆的管理者，而不是一个画家，我将一个领域的伟大思想进行搜集、组织和打包，使之广为所用。这同样不失为一项有用的工作。

一些人在读过本书的初稿之后，说它是一本“通俗易懂”的机器学习书。这是一种盛誉，而我完全赞同。机器学习有其魔力所在，不过这个领域中有许多研究性的著作对于非专业人员而言就像天书，它们也与该技术的应用实践相去甚远。而本书旨在让读者易于理解，为爱好者揭示领悟的快乐，并为实践者节省工作时间。我希望你阅读本书时的惊喜比疑问多。

——Sean Owen

我对机器学习的兴趣可以回溯到2006年上大学的那段日子。那时，我作为实习生和一组人共同设计一个个性化的推荐引擎。这个小组后来成长为Minekey公司；我也被邀请加入，成为其核心开发人员。后来的四年，我一直从事机器学习技术的实现与试验。在此期间，我偶然间发现了Mahout，并开始作为一个Google Summer of Code的参赛学生加入这个项目。我记得，接下来的事情就是不断为它的代码库贡献算法和补丁，做性能调优，以及帮助邮件列表中的其他人。

由机器学习开发者、研究者和爱好者组成的社区非常出色，正在不断成长，而我有幸成为这个团队的一员。随着越来越多的公司采用Mahout，它正在成为机器学习的主流软件库。我衷心希望你在阅读本书时能够乐在其中。

——Robin Anil

我（Ted）在机器学习上是先做研究后做项目。我早期从事的是学术工作，后来参与了一些创业团队，从而得以将机器学习在实际中应用。

我（Ellen）以前在生物化学和分子生物学实验室工作。在研究大量数据的同时，我还写了许多技术文章。此番经历，让我痴迷于数据及其蕴含的意义。我努力把这种内在的东西写进本书里。

我们两个人一致认为开源有赖于一个有大量活跃用户参与的社区。Mahout的成功主要来自于那些使用这个软件的人，他们通过在邮件列表中展开讨论、修复bug，以及提供建议，把使用经验回馈到这个项目中。

为此，本书不仅给出代码的实用注解，而且引出了代码背后的一些概念。介绍隐藏于代码背后的框架，会使你更有效地加入到Mahout的讨论中，并从中获益。我们希望本书不仅能够帮助读者，而且能够使Mahout自身得到完善和发展。

——Ted Dunning和Ellen Friedman

致谢

本书的出版离不开众人的努力。作者对他们致以衷心的感谢，但限于篇幅，致谢名单只列出了其中一部分人，排名不分先后。

- 在机器学习领域发表核心文章的研究者，详见附录C。
- 花时间测试试用版软件的Mahout用户，他们寻找与解决bug，为软件打补丁乃至提出建议。
- Mahout提交者，他们致力于Mahout的发展、完善和提升。
- Manning出版社投入了大量时间和精力将本书出版并投入市场。特别感谢Katharine Osborne、Karen Tegtmeyer、Jeff Bleiel、Andy Carroll、Melody Dolab和Dottie Marsico，你看到的最终稿与他们的工作密不可分。
- 在本书写作过程中提供了宝贵反馈的审校者：Philipp K. Janert、Andrew Oswald、John Griffin、Justin Tyler Wiley、Deepak Vohra、Grant Ingersoll、Isabel Drost、Kenneth DeLong、Eric Raymond、David Grossman、Tom Morton，以及Rick Wagner。
- Alex Ott在本书印刷前一刻对全稿进行了细致的技术审核。
- 在作者在线论坛上发帖评价本书的MEAP读者^①。
- 每个在Mahout邮件列表中提问的人。
- 在本书长时间写作过程之中，给予我们无尽支持的家人和朋友！

^① MEAP，全称Manning Early Access Program。因为图书出版周期较长，为让读者可以时刻了解热门技术，Manning出版社推出了这一图书抢鲜阅读项目。参与其中的读者可以在图书未编辑完成之际，一章一章地阅读。——编者注

关于本书

你可能还有疑问：这本书是否适合我？

如果你正在寻找一本机器学习教材，答案是否定的。本书不会对诸多算法和技术的理论以及推导过程给出全面解释。如果你了解机器学习技术，并熟悉矩阵和向量等相关的数学概念，这有助于阅读本书，但并非必要条件。

如果你正在开发先进、智能的应用，答案则是肯定的。本书从实践而非理论入手来诠释这些技术，并给出完整的例子与解决方案。它在教授用Mahout解决问题的同时，带给你实践者的经验与领悟。

如果你是人工智能、机器学习及相关领域的研究人员，答案亦是肯定的。你所面对的最大障碍，很可能就是把新的算法应用到实践中。对于新的大规模算法的测试与部署，Mahout提供了一个成熟的框架、一系列模式以及现成的组件。本书是你在复杂的分布式计算框架上学习开发机器学习系统的“快车票”。

如果你正领导一个产品小组或创业团队，想利用机器学习创造一种竞争优势，那么本书同样适合你。通过实际示例，它让你了解这些技术的多种应用方式。它还会让小型技术团队变得高效，能够处理大量数据，而这在以前只有具有大量技术资源的组织机构才做得到。

路线图

本书分为三部分，分别介绍了Apache Mahout中的协同过滤、聚类和分类。

首先，第1章整体介绍Apache Mahout。这一章为你阅读后续各章奠定基础。

第一部分（第2章~第6章）由Sean Owen编写，主要介绍协同过滤与推荐。第2章基于Mahout构造推荐引擎并评估其性能。第3章探讨如何高效呈现推荐引擎使用的数据。第4章介绍可在Mahout中利用的所有推荐算法，并比较它们的优缺点。在此背景之下，第5章给出一个案例，将第4章介绍的推荐系统实现应用在该案例的真实问题中，配以某些特定属性的数据，从而建立一个可为生产环境所用的推荐引擎。第6章介绍Apache Hadoop，通过研究基于Hadoop的推荐引擎，首次为你展现分布式环境中的机器学习算法。

第二部分（第7章~第12章）探索Apache Mahout上的聚类算法。通过Robin Anil所做的技术说明，你可以把看起来类似的数据片段组织为一个个集合或者说簇（cluster）。聚类有助于揭示大规模数据中有趣的信息组合。这部分从聚类中的简单问题开始介绍，并给出了Java示例。接下来，

作者引入更多实际示例，并展示如何让Apache Mahout以Hadoop作业的方式运行，从而轻而易举地聚类大量数据。

第三部分（第13章~第17章）由Ted Dunning和Ellen Friedman编写，探索如何用Mahout进行分类。首先，作者带你了解如何通过一组示例“教会”一个算法，从而建立和训练分类器模型。接下来，你会了解如何评估并微调分类器模型以得到更好的结果。这部分最后以一个分类实战案例结束。

代码约定及下载

本书源代码均采用等宽字体印刷，列为代码清单，并对重点进行注释。代码清单旨在简单明了，重点突出。它们通常不给出Java导入包、类声明、Java注释，以及其他对代码的讨论无关紧要的东西。

本书中的类名亦采用等宽字体，放于文本之间，以显示它们是可以在Apache Mahout源代码中找到并研究的类名。例如，`LogLikelihoodSimilarity`是Mahout中的一个Java类。

一些代码清单中列出了可执行命令。它们是为Mac OS X和Linux发行版等类Unix环境而写的。如果使用了类Unix的Cygwin环境，它们也可以在微软Windows系统下运行。

本书关键代码清单中的源代码均可编译，且均可从www.manning.com/MahoutinAction下载^①。这些都是独立的Java源文件，并不包括编译脚本。为了方便起见，你可以把它们解压到Mahout源代码发布包的examples/src/java/main目录下。这样，Mahout的编译环境将会自动编译这些代码。

多媒体资料

四位作者均录制了音频和视频片段，与书中多数章中的特定节互相补充，为相应话题提供了附加信息。你可以从本书英文版电子书中看到或听到这些音视频片段，该电子书对英文纸质书的拥有者免费；你还可以从www.manning.com/MahoutinAction/extras免费获取。通过书中的音频和视频图标，你可以获知其所涉及的话题，以及发言者是谁。这些多媒体资料的清单详见“关于多媒体资料”。

作者在线

本书英文版读者可以免费访问Manning出版社专门维护的一个论坛，并可以发表评论、提出技术问题，并获得作者和其他论坛用户的帮助。你可以通过网页www.manning.com/MahoutinAction进入和订阅该论坛。完成注册后，你可以了解如何使用该论坛、该论坛所能提供的帮助，以及论坛的行为规范。

Manning出版社承诺为读者和作者提供一个进行深入对话的场所，但不对作者的参与程度做要求，他们对于该论坛的贡献是出于自愿且无报酬的。我们建议读者尽量向作者提一些具有挑战性的问题，让他们保持兴趣！

本书在印期间，读者均可访问作者在线论坛，并查看之前的讨论。

^① 亦可在图灵社区（iTuring.cn）本书页面免费注册下载。——编者注

关于多媒体资料

本书附带的多媒体资料可以在www.manning.com/MahoutinAction/extras/上免费收听或收看。本书中空白处的音频或视频文件图标（如下所示），指出了书中哪些地方可参考附加资料。



Audio icon



Video icon

No. 1 音频 p2

Sean介绍了Mahout项目以及他参与的事项。

No. 2 音频 p19

Sean讨论了推荐系统的工作。

No. 3 音频 p29

Sean阐述为什么他认为人们有可能过度“聆听”数据。

No. 4 音频 p42

Sean谈论皮尔逊相关系数的实现。

No. 5 音频 p63

Sean讨论了诠释性能指标的价值。

No. 6 音频 p84

Sean解释了Mahout和Hadoop之间的关系。

No. 7 音频 p108

Robin解释了如何为一个数据集选择正确的距离测度方法。

No. 8 音频 p114

Robin扩展了苹果的类比示例。

No. 9 音频 p127

Robin解释了k-means聚类迭代过程。

No. 10 音频 p165

Robin讨论改善聚类质量的策略。

No. 11 音频 p179

Robin解释了如何改进大规模聚类的性能。

2 关于多媒体资料

No. 12 视频 p208

Ellen展示了如何训练一个模型使之逐步优化。

No. 13 视频 p234

Ted和Ellen展示了Logistic回归的内部机制。

No. 14 视频 p238

Ted比较了使用串行算法与并行算法的优势。

No. 15 音频 p249

Ted和Ellen讨论了AUC评估方法。

No. 16 音频 p252

Ted和Ellen讨论了为什么对数似然法意味着“永不说不”。

关于封面

封面上是“一个来自Rakov-Potok的男人”。Rakov-Potok是克罗地亚北方的一个村庄。该图取自克罗地亚19世纪中叶传统服饰影集的一个副本，作者为Nikola Arsenovic，由Ethnographic博物馆在2003年出版于克罗地亚的斯普利特。该图得自于乐于助人的Ethnographic博物馆馆员，这个博物馆位于该城镇在中世纪罗马时的核心位置，是公元304年左右罗马皇帝戴克里先的宫殿遗址。这本书包含来自克罗地亚不同地域的颜色精美的插图，附有服饰和日常生活的说明。

Rakov-Potok是一个风景如画的乡村，位于Samobor山脚下、萨瓦河土地肥沃的河谷中，距Zagreb城不远。它有着悠久的历史，在那里，你会与许多城堡、教堂和中世纪甚至罗马时期的遗迹不期而遇。封面上的人物身着白色羊毛长裤和白色羊毛外套，上面有着大量的红色和蓝色绣花——这是该地区山区居民的典型装束。

过去200年间，人们的着装和生活方式已经发生变化，曾经如此丰富的地域多样性已渐渐消失了。现在，各大洲的居民已经很难分辨，更遑论不同小村或距离只有几英里的人。也许我们用文化多样性换来的是更多样化的个人生活——必然是更为丰富和快节奏的技术生活。

Manning出版社在此类古老书籍的插图中取材，基于两个世纪前丰富多样的地域生活来制作图书封面，借此颂扬计算机行业的创造力和首创精神。

目 录

第 1 章 初识 Mahout	1
1.1 Mahout 的故事.....	1
1.2 Mahout 的机器学习主题	2
1.2.1 推荐引擎	2
1.2.2 聚类	3
1.2.3 分类	4
1.3 利用 Mahout 和 Hadoop 处理大规模 数据.....	4
1.4 安装 Mahout	6
1.4.1 Java 和 IDE	6
1.4.2 安装 Maven	7
1.4.3 安装 Mahout.....	7
1.4.4 安装 Hadoop.....	8
1.5 小结	8
 第一部分 推荐	
第 2 章 推荐系统	10
2.1 推荐的定义	10
2.2 运行第一个推荐引擎.....	11
2.2.1 创建输入	11
2.2.2 创建一个推荐程序	13
2.2.3 分析输出	14
2.3 评估一个推荐程序	14
2.3.1 训练数据与评分.....	15
2.3.2 运行 RecommenderEvaluator.....	15
2.3.3 评估结果	16
2.4 评估查准率与查全率.....	17
2.4.1 运行 RecommenderIRStats- Evaluator	17
2.4.2 查准率和查全率的问题	19
第 3 章 推荐数据的表示	21
3.1 偏好数据的表示	21
3.1.1 Preference 对象	21
3.1.2 PreferenceArray 及其实现	22
3.1.3 改善聚合的性能	23
3.1.4 FastByIDMap 和 FastIDSet	23
3.2 内存级 DataModel	24
3.2.1 GenericDataModel	24
3.2.2 基于文件的数据	25
3.2.3 可刷新组件	25
3.2.4 更新文件	26
3.2.5 基于数据库的数据	26
3.2.6 JDBC 和 MySQL	27
3.2.7 通过 JNDI 进行配置	27
3.2.8 利用程序进行配置	28
3.3 无偏好值的处理	29
3.3.1 何时忽略值	29
3.3.2 无偏好值时的内存级表示	30
3.3.3 选择兼容的实现	31
3.4 小结	33
第 4 章 进行推荐	34
4.1 理解基于用户的推荐	34
4.1.1 推荐何时会出错	34
4.1.2 推荐何时是正确的	35
4.2 探索基于用户的推荐程序	36

2 目录

4.2.1 算法	36	5.2.1 基于用户的推荐程序.....	61
4.2.2 基于 GenericUserBased-Recommender 实现算法	36	5.2.2 基于物品的推荐程序.....	62
4.2.3 尝试 GroupLens 数据集	37	5.2.3 slope-one 推荐程序.....	63
4.2.4 探究用户邻域	38	5.2.4 评估查准率和查全率.....	63
4.2.5 固定大小的邻域	39	5.2.5 评估性能.....	64
4.2.6 基于阈值的邻域	39	5.3 引入特定域的信息	65
4.3 探索相似性度量	40	5.3.1 采用一个定制的物品相似性度量	65
4.3.1 基于皮尔逊相关系数的相似度	40	5.3.2 基于内容进行推荐	66
4.3.2 皮尔逊相关系数存在的问题	42	5.3.3 利用 IDRescorer 修改推荐结果	66
4.3.3 引入权重	42	5.3.4 在 IDRescorer 中引入性别	67
4.3.4 基于欧氏距离定义相似度	43	5.3.5 封装一个定制的推荐程序	69
4.3.5 采用余弦相似性度量	43	5.4 为匿名用户做推荐	71
4.3.6 采用斯皮尔曼相关系数基于相对排名定义相似度	44	5.4.1 利用 PlusAnonymousUser-DataModel 处理临时用户	71
4.3.7 忽略偏好值基于谷本系数计算相似度	45	5.4.2 聚合匿名用户	73
4.3.8 基于对数似然比更好地计算相似度	46	5.5 创建一个支持 Web 访问的推荐程序	73
4.3.9 推测偏好值	47	5.5.1 封装 WAR 文件	74
4.4 基于物品的推荐	47	5.5.2 测试部署	74
4.4.1 算法	48	5.6 更新和监控推荐程序	75
4.4.2 探究基于物品的推荐程序	49	5.7 小结	76
4.5 Slope-one 推荐算法	50	第 6 章 分布式推荐	78
4.5.1 算法	50	6.1 分析 Wikipedia 数据集	78
4.5.2 Slope-one 实践	51	6.1.1 挑战规模	79
4.5.3 DiffStorage 和内存考虑	52	6.1.2 分布式计算的优缺点	80
4.5.4 离线计算量的分配	53	6.2 设计一个基于物品的分布式推荐算法	81
4.6 最新以及试验性质的推荐算法	53	6.2.1 构建共现矩阵	81
4.6.1 基于奇异值分解的推荐算法	53	6.2.2 计算用户向量	82
4.6.2 基于线性插值物品的推荐算法	54	6.2.3 生成推荐结果	82
4.6.3 基于聚类的推荐算法	55	6.2.4 解读结果	83
4.7 对比其他推荐算法	56	6.2.5 分布式实现	83
4.7.1 为 Mahout 引入基于内容的技术	56	6.3 基于 MapReduce 实现分布式算法	83
4.7.2 深入理解基于内容的推荐算法	57	6.3.1 MapReduce 简介	84
4.8 对比基于模型的推荐算法	57	6.3.2 向 MapReduce 转换：生成用户向量	84
4.9 小结	57	6.3.3 向 MapReduce 转换：计算共现关系	85
第 5 章 让推荐程序实用化	59	6.3.4 向 MapReduce 转换：重新思考矩阵乘	87
5.1 分析来自约会网站的样本数据	59		
5.2 找到一个有效的推荐程序	61		

6.3.5 向 MapReduce 转换：通过部分乘积计算矩阵乘.....	87	8.3 从文档中生成向量	119
6.3.6 向 MapReduce 转换：形成推荐.....	90	8.4 基于归一化改善向量的质量	123
6.4 在 Hadoop 上运行 MapReduce	91	8.5 小结	124
6.4.1 安装 Hadoop.....	92	第 9 章 Mahout 中的聚类算法	125
6.4.2 在 Hadoop 上执行推荐	92	9.1 k-means 聚类	125
6.4.3 配置 mapper 和 reducer.....	94	9.1.1 关于 k-means 你需要了解的.....	126
6.5 伪分布式推荐程序	94	9.1.2 运行 k-means 聚类	127
6.6 深入理解推荐	95	9.1.3 通过 canopy 聚类寻找最佳 k 值.....	134
6.6.1 在云上运行程序.....	95	9.1.4 案例学习：使用 k-means 对新闻聚类	138
6.6.2 考虑推荐的非传统用法	97	9.2 超越 k-means：聚类技术概览	141
6.7 小结	97	9.2.1 不同类型的聚类问题	141
第二部分 聚类			
第 7 章 聚类介绍	100	9.2.2 不同的聚类方法	143
7.1 聚类的基本概念	100	9.3 模糊 k-means 聚类	145
7.2 项目相似性度量	102	9.3.1 运行模糊 k-means 聚类	145
7.3 Hello World：运行一个简单的聚类示例.....	103	9.3.2 多模糊会过度吗	147
7.3.1 生成输入数据	103	9.3.3 案例学习：用模糊 k-means 对新闻进行聚类	148
7.3.2 使用 Mahout 聚类	104	9.4 基于模型的聚类	149
7.3.3 分析输出结果	107	9.4.1 k-means 的不足	149
7.4 探究距离测度	108	9.4.2 狄利克雷聚类	150
7.4.1 欧氏距离测度	108	9.4.3 基于模型的聚类示例	151
7.4.2 平方欧氏距离测度	108	9.5 用 LDA 进行话题建模	154
7.4.3 曼哈顿距离测度.....	108	9.5.1 理解 LDA	155
7.4.4 余弦距离测度	109	9.5.2 对比 TF-IDF 与 LDA	156
7.4.5 谷本距离测度	110	9.5.3 LDA 参数调优	156
7.4.6 加权距离测度	110	9.5.4 案例学习：寻找新闻文档中的话题	156
7.5 在简单示例上使用各种距离测度	111	9.5.5 话题模型的应用	158
7.6 小结	111	9.6 小结	158
第 8 章 聚类数据的表示	112	第 10 章 评估并改善聚类质量	160
8.1 向量可视化	113	10.1 检查聚类输出	160
8.1.1 将数据转换为向量	113	10.2 分析聚类输出	162
8.1.2 准备 Mahout 所用的向量	115	10.2.1 距离测度与特征选择	163
8.2 将文本文档表示为向量	116	10.2.2 簇间与簇内距离	163
8.2.1 使用 TF-IDF 改进加权.....	117	10.2.3 簇的混合与重叠	166
8.2.2 通过 n-gram 搭配词考察单词的依赖性.....	118	10.3 改善聚类质量	166
		10.3.1 改进文档向量生成过程	166

