Broadview

MANNING

智能Web算法

Algorithms of the Intelligent Web

Haralambos Marmanis
Dmitry Babenko 著

阿稳 陈钢 译

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

智能Web算法

Algorithms of the Intelligent Web

Haralambos Marmanis
Dmitry Babenko 著

阿稳 陈钢 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书涵盖了五类重要的智能算法：搜索、推荐、聚类、分类和分类器组合，并结合具体的案例讨论了它们在 Web 应用中的角色及要注意的问题。除了第 1 章的概要性介绍以及第 7 章对所有技术的整合应用外，第 2～6 章以代码示例的形式分别对这五类算法进行了介绍。

本书面向的是广大普通读者，特别是对算法感兴趣的工程师与学生，所以对于读者的知识背景并没有过多的要求。本书中的例子和思想应用广泛，所以对于希望从业务角度更好地理解有关技术的技术经理、产品经理和管理层来说，本书也有一定的价值。

Original English language edition published by Manning Publications, 178 South Hill Drive, Westampton, NJ 08060 USA. Copyright © 2009 by Manning Publications. Simplified Chinese-language edition copyright © 2011 by PHEI. All rights reserved.

本书简体中文版专有出版权由 Manning Publications 授予电子工业出版社，专有出版权受法律保护。

版权贸易合同登记号　图字：01-2010-6423

图书在版编目（CIP）数据

智能 Web 算法 /（美）玛若曼尼斯（Marmanis, H.），（美）巴宾寇（Babenko, D.）著；阿稳，陈钢译.
北京：电子工业出版社，2011.7
书名原文：Algorithms of the Intelligent Web
ISBN 978-7-121-13919-2

Ⅰ. ①智… Ⅱ. ①玛… ②巴… ③阿… ④陈… Ⅲ.①互联网络－程序设计 Ⅳ. ①TP393.4

中国版本图书馆 CIP 数据核字(2011)第 124818 号

策划编辑：张春雨
责任编辑：李利健
印　　刷：
装　　订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×980　1/16　印张：25　字数：440 千字
印　　次：2011 年 7 月第 1 次印刷
定　　价：65.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

译 者 序

在这篇文章里，我会谈谈自己在翻译本书过程中的一些比较深刻的感受，以期让读者在正式阅读本书之前对它有一个整体的了解。比如，书中算法讲解背后所隐含的更重要的思想，又如作者使用了什么样的方式来传递书中的知识，以及本书的受众群体。

虽然书中各章论述的是不同的理论和不同的应用，但作者一直在试图传递一些这个领域里不变的朴素而实用的思想，其中一个最重要的思想是：组合不同的技术，以得到更好的结果。这无论是在学术界，还是在业界，都已经逐渐成为风气。不久前的 2011 中国推荐系统峰会上，张栋博士在总结自己参加 netflix 竞赛感受时就曾说到：一个好的推荐器无法打败无数技术组合起来的推荐器。而书中第 3 章也引述了 netflix 竞赛获胜者的话说："我们没有发现完美的模型，而我们最好的结果是来自对具有互补作用的模型预测结果的组合"。

此外，书中还反复强调了一些很具有现实参考意义的观点，限于篇幅，简要列举几点如下：

<1>对问题本质及数据性质的理解比使用什么算法更重要。2011 中国推荐系统峰会同样有一个类似的为人们所争议的热点，有与会者认为以重要性而论，领域知识>数据>算法。作为有过一定从业经验的算法工作者，对此应能有深刻体会。

<2>实验环境的数据要能代表现实数据。从统计与抽样理论的角度来说，这个命题的重要性毋庸置疑。而对于先在离线环境中训练模型，然后在生产环境中使用模型的智能应用模式而言，这点尤其重要，你是不可能把一个在北京计算得到的空气

质量预测模型直接拿到北海道去用的。

<3>当你想了解你的改变所产生的效果时，最好一次只改变一个因素。这其实与 AB-Test 的思想具有异曲同工之妙，要想研究某个参数或某个变量修改前后的差异，需要尽量保证除此之外的其他因素不变。