10.3.2 编写自定义距离测度	169	13.2.2 分类的应用	201
10.4 小结	171	13.3 分类的工作原理	202
第 11 章 将聚类用于生产环境	172	13.3.1 模型	203
11.1 Hadoop 下运行聚类算法的快速入门	172	13.3.2 训练、测试与生产	203
11.1.1 在本地 Hadoop 集群上运行聚类算法	173	13.3.3 预测变量与目标变量	204
11.1.2 定制 Hadoop 配置	174	13.3.4 记录、字段和值	205
11.2 聚类性能调优	176	13.3.5 预测变量值的 4 种类型	205
11.2.1 在计算密集型操作中避免性能缺陷	176	13.3.6 有监督学习与无监督学习	207
11.2.2 在 I/O 密集型操作中避免性能缺陷	178	13.4 典型分类项目的工作流	207
11.3 批聚类及在线聚类	178	13.4.1 第一阶段工作流：训练分类模型	208
11.3.1 案例分析：在线新闻聚类	179	13.4.2 第二阶段工作流：评估分类模型	212
11.3.2 案例分析：对维基百科文章聚类	180	13.4.3 第三阶段工作流：在生产中使用模型	212
11.4 小结	181	13.5 循序渐进的简单分类示例	213
第 12 章 聚类的实际应用	182	13.5.1 数据和挑战	213
12.1 发现 Twitter 上的相似用户	182	13.5.2 训练一个模型来寻找颜色填充：初步设想	214
12.1.1 数据预处理及特征加权	183	13.5.3 选择一个学习算法来训练模型	215
12.1.2 避免特征选择中的常见陷阱	184	13.5.4 改进填充颜色分类器的性能	217
12.2 为 Last.fm 上的艺术家推荐标签	187	13.6 小结	221
12.2.1 利用共现信息进行标签推荐	187	第 14 章 训练分类器	222
12.2.2 构建 Last.fm 艺术家词典	188	14.1 提取特征以构建分类器	222
12.2.3 将 Last.fm 标签转换成以艺术家为特征的向量	190	14.2 原始数据的预处理	224
12.2.4 在 Last.fm 数据上运行 k-means 算法	191	14.2.1 原始数据的转换	224
12.3 分析 Stack Overflow 数据集	193	14.2.2 一个计算营销的例子	225
12.3.1 解析 Stack Overflow 数据集	193	14.3 将可分类数据转换为向量	226
12.3.2 在 Stack Overflow 中发现聚类问题	193	14.3.1 用向量表示数据	226
12.4 小结	194	14.3.2 用 Mahout API 做特征散列	228
第三部分 分类		14.4 用 SGD 对 20 Newsgroups 数据集进行分类	231
第 13 章 分类	198	14.4.1 开始：数据集预览	231
13.1 为什么用 Mahout 做分类	198	14.4.2 20 Newsgroups 数据特征的解析和词条化	234
13.2 分类系统基础	199	14.4.3 20 Newsgroups 数据的训练代码	234
13.2.1 分类、推荐和聚类的区别	201		

14.5 选择训练分类器的算法	238	16.2 确定规模和速度需求	270
14.5.1 非并行但仍很强大的算法： SGD 和 SVM	239	16.2.1 多大才算大	270
14.5.2 朴素分类器的力量：朴素贝 叶斯及补充朴素贝叶斯	239	16.2.2 在规模和速度之间折中	272
14.5.3 精密结构的力量：随机森林 算法	240	16.3 对大型系统构建训练流水线	273
14.6 用朴素贝叶斯对 20 Newsgroups 数据 分类	241	16.3.1 获取并保留大规模数据	274
14.6.1 开始：为朴素贝叶斯提取 数据	241	16.3.2 非规范化及下采样	275
14.6.2 训练朴素贝叶斯分类器	242	16.3.3 训练中的陷阱	276
14.6.3 测试朴素贝叶斯模型	242	16.3.4 快速读取数据并对其进行 编码	278
14.7 小结	244	16.4 集成 Mahout 分类器	282
第 15 章 分类器评估及调优	245	16.4.1 提前计划：集成中的关键 问题	283
15.1 Mahout 中的分类器评估	245	16.4.2 模型序列化	287
15.1.1 获取即时反馈	246	16.5 案例：一个基于 Thrift 的分类服 务器	288
15.1.2 确定分类“好”的含义	246	16.5.1 运行分类服务器	292
15.1.3 认识不同的错误代价	247	16.5.2 访问分类器服务	294
15.2 分类器评估 API	247	16.6 小结	296
15.2.1 计算 AUC	248	第 17 章 案例分析——Shop It To Me	297
15.2.2 计算混淆矩阵和熵矩阵	250	17.1 Shop It To Me 选择 Mahout 的原因	297
15.2.3 计算平均对数似然	252	17.1.1 Shop It To Me 公司简介	298
15.2.4 模型剖析	253	17.1.2 Shop It To Me 需要分类系 统的原因	298
15.2.5 20 Newsgroups 语料上 SGD 分类器的性能指标计算	254	17.1.3 对 Mahout 向外扩展	298
15.3 分类器性能下降时的处理	257	17.2 邮件交易系统的一般结构	299
15.3.1 目标泄漏	258	17.3 训练模型	301
15.3.2 特征提取崩溃	260	17.3.1 定义分类项目的目标	301
15.4 分类器性能调优	262	17.3.2 按时间划分	303
15.4.1 问题调整	262	17.3.3 避免目标泄漏	303
15.4.2 分类器调优	265	17.3.4 调整学习算法	303
15.5 小结	267	17.3.5 特征向量编码	304
第 16 章 分类器部署	268	17.4 加速分类过程	306
16.1 巨型分类系统的部署过程	268	17.4.1 特征向量的线性组合	307
16.1.1 理解问题	269	17.4.2 模型得分的线性扩展	308
16.1.2 根据需要优化特征提取过程	269	17.5 小结	310
16.1.3 根据需要优化向量编码	269	附录 A JVM 调优	311
16.1.4 部署可扩展的分类器服务	270	附录 B Mahout 数学基础	313
		附录 C 相关资源	318
		索引	320

第1章

初识Mahout



本章内容

- Apache Mahout是什么？从哪里来？
- 现实中的推荐引擎、聚类和分类一览
- Mahout安装

大概你已经从书名中猜到了，本书主要讲解一个特殊的工具——Apache Mahout——在现实生活中的高效应用。它具备三个明显的特征。

首先，Mahout是一个来自Apache的、开源的机器学习（machine learning）软件库。它所实现的算法归属于机器学习或集体智慧（collective intelligence，也常常译为群体智慧或群体智能）这个广阔的领域。这意味着有许多事情可做，但对于此时此刻的Mahout，它主要关注于推荐引擎（协同过滤）、聚类和分类。

其次，Mahout是可扩展的。它旨在当所处理的数据规模远大于单机处理能力时成为一种可选的机器学习工具。在当前的Mahout系统中，这些可扩展的机器学习实现都是用Java来写的，而且有些部分是建立在Apache的Hadoop分布式计算项目之上的。

最后，它是一个Java软件库，并不提供用户接口、预装服务器（prepackaged server）或安装程序（installer）。它打算为开发者提供一个可用可改的工具框架。

出于阶段安排的需要，本章将通过一些常见的真实案例简要介绍一下推荐引擎、聚类和分类这几种机器学习手法，而Mahout通过它们帮助你处理数据。

若要在阅读本书时做到对Mahout随学随用，还要做一些必要的系统搭建与安装工作。

1.1 Mahout 的故事

首先来了解Mahout的背景知识。你可能还搞不清Mahout该如何发音：就是其通常的英语发音（[mə'haut]），它和trout押韵。它来自北印度语，意为驱象人，若要解释它，还有个小故事。

Mahout是2008年作为Apache Lucene的子项目出现的。Lucene项目推出了一个同名的著名开源搜索引擎，并给出了搜索、文本挖掘（text mining）和信息检索技术的先进实现方法。在计算机科学领域，这些术语和机器学习技术中的概念很相近，比如聚类（clustering），并在某种程度

上与分类(classification)相近。这样一来，某些Lucene贡献者的工作更多落入机器学习领域，从而逐渐脱离出来形成了独立的子项目。之后不久，Mahout吸纳了开源的协同过滤项目Taste。

No. 1 图1-1给出了Mahout在ASF(Apache Software Foundation, Apache软件基金会)中的部分传承关系。到2010年4月，Mahout已经成为了一个独立的顶级Apache项目，并发布了一个全新的驱象人徽标。

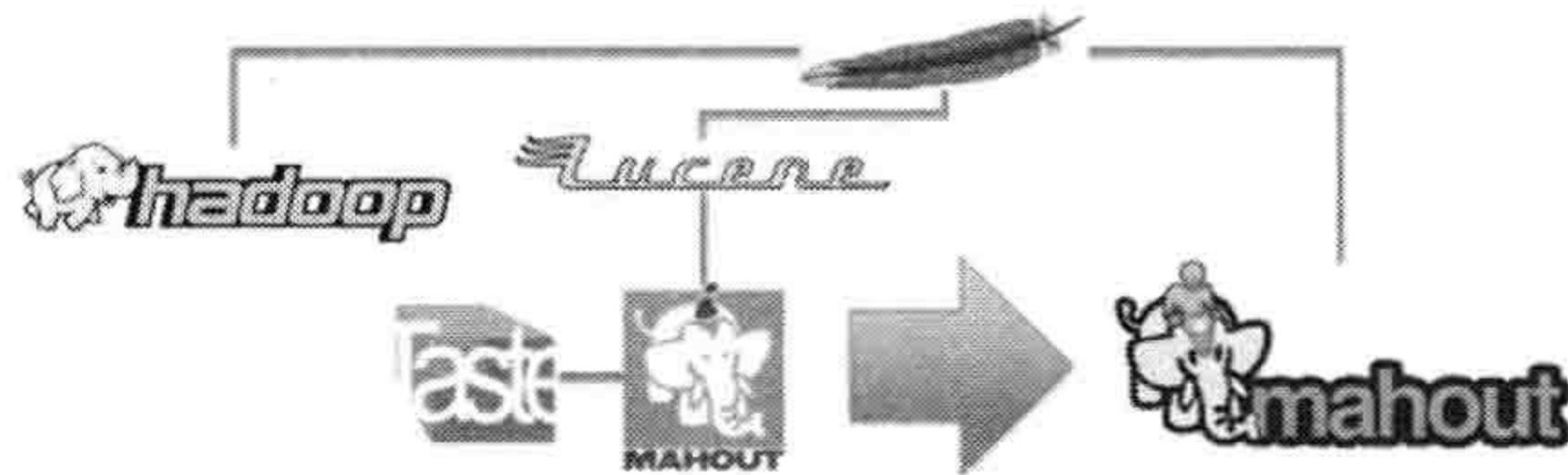


图1-1 Apache Mahout及其在ASF中的相关项目

Mahout所做的大量工作不仅体现在以高效和可扩展的方式实现这些经典算法，而且将部分算法进行转换使其可以在Hadoop上处理大规模的问题。Hadoop的吉祥物是一头象，Mahout项目的名字便由此而来！

从Mahout孵化出了许多技术和算法，其中有许多仍在开发或实验阶段(<https://cwiki.apache.org/confluence/display/MAHOUT/Algorithms>)。在该项目的早期，有3个明确的核心主题：推荐引擎(协同过滤)、聚类和分类。虽然它们绝非Mahout的全部，但在本书写作时，它们是最突出和最成熟的主题，也因此成为了本书的焦点。

也许你在阅读本书时已经了解了这三种技术的魅力，但为了不漏掉什么，请继续读下去。

1.2 Mahout的机器学习主题

虽然Mahout项目在理论上可以实现所有类型的机器学习技术，但实际上当前它仅关注机器学习的三个主要领域，即推荐引擎(协同过滤)、聚类和分类。

1.2.1 推荐引擎

在目前采用的机器学习技术中，推荐引擎是最容易一眼就被认出来的。服务商或网站会根据你过去的行为向你推荐书籍、电影或文章。它们会推测你的品味与爱好，并找到某些你可能感兴趣的物品。

- 在部署了推荐系统的电子商务网站中，亚马逊大概是最有名的。亚马逊基于交易行为和网站记录为你推荐你可能感兴趣的书籍和其他物品(见图1-2)。
- 与之类似，Netflix为用户推荐其可能感兴趣的DVD，为了鼓励研究者改善其推荐质量，它给出了一份1 000 000美元的奖金，这使它颇具盛名。
- 像Líbímseti这样的约会网站(稍后讨论)还能把一个人推荐给另一个人。

- 而Facebook这样的社交网络则利用推荐技术为你找到最可能尚未关联的朋友。

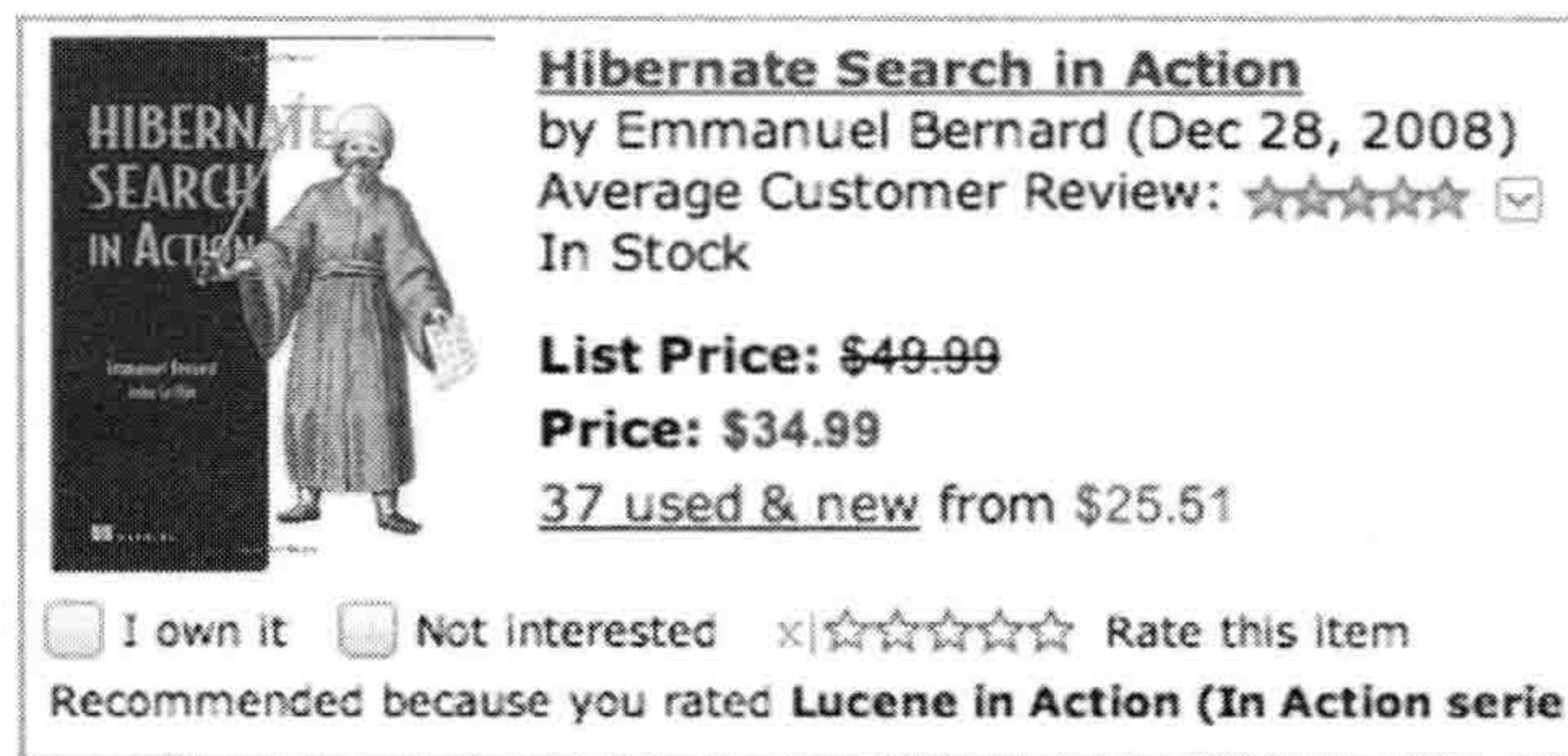


图1-2 亚马逊的推荐结果。基于该用户的交易历史和同类顾客的一些行为，亚马逊认为该用户会对这个推荐结果感兴趣。它甚至可以列出这次推荐的部分依据，即该用户已购或喜欢的类似物品

正如亚马逊等网站所展现的，推荐系统通过提供绝佳的交叉销售机会，从而产生实在的商业价值。某公司的报告显示，向用户推荐的产品能够使销售额增长8%~12%。^①

1.2.2 聚类

聚类（clustering）的概念没有那么浅显易懂，但它也是在同样的应用场景下出现的。顾名思义，聚类技术试图将大量的事物组合为拥有类似属性的簇（cluster），借以在一些规模较大或难于理解的数据集上发现层次结构和顺序，以揭示一些有用的模式或让数据集更易于理解。

- Google News使用聚类技术通过标题把新闻文章进行分组，从而按照逻辑线索来显示新闻，而非给出所有文章的原始列表。如图1-3所示。
- 出于类似的原因，像Clusty这样的搜索引擎也将其查询结果进行分组。
- 聚类技术可以根据如收入、居住地和购买习惯等属性，将消费者分为许多段（簇）。

Obama to Name 'Smart Grid' Projects
Wall Street Journal - Rebecca Smith - 1 hour ago

The Obama administration is expected Tuesday to name 100 utility projects that will share \$3.4 billion in federal stimulus funding to speed deployment of advanced technology designed to cut energy use and make the electric-power grid ...

[Cobb firm wins "smart-grid" grant](#) Atlanta Journal Constitution
[Obama putting \\$3.4B toward a 'smart' power grid](#) The Associate
[Baltimore Sun - Bloomberg - New York Times - Reuters](#)
[all 594 news articles »](#) [Email this story](#)

图1-3 Google News分组出的一段新闻样本。展现了一个代表性事件的详细片段，并且呈现了该主题下同一个集群内其他几个类似事件的链接。你也可以得到在该主题下聚类在一起的所有事件的链接

^① 实用电子商务，“10 Questions on Product Recommendations”（产品推荐十问），<http://mng.bz/b6A5>。

聚类可以帮你在一大堆东西中找到脉络，甚至层次关系，否则理解这些数据将非常困难。利用这种技术，企业可以发现用户中潜在的群体，可以合理地组织大量文档，还可以根据日志来发现用户使用网站的常见模式。

1.2.3 分类

分类技术决定了一个事物多大程度上从属于某些类别或类型，或者多大程度上具有或不具有某些属性。与聚类一样，分类也无处不在，但是更多隐身于幕后。通常这些系统会考察类别中的大量实例，来学习推导出分类的规则。这种通常的方法有许多应用。

- 雅虎邮箱基于用户以前对正常或垃圾邮件的报告，以及电子邮件自身的特征，来判别到来的消息是否为垃圾邮件。几个被归类为垃圾邮件的消息如图1-4所示。
- 谷歌的Picasa和其他照片管理应用可以判断出一张照片中是否包含了人脸。
- OCR（Optical Character Recognition，光学字符识别）软件将一个个小块中的扫描文本分类成不同的字符。
- 据称iTunes中苹果公司的Genius特性使用分类来处理歌曲，为用户生成可能的播放列表。

Spam (49)	Empty	<input type="checkbox"/>	Hevnerco	DishView	Wed 10/28, 12:34 PM
Trash	Empty	<input type="checkbox"/>	Customer Service	FINAL NOTIFICATION:..Please r...	Wed 10/28, 4:53 AM
Contacts	Add	<input type="checkbox"/>	MmddDdhbh	From: MmddDdhb Read The File.	Wed 10/28, 12:58 AM

图1-4 由雅虎邮件检测出的垃圾消息。基于来自用户的垃圾邮件报告，结合其他分析，系统就能习得一些通常可用于确定垃圾邮件的属性。例如，提到“Viagra”的消息通常为垃圾邮件，故意错拼为“vlagra”的消息也一样。这些词项（term）^①的出现就是垃圾邮件过滤器可以习得的一个属性

分类有助于判断一个新的输入或新的事物是否与以前观察到的模式相匹配，它通常还被用于遴选异常的行为或模式，来检测可疑的网络活动或欺骗行为。它还可用于“察觉”某个用户的消息是否存在失望或满意情绪。

如果输入数据的质量好，这些技术都可以完美地处理大量数据。但有时不仅需要处理大量的输入，还必须快速生成结果，这就使可扩展性（scalability）成为一个主要问题。并且，如前所述，Mahout存在的一个重要原因是能够为这些技术提供实现手段，从而使之向上扩展到处理庞大的输入数据。

1.3 利用Mahout和Hadoop处理大规模数据

规模问题在机器学习算法中有什么现实意义？让我们考虑你可能需要部署Mahout来解决的

^① 词项是信息检索（information retrieval）领域的标准术语，意指用于表示查询或文档的特征，实际中文本常用单词（word）来表示词项，但是词项不一定就是单词。严格地说，词项、词条（token）和单词都不完全一样。本书并没有严格区分。——译者注

几个问题的大小。

据粗略估计, Picasa三年前就拥有了5亿张照片。^①这意味着每天有百万级的新照片需要处理。一张照片的分析本身不是一个大问题, 即使重复几百万次也不算什么。但是在学习阶段可能需要同时获取数十亿张照片中的信息, 而这种规模的计算是无法用单机实现的。

据报道, Google News每天都会处理大约350万篇新的新闻文章。虽然它的绝对词项数量看似不大, 但试想一下, 为了及时提供这些文章, 它们连同其他近期的文章必须在几分钟的时间内完成聚类。

Netflix为Netflix Prize公布的评分数据子集中包含了1亿个评分。因为这仅仅是针对竞赛而公布的数据, 据推测Netflix为形成推荐结果所需处理的数据总量与之相比还要大出许多倍。

机器学习技术必须部署在诸如此类的应用场景中, 通常输入数据量都非常庞大, 以至于无法在一台计算机上完全处理, 即使这台计算机非常强大。如果没有Mahout这类的实现手段, 这将是一项无法完成的任务。这就是Mahout将可扩展性视为重中之重的道理, 以及本书将焦点放在有效处理大数据集上的原因, 这一点与其他书有所不同。

将复杂的机器学习技术应用于解决大规模的问题, 目前仅为大型的高新技术公司所考虑。但是, 今天的计算能力与以往相比, 已廉价许多, 且可以借助于Apache Hadoop这种开源框架更轻松地获取。Mahout通过提供构筑在Hadoop平台上的、能够解决大规模问题的高质量的开源实现以期完成这块拼图, 并可为所有技术团体所用。

Mahout中的有些部分利用了Hadoop, 其中包含一个流行的MapReduce分布式计算框架。MapReduce被谷歌在公司内部得到广泛使用 (<http://labs.google.com/papers/mapreduce.html>), 而Hadoop是它的一个基于Java的开源实现。MapReduce是一个编程范式, 初看起来奇怪, 或者说简单得让人很难相信其强大性。MapReduce范式适用于解决输入为一组“键–值对”的问题, map函数将这些键值对转换为另一组中间键值对, reduce函数按某种方式将每个中间键所对应的全部值进行合并, 以产生输出。实际上, 许多问题可以归结为MapReduce问题, 或它们的级联。这个范式还相当易于并行化: 所有处理都是独立的, 因此可以分布到许多机器上。这里不再赘述MapReduce, 建议读者参考一些入门教程来了解它, 如Hadoop所提供的http://hadoop.apache.org/mapreduce/docs/current/mapred_tutorial.html。

Hadoop实现了MapReduce范式, 即便MapReduce听上去如此简单, 这仍然称得上是一大进步。它负责管理输入数据、中间键值对以及输出数据的存储; 这些数据可能会非常庞大, 并且必须可被许多工作节点访问, 而不仅仅存放在某个节点上。Hadoop还负责工作节点之间的数据分区和传输, 以及各个机器的故障监测与恢复。理解其背后的工作原理, 可以帮你准备好应对使用Hadoop可能会面对的复杂情况。Hadoop不仅仅是一个可在工程中添加的库。它有几个组件, 每个都带有许多库, 还有(几个)独立的服务进程, 可在多台机器上运行。基于Hadoop的操作过程并不简单, 但是投资一个可扩展、分布式的实现, 可以在以后获得回报: 你的数据可能会很快增长到很大的

^① Google Blogoscoped, “Overall Number of Picasa Photos” (2007年3月12日), 参见<http://blogoscoped.com/archive/2007-03-12-n67.html>。

规模，而这种可扩展的实现让你的应用不会落伍。

在第6章，本书将尝试克服这种复杂性，让你可以很快地在Hadoop上运行程序，之后你可以探索或研究关于集群操作和框架调优的细节。鉴于这种需要大量计算能力的复杂框架正变得越来越普遍，云计算提供商开始提供Hadoop相关的服务就不足为奇了。例如，亚马逊提供了一种管理Hadoop集群的服务Elastic MapReduce (<http://aws.amazon.com/elasticmapreduce/>)，该服务提供了强大的计算能力，并使我们可通过一个友好的接口在Hadoop上操作和监控大规模作业，而这原本是一个非常复杂的任务。

1.4 安装Mahout

之后的章节将会出现一些代码，你需要先配备一些工具才能随意使用这些代码。我们假设你对Java开发环境已经很熟悉了。

Mahout及其相关框架是基于Java实现的，因此具有平台独立性，你能够在任何一个可运行较新版JVM的平台上使用它。不过，我们有时仍然需要针对平台之间的差异性给出示例和解释。特别是在Windows shell上，其命令行命令与FreeBSD tcsh shell的不同。我们会使用bash中可用的命令和语法，它是大多数类Unix平台所采用的shell。默认情况下，大多数Linux发布包、Mac OS X、许多Unix变种和Cygwin（Windows上一种流行的类Unix环境）都使用它。打算使用Windows shell的用户对此很可能不习惯。不过，这些用户仍可以简单地使用本书所提供的代码清单，把命令翻译为在bash shell中可用的形式。

1.4.1 Java和IDE

如果做过Java开发，你的个人电脑上很可能已经安装了Java环境。注意，Mahout需要Java 6的支持。如果你不确定使用了哪个Java版本，可以打开一个终端并输入java-version查看。如果显示的版本低于1.6，你仍需要安装Java 6。

Windows和Linux用户可以在Oracle找到Java 6的JVM，网址为<http://www.oracle.com/technetwork/java/>。苹果为Mac OS X 10.5和10.6提供了Java 6的JVM。在Mac OS X，如果显示所用版本不是Java 6，可以在/Applications/Utilities文件夹下打开Java Preferences应用。这里允许你将Java 6设为默认选项。

借助IDE（集成开发环境），大多数人可以轻松地编辑、编译和运行本书的示例；我们强烈推荐你使用IDE。Eclipse (<http://www.eclipse.org>) 是最流行的免费Java IDE。本书不会涉及Eclipse的安装和配置，但继续阅读本书之前，你最好花点时间熟悉它。NetBeans (<http://netbeans.org/>) 也是一个流行的免费IDE。另一个强大而流行的IDE是IntelliJ IDEA (<http://www.jetbrains.com/idea/index.html>)，目前可获得免费的社区版本。

举一个使用IDE的例子，IDEA可以从现有的Maven模型中创建一个新的项目；如果你在创建项目时指定了Mahout源码的根目录，它会将整个项目按组织好的方式进行自动配置和展示。因此，我们可以将本书中所有的源代码放入examples/src/main/java/源码根目录中，并在IDEA中一键式运

行——依赖关系和编译的细节都会得到自动管理。这比手动编译和运行代码要容易得多。

注意 如果测试程序使用输入数据中的一个文件，它通常应该在与数据文件相同的目录中运行。查看你所用IDE的手册，了解如何为每个示例配置一个工作目录。

1.4.2 安装Maven

如同许多Apache的项目一样，Mahout利用Maven (<http://maven.apache.org>) 来构建和发布项目。Maven是一个命令行工具，它管理依赖关系、编译代码、形成软件包、生成文档并发布正式版本。虽然它表面上类似于同样流行的工具Ant，实际却并不与之相同。Ant是一个灵活的低级脚本语言，而Maven是一个更重视依赖关系和发布管理的高级工具。鉴于Mahout使用了Maven，你最好把它安装好。

Mac OS X的用户会很高兴地发现Maven已经安装好了。如果没有，可以安装苹果的Developer Tools。在命令行输入mvn --verison。如果你成功地看到版本号，且版本大于或等于2.2，你就 can 使用它了。否则，你需要在本地安装Maven。

用户若使用一个带有适当包管理系统的Linux发行版，便可以很快获得一个Maven的当前版本；否则就需要按照标准的流程进行安装，即下载一个二进制发行版，在一个类似/usr/local/maven的公共目录中解压，再编辑bash的配置文件~/.bashrc并添加一行，如export PATH=/usr/local/maven/bin:\$PATH。它确保你随时可以使用mvn命令。

如果你正在使用Eclipse或IntelliJ这样的IDE环境，Maven已经被集成在其中了。参考其文档可以了解如何打开Maven集成的功能。这会大大简化Mahout在IDE中的使用，因为IDE可以使用一个项目中的Maven配置文件 (pom.xml)，来即刻配置并导入这个项目。

注意 对于Eclipse，你需要安装m2eclipse插件 (<http://www.eclipse.org/m2e/>)。对于NetBeans，自6.7版之后就已经支持了Maven；而对于以前的版本，你需要额外安装一个插件。

1.4.3 安装Mahout

Mahout仍在不断发展，本书使用的是Mahout的0.5发布版。在<https://cwiki.apache.org/confluence/display/MAHOUT/Downloads>上可以找到下载这个发布版及其他版本的提示；你可以在计算机上找一个方便的地方将源码的压缩包解压。

因为Mahout的变更很频繁，定期会加入bug修复和一些改进，也许使用0.5的后续版本会更好（甚至可以用Subversion上仍未发布的最新代码，参见<https://cwiki.apache.org/confluence/display/MAHOUT/Version+Control>）。后续的发布包可以向后兼容地运行本书所提供的示例。

一旦你获得了源码，无论是从Subversion还是从发布包获得，都可以在IDE中为Mahout创建

一个新项目。IDE各不相同，参考其文档可以掌握它们在创建项目中的特殊用法。最简单的办法是使用IDE所集成的Maven，从项目源代码根目录中的pom.xml文件导入Maven项目。

一旦完成了这些步骤，你就可以很轻松地在这个项目中创建一个新的源代码目录，用来存放后续章节中所介绍的示例代码。正确地配置这个项目，以便可以顺利地编译并运行这些代码，这样你就无须付出额外的努力。

这些示例的源代码可以从Manning的网站（<http://www.manning.com/MahoutinAction/>）或GitHub（<https://github.com/tdunning/MiA>）上获得。你可以根据源码所提供的指导来建立你的工作环境。

1.4.4 安装Hadoop

你需要在本地安装一个Hadoop，来完成本书稍后所涉及的操作。你不必用一个集群来运行Hadoop。安装Hadoop虽不困难，但却有些烦琐。这里并不重复这个过程，我们将指导你从Hadoop网站<http://hadoop.apache.org/common/releases.html>获取一个0.20.2版的Hadoop副本，并遵照单节点安装文档（http://hadoop.apache.org/common/docs/current/single_node_setup.html）来安装一个伪分布模式的Hadoop。

1.5 小结

Mahout来自Apache，它是一个“年轻”、开源、可扩展的机器学习库，而本书将指引你在Mahout上使用机器学习技术解决实际问题。特别地，你会很快了解推荐引擎、聚类和分类。如果你是一个熟知机器学习理论的研究者，正在寻找一个实用的how-to指南，或者是一个希望快速掌握从业者宝贵经验的开发者，那么这本书正是为你而写。

这些技术已经不再只是理论。我们已经知道在现实世界中有许多广为人知的机器学习案例，它们采用了推荐引擎、聚类和分类：电子商务、电子邮件、视频网站、照片网站以及更多。这些技术已经被用于解决实际问题，甚至为企业创造价值——现在它们都可以借助Mahout来实现。

我们已经发现这些技术有时会涉及大量的数据——可扩展性是这个领域中一个独特而永恒的话题。初步审视MapReduce和Hadoop，我们了解到它们是如何承载了Mahout所提供的可扩展性。

因为本书注重实际操作，所以我们让你一上来就准备好去使用Mahout。现在，你应该已经安装了Mahout工作所需的工具，并准备开始行动了。因为本书旨在实战，让我们现在就结束开篇，来看看Mahout的实际代码。请看后续篇章！

Part 1

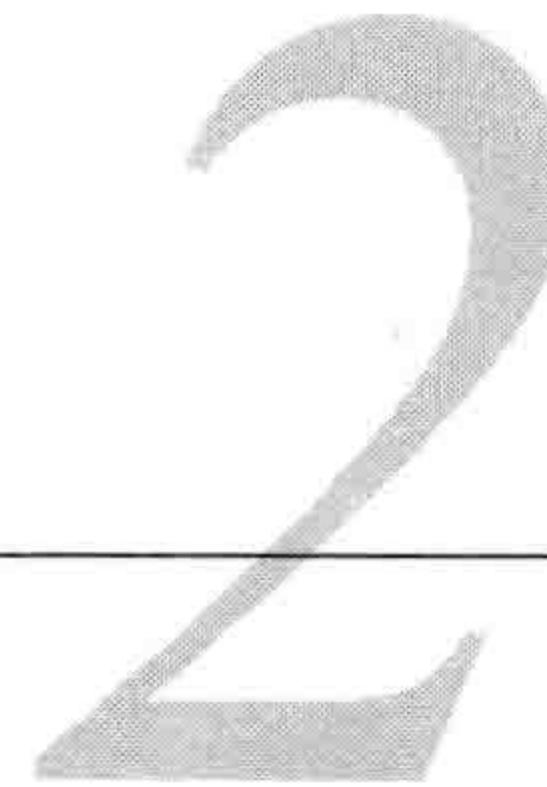
第一部分

推荐

本书第一部分涵盖第2章至第6章，探讨Apache Mahout机器学习实现的三大支柱之一：协同过滤（collaborative filtering）和推荐（recommendation）。通过这些技术，你能够了解一个人的品味，并自动找到新内容来投其所好。本部分仍为后续章节的铺垫，后续章节将高度依赖于Apache Hadoop的分布式计算框架。我们先通过简单的Java程序来了解Apache Mahout的机器学习，再使用Hadoop来实现它。

第2章介绍由Mahout实现的推荐引擎（recommender engine），并在一个可运行的示例中评价性能。第3章讨论Mahout中推荐程序（recommender）的高效数据表示。第4章分类说明Mahout中推荐引擎的各种实现及其不同的属性特征。

第5章给出一个实例，其数据来自一个约会网站，由此讨论如何采用Mahout中的方法来处理真实数据，从而形成一个可供生产环境使用的推荐程序。最终，第6章会初步在Apache Hadoop上使用Mahout，以实现一个大型的分布式推荐引擎。



本章内容

- Mahout中的推荐系统
- 推荐系统实战初探
- 评估推荐引擎的精度和质量
- 评估基于实际数据集GroupLens的推荐程序

我们每天都对事物形成观点：喜欢、不喜欢，甚或不关心。这都是无意识中发生的。当你在广播中听到一首歌时，你可能因为它动听而注意它，也可能因为它难听而注意它，也有可能压根儿就没有注意到它。同样的情形还适用于T恤衫、色拉、发型、滑雪场、容貌和电视节目。

人们的嗜好各异，却有规律可循。人们倾向于喜欢那些与其爱好相似的东西。由于Sean喜欢吃火腿-莴苣-番茄三明治，你就可以猜测他可能会喜欢总会三明治（club sandwich），因为它们基本上是一样的，只是后者使用了火鸡肉。而且，人们容易爱上类似人群所喜欢的东西。

这些模式可用于预测人们的好恶。推荐就是通过对嗜好的这些模式进行预测，借以发现你尚未知晓，却合乎心意的新事物。

在更深入地介绍推荐思想之后，本章将帮助你体验Mahout的一段代码，用以运行一个简单的推荐引擎并了解其执行效果，从而让你直观感受一下Mahout是如何实现推荐的。

2.1 推荐的定义

你从书架上拿起这本书是有原因的。也许它恰好放在对你有用的其他书籍的旁边，而你明白之所以书店会把它放在那里，是因为喜欢那些书的人很可能也会喜欢这本书。或许它恰好放在你同事的书架上，而你们在机器学习方面志趣相投，也有可能是他们直接向你推荐了本书。

这些策略虽然各不相同，但在发掘新鲜事物上都是有效的：要找到你可能喜欢的物品，你可以观察与你志趣相投的人喜欢些什么。另一方面，通过观察其他人的明显偏好，你可以弄清楚哪些东西和你已然喜欢的物品相似。实际上，它们是推荐引擎算法中应用最广的两大类：基于用户（user-based）和基于物品（item-based）的推荐程序，它们均在Mahout中得到了充分展现。

严格说来，上述场景均为协同过滤的范例——仅仅通过了解用户与物品之间的关系进行推荐。这些技术无须了解物品自身的属性。从某种意义上讲，这是一个优点。该推荐框架并不关心物品是否为书籍、主题公园、鲜花或其他人，因为根本不会导入它们的属性。

其他一些方法则立足于物品的属性，通常称为基于内容（content-based）的推荐技术。例如，如果有朋友向你推荐本书，原因是它是Manning出版的，而且他也喜欢Manning出版的其他书，那么这个朋友所做的就是类似于基于内容的推荐。它给你的建议是基于书的属性，即基于“出版商”作出的。

基于内容的推荐技术没有什么问题，相反，它们很有用。但是，它们必须与特定领域相结合，而难以规整为一个框架。为了构造一个有效的基于内容的图书推荐程序，人们不得不确定图书的哪种属性（页数、作者、出版商、颜色、字体）是有意义的，以及有多大意义。这些知识无法转换以用于其他领域，比如这种推荐图书的方法对于推荐比萨配料毫无用处。

因此，Mahout对基于内容的推荐所言甚少。这些思想能够融入并构建在Mahout之上；故而，Mahout在技术上可称为一种协同过滤框架。第5章会给出一个示例，指导你为约会网站创建一个推荐程序。

但在现阶段，我们先生成一些简单的输入并据此找出推荐结果，来体验一下Mahout中的协同过滤。

2.2 运行第一个推荐引擎

Mahout包含一个推荐引擎，其中有几种类型实际来自于传统的基于用户和基于物品的推荐程序。它也包含了其他几种算法实现，但是现在我们先看一个简单的基于用户的推荐程序。

2.2.1 创建输入

为了探究Mahout中的推荐，最好从一个简单的例子开始。

推荐程序需要有输入——构成推荐的基础数据。在Mahout的语言中，数据是以偏好（preference）的形式来表达的。因为最常见的推荐引擎总是把项目推荐给用户，所以谈论偏好最简便的方法是建立从用户到物品的关联，尽管如前所述，这些用户和物品是可以任意指定的。一个偏好包含一个用户ID、一个物品ID，通常还有一个表达用户对物品的偏爱程度的数值。实际上，Mahout中的ID通常也为数字——整数。偏好值（preference value）可任意设定，只需保证更大的值代表更强的正向偏好。例如，这些值可能按从1到5来定级，其中1表示用户非常不喜欢该物品，而5表示物品是用户的至爱。

创建一个包含关于用户数据的文本文件，巧妙地从1到5为用户命名，从101到107为他们喜欢的7本书命名。在现实世界，这些数据可能是来自某公司数据库的顾客ID和产品ID；Mahout并不要求用户和物品必须按照数字命名。我们用一种简单的以逗号分隔值的格式把这些数据写入文件。

复制下面的示例到一个文件中并将之存为intro.csv。

代码清单2-1 推荐程序的输入文件intro.csv

```

1,101,5.0
1,102,3.0
1,103,2.5
2,101,2.0
2,102,2.5
2,103,5.0
2,104,2.0
3,101,2.5
3,104,4.0
3,105,4.5
3,107,5.0
4,101,5.0
4,103,3.0
4,104,4.5
4,106,4.0
5,101,4.0
5,102,3.0
5,103,2.0
5,104,4.0
5,105,3.5
5,106,4.0

```

一番研究之后，倾向就明显了。用户1和5似乎有相似的喜好。他们都最喜欢101这本书，其次喜欢102，再次喜欢103。同样，对于用户1和4，他们似乎都喜欢101和103（但用户4对102的关系不明）。另一方面，用户1和2的喜好基本上是对立的：用户1喜欢101，而用户2对其不感兴趣；用户1喜欢103，而用户2则恰恰相反。用户1和3的喜好迥异——他们仅同时喜欢101。图2-1显示了用户和物品之间正面和负面的关系。