所以，这不单是一本介绍算法之术的书，更是一本介绍算法之道的书。读者在学习其中各种具体算法的同时，不妨更多地思考作者总结出来的这些实战性很强的经验，因为那些算法不一定是最新最有效的，但那些经验教训则是需要经历实际的成败方能总结出来的。此外，我们建议读者不要只看代码的实现，而更要关注蕴含于其中的一些设计原理以及对问题建模的方式。因为在实际的应用环境中，你需要考虑的可能不仅仅是一两个算法，而是一个具体的系统问题。

无论是多么朴素和实用的思想，传授起来也难免让人觉得枯燥，所以作者尽其最大努力希望从工程化的角度来介绍这些算法。其中一种尝试就是尽量摒弃对数学公式的使用，而代之以论述具体问题与代码说明的方式来解释算法。是否依赖数学公式来说明问题是一个值得商榷的事情，数学公式就好像是一种共同的语言或标准一样，只要大家都懂这种语言，沟通基本上不会存在问题，而且显得简练、优雅，无二义性。但霍金也曾经说过，多一个数学公式，就会吓跑一半的读者，所以在他那本经典著作《时间简史》中只保留了最重要的一个公式。近年来，面向大众的技术书籍也有这种尽量摆脱乏味的数学公式堆砌的风潮，特别是国外的很多优秀著作，即使只用很少的公式也能把问题说清楚。

依我的观点，一本书籍对数学公式的取舍与使用量的多少，取决于作者对读者范围的界定，以及如何找到这一读者群中公认的沟通标准。无疑，在通常的 IT 技术从业者中，按通用性而论：自然语言>代码>数学公式。虽然作为这一领域的从业者，懂得基本的数学知识是必需的，但为了使本书适用于更广泛的程序员阶层人群，作者尽力使用代码与自然语言来描述那些算法，而不是数学公式。于是，你将会很惊讶地看到，一本讲述算法的著作里，居然找不到几个数学公式，而且这一点都不妨碍你对这些算法的含义的理解。

另一方面，要填平从理论到实践、从学校到业界的鸿沟并不是件容易的事情，特别是在我国现有的教育体系下，在校期间的学习与现实需求的差异尤其大。本书作者既有工程的从业背景，又有非常丰富的机器学习研究经历，使得他可以用一种实用性较强的方式把业界所需要的知识点传递出来。被证明最直接有效的传递知识

的方式当然就是案例化的教学，书中每一个讲授具体算法的章节都会辅之以一个现实中的问题，比如文档搜索引擎、在线音乐推荐商店、用户的信用等级分类等。

除了案例化的知识传授方式，每章后的 To Do 事项则充分体现了西方人所崇尚的启发式的教学思想，虽然经过我们传统教育的读者也许并不乐意自己再去探索，而更愿意作者把所有的东西都一一告诉我们，但如果对照着现成的代码把一个个具有探索性意味的 To Do 事项完成，你获得的知识都是自己的，也是超越本书所教授的内容的。学会独立思考、独立实践，也是一名普通工程师走向优秀工程师的重要分水岭。

虽然有不少的优点，但本书也仍有其不足之处。书中对智能技术的主要方面都有所论及，但在如此篇幅的书中把所有问题都论述清楚是不可能的，过于追求全面就会导致深度方面稍有缺乏，所有领域相关的问题未必能方方面面都讨论透彻，对于资深从业人员来说，未免会有点意犹未尽的感觉。此外，虽然本书是本着工程的理念来讲述，在代码实现方面也考虑了一些工程上效率的因素，但毕竟只是一套教学演示代码，未免为追求简明易懂而显得简陋，所以也不能寄希望于把它们直接用于自己的生产环境中。