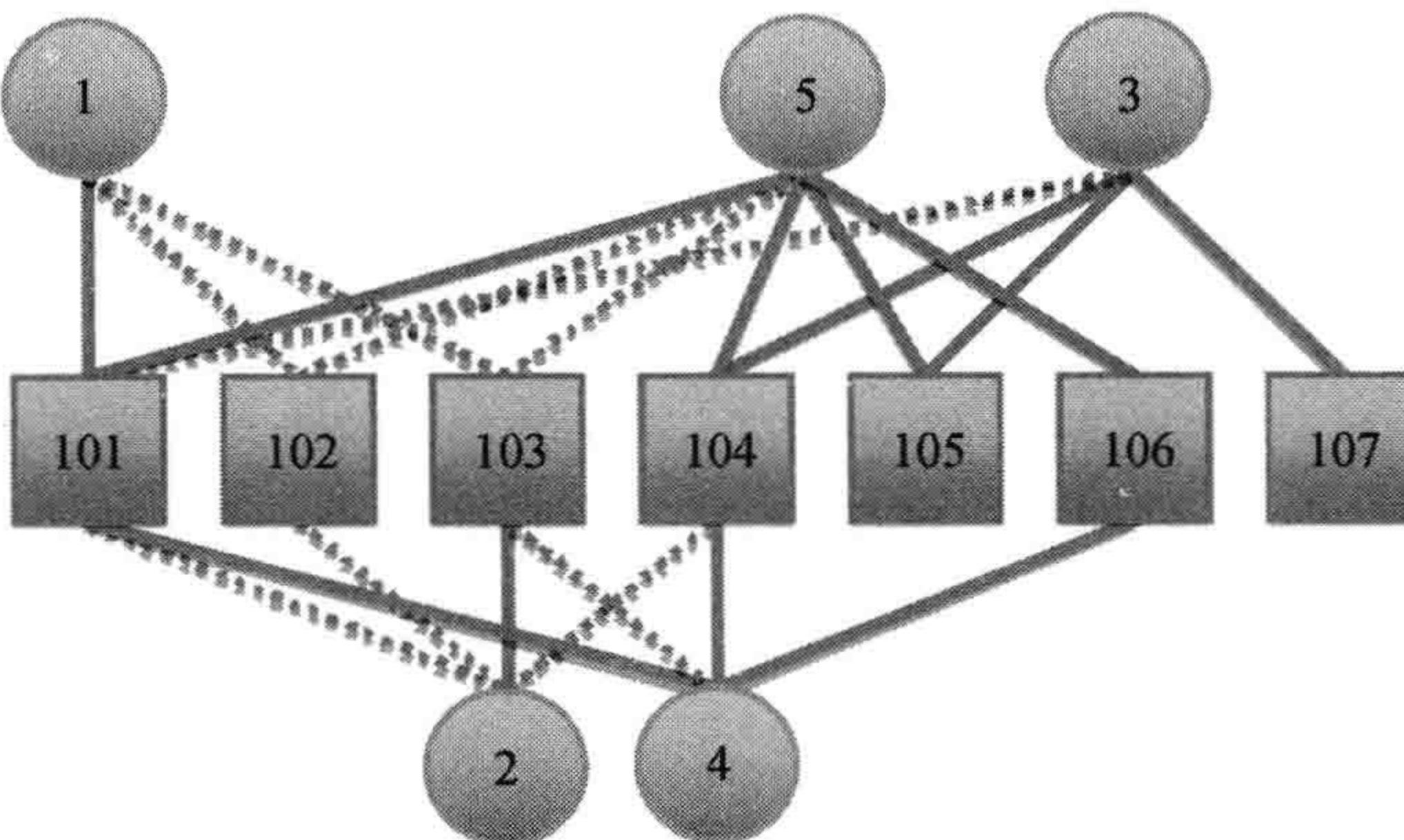


图2-1 用户1到5和物品101到107的关系。虚线表示看似负面的关系，即用户似乎不太喜欢这个物品，但也表达了对物品的态度

2.2.2 创建一个推荐程序

那么，可以为用户1推荐什么书呢？不是101、102和103，因为用户1显然已经知道这些书了，而推荐是用来发现新事物的。直观上看，既然用户4和5与用户1类似，那么把用户4或5喜欢的东西推荐给用户1是个好主意。这样一来，可能的推荐结果就是书104、105和106。总体上看，104似乎最有可能，因为物品104对应的偏好值是4.5和4.0。

现在，运行如下代码。

代码清单2-2 一个简单的基于用户的Mahout推荐程序

```
class RecommenderIntro {
    public static void main(String[] args) throws Exception {
        DataModel model =
            new FileDataModel (new File("intro.csv"));           ←—— 装载数据文件
        UserSimilarity similarity =
            new PearsonCorrelationSimilarity (model);
        UserNeighborhood neighborhood =
            new NearestNUserNeighborhood (2, similarity, model);
        Recommender recommender = new GenericUserBasedRecommender (
            model, neighborhood, similarity);                  ←—— 生成推荐引擎
        List<RecommendedItem> recommendations =
            recommender.recommend(1, 1);                         ←—— 为用户1推荐物品1
        for (RecommendedItem recommendation : recommendations) {
            System.out.println(recommendation);
        }
    }
}
```

图2-2形象地表示了这些基础组件之间的关系。并非所有基于Mahout的推荐程序都是如此，有些会采用不同的组件、不同的关系。但这个例子先让我们对此有一些感觉。

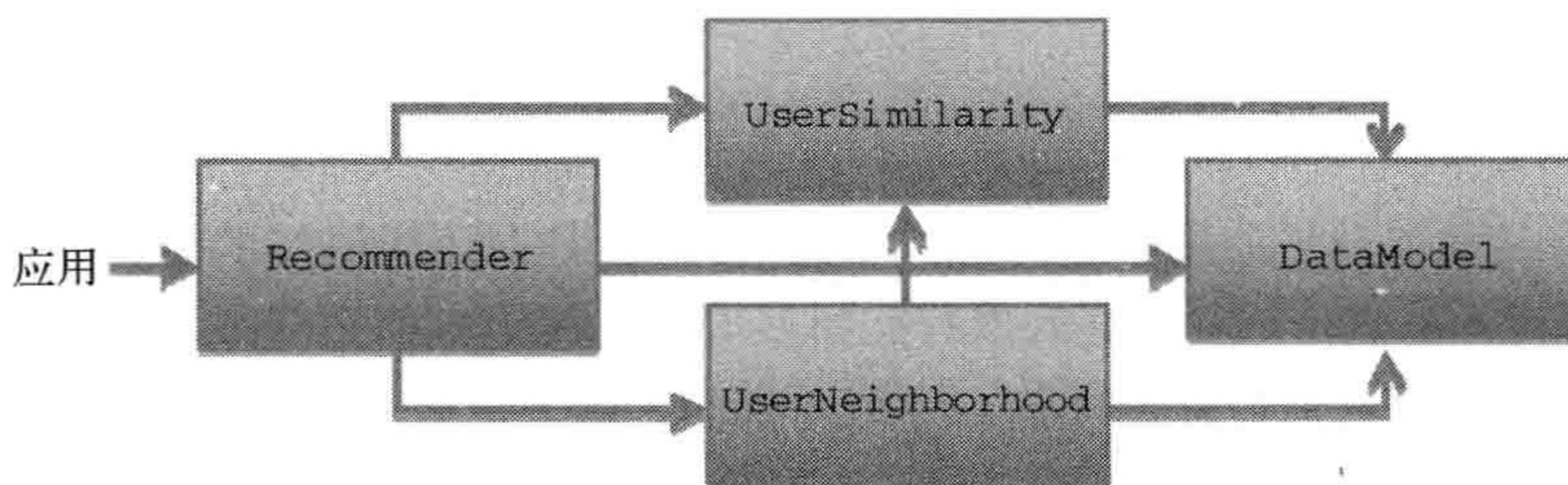


图2-2 Mahout基于用户推荐程序中组件间关系的简单示意图

在接下来的两章中，我们将详细地逐一讨论这些组件，但现在先对每个组件的角色做一个概览。DataModel实现存储并为计算提供其所需的所有偏好、用户和物品数据。UserSimilarity

实现给出两个用户之间的相似度，可从多种可能度量或计算中选用一种来作为依据。UserNeighborhood实现明确了与给定用户最相似的一组用户。最后，Recommender实现合并所有这些组件为用户推荐物品。

2.2.3 分析输出

当运行代码清单2-2中的代码时，你的终端或IDE（集成开发环境）会输出如下结果：

```
RecommendedItem [item:104, value:4.257081]
```

该请求寻找一个最优的推荐结果，并最终找到了一个。推荐引擎把书104推荐给用户1。而且推荐引擎之所以这样做，是因为它估计出用户1对书104的偏好值约为4.3，而这在所有适合推荐的物品中是最高的。

推荐结果还不错。没有出现书107；它虽然也在可被推荐之列，但它仅和一个嗜好不同的用户相关。推荐引擎选择104而没选106也是合理的，因为能看到104的总体评分略高。此外，输出还包含了一个合理的估计，即用户1有多喜欢物品104——大致在用户4和5所表达的偏好值4.0和4.5之间。

从数据中不能一眼看出正确答案，但是推荐引擎找到了它的踪迹，并返回了一个合理的答案。这个简单的程序找到了一个不易发现的有用结果，如果你为此感到欢欣鼓舞，那就表示机器学习的世界很适合你。

对于干净的小数据集，生成推荐结果就像前面的示例一样简单。但现实中，数据集往往非常庞大，而且其中很多信息没有价值。例如，假设一个受欢迎的新闻网站要为读者推荐新闻文章。可以根据文章点击率推断出偏好，但也可能会产生很多假的偏好——或许读者点击了并不喜欢的文章，或错误地点击了一个故事。或许很多点击发生在未登录状态下，因而不能与某个用户进行对应。再试想一下数据集的大小——也许每个月的点击量有几十亿次。

要在该数据之上快速生成准确的推荐结果并不简单。后面的案例研究中，我们将使用Mahout提供的工具来解决一组这样的问题。它们会为你呈现标准的方法会如何导致糟糕的推荐结果，或是耗费大量的内存和CPU时间，同时展示如何配置和定制Mahout来提高性能。

2.3 评估一个推荐程序

推荐引擎是一种工具，一种解答问题的手段。“什么是对用户最好的推荐？”在探寻其答案之前，最好先深究一下这个问题。好的推荐需要多准确？用户如何获知推荐程序正在输出最佳结果？本章后续部分将转而探讨如何评估一个推荐程序，因为这会有助于审视特定的推荐系统。

最佳的推荐程序应该就像是一个“巫师”，它能够在你行动之前设法准确地获知你喜欢的每一种可能的物品，而且这些物品是你尚未见过或没有对其表达过任何喜好意见的。能够准确预测你所有喜好和行为的推荐程序还应按你未来的喜好把物品进行排队。最优的潜在推荐结果应该就是这样。

而实际上，大多数推荐引擎仅会试图给出某些或其他所有物品的估计评分。由此，一种评估推荐程序推荐结果的方法是评估其估计偏好值的质量——即评估所估计的偏好在多大程度上与实际偏好相匹配。

2.3.1 训练数据与评分

但是，没有现成的实际偏好值可用。没有人能确切地知道你将来有多喜欢某些新东西（包括你之内）。在推荐引擎中，这可以通过提取一小段真实数据作为测试数据来仿真。这些用于测试的偏好不会作为训练数据导入到被评估的推荐引擎。相反，推荐程序需要为这些缺失的测试数据估计出偏好值，然后估计结果用于与真实值进行对照。

进而，我们可以非常简单地为推荐程序做一种评分。例如，可以计算出在估计和实际偏好之间的平均差值。在这种评分中，值越低越好，因为值越低意味着估计值与实际偏好值的差别越小。评分为0.0意味着完美的估计，即在估计值和实际偏好值之间根本没有差别。

有时会使用差值的均方根：计算出实际偏好值和估计值之间的差值之后，先进行平方再求其均值的平方根。见表2-1。值同样是越低越好。

表2-1 平均差值与均方根的计算说明

	物品1	物品2	物品3
真实值	3.0	5.0	4.0
估计值	3.5	2.0	5.0
差值	0.5	3.0	1.0
平均差值	= $(0.5+3.0+1.0)/3=1.5$		
均方根	$=\sqrt{(0.5^2 + 3.0^2 + 1.0^2) / 3} = 1.8484$		

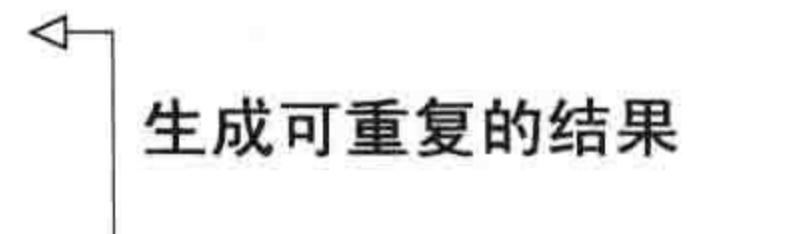
表2-1显示了一组实际偏好和估计之间的差值，以及如何将它们转换为评分。均方根使得估计值的偏离显得更严重，正如这里的物品2，这在有时是需要的。例如，相比于偏离1颗星的估计值，偏离2颗星对推荐所造成的不良影响也许会超过2倍。鉴于简单地对差值求平均可能更直观和易于理解，后续的例子中都会采用这种方法。

2.3.2 运行RecommenderEvaluator

让我们重温示例程序，在简单的数据集上评估这个简易的推荐程序，如下列代码清单所示。

代码清单2-3 配置并评估一个推荐程序

```
RandomUtils.useTestSeed();
DataModel model = new FileDataModel (new File("intro.csv"));
RecommenderEvaluator evaluator =
    new AverageAbsoluteDifferenceRecommenderEvaluator ();
```



```

RecommenderBuilder builder = new RecommenderBuilder() {
    @Override
    public Recommender buildRecommender(DataModel model)
        throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity (model);
        UserNeighborhood neighborhood =
            new NearestNUserNeighborhood (2, similarity, model);
        return
            new GenericUserBasedRecommender (model, neighborhood, similarity);
    }
};

double score = evaluator.evaluate(
    builder, null, model, 0.7, 1.0);           ← 训练70%的数据，测试30%
System.out.println(score);

```

构建如代码清单2-2所示的推荐程序

大多数行为发生在evaluate()中：RecommenderEvaluator将数据分为训练集和测试集，构建一个新训练的DataModel与Recommender用于测试，并将估计的偏好值与实际测试数据进行比较。

注意传递给evaluate()的参数中没有Recommender。这是因为在该方法中，Recommender是由新训练的DataModel来构建的。该方法的调用者必须提供一个对象——RecommenderBuilder，可以使用DataModel构建出Recommender。这里，该方法所用的是和本章之前所述相同的实现。

2.3.3 评估结果

代码清单2-3中的程序输出评价的结果：一个显示Recommender表现如何的分数。在这个例子中，你只会看到1.0这一个值。即使evaluator在选择测试数据时引入许多随机量，结果仍是相同的，因为对RandomUtils.useTestSeed()的调用会强制每次选择相同的随机值。这仅仅是为了获得可重复的结果，而被用在这样的示例或单元测试中。请不要在实际代码中这样用！

这个分值的意义取决于所采取的实现方法，这里是AverageAbsoluteDifferenceRecommenderEvaluator。在该实现中分值为1.0，这意味着平均而言推荐程序所给出的估计值与实际值的偏差为1.0。

在从1至5的区间中，1.0这个值并不大，但我们这里只采用了非常少的数据。你来执行时所获得的结果也许会不同，因为对数据集的分片是随机的，而且程序每次运行所用的训练集和测试集也可能不一样。

这个技术可以应用于任何Recommender和DataModel。如要使用均方根来评分，可以用RMSRecommenderEvaluator取代AverageAbsoluteDifferenceRecommenderEvaluator。

你可以选择不向evaluate()传递null(空)参数，而是传递DataModelBuilder的一个实例(instance)，它可用于控制如何从训练数据中生成DataModel。通常默认地传空参数就够了，除非你使用了一个特殊的DataModel实现——一个你希望插入到评估过程中的DataModelBuilder。

最后传递给evaluate()的参数1.0是用来控制总共使用多少输入数据的。这里，它是指100%的数据。这个参数可用于仅通过庞大数据集中的很小一部分数据，来生成一个精度较低但

更快的评估。例如，0.1代表使用10%的数据，而90%的数据被忽略。当你希望快速测试Recommender中一些小的更改时，这个参数会很有用。

2.4 评估查准率与查全率

我们还应该更全面地看待推荐问题：通过估计偏好值来生成推荐结果并非绝对必要。给出一个从优到劣排列的推荐列表对于许多场景都够用了，而不必包含估计的偏好值。事实上，有时精确的列表顺序也不那么重要——有几个好的推荐结果就可以了。

从这种更普遍的视角，我们还可以运用经典的信息检索（information retrieval）度量标准来评估推荐程序：查准率（precision）和查全率（recall）。这些术语通常用在像搜索引擎这样的系统中，即从许多可能的搜索结果中返回一组最佳结果。

搜索引擎应避免在top结果中返回无关信息，而应竭力返回尽可能相关的结果。在一些对“相关”的定义中，查准率是指在top结果中相关结果的比例。“Precision at 10”（推荐10个结果时的查准率）是指这个比例来自对前10个top结果的判定。查全率是指所有相关结果包含在top结果中的比例。图2-3给出了它们的图示。

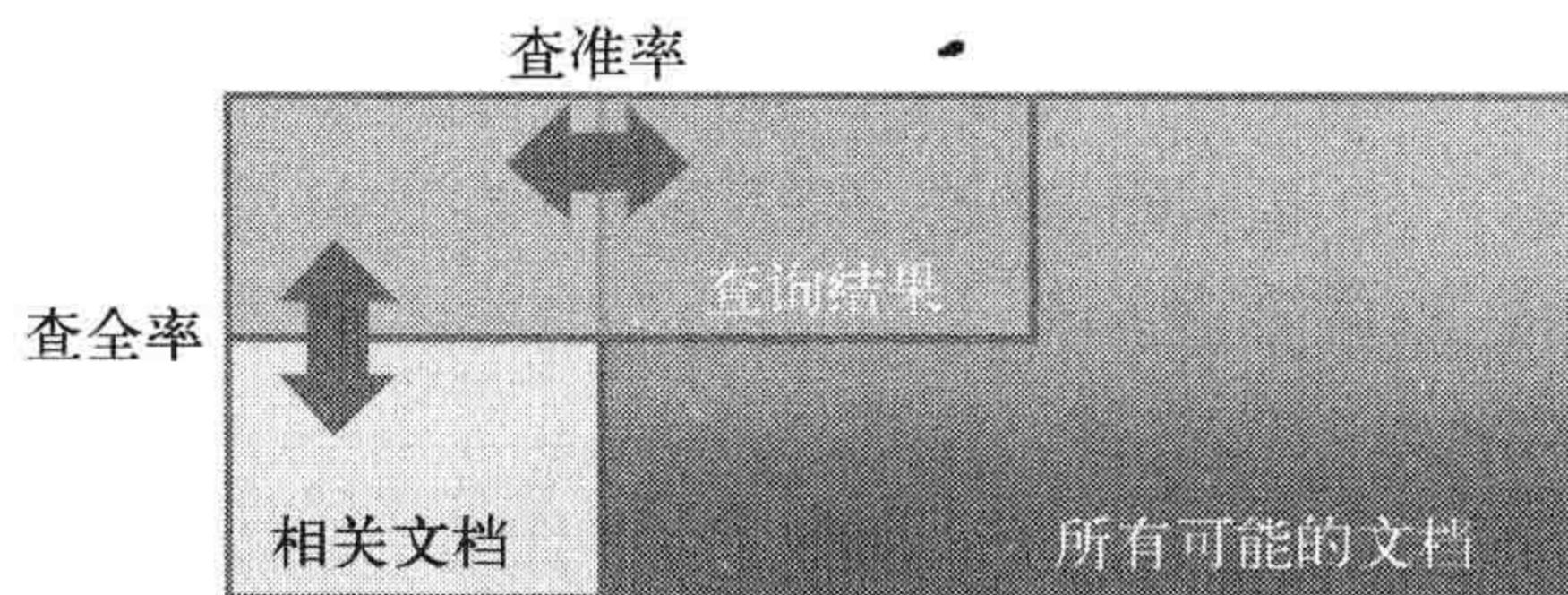


图2-3 在搜索结果中查准率和查全率的说明

这些术语很容易用在推荐程序中：查准率是top推荐中间有“好”结果的比例，而查全率是“好”结果出现在top推荐中的比例。下一节将定义何为“好”。

2.4.1 运行RecommenderIRStatsEvaluator

Mahout同样提供了一个相当简单的方法，为Recommender计算出这些值，如下面的代码清单所示。

代码清单2-4 查准率和查全率评估的配置与运行

```
RandomUtils.useTestSeed();
DataModel model = new FileDataModel (new File("intro.csv"));

RecommenderIRStatsEvaluator evaluator =
    new GenericRecommenderIRStatsEvaluator ();
```

```

RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
    @Override
    public Recommender buildRecommender(DataModel model)
        throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity (model);
        UserNeighborhood neighborhood =
            new NearestNUserNeighborhood (2, similarity, model);
        return
            new GenericUserBasedRecommender (model, neighborhood, similarity);
    }
};

IRStatistics stats = evaluator.evaluate(
    recommenderBuilder, null, model, null, 2,
    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD,
    1.0);                                     ← 评估推荐2个结果时的
                                                查准率和查全率

System.out.println(stats.getPrecision());
System.out.println(stats.getRecall());

```

如果不调用`RandomUtils.useTestSeed()`，你会看到完全不同的结果，因为训练数据和测试数据是随机选择的，而且这里选用的数据集也非常小。但是加入这个调用之后，结果就应该为：

0.75
1.0

“Precision at 2”（推荐2个结果时的查准率）为0.75；平均有3/4的推荐结果是好的。“Recall at 2”（推荐2个结果时的查全率）为1.0；所有好的推荐都包含在这些推荐结果中。^①

但是，到底什么才是好的推荐呢？框架要负责作出决定，而没有人给它一个定义。直观上看，在测试集中最受欢迎的物品为好的推荐，其他则不是。

代码清单2-5 在测试数据集中用户5的偏好值

5,101,4.0
5,102,3.0
5,103,2.0
5,104,4.0
5,105,3.5
5,106,4.0

重新看一下该样本数据集中的用户5。我们把物品101、102和103的偏好值分离出来作为测试数据。它们的偏好值分别为4.0、3.0和2.0。当这些值不在训练数据集中时，推荐引擎应该先推荐101，再是102，最后是103，因为这是用户5对这些物品的偏好顺序。但是推荐103会不会是个好主意呢？它位于列表的末尾，用户5不会很喜欢它。而用户5对书102的喜爱也只是一般而已。书101看起来不错，因为它的偏好值远远超过平均值。或许101是一个好的推荐，102和103也不错，但算不上是好的推荐。

但这是`RecommenderEvaluator`的思维方式。当没有明确的阈值可将推荐分出好坏时，框架会为每个用户取一个阈值，它等于该用户的平均偏好值 μ ，加上一个标准方差 σ : ^②

^① 查准率和查全率均为对所有用户的推荐分别评估后取的平均值。——译者注

^② 注意这里讨论的是`RecommenderIRStatsEvaluator`中阈值的设定，`RecommenderEvaluator`直接将估计值与实际值相比较，故不需使用阈值。——译者注

$$\text{阈值} = \mu + \sigma$$

即使你已经忘记了统计数据，也没关系。确定阈值所用的物品偏好值不是略高于平均值(μ)，而是比平均值高出很多(σ)。在现实场景下，这意味着大约有16%的物品最受欢迎，它们可以被视为好的推荐并反馈给用户。该方法所用的其他参数和以前类似，它们在项目的Javadoc中有完整的文档说明。

2.4.2 查准率和查全率的问题

 No. 2 在推荐程序中，查准率和查全率测试的有效性完全依赖于怎样定义“好的推荐”。在前一节中，阈值要么是特别指定的，要么是由框架定义的。阈值选择不当会损害到对推荐结果评分的有效性。

但是，这些测试还有一个更细节的问题。这里，它们必然是从那些用户已经表达过一些偏好的物品中挑选一组好的推荐结果。但是，最好的推荐结果并不一定在那些用户已知的物品中！

试想为一个用户运行这个测试，这个用户肯定喜欢小众的法国非主流电影*My Brother the Armoire*。平心而论，这是给用户的一个非常棒的推荐，但这个用户从来没有听说过这部电影。假如推荐程序推荐这部电影，会被认为是推荐错误；测试框架仅会从用户已有的偏好集合中选择好的推荐。

如果偏好是布尔型，不包含偏好值，那么事情就更复杂了。这时，甚至没有相对偏好的概念可用于选出包含好物品的数据子集。该测试可做的最好选择就是随机选择一些受欢迎的物品作为好的推荐。

这个测试仍然有些用处。用户偏好的物品可以很好地代表对用户的最佳推荐，不过它们绝非完美的选择。在布尔型偏好数据的案例中，只能做查准-查全测试(precision-recall test)。理解这个测试在该场景下的局限是必要的。

2.5 评估 GroupLens 数据集

有这些工具在手，我们不仅可以评估推荐引擎的速度，还可以评估其质量。虽然几章之后才会讨论有关大规模真实数据的示例，但现在我们已经可以快速评估一个小数据集的性能了。

2.5.1 提取推荐程序的输入

GroupLens (<http://grouplens.org/>) 是一个研究项目，提供多个大小不同的数据集，每个都来自真实用户对电影的评分。它是几个可用的大规模真实数据集中的一员，本书稍后还会为你介绍更多的数据集。

在GroupLens网站上，找到并下载“100K data set”，当前其地址为<http://www.grouplens.org/node/73>。将下载的文件解压，在其中找到名为ua.base的文件。这是一个以制表符(tab)分隔的文件，包含用户ID、物品ID、评分(偏好值)，以及一些附加信息。

这个文件的字段用制表符分隔，而不是逗号，结尾还包含一个额外的信息字段。它可用吗？是的，这个文件可用于FileDataModel。回到代码清单2-3的代码，创建一个Recommender-Evaluator，然后把ua.base的位置传递给它，而不再是传递一个小数据文件。再次运行。这次，

评估会花上几分钟，因为现在是基于100 000个偏好值，而不是少数几个。

最终，你会得到一个大约为0.9的值。这不算坏，但放在1到5的区间内，这个值偏离了将近1个点，看起来不算太好。对这类数据，也许我们正在使用的这个特定的Recommender实现并不是最优的？

2.5.2 体验其他推荐程序

让我们试着在这个数据集上运行一个slope-one推荐程序，这是一个还会在第4章中出现的简单算法。如下所示，用`org.apache.mahout.cf.taste.impl.recommender.slopeone.SlopeOneRecommender`替代`RecommenderBuilder`即可。

代码清单2-6 改变评估程序后运行SlopeOneRecommender

```
RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
    @Override
    public Recommender buildRecommender(DataModel model)
        throws TasteException {
        return new SlopeOneRecommender(model);
    }
};
```

再次运行该评估。你会发现它快了很多，而且生成的评估结果大约为0.748。正在向正确的方向前进。

这并不是说slope-one总是更好或更快。每个算法都有其特征与属性，无法预知它们在给定数据集上的行为。例如，虽然slope-one在运行时会很快算出推荐结果，但在运行之前却需要大量的时间来预先算出其内在的数据结构。因此，我们最初介绍的基于用户的推荐程序也许在其他数据集上会更快和更准确。第4章会探讨每种算法的相对优势。

这种区别彰显了在真实数据上做测试与评估的重要性，以及使用Mahout如何相对消除了一些麻烦。

2.6 小结

本章中，我们介绍了推荐引擎的思想。我们选用一个简单的Mahout Recommender，为之创建了一些小规模的输入数据、运行了一个简单计算，并解释了其结果。

接着我们评估了推荐引擎输出结果的质量，这是在后续章节中需要经常使用的。本章涵盖对Recommender所估计偏好的精度评估，以及传统的查准率和查全率度量标准在推荐中的应用。最终，我们尝试评估一个来自GroupLens的真实数据集，并观察如何借由评估在现实场景中探索对推荐引擎的改进。

在我们继续详解推荐引擎之前，还有一个重要的事情要做，即了解Mahout中推荐程序的另一个基本概念：数据表示。我们将在下一章讨论它。

本章内容

- Mahout如何表示推荐数据
- DataModel的实现和用法
- 无偏好值时的数据处理

推荐的质量很大程度上取决于数据的数量和质量。“种瓜得瓜，种豆得豆”，没有比用在这里更恰当的了。拥有高质量的数据当然是件好事，而且通常越多越好。

但是，推荐算法天生是数据密集型的，其计算涉及对大量信息的访问。因此，数据的数量和表示方式会很大程度上影响执行性能。智能地选择数据结构能够极大地改善性能，数据达到一定规模的时候，这并非小事。

本章探讨Mahout在表示和访问推荐程序的相关数据时所用的关键类。你会更好地理解为什么Mahout采用这样的方式来表示用户和物品及其相关的偏好，以达到高效和可扩展性。本章还会详细解析在Mahout中用于访问数据的关键抽象：DataModel。

最后，让我们来看看当用户和物品的数据没有评分或偏好值时的情况，即所谓的布尔偏好（Boolean preference），这时就需要做特殊的处理。

第一节介绍推荐数据的基本单元：用户对物品的偏好（user-item preference）。

3.1 偏好数据的表示

推荐引擎的输入是偏好数据（preference data）：什么人喜欢什么物品以及喜欢的程度。这意味着该输入就是一个用户ID、物品ID和偏好值的元组集合——这自然是一个大数据集。有时，偏好值会被忽略。

3.1.1 Preference对象

Preference是最基本的抽象，表示单个用户ID、物品ID和偏好值。一个对象代表一个用户对一个物品的偏好。Preference是一个接口，你最有可能使用的实现是GenericPreference。例如，下面一行代码所生成的表示形式意味着用户123对于物品456的偏好值为3.0：

```
new GenericPreference(123, 456, 3.0f)
```

那么一组 Preference该如何表示呢？如果你给出像 `Collection<Preference>` 或者 `Preference[]` 这类答案，虽然看似合理，但对于大多数 Mahout API 而言通常都是错误的。聚合（collection）和数组（Array）在表示大量 Preference 对象时会变得相当低效。如果你从未见识过 Java 中一个 Object 的开销，你一定会被吓到！

一个 GenericPreference 包含 20 字节的有用数据：一个 8 字节的用户 ID（Java `long`）、一个 8 字节的物品 ID（`long`）和一个 4 字节的偏好值（`float`）。而该对象的存在所需要的开销令人吃惊：28 字节！这个对象的表示形式包含一个 8 字节的对该对象的引用，以及由于 Object 开销和其他对齐问题所带来的另外 20 字节。于是 GenericPreference 对象仅由于引用的开销上就比实际多消耗了 1.4 倍的内存。

注意 实际的开销大小因 JVM 实现而不同；上述数据是针对苹果 Mac OS X 10.6 的 64 位 Java 6 虚拟机而言的。

该如何表示大量 Preference 对象呢？在推荐算法中，通常需要一个与某个用户或某个物品关联的所有偏好的聚合。在这种聚合里，所有 Preference 对象的用户 ID 或 物品 ID 都是一样的，这似乎是冗余的。

3.1.2 PreferenceArray 及其实现

看一下 PreferenceArray，这是一个接口，它的实现表示一个偏好的聚合，具有类似数组的 API。例如， GenericUserPreferenceArray 表示的是与某个用户关联的所有偏好。其内部包含一个单一用户 ID、一个物品 ID 数组，以及一个偏好值数组。在这个表示形式中，每个偏好的边界内存（marginal memory）仅需要 12 字节（一个数组有一个 8 字节的物品 ID 和一个 4 字节的偏好值）。与此对应，一个完整的 Preference 对象需要大约 48 字节。这种特殊的实现仅在内存上就节省了 4 倍空间，而且需要由垃圾回收器分配和检查的对象也少多了，因此性能也能获得一定的提升。比较图 3-1 和图 3-2 就能理解这种节省是如何达成的。

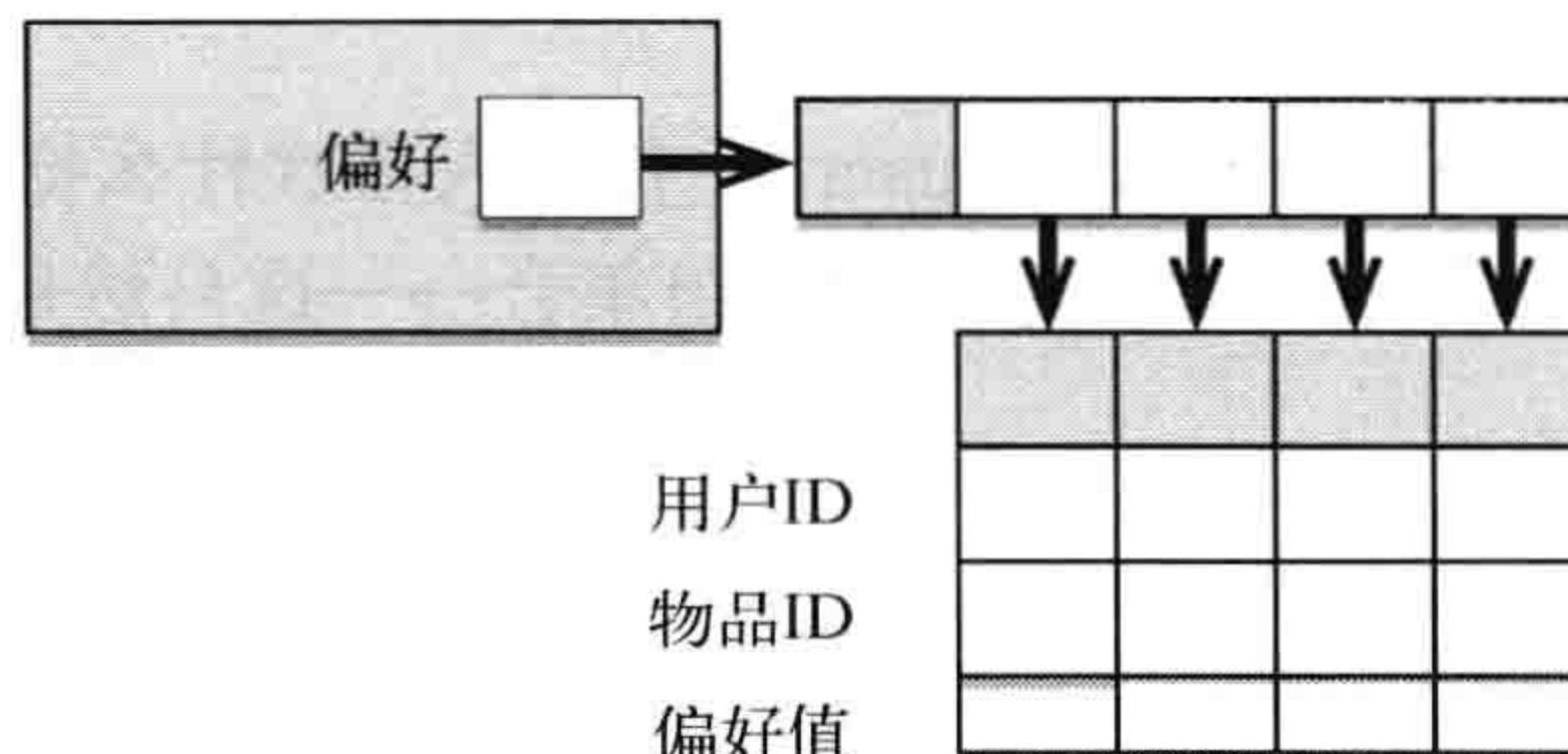


图 3-1 一种基于 Preference 对象数组的相对低效的偏好表示形式。灰色区域大体表示 Object 的开销。白色区域为数据，包括 Object 的引用

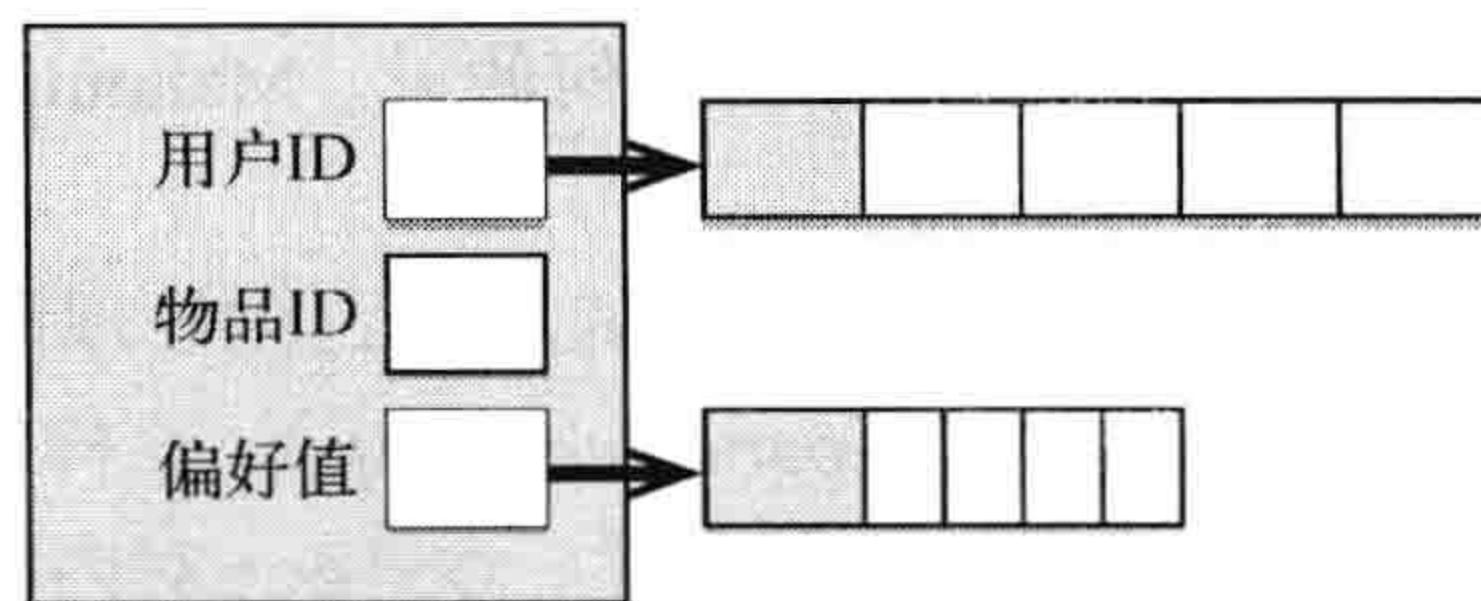


图3-2 基于GenericUserPreferenceArray的更高效的表现形式

3

下列代码显示了PreferenceArray典型的构建和访问方式。

代码清单3-1 设置PreferenceArray中的偏好值

```
PreferenceArray user1Prefs = new GenericUserPreferenceArray(2);

user1Prefs.setUserID(0, 1L);           ◀ 设置这些偏好的用户ID
user1Prefs.setItemID(0, 101L);
user1Prefs.setValue(0, 2.0f);

user1Prefs.setItemID(1, 102L);          | 表示这些偏好
user1Prefs.setValue(1, 3.0f);
Preference pref = user1Prefs.get(1);    ◀ 提取物品102的
                                         | Preference
```

同样，存在一个称为GenericItemPreferenceArray的实现，它封装了所有与某一物品相關的偏好，而不是关联到某个用户。它的用途与用法完全类似。

3.1.3 改善聚合的性能

你可能会想：“太棒了！Mahout已经创造了一个Java对象的数组。”哦，先别急，因为还有惊喜。你还记得我们曾提到过规模的重要性吗？希望你已经明白使用这种技术将要面对的数据大得非同寻常，而这可能会带来出乎意料的后果。

PreferenceArray及其实现降低了对内存的需求，即便引入复杂性也是值得的。将内存需求砍掉3/4并不只是节省了几兆字节——在一定规模下这会节省出几十GB的内存容量。这也许就是你的现有硬件能否容纳下这些数据的区别。也许这意味着你是否需要花费许多钱来购买更多RAM，或者一个新的64位系统。这是一个看似很小，却很实在的节省。

3.1.4 FastByIDMap和FastIDSet

你一定不会吃惊，Mahout的推荐程序中大量使用了Map和Set这些典型的数据结构，但它们用的并不是通常的Java集合（collection）的实现，如TreeSet和HashMap。相反，通览全部的实现与API，你会找到FastMap、FastByIDMap和FastIDSet。它们类似于Map和Set，但做了特殊定制，仅为满足Mahout中推荐程序的需要。它们降低了对内存的占用，而不是去显著地改善性能。

不能把它们当做是对Java Collections框架的批评。相反，集合因为良好的设计，可以有效地

适用于很多场景。只是，它们无法对使用方式作出任何假设。Mahout的需求则更具针对性，从而可以对用途作出更强的设定。主要区别如下。

- 与HashMap类似，FastByIDMap是基于散列的。但它在处理散列冲突时使用的是线性探测（linear probing），而非分离链接（separate chaining）。这样便不必为每个条目（entry）都增加一个额外的Map.Entry对象；如前所述，Object对内存的消耗是惊人的。
- 在Mahout推荐程序中键（key）和成员（member）通常采用原始类型long，而非Object。使用long型的键可以节约内存并提升性能。
- Set实现的内部没有使用Map。
- FastByIDMap可以作为高速缓存，因为它有一个最大空间的概念；超过这个大小时，若要新加入条目则会把不常用的移走。