这不是一本适用于所有对该领域感兴趣的读者的书，它有其适用人群。适用人群之一：软件或互联网从业工程师，如果你原来没有接触过相关的知识，而又想让自己的代码拥有更多的智能化特性，这本书也许能给你带来新的启发和新的思维。人群之二：如果你已经是书中所涉及领域的从业人员，那么阅读该书也许会帮助你梳理原有的知识体系，但如果你在寻求某个领域的最新技术动向，此书则不合适，因为该书讲述的都是业界上已经获得成熟应用的概念与算法。人群之三：想在数据挖掘、智能技术上用力的企业管理人员或产品经理。据我的了解，一些互联网公司在发展到达一定的阶段后，纯粹的人力与资金上的增加已经无法带动效益的同比增长，他们都在寻求发展模式上的转变，转粗放型增长为技术型增长，希望这本书关于智能技术的概念与案例能给有这方面需求的人带去一定的帮助。人群之四：在自己的职业规划之路中，有算法工程师、数据挖掘工程师、数据分析师这些关键词的在校学生。

本书由我及我的搭档陈钢联合翻译完成。我们很希望，该书的出版能为正在思考自己出路的在校学生，或每天在计算机面前敲打着代码的工程师们，提供多一种职业发展的可能性。只有更多优秀的人才投身于这个领域，才能使国内 IT 公司对智

能技术应用产生广泛的重视，进而推动国内 IT 事业的创新性。如果这本书能成为未来许多年大量算法工程师的入门书籍，则是它最大的成功之处。

这本书的翻译出版，首先要感谢我所任职的公司豆瓣网，如果没有这个宽松自主的工作环境，我很难把两三个月几乎所有的业余时间都花在这件事情上。当然，如果没有在这个地方三年来的从业经验，以及在那帮优秀的同事身边成长的经历，我从一开始就不会有信心把这件事情做好。我还要特别感谢豆瓣电台，正是它知心动听的歌曲陪伴我度过了那一个个枯燥的夜晚。还有有道词典，它在网络释义与例句上给予了我莫大的帮助。而恰巧，这两个个性鲜明的互联网产品正是本书所叙述的智能 Web 技术应用的杰出代表。同样要感谢电子工业出版社博文视点公司的编辑们，以及给我们提出过宝贵意见的中国推荐社区（http://resyschina.com）的朋友们。

因为自身水平有限，在理解与翻译本书[1]的过程中，一些知识的传递未必到位，但认识总是可以通过交流与思考来加深的。所以，接下来我们也希望借助豆瓣的读书笔记功能把我们在翻译过程中得到的一些认识写下来与大家分享交流。大家在阅读过程中也不妨通过这种形式来分享自己的理解，提出自己的问题，形成一种思想上的互动。因为与同行交流是促进思考与进步的最重要手段。读者可以在博客或豆瓣上找到我们。

阿稳：http://www.wentrue.net/blog/或http://www.douban.com/people/wentrue/

阿稳

感谢昆明理工大学的彭玮老师和中国推荐系统小组中的各位同学在百忙之中抽时间审读我们的译稿。感谢电子工业出版社博文视点的各位编辑在本书翻译过程中给予的指点和帮助。特别要感谢本书的另外一位译者阿稳在技术上的指点。没有你们的支持和帮助，本书的翻译工作不可能这么顺利。在翻译本书的几个月里，尤其是最后审稿的阶段，我不得不将本应该陪伴已怀孕的妻子的时间投入到本书的翻译工作中，特将本书献给我的妻子王倩和我们即将到来的孩子。

陈钢：http://gossipcoder.com/或 http://www.douban.com/people/oldbeggar/

陈钢

1 由于本书篇幅较大，为了节省成本和便于读者对照原书阅读，我们用“▭”标出了原书对应的页码，本书的索引所列页码为原英文版页码。

前　言

在读研究生时，我开始接触到机器学习，尤其是模式识别。我的工作主要是数学建模和数值模拟，而海量数据的模式识别其实在很多领域都有着广泛的应用。以前也未曾想到，这些年我会如此深入地进入机器学习领域。