存储上的差异是非常明显的：FastIDSet平均每个成员需要大约14字节，而HashSet需要84字节。FastByIDMap每个条目需要大约28字节，而HashMap每个条目需要大约84字节。这说明当能够在用途上作出更强假设时，就有可能进行大幅的优化——这里主要是在内存需求上。考虑到推荐系统所处理的数据量，这些定制化的实现并非自卖自夸。

那么，这些精心设计的类被用在哪里了呢？

3.2 内存级 DataModel

在Mahout中使用DataModel这种抽象机制对推荐程序的输入数据进行封装，而DataModel的实现为各类推荐算法提供了对数据的高效访问。例如，DataModel可以提供输入数据中所有用户ID的计数或列表、提供与某个物品相关的所有偏好，或给出所有对一组物品ID表达过偏好的用户的个数。

本节仅关注一些要点；更多关于DataModel的内容参见在线的Javadoc文档（<https://builds.apache.org/job/Mahout-Quality/javadoc/>）。

3.2.1 GenericDataModel

内存级（in-memory）实现GenericDataModel是现有DataModel实现中最简单的。它适用于通过程序在内存中构造数据的表示形式，而不是基于来自外部的数据源，如文件或关系数据库。它简单地将偏好作为输入，采用FastByIDMap的形式，将用户ID映射到这些用户的数据所在的PreferenceArray上，如下所示。

代码清单3-2 利用GenericDataModel在程序中定义输入数据

```
FastByIDMap<PreferenceArray> preferences =
    new FastByIDMap<PreferenceArray>();
PreferenceArray prefsForUser1 = new GenericUserPreferenceArray(10);

prefsForUser1.setUserID(0, 1L);
prefsForUser1.setItemID(0, 101L);
prefsForUser1.setValue(0, 3.0f);
```

| 增加10个偏好中的第1个

```

prefsForUser1.setItemID(1, 102L);
prefsForUser1.setValue(1, 4.5f);
... (8 more)
preferences.put(1L, prefsForUser1);           ← 在输入中附上用户1的偏好

DataModel model = new GenericDataModel(preferences);

```

3

GenericDataModel使用了多少内存呢？被存储的偏好的个数决定了对内存的需求。根据一些经验性的测试得知，每个偏好大约消耗28字节的Java堆空间。这包括所有数据以及其他支持性的数据结构，如索引。如果你乐意，可以尝试：加载GenericDataModel，调用几次System.gc()，再比较Runtime.totalMemory()和Runtime.freeMemory()的结果。这种比较非常粗略，但会对数据所消耗的内存有一个合理的估计。

3.2.2 基于文件的数据

你通常不会直接使用GenericDataModel，而是借助于FileDataModel，后者从文件中读取数据，并将所得到的偏好数据存储到内存中，即存储到GenericDataModel中。

几乎任何正常的文件都可以用，比如第2章中采用的CSV（Comma-Separated Value，逗号分隔值）格式的那种文件。每行包含一个数据：用户ID、物品ID和偏好值。采用制表符分隔也是可以的。用zip或gzip压缩的文件同样可以，只要名字分别以.zip或.gz结尾。将数据以压缩格式存储是一个很好的主意，因为它会非常大且很容易压缩。

3.2.3 可刷新组件

虽然我们谈论的是加载数据，但仍有必要讲一讲重加载数据（reloading data），以及Refreshable接口，即在Mahout推荐程序相关类中所实现的几个组件。它只公开了一个方法refresh(Collection<Refreshable>)。该方法简单地请求组件在最新的输入数据上进行：重加载（reload）、重算（recompute）并刷新（refresh）自身状态，并事先让它的依赖（dependency）也这样做。

例如，Recommender在重新计算其内部数据索引时，多会在它所依赖的DataModel上调用refresh()。循环依赖（cyclical dependency）和共享依赖（shared dependency）被管理得很好，如图3-3所示（基于图2-2）。

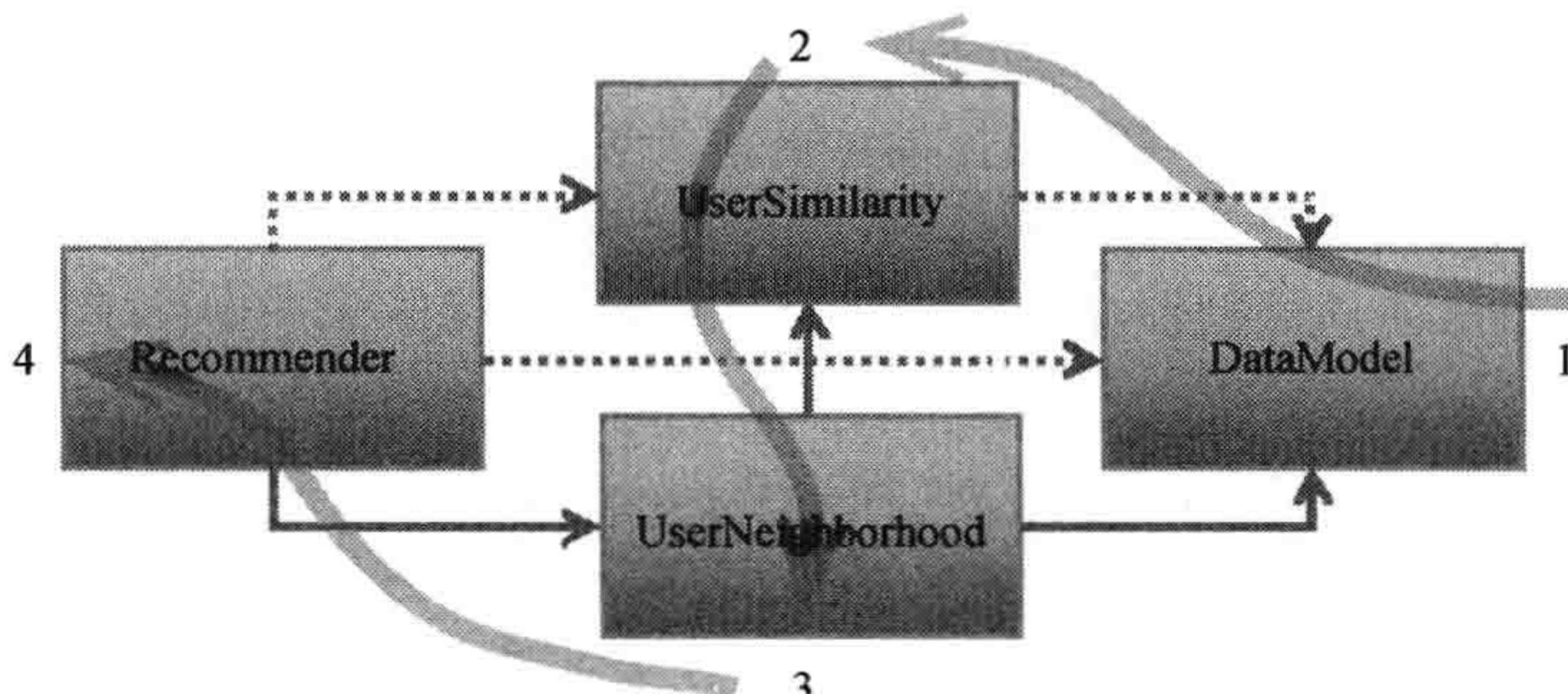


图3-3 一个简单的基于用户的推荐系统，箭头所示为组件之间刷新数据结构的顺序

注意，`FileDataModel`仅会在被请求的时候才从底层文件重新加载数据。出于性能考虑，它不会自动检测更新或定期重新加载文件的内容。这是`refresh()`方法该做的事。你大概不会只想刷新`FileDataModel`，而是希望所有依赖该数据的对象都被刷新。实际上，这就是为什么你总要在如下`Recommender`中明确调用`refresh()`的原因。

代码清单3-3 触发一个推荐系统的刷新

```
DataManager dataModel = new FileDataModel(new File("input.csv"));
Recommender recommender = new SlopeOneRecommender(dataModel);
...
recommender.refresh(null);           ← 刷新DataManager, 然后刷新自己
```

因为规模是贯穿本书的主题，我们应该强调`FileDataModel`另一个有用的特性：更新文件。数据总在改变，通常改变的数据只是所有数据中很小的一部分，甚至可能仅仅是十亿个数据中的几个点。只为了几个数据的更新对一个包含十亿个偏好的数据做一个全新的复制，这是非常低效的。

3.2.4 更新文件

`FileDataModel`支持更新文件。它们就是在读取主数据文件之后额外生成的数据文件，并可以覆盖任何以前读取的数据。通过添加来形成新的偏好，还可以更新现有偏好。通过设一个偏好值为空的字符串来实现删除。

例如，考虑如下的更新文件：

```
1,108,3.0
1,103,
```

就是说，“更新（或生成）用户1对物品108的偏好，并将值设为3.0”，以及“删除用户1对物品103的偏好”。

这些更新文件必须和主数据文件在同一个目录下，且文件名的前缀（第一个域）相同。例如，如果主数据文件为`foo.txt.gz`，更新文件可为`foo.1.txt.gz`和`foo.2.txt.gz`。它们可以是压缩文件。

3.2.5 基于数据库的数据

有时数据就是太大了，无法放入内存。一旦数据集有几千万个偏好，内存需求会增长到几GB，在某些场景下可能无法支持这么大的内存容量。

偏好数据是有可能存储到一个关系数据库中并进行访问的，而Mahout支持这样做。在Mahout推荐程序中，一些类的实现出于性能考虑会把计算下放到数据库中。

要知道，当推荐引擎所用的数据来自数据库时，它的运行要比使用内存级的数据表示慢很多倍。这并不是数据库的错；通过合理地调优和配置，一个现代数据库可以用于极其高效地对信息进行索引和检索，但检索、整理（marshalling）、序列化（serializing）、传输和反序列化（deserializing）结果集的开销仍远大于从优化的内存级数据结构中读取数据的开销。由于推荐算法是数据密集型的，这种开销会快速积累。不过，当没有其他选择时，数据库仍是理想选择，或者虽然所用数据集不太大，但为了集成还需要重用一个现有的数据表，此时也应选择数据库。

3.2.6 JDBC和MySQL

偏好数据是通过JDBC访问的，使用了JDBCDataModel的实现。现在，JDBCDataModel的主要子类是为使用MySQL 5.x而写的：MySQLJDBCDataModel。它在MySQL的早期版本上也很好用，甚至可用于其他数据库，因为尽可能地使用了标准的ANSI SQL。变种的实现也不难，可以结合需求使用数据库所专有的语法和特性。

注意 在Mahout的开发版本中有一个专为PostgreSQL而做的JDBCDataModel的实现。还有一个GenericJDBCDataModel类，它允许你使用那些没有做专有实现的数据库中的数据。

默认情况下，这个实现假设所有的偏好数据位于一个名为taste_preferences的表中，其中用户ID的列为user_id，物品ID的列为item_id，偏好值的列为preference。其模式如表3-1^①所示。该表还可以包含一个名为timestamp的字段，它的类型应该兼容于Java的long型。

表3-1 MySQL中taste_preferences表默认的模式

user_id	item_id	preference
BIGINT NOT NULL INDEX	BIGINT NOT NULL INDEX PRIMARY KEY	FLOAT NOT NULL

3.2.7 通过JNDI进行配置

JDBCDataModel实现还假设包含这个表的数据库可以通过一个DataSource对象来访问，这个对象已经注册到名为jdbc/taste的JNDI中。

你也许会问：什么是JNDI^②？它的全称为Java Naming and Directory Interface，即Java命名与目录接口，它是J2EE（Java 2 Enterprise Edition）规范的核心。如果你正在一个Web应用中使用推荐引擎，并正在使用Tomcat或Resin这样的servlet容器，那么你很可能已经间接用到了JNDI。如果正通过容器（例如Tomcat的server.xml文件）配置数据库，你会发现这个配置通常会被JNDI中的DataSource所引用。

你可以将数据库配置为jdbc/taste，其中包含JDBCDataModel会使用的细节。这里有Tomcat可用配置的一个片段。

① 在MySQL中创建该表的命令可以写为：CREATE TABLE taste_preferences (user_id BIGINT NOT NULL, item_id BIGINT NOT NULL, preference FLOAT NOT NULL, PRIMARY KEY (user_id, item_id), INDEX (user_id), INDEX (item_id))。——译者注

② JNDI避免了数据库和程序之间的紧耦合。当数据库相关参数发生变更时，仅需在JNDI中修改相关配置，而无须修改程序。——译者注

代码清单3-4 在Tomcat中配置一个JNDI DataSource

```
<Resource
    name="jdbc/taste"
    auth="Container"
    type="javax.sql.DataSource"
    username="user"
    password="password"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/mydatabase"/>
```

默认的名字（`jdbc/taste`）可以根据环境需要更换。你还可以像上面这样不去明确地为数据库和列命名。

3.2.8 利用程序进行配置

你也不必直接使用JNDI，而是将DataSource直接传递到MySQLJDBCDataModel的构造函数中。下面的代码清单显示了配置MySQLJDBCDataModel的一个完整示例，其中说明了如何使用MySQL Connector/J驱动（<http://www.mysql.com/products/connector/>），以及指定了表和列名的DataSource。

代码清单3-5 利用程序配置DataSource

```
MysqlDataSource dataSource = new MysqlDataSource ();
dataSource.setServerName("my_database_host");
dataSource.setUser("my_user");
dataSource.setPassword("my_password");
dataSource.setDatabaseName("my_database_name");
JDBCDataModel dataModel = new MySQLJDBCDataModel(
    dataSource, "my_prefs_table", "my_user_column",
    "my_item_column", "my_pref_value_column");
```

这就是将数据库中的数据用于推荐所需做的所有事情。

你现在已经得到了一个与所有推荐程序组件兼容的DataModel！但是正如MySQLJDBCDataModel的文档所说的，高效地推荐需要正确配置数据库与驱动。具体如下所述。

- 用户ID和物品ID列应为非空，而且必须被索引。
- 主键必须为用户ID和物品ID的组合。
- 列的数据类型根据Java中对应的long和float型来选择。在MySQL中，它们应为BIGINT和FLOAT。
- 注意调节缓冲区和查询高速缓存（query cache），见MySQLJDBCDataModel的Javadoc。
- 当使用MySQL的Connector/J驱动时，将驱动的参数（如cachePreparedStatements）设为true，细节同样见Javadoc。

上述讨论已经涵盖了使用Mahout推荐引擎框架中DataModel的基础。在这些实现中还有一个重要的变体需要讨论：如何表示偏好值缺失的数据。这听起来有些奇怪，因为偏好值似乎是推荐引擎所必须的输入数据。但有时，偏好值不存在或者忽略偏好值是有好处的。

3.3 无偏好值的处理

有时，输入推荐引擎的偏好没有值。也就是说，用户和物品是关联的，但是没有这种关联的强度描述。例如，一个新闻网站要根据用户以前浏览的新闻文章做推荐，但只知道一些用户和物品之间的关联，而没有更多的信息，因为用户通常不会去评价文章。用户在浏览文章之外甚至都很少会去做其他的事情。这时，我们仅能得知与用户关联的是哪篇文章，以及少量的其他信息。

在这里，我们别无选择；在输入中没有偏好值可以作为初始值。本章后续的技术和建议仍适用该场景。即便在输入中的确存在偏好值，有时忽略它们也会有好处。至少在有的时候，这样做没有坏处。

这并非要忘记用户和物品之间的关联，而是忽略其中的偏好强度。例如，当推荐一部新电影时，不是考虑你看过哪些电影以及你是如何评价它的，而只是简单地考虑你看过的是哪些电影。不是获取“用户1对电影103表达的偏好为4.5”，而是忘记4.5这个值，将“用户1和电影103有关联”这样的数据作为输入，这会很有用。图3-4说明了这种区别。

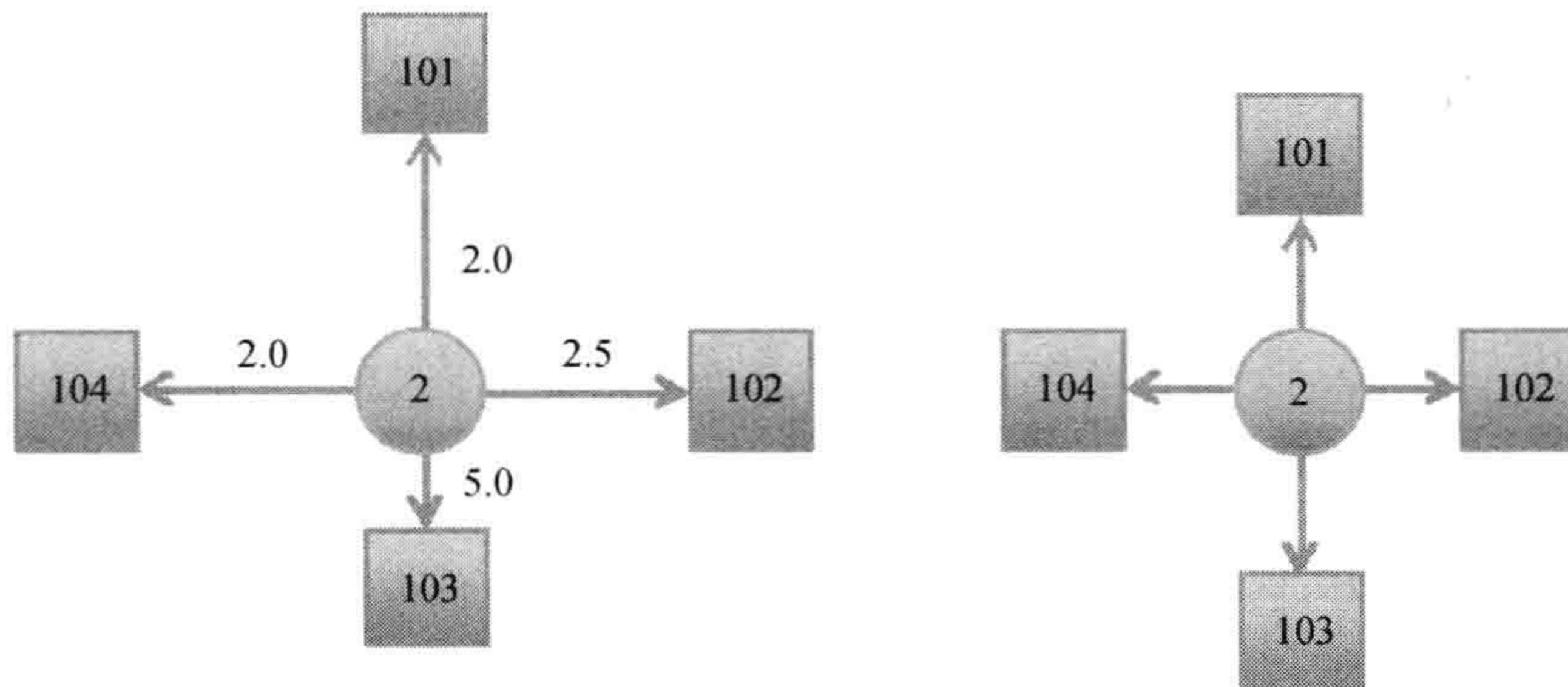


图3-4 用户和物品之间具有偏好值的关系（左图）和不具有偏好值的关系（右图）

由于缺少更好的术语来表达，在Mahout的语言里，这种没有偏好值的关联称为布尔型偏好（Boolean preference），因为一个关联只可能有两个值：存在或不存在。这并不意味着数据中的物品偏好是yes或no，而是会让用户-物品关联具有全部的三种可能状态：喜欢、不喜欢或无所谓。

3.3.1 何时忽略值

为什么要忽略偏好值？因为这么做在一定场景下是有好处的，此时喜欢或不喜欢一个物品相对而言都差不多，至少和根本没有关联相比是这样的。
No. 3

让我们举例说明。想象有这么一个人，他不喜欢古典作曲家Rachmaninoff的作品。事实上，他在自己的iTunes库中对Rachmaninoff的几个作品给出了1星或2星的评价。除了这些作品，世界上还有无数的音乐，其中一些是他从来没有听过的（就像挪威死亡金属乐，即Norwegian death

metal)。他甚至是因为足够了解Rachmaninoff而不喜欢他的作品，甚至在iTunes库的开头还留有几个Rachmaninoff的作品，这些都显示了他和这个作曲家的关联，甚至透漏出他对类似作品的一种偏好。与大千世界中他完全不知道的作品相比，这种关联是非常明显的。虽然他也许会给Rachmaninoff一个1星评价，而给Brahms一个5星，实际上这都传递了一些类似的信息：一种对古典音乐的兴趣。因此，忘记实际的评分反，认真思考这一事实，甚至可以给出更好的推荐。