1999 年，我完成学业，开始进入企业工作。在我担任顾问的一个项目中，我们试图根据患者的心电图判断出他们患心脏病的概率。显然，对这种问题，不存在也不可能推导出一个精确的数学公式。现实中，心脏病专家已经对大量的患者患心脏病的风险做出了诊断，而我们建模所使用的方法要能从这些病历中学习如何预测患心脏病的风险。通俗地说，我们要寻找的是能从用户输入的数据中不断地“学习”新知识的方法。

同时，在 20 世纪 90 年代，各种因素汇聚在一起导致了一个新产业的飞速发展——网络变得无处不在！根据摩尔定律，CPU 的运行速度变得更快，而且价格更便宜。RAM 模组、硬盘等各种计算机硬件的性能也日新月异，而价格则是一降再降。随之而来的是，网络连接的带宽不断增长，价格也能被更多的人接受。此外，健壮的 Web 应用开发技术已经成熟，而各种开源项目的蓬勃发展更是促进了相关技术的进步。所有的这些因素构成了现在我们称为 Web 的庞大生态系统。

显然，作为软件工程师和 Web 开发人员，我们首要的任务就是为构建健壮、可扩展、美观的 Web 应用提供足够的技术保障。正是如此，在过去的十年里，人们为此做出了巨大的努力，也获得了可观的成绩。当然，没有最好，只有更好，我们依然有进步的空间。虽然我们一直在追求更健壮、可扩展性更好、更美观的 Web 应用，

然而我们已经遇到了瓶颈。在我们看来，单调乏味的互联网应用已经成为过去，仅仅是聚合数据，简单地根据预定逻辑工作的用户请求/响应模型也已经走到了尽头。

现在，在某些应用中已经出现了一股新的浪潮，让人们对互联网应用有了新的认识。这就是本书中所说的智能应用（intelligent application）。不同于传统的应用，智能应用能根据用户的输入调整自己的行为，就像我那个能根据心电图预测患心脏病概率的建模软件。

最近五年，我渐渐地发现，对于大部分软件开发人员来说，构建智能应用的技术依然未曾掀开神秘的面纱。在我看来，这是由两方面的原因造成的。一方面，这些技术潜在的商业价值可以带来巨大的经济回报。所以从经济方面考虑，对这些应用进行保护，隐藏其中的关键细节是可以理解的。另一方面，几乎所有的相关技术都源自学术研究，需要较强的数学背景才能理解。对于第一个原因，我们无能为力，但在随时能获取海量知识的今天，第二个原因依然是不可逾越的障碍吗？我可以简短而明确地回答“不是!”。如果想要详细地回答，那就阅读本书吧！

我决定写这本书，是为了说明这些技术是可以用算法来表示的，并不需要读者有很强的数学基础。本书的目的是让读者掌握一些有助于在应用中构建智能行为的技术，同时尽可能地降低掌握这些技术的数学门槛。代码以算法的形式包含了所有必要的数学知识。

最初，我想用开源的库来演示这些技术，但大部分的此类库都是为了解决具体问题而开发的，并不是为了演示底层的技术。因此，这些库的源代码通常都是冗长且晦涩难懂的。显然，如果能有清晰、带注释的代码，一定会让本书的读者获益更多。Dmitry 就是在这个时候加入了本书的写作，并最终编写完成了本书中的大部分代码。

尽管增长缓慢，但关于这个激动人心的新领域的书籍肯定将逐渐增多。本书只是一本有关这个依然在迅速增长的大领域的入门书籍。所以，本书所涉及的算法是很有限的，对算法的解释也比较简要。我的目标是选择并探讨一些有代表性的话题，而不是写一本代码手册或是有可能让读者晕头转向，内容包罗万象的书。

我希望能通过以下四个方面来实现我的目标：

- 集中精力关注清晰易懂的例子。
- 使用高级脚本语言来演示算法的使用，就像读者在自己的应用中使用这些算法一样。
- 通过大量的 ToDo 事项让读者有机会尝试并思考这些代码。