你可能会反驳说这是用户的错误。难道他不会给Rachmaninoff一个4星？因为还有挪威死亡金属乐，而这可能才是他会给出1星评价的作品。也许如此，但这就是生活。输入常常是有问题的。你可能还会反驳说，虽然这对于从所有类型的音乐中做推荐是合理的，但忽略这些数据的话，在只推荐古典作曲家时可能使推荐效果变差。的确如此；但在一个领域中的好方案并不总是能移植到其他领域的。

3.3.2 无偏好值时的内存级表示

没有了偏好值会极大地简化偏好数据的表示，这会获得更优的性能并显著降低对内存的占用。如前所述，Mahout的Preference对象将偏好值存为4字节的Java float型。没有了偏好值，在内存中每个偏好能够节省4字节。实际上，重复前面的粗略测试可以看到，每个偏好的内存消耗平均减少了4字节，降为24字节。

这来自于对GenericDataModel孪生兄弟GenericBooleanPrefDataModel的测试。这是另一个内存级的DataModel实现，但其内部并不存储偏好值。它简单地将关联存为FastIDSet；例如，每个用户用1个，来代表与用户关联的所有物品ID。其中不包含偏好值。

因为GenericBooleanPrefDataModel也是一个DataModel，它有时可以代替GenericDataModel。DataModel的一些方法使用这个新的实现会更快，如getItemIDsForUser()，因为新的实现已经有现成的结果。有些则会变慢，如getPreferencesFromUser()，因为新的实现不使用PreferenceArray，必须实例化一个才能实现这个方法。

你也许想知道getPreferenceValue()会返回什么，因为这里并没有偏好值。它并不抛出UnsupportedOperationException，而会大概返回相同的假值：1.0。必须注意这一点，因为依赖于偏好值的组件仍会从该DataModel中获取一个值。这些偏好值是假值且不会改变，这会带来一些小问题。

让我们回到上一章的GroupLens示例。但代码改为使用GenericBooleanPrefDataModel，如代码清单3-6所示。

代码清单3-6 布尔型数据的生成与评估

```
DataManager model = new GenericBooleanPrefDataModel(
    GenericBooleanPrefDataModel.toDataMap(
        new FileDataManager(new File("ua.base"))));
RecommenderEvaluator evaluator =
    new AverageAbsoluteDifferenceRecommenderEvaluator();
RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
```

<div style="position: absolute; left: 50%; top: 1045px; width: 0; height: 0; border-left: 10px solid transparent; border-right: 10px solid transparent; border-top: 2

```

public Recommender buildRecommender(DataModel model)
    throws TasteException {
    UserSimilarity similarity =
        new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood =
        new NearestNUserNeighborhood(10, similarity, model);
    return
        new GenericUserBasedRecommender(model, neighborhood, similarity);
}
};

DataModelBuilder modelBuilder = new DataModelBuilder() {
    public DataModel buildDataModel(
        FastByIDMap<PreferenceArray> trainingData) {
        return new GenericBooleanPrefDataModel(
            GenericBooleanPrefDataModel.toDataMap(
                trainingData));
    }
};

double score = evaluator.evaluate(
    recommenderBuilder, modelBuilder, model, 0.9, 1.0);
System.out.println(score);

```

3 构造一个
GenericBooleanPrefDataModel

该示例的关键在于DataModelBuilder。你可以用它控制评估过程构造用于训练数据的DataModel。GenericBooleanPrefDataModel获取输入的方式略为不同——通过一组FastIDSet而非PreferenceArray——有一个toDataMap()方法可以方便地转换它们。

阅读下一节之前，试着运行这段代码——它不会成功地结束。

3.3.3 选择兼容的实现

你会发现运行代码清单3-6中的代码会导致一个来自PearsonCorrelationSimilarity构造函数的异常IllegalArgumentException。初看这会让人感到奇怪：GenericBooleanPrefDataModel不也是一个DataModel吗？而且它除了不存储明确的偏好值之外，几乎与GenericDataModel相同。

如果缺少偏好值，像EuclideanDistanceSimilarity这样的相似性度量会拒绝工作，因为其结果会是未定义的（undefined）或无意义的，从而导致无用的结果。如果两个数据集是相同数值的简单重复，它们之间的皮尔逊相关系数是未定义的。这里，DataModel假设所有偏好值均为1.0。类似地，计算对应于空间上同一个点的所有用户之间的欧氏距离（Euclidean distance，又称欧几里得距离），即这里的(1.0, 1.0, …, 1.0)是无意义的，因为所有的相似性均为1.0。

注意 皮尔逊相关系数是两个数据集的协方差与其标准差之间的比值。当所有数据为1时，两个值均为0，而目前Java在计算0/0的相关结果时一定会返回“not a number”。^①

^① 在PearsonCorrelationSimilarity中通过return Double.NaN;来实现。——译者注

这个例子具有普遍意义，它说明了即使组件会采取一系列的标准接口来获得交互性，也无法保证每个实现都彼此相容。为了解决这个现实问题，需要一个合适的相似性度量。`LogLikelihoodSimilarity`就是这样的一个实现，因为它并非基于实际的偏好值。（我们稍后会讨论相似性度量。）用它来替代`PearsonCorrelationSimilarity`，结果为0.0。这很棒，因为这意味着完美的预测结果。是不是好得过火了呢？

很遗憾，的确如此。这个结果是当每个偏好值为1时，估计偏好和实际偏好之间的平均差值。结果自然会等于0；这个测试是无效的，因为它只能输出0。

但是查准率和查全率的评估仍是有效的。让我们尝试在下面的代码清单中来实现它。

代码清单3-7 利用布尔型数据评估查准率和查全率

```
DataModel model = new GenericBooleanPrefDataModel(
    new FileDataModel(new File("ua.base")));

RecommenderIRStatsEvaluator evaluator =
    new GenericRecommenderIRStatsEvaluator();
RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
    @Override
    public Recommender buildRecommender(DataModel model) {
        UserSimilarity similarity = new LogLikelihoodSimilarity(model);
        UserNeighborhood neighborhood =
            new NearestNUserNeighborhood(10, similarity, model);
        return new GenericUserBasedRecommender(
            model, neighborhood, similarity);
    }
};

DataModelBuilder modelBuilder = new DataModelBuilder() {
    @Override
    public DataModel buildDataModel(
        FastByIDMap<PreferenceArray> trainingData) {
        return new GenericBooleanPrefDataModel(
            GenericBooleanPrefDataModel.toDataMap(trainingData));
    }
};

IRStatistics stats = evaluator.evaluate(
    recommenderBuilder, modelBuilder, model, null, 10,
    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD,
    1.0);
System.out.println(stats.getPrecision());
System.out.println(stats.getRecall());
```

所得查准率和查全率都是大约24.7%。这不算太好；回顾一下，这意味着返回的推荐中只有1/4是好的，而好的推荐中只有1/4在返回结果中。

这可追查出另一个问题：仍有一个地方隐藏着偏好值：`GenericUserBasedRecommender`。这个推荐程序仍基于其估计的偏好对推荐进行排序，但这些值均为1.0。因此顺序基本上是随机的。相反，你可以引入`GenericBooleanPrefUserBasedRecommender`（顾名思义）。这个变体可以让推荐形成更有意义的顺序。它为与其他类似用户相关的物品计算权重，用户相似度越高，这个权重越大。它并不生成加权平均。

尝试替代这个实现并重新运行代码。结果大约为22.9%——大致相同。这显然说明在这个数据上我们并未使用一个超高效的推荐系统。这里的目的并不是修复它，而仅仅为了审视如何在Mahout推荐程序中高效地部署布尔型数据。

还有其他DataModel的布尔型变种。FileDataModel会在输入数据不包含偏好值时（行只采用userID, itemID的形式），在内部自动使用GenericBooleanPrefDataModel。类似地，MySQLBooleanPrefDataModel适合在数据库表中无偏好值列时使用。否则它完全类似于MySQLJDBCDataModel。特别地，这种实现可以充分利用数据库中更多的快捷方式来提高性能。

最后，如果你想知道是否可以将布尔型和非布尔型数据在一个DataModel中混合使用，那么答案是：不行。一个解决办法是忽略偏好值，而将之视为布尔型数据。或者，如果你出于某种原因不希望抛弃它们，那些缺失的偏好值可以通过一些办法推测出来，即便只是简单地填充一个现有偏好值的平均数。

3.4 小结

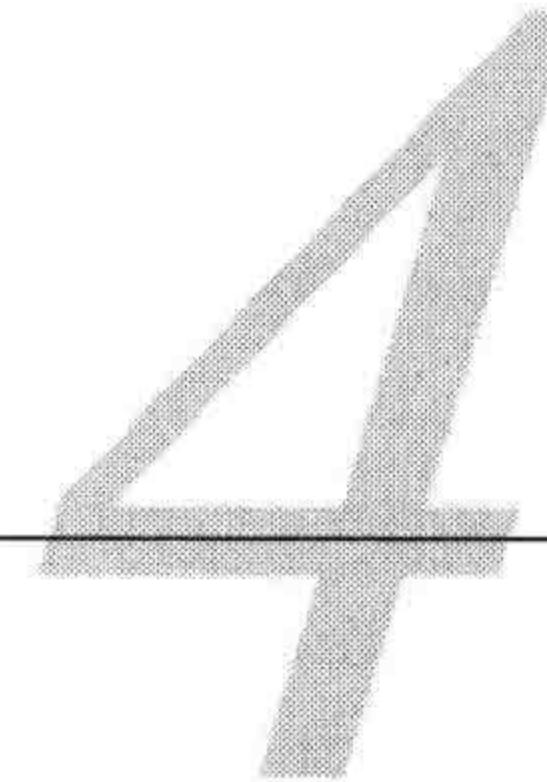
在本章，我们探讨了如何在Mahout的推荐程序中表示偏好数据。其中涉及Preference对象，还有特定的数组和类似聚合的实现，如PreferenceArray和FastByIDMap。它们主要用来降低内存占用。

我们研究了DataModel，它是推荐程序输入的一个整体抽象。GenericDataModel把数据放在内存中，就像FileDataModel从文件中读取输入数据之后所做的一样。JDBCDataModel和其他实现支持基于关系库表的数据；我们特别观察了与MySQL的集成。

最后，我们查看了当输入数据仅包含用户-物品关联，而不包含偏好值时所带来的变化。有时，这就是可用的所有数据，而这必然减少对存储的需求。我们看到这类数据不兼容于PearsonCorrelationSimilarity等标准组件。我们还考虑如何解决这类问题并得到一个基于布尔型输入数据的函数级推荐程序。

下一章，我们会继续检视数据表示的各种可能并窥探其可用的推荐程序实现。

进行推荐



本章内容

- 进一步了解基于用户的推荐程序
- 相似性度量
- 基于物品的推荐程序及其他

我们用了一章的篇幅来讨论如何评价推荐程序，以及推荐程序的输入数据形式，是时候来深入探究推荐程序本身了。我们就此开始切入正题。

前面的章节提到两类典型的推荐算法，它们均在Mahout中得到实现：基于用户的推荐程序和基于物品的推荐程序。实际上，在第2章我们已经遇到过一个基于用户的推荐程序。本章将仔细探究和讨论这些算法背后的理论及其在Mahout中的实现。

两种算法均依赖于两个事物（用户或物品）之间的相似性度量，或者说等同性定义。相似性的定义有多种，本章将详细介绍Mahout中可供选择的方法。它们包括基于皮尔逊相关系数（Pearson correlation）、对数似然值（log likelihood）、斯皮尔曼相关系数（Spearman correlation）、谷本系数（Tanimoto coefficient）等的实现。

最终，本章还会介绍Mahout中实现的其他几种推荐算法，包括slope-one、基于SVD（SVD-based）和基于聚类（clustering-based）的推荐算法。

4.1 理解基于用户的推荐

如果你看过前面所讲的推荐算法，就会知道它是一种基于用户的推荐算法。它是在这个领域早期研究中阐述的方法，Mahout自然会有它的实现。“基于用户”这个说法有些不准确，因为所有推荐算法都建立在与用户和物品相关的数据上。基于用户的推荐算法的典型特征是，它建立在用户间有某种相似性的基础之上。事实上，这种算法在日常生活中很常见。

4.1.1 推荐何时会出错

你是否曾收到CD这样的礼物？我（Sean）在小时候从好心的成年人那里收到过。其中一个成年人走进当地的音乐商店并询问店员，于是有了下面的场景：

成年人：我要为一个男孩儿买张CD。

店 员：好的，他喜欢什么？

成年人：呃，现在的孩子都喜欢些什么？

店 员：他喜欢什么音乐或乐队呢？

成年人：对我而言，那些都太吵了，呃，我不知道。

店 员：嗯，好吧……我猜大多数年轻人都会购买New 2 Town这个男生组合的专辑。

成年人：就它了！

4

结果可想而知。不用说，他们送的礼物并非是我想要的。遗憾的是，这种基于用户进行推荐导致出错的事情比比皆是。但这种直觉还是对的：因为年轻人在音乐上的品味通常比较接近，一个年轻人很可能会喜欢其他年轻人追捧的专辑。根据人群之间的相似度进行推荐是非常合理的。

当然，推荐一个女孩儿们追捧的乐队专辑给男孩儿们可能并不合适。这里的问题在于相似性度量不再有效。是的，一群年轻人会有相对一致的品味：相对于柴迪科舞 (zydeco) 和古典音乐，流行音乐可能更受欢迎。但是，这种相似性太脆弱而难以用：当把音乐作为推荐对象时，女孩儿们与男孩儿们并没有足够多的共性。

4.1.2 推荐何时是正确的

让我们回到前面的场景，来想象一个更好的情景：

成年人：我要为一个男孩儿买张CD。

店 员：他喜欢哪种音乐或者乐队？

成年人：我不知道，但他最好的朋友经常穿一件Bowling In Hades的T恤。

店 员：我知道，一个来自克利夫兰的非常流行的新金属乐队。我们正好有Bowling In Hades的最新专辑*Impossible Split: The Singles 1997–2000*。

这次好多了。这个推荐基于这样的假设，即两个好朋友在音乐上的品味会有些类似。相似性度量比较可靠时，结果就可能会更好。两个好朋友都喜欢Bowling In Hades的可能性比任意两个年轻人大得多。还有一些其他的途径能让结果更好：

成年人：我要为一个男孩儿买张CD。

店 员：他喜欢哪种音乐或者乐队？

成年人：“音乐？”，哈，很好，我从他卧室墙上的海报里抄下了乐队名。The Skulks、Rock Mobster、Wild Scallions……你这里有吗？

店 员：我看看。这些专辑我的孩子也有一些。他总是在不停地谈论一些Diabolical Florist的新专辑，那么也许……

现在相似度的推断直接来自于对音乐的品味。因为其中所提及的两个孩子都喜欢一些相同的乐队，有理由相信他们都会喜欢对方的其他收藏。这比基于他们是好朋友来猜测他们的品味更为可靠。这种思路通过观察年轻人对音乐的品味来推断他们之间的相似度。这是基于用户的推荐系统最基本的逻辑。

4.2 探索基于用户的推荐程序

如果那两个人继续谈下去，可能还会得到更好的推测。为什么只根据一个孩子的音乐收藏来挑选礼物呢？何不多考虑几个类似的孩子？他们会留意哪些孩子更为相似（那些海报、T恤和散放在唱片机上的CD大多相同的），还会观察那些最相似的孩子都关注什么乐队，并据此来选择最合适的礼物。（然后，他们也许会成为Mahout的用户！）

4.2.1 算法

基于用户的推荐算法就来自这种直觉。下面是一个为用户（记为 u ）进行推荐的过程：

```
for (用户u尚未表达偏好的) 每个物品i
    for (对i有偏好的) 每个其他用户v
        计算u和v之间的相似度s
        按权重为s将v对i的偏好并入平均值
return值最高的物品 (按加权平均排序)
```

外层循环简单地把每个已知物品（用户未对其表达过偏好的）作为候选的推荐项。内层循环逐个查看对候选物品做过评价的其他用户，并记下他们对该物品的偏好值。最终，将这些值的加权平均作为目标用户对该物品偏好值的预测。每个偏好值的权重取决于该用户与目标用户之间的相似度。与目标用户越相似，他的偏好值所占权重越大。

但是，每个物品都检查实在是太慢了。实际应用中，通常会先计算出一个最相似用户的邻域，然后仅考虑这些用户评价过的物品：

```
for 每个其他用户w
    计算用户u和用户w的相似度s
    按相似度排序后，将位置靠前的用户作为邻域n
    for (n中用户有偏好，而u中用户无偏好的) 每个物品i
        for (n中用户对i有偏好的) 每个其他用户v
            计算用户u和用户v的相似度s
            按权重s将v对i的偏好并入平均值
```

这一过程与前面的主要区别在于首先确定相似的用户，再考虑这些最相似用户对什么物品感兴趣。这些物品就成为推荐的候选项。后续的过程是一样的。这就是标准的基于用户的推荐算法，也是它在Mahout中的实现方式。

4.2.2 基于**GenericUserBasedRecommender**实现算法

本书最早的推荐程序示例展示了Mahout中一个实际的基于用户的推荐程序（代码清单2-2）。现在我们回顾一下它的构成，并对它的性能进行评估。

代码清单4-1 回顾一个简单的基于用户的推荐系统

```
DataModel model = new FileDataModel(new File("intro.csv"));
UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
```

```
UserNeighborhood neighborhood =  
    new NearestNUserNeighborhood(2, similarity, model);  
Recommender recommender =  
    new GenericUserBasedRecommender(model, neighborhood, similarity);
```

UserSimilarity封装了用户间相似性的概念，而UserNeighborhood封装了最相似用户组的概念。它们是标准的基于用户推荐算法的必要组件。

相似性的定义不是唯一的——前面选择CD的对话反映了在现实生活中人们关于相似性的几种看法。同样，最近邻用户也有多种不同的定义：最相似的5个，还是20个，还是所有相似度大于某一阈值的用户？为帮助理解这些选项，想象一下，假设你正在列一个婚礼客人的清单。你想邀请最亲密的朋友和家庭成员出席，但是你的朋友和家人远远超过预算允许的人数。为了决定邀请谁以及不邀请谁，你是不是会先定一个人数（比如50人），然后来选择最亲近的50个朋友或家人？50是一个合适的数字吗？40或100怎么样？或者，你会邀请每一个你认为很亲近的人吗？你会仅仅邀请真正的好朋友吗？谁会让你的婚礼派对非常成功？选择用户邻域与此类似。

引入新的相似性度量，结果就会发生显著变化。由此可知，提供推荐的方式是多种多样的——而这还只是调整了方法的一个侧面。Mahout是由多个组件混搭而成的，而非单一的推荐引擎，其各个组件的组合可以定制，从而针对特定应用提供理想的推荐。通常包括如下组件：

- 数据模型，由DataModel实现；
- 用户间的相似性度量，由UserSimilarity实现；
- 用户邻域的定义，由UserNeighborhood实现；
- 推荐引擎，由一个Recommender实现（此处为GenericUserBasedRecommender）。

要想推荐得更好更快，就必然需要经历一个漫长的试验和调优过程。

4.2.3 尝试GroupLens数据集

让我们回到GroupLens数据集，并将所用数据增加100倍。到<http://grouplens.org>下载包含1000万个评分的MovieLens数据集，目前它可以从地址<http://www.grouplens.org/node/73>获得。在本地解压后找到其中的ratings.dat文件。

出于某种原因，该数据的格式有别于之前的100 000评分数据集。其中ua.base文件可直接用于FileDataModel，但该数据集的ratings.dat文件则不能。简单的做法是使用标准的命令行文本处理工具将其转换为逗号分隔的形式，通常这也是最好的办法。专门编写代码来转换文件格式，或使用定制的DataModel，这不仅烦琐而且容易出错。

幸运的是，针对这个特例^①还有一个更简单的办法。Mahout的示例模块（examples）包含了一个定制的GroupLensDataModel实现，它扩展了FileDataModel以读取这个文件。你需要确保这个代码在IDE项目的examples/src/java/main目录下。然后，如代码清单4-2所示的内容替换FileDataModel。

^① 仅针对Gouplens数据集。——译者注

代码清单4-2 更新代码清单4-1来使用为GroupLens定制的DataModel

```

DataModel model = new GroupLensDataModel(new File("ratings.dat"));
UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
UserNeighborhood neighborhood =
    new NearestNUserNeighborhood(100, similarity, model);
Recommender recommender =
    new GenericUserBasedRecommender(model, neighborhood, similarity);
LoadEvaluator.runLoad(recommender);

```

运行这段代码，首先遇到的问题可能是`OutOfMemoryError`。在这里我们第一次碰到了规模问题。默认情况下，Java不会把堆（heap）大小设得过大。而这里，必须增加Java可用的堆空间。

这是一个探讨如何调节JVM来改善性能的好机会。可参考附录A更深入地了解JVM调优。

4.2.4 探究用户邻域

下面我们来评估推荐程序的精度。代码清单4-3再次给出了评估代码示例；在此之后，我们会认为你已经对它有了充分的了解，并可以独立构造和进行评估。

现在，我们尝试配置并调整该邻域的实现，如下面的代码清单所示。记住，这里使用的数据同样多出了100倍。

代码清单4-3 对这个简单的推荐引擎进行评估

```

DataModel model = new GroupLensDataModel (new File("ratings.dat"));
RecommenderEvaluator evaluator =
    new AverageAbsoluteDifferenceRecommenderEvaluator ();
RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
    @Override
    public Recommender buildRecommender(
        DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood =
            new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(
            model, neighborhood, similarity);
    }
};
double score = evaluator.evaluate(
    recommenderBuilder, null, model, 0.95, 0.05);
System.out.println(score);

```

注意，`evaluate()`的最后一个参数是0.05。这意味着仅有5%的数据用于评估。这纯粹是为了方便；评估是一个耗时的过程，使用全部数据会花上几个小时。为了快速评估变化，比较简便的做法是减小这个值。但是使用的数据太少可能会影响到评估结果的精度。参数0.95就是说使用95%的数据来构建要评估的模型，然后使用余下的5%来做测试。

代码运行所得到的结果可能会有出入，但约为0.89。

4.2.5 固定大小的邻域

此时，代码清单4-3中代码所给出的推荐来自于100个最相似用户构成的邻域（`NearestNUserNeighborhood`被设为邻域大小100）。推荐所依赖的最相似用户为100个，这个选择是随意的。如果选择10个会怎么样？推荐所依赖的相似用户虽然少了，但也会排除一些相似度较低的用户。包含3个最相似用户的邻域如图4-1所示。

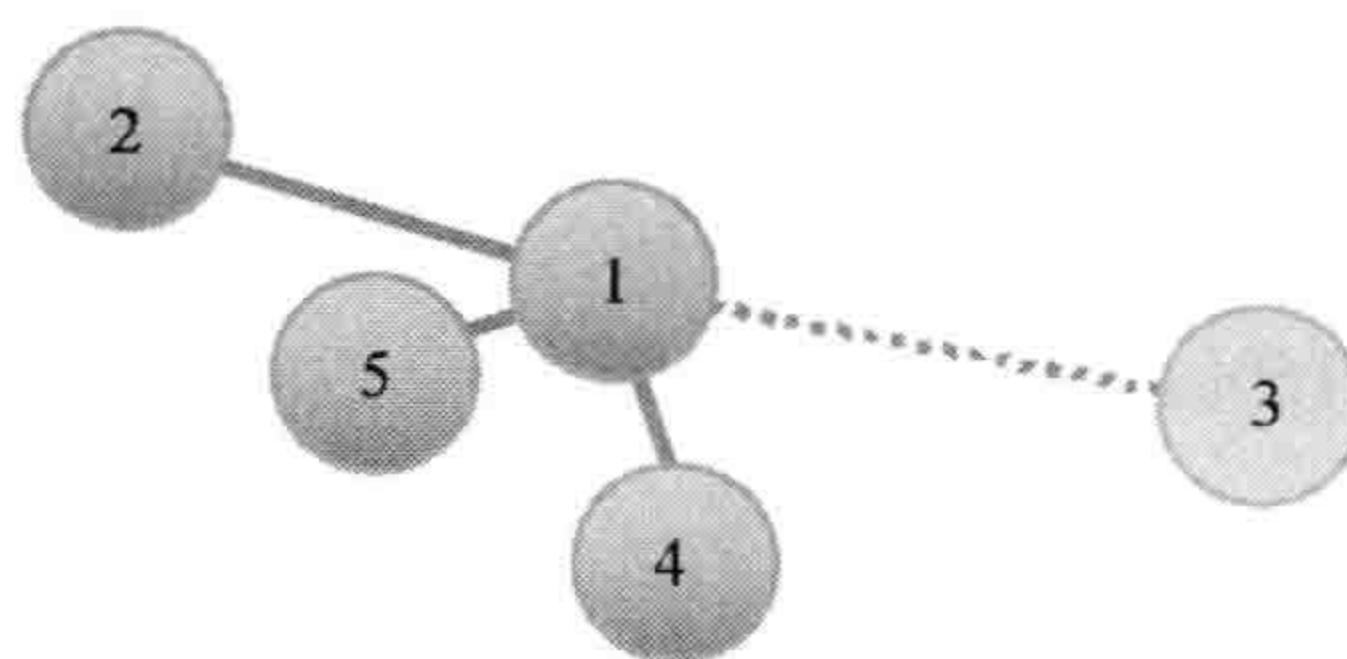


图4-1 通过确定最相似用户的数量来定义用户的邻域。这里，距离表示相似度：越远则越不相似。用户1的邻域由3个最相似的用户组成：5、4和2

尝试用10来替代100。推荐的评估结果，即估计值与实际偏好值的平均差异，为0.98左右。考虑到这一评估值越大越不好，这意味着选错方向了。最可能的解释是10个用户太少了。很可能后面的用户会有价值，如最相似的第11个、第12个用户等。他们不仅仍有很大的相似度，而且可能会关联到前10个用户没有涉及的一些物品。

尝试500个用户的邻域；结果降为0.75，这个结果自然较优。你可以多试一些值来为这个数据集找到最佳选项，但事实上并不存在一个万能的值；在真实数据上做一些试验对推荐程序的调优来说是很必要的。

4.2.6 基于阈值的邻域

假如不想用n个最相似用户构建邻域，那么如何直接选择那些很类似的用户并忽略其他人呢？你可以确定一个相似度阈值，并选择所有相似度超过这个阈值的用户。图4-2展示了一个基于阈值的用户邻域定义，你可将其与图4-1中的固定大小的邻域相对照。

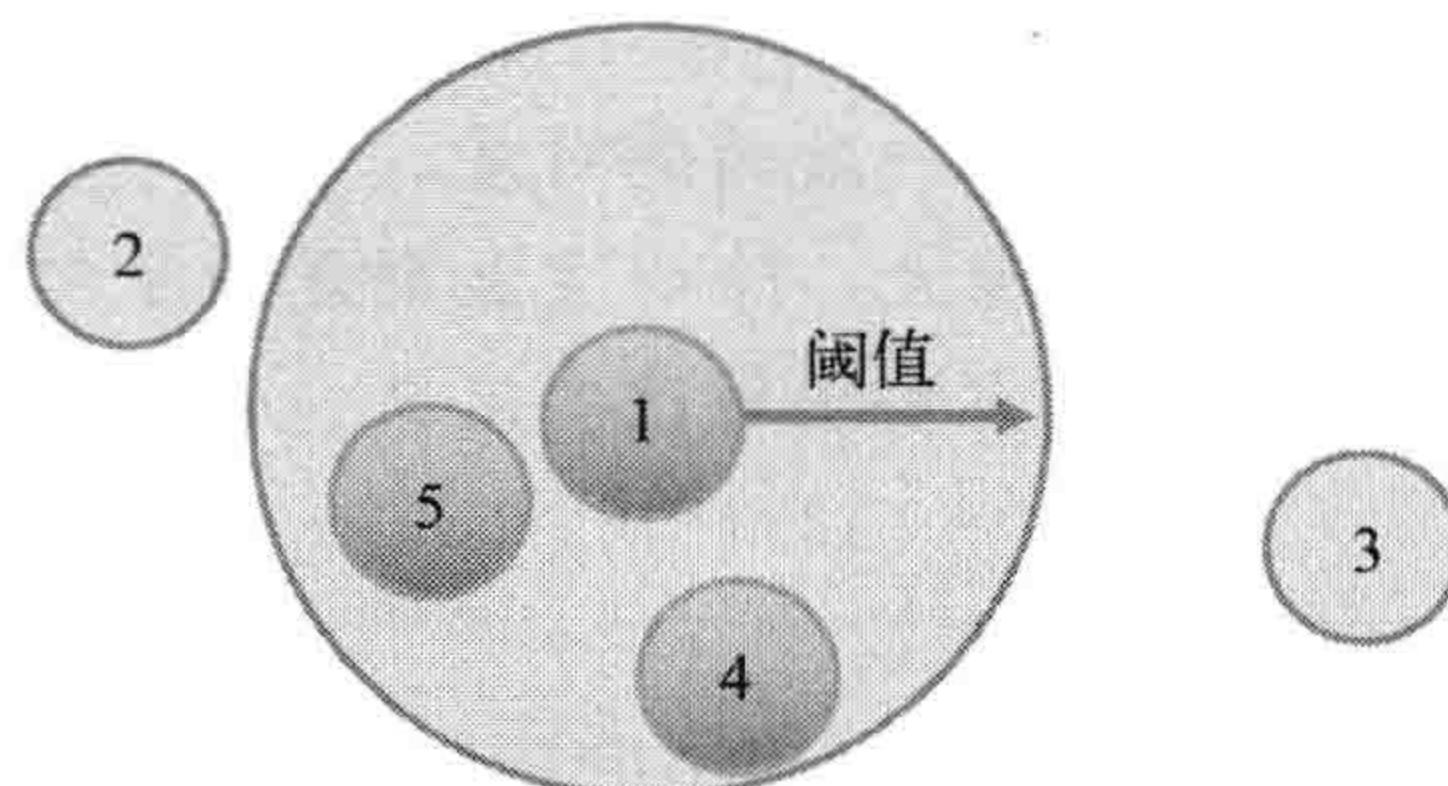


图4-2 通过一个相似度阈值来定义最相似用户的邻域

阈值应该设在-1和1之间，因为所有的相似性度量返回的相似度值都在该区间内。目前，我们的示例使用标准的皮尔逊相关系数作为相似性的度量标准。熟悉这种相关方法的读者应该知道0.7及以上的值意味着高度相关，可作为非常相似的一个合理定义。让我们改用ThresholdUserNeighborhood。简单地修改一行来实例化ThresholdUserNeighborhood：

```
new ThresholdUserNeighborhood(0.7, similarity, model)
```

现在评估程序给推荐程序的评分为0.84。如果更严格的限定阈值，使用0.9会如何？评分为0.92，即性能更差了；前面的解释同样适用于此处——此阈值限定的邻域包含的用户数太少。如果设为0.5呢？评分变好了，可降至0.78。后面的示例会将这种邻域的阈值设为0.5。

现在，你可能想在真实数据上尝试更多的阈值以得到一个最优结果，不过我们通过简单的试验已经将估计精度提高了大约15%。

4.3 探索相似性度量

基于用户的推荐程序的另一个重要部分是UserSimilarity实现。基于用户的推荐程序非常依赖这个组件。如果对用户之间的相似性缺乏可靠并有效的定义，这类推荐方法是没有意义的。这也适用于基于用户的推荐程序的“近亲”——基于物品的推荐程序，它同样依赖于相似性。这一组件十分重要，我们将用接近本章1/3的篇幅来讨论标准的相似性度量及其在Mahout中的实现。

4.3.1 基于皮尔逊相关系数的相似度

到目前为止，示例都使用了PearsonCorrelationSimilarity这一实现，它是一个基于皮尔逊相关系数的相似性度量标准。

皮尔逊相关系数是一个介于-1和1之间的数，它度量两个一一对应的数列之间的线性相关程度。也就是说，它表示两个数列中对应数字一起增大或一起减小的可能性。它度量数字一起按比例改变的倾向性，也就是说两个数列中的数字存在一个大致的线性关系。当该倾向性强时，相关值趋于1。当相关性很弱时，相关值趋于0。在负相关的情况下——一个序列的值高而另一个序列的值低——相关值趋于-1。

注意 对于熟悉统计学的读者而言，皮尔逊相关系数是两个序列协方差与二者方差乘积的比值。

协方差计算的是两个序列变化趋势一致的绝对量。当两个序列相对于各自的均值点向同一方向移动得越远，协方差值就越大。除以方差则是为了对这一变化进行归一化。使用Mahout中的皮尔逊相关系数并不需要理解这些定义，但如果你有兴趣，可以从网络上找到大量相关信息。

这一统计学中广泛使用的概念，同样可以用于度量用户之间的相似性。它度量两个用户针对同一物品的偏好值变化趋势的一致性——都偏高或都偏低。举个例子，再看看我们用过的第一个

样本数据文件intro.csv，如下所示。

代码清单4-4 一个简单的推荐系统输入文件

```
1,101,5.0
1,102,3.0
1,103,2.5

2,101,2.0
2,102,2.5
2,103,5.0
2,104,2.0

3,101,2.5
3,104,4.0
3,105,4.5
3,107,5.0

4,101,5.0
4,103,3.0
4,104,4.5
4,106,4.0

5,101,4.0
5,102,3.0
5,103,2.0
5,104,4.0
5,105,3.5
5,106,4.0
```

我们注意到用户1和5看起来相似，因为他们的偏好值好像在同一改变。对于物品101、102和103，他们大体达成一致：101最好，102略差，而103不理想。类似地，用户1和用户2则不那么相似。注意我们关于用户1的分析不包括物品104~106，因为用户1对它们的偏好是未知的。相似度的计算仅能在用户都表达了偏好的物品上进行。（在4.3.9节，我们将看到缺失偏好值的时候该怎么进行推测。）

皮尔逊相关系数可表达这些相似性，如表4-1所示。这里不再重复计算的细节；可参考在线资源，如<http://www.socialresearchmethods.net/kb/statcorr.php>，了解相关系数计算的说明。

表4-1 在用户1和其他用户之间基于3个共有物品的皮尔逊相关系数

	物品101	物品102	物品103	与用户1的相关性
用户1	5.0	3.0	2.5	1.000
用户2	2.0	2.5	5.0	-0.764
用户3	2.5	—	—	—
用户4	5.0	—	3.0	1.000
用户5	4.0	3.0	2.0	0.945

* 用户与其自身的皮尔逊相关系数总是1.0。

4.3.2 皮尔逊相关系数存在的问题

尽管结果很直观，但某些情况下皮尔逊相关系数在推荐引擎中的表现会有点奇怪。

首先，它没有考虑两个用户同时给出偏好值的物品数目，在推荐引擎中这可能不太可靠。例如，两个看过200部相同电影的用户，即便他们给出的评分偶尔不一致，但可能要比两个仅看过两部相同电影的用户更相似。这在之前的数据中有所体现；注意，用户1和5对三个共同物品表达了偏好，他们的品位看似比较相近。但是，用户1和4的交集仅包含两个物品，却得到了1.0这个更高的相关值。这有点不符合常规。

其次，基于该计算的定义，如果两个用户的交集仅包含一个物品，则无法计算相关性。这也是没有计算用户1和3之间相关性的原因。在小的或稀疏的数据集上，这个问题就会凸现出来，因为其中用户的物品集很少重叠。当然，这也可能是一种优点：直观上讲，如果两个用户的交集仅有一个物品的话，他们可能并不太相似。

最后，只要任何一个序列中出现偏好值相同的情况^①，相关系数都是未定义的（*undefined*）。这种情况并不需要两个序列中的偏好值都完全一样。例如，若用户5对所有三个物品的偏好值都是3.0，即使用户1有3.0以外的偏好值，也无法计算用户1与用户5之间的相似度（因为皮尔逊相关系数将是未定义的）。这一问题同样很可能出现在两个用户的偏好交集很小的情形。



尽管皮尔逊相关系数在早期关于推荐系统的论文中很常见^②，并且在很多介绍推荐系统的书

No. 4 中被提及，但它未必是最优的。当然，它也并不差；你只需要理解它是如何工作的。

4.3.3 引入权重

为了解决上述问题，PearsonCorrelationSimilarity在标准计算公式的基础上提供了一个扩展，即加权（weighting）。

皮尔逊相关系数并不直接反映其用到的物品数目，而我们是需要这个数字的。考虑的信息越多，所得的相关结果就越可靠。为了体现这一观点，最好在基于较多物品计算相关系数时，使正相关值向1.0偏移，而使负相关值向-1.0偏移。或者，当基于较少的物品计算相关系数时，可以让相关值向偏好值的均值偏移；这与前面的效果类似，但实现会较为复杂，因为它需要记录用户对的平均偏好值。

在代码清单4-3中，将值Weighting.WEIGHTED作为第二个参数传递给PearsonCorrelationSimilarity的构造函数即可实现上述方法。它会根据计算相关系数所用的数据点数，使偏好值向1.0或-1.0偏移。在这种情况下，重新运行前面的评估程序，可以看到分值有所改善，变为0.77。

^① 此时该序列方差为0，导致皮尔逊相关系数计算公式中的分母为0。——译者注

^② 附录C列出了一些相关文献。可重点参考Breese、Heckerman、Kadie的“Empirical Analysis of Predictive Algorithms for Collaborative Filtering”以及Herlocker、Konstan、Borchers和Riedl的“An Algorithmic Framework for Performing Collaborative Filtering”这两篇文章。

4.3.4 基于欧氏距离定义相似度

下面让我们尝试使用EuclideanDistanceSimilarity——将代码清单4-3中UserSimilarity的实现改为new EuclideanDistanceSimilarity(model)即可。

这一实现基于用户之间的距离。你可以将用户想象成多维空间中的点（维数等于总的物品数），偏好值是坐标。这种相似性度量计算两个用户点之间的欧氏距离 d ^①。这个值本身并不代表相似度，因为该值越大表示距离越远，也就是说两个用户越不相似。用户越相似，这个值应该越小。因此，实际应用中取 $1/(1+d)$ 为相似度。表4-2展示了一些示例。可以证明，距离为0（用户间的偏好完全相同）时，它的结果为1，而随着 d 的增加，会逐渐递减为0。这种相似性度量不会返回负数，而且值越大表示相似度越高。

表4-2 用户1与其他用户之间的欧氏距离及所得到的相似度评分

	物品101	物品102	物品103	距离	与用户1的相似度
用户1	5.0	3.0	2.5	0.000	1.000
用户2	2.0	2.5	5.0	3.937	0.203
用户3	2.5	—	—	2.500	0.286
用户4	5.0	—	3.0	0.500	0.667
用户5	4.0	3.0	2.0	1.118	0.472

在代码清单4-3改用EuclideanDistanceSimilarity后，得到结果0.75；比之前强一点，但差别不大。注意这里可以计算出任何用户之间的相似度，而皮尔逊相关系数就无法得出用户1与用户3之间的相似度。这算是欧氏距离的一个优点，但是根据一个共同物品得到的结果并不可靠。基于欧氏距离的实现同样可能得出一些与直觉不符的结果：用户1和用户4之间的相似度高于用户1和用户5。

4.3.5 采用余弦相似性度量

余弦相似性度量（cosine measure similarity）也将用户偏好值视为空间中的点，并基于此进行相似性度量。你需要将用户偏好值视为 n 维空间中的点。现在，假设有两条从原点——或者说 $(0,0,\dots,0)$ ——出发，分别到这两个点的射线。如果两个用户相似，则他们的打分也相似，也就是说他们的空间位置是很接近的，这样一来，至少这两条射线的方向也会差不多，两条射线之间的夹角会比较小。反之，如果两个用户不相似，则相应的两个点会相隔较远，从原点到这两点的射线很有可能指向不同的方向，形成的夹角会比较大。

与欧氏距离类似，这个夹角同样可以用来度量相似性。在这种情况下，夹角余弦代表相似度值。如果你对三角函数不熟悉，那么记住这点就行了：余弦取值范围在-1到1之间，小的夹角余弦接近1，大的夹角（接近 180° ）余弦接近-1。这个性质很好，因为小的夹角映射到了较高的相

① 回忆一下，欧氏距离就是各维坐标之差的平方和的平方根。

有关此电子图书的说明

本人由于一些便利条件，可以帮您提供各种中文电子图书资料，且质量均为清晰的 PDF 图片格式，质量要高于网上大量传播的一些超星 PDG 的图书。方便阅读和携带。只要图书不是太新，文学、法律、计算机、人文、经济、医学、工业、学术等方面的图书，我都可以帮您找到电子版本。所以，当你想要看什么图书时，可以联系我。我的 QQ 是：85013855，大家可以在 QQ 上联系我。

此 PDF 文件为本人亲自制作，请各位爱书之人尊重个人劳动，敬请您不要修改此 PDF 文件。因为这些图书都是有版权的，请各位怜惜电子图书资源，不要随意传播，否则，这些资源更难以得到。