- 编写高水平的、易读的代码。

那么，端着您最喜爱的热饮，坐好，来试试这些聪明的应用吧！它们就在本书中！

HARALAMBOS MARMANIS

致 谢

我们要感谢 Manning 出版公司的人让我们有机会出版这本书。他们不仅让本书从书稿最终变成一本书，而且在我们的写作进度一再延迟的情况下非常耐心地帮助我们最终完成了本书。我们尤其要感谢 Marjan Bace、Jeff Bleiel、Karen Tegtmeyer、Megan Yockey、Mary Piergies、Maureen Spencer、Steven Hong、Ron Tomich、Benjamin Berg、Elizabeth Martin，以及 Manning 出版公司所有为本书做出贡献的人们，感谢你们的努力工作。

对审稿人和 Author Online 论坛访问者在本书上耗费的时间和精力，以及给我们的宝贵意见，我们也深表谢意，你们的建议从诸多方面提升了本书的质量。我们知道真正的“自由”时间对于专业人员是非常有限的，所以请接受我们对你们的贡献表示诚挚的谢意。

我们尤其要感谢下面这些在本书撰写过程中反复阅读书稿，并给我们提出宝贵意见的审稿人：Robert Hanson、Sumit Pal、Carlton Gibson、David Hanson、Eric Swanson、Frank Wang、Bob Hutchison、Craig Walls、Nicholas C. Heinle、Vlad Gorsky、Alessandro Gallo、Craig Lancaster、Jason Kolter、Martyn Fletcher 和 Scott Dawson。很重要的是，我们要诚挚地感谢本书的技术校对员 Ajay Bhandari，他在本书付梓之前仔细地阅读了每一章的内容，并检查了所有的代码。

H. Marmanis

我要感谢我的父母——Eva 和 Alexander，他们循序渐进地引导我对学习的好奇

心和热情，这正是我废寝忘食写作和研究的原动力。穷我一生都难以报答他们的养育之恩。

我由衷地感谢我亲爱的妻子 Aurora 和我的三个儿子：Nikos、Lukas、Albert，他们是我生活中最大的骄傲和乐趣，感谢他们对我的爱、耐心和理解。孩子们永不停歇的好奇心促使我不断地学习。此外，还要感谢我的岳父岳母 Cuchi 和 Jose、我的姐妹 Maria 和 Katerina，以及我最好的朋友 Micheal 和 Antonio，感谢他们长期以来给我的鼓励和无条件的支持。

无论如何，我都不会忘记 Drs. Amilcar Avendaño 和 Maria Balerdi 的大力支持，他们教给我很多关于心脏的知识，还资助了我关于“学习”的早期研究。此外，还要感谢 Leon Cooper 教授和布朗大学所有让人啧啧称奇的人们，他们对人类大脑的研究热情感染了很多像我一样的人，并激发了我对智能应用的研究兴趣。

对于曾经和现在的同僚 Ajay Bhandari、Kavita Kanetkar、Alexander Petrov、Kishore Kirdat 等在智能研究中给予我的鼓励和支持，我的感激之情远远超出这寥寥数行文字。

D. Babenko

首先，我要感谢我亲爱的妻子 Elena。在撰写本书一年多的时间里，作为丈夫，我的时间几乎都花在了工作或是本书的撰写上，她对此却毫无怨言。她的支持和鼓励为本书的顺利完成提供了一个完美的氛围。

我还要感谢所有现在或曾经与我共事，并影响了我的专业生涯，给我带来无数灵感的同事：Konstantin Bobovich、Paul A. Dennis、Keith Lawless 和 Kevin Bedell。

最后，我要感谢我的共同作者 Marmanis 博士让我参与这项工作。

PDG

关于本书

现代 Web 应用那绚丽流畅的用户界面经常为人所津津乐道。这些应用的另一个方面则不太为人所知，那就是利用各种技术对信息进行智能化的处理，从而带来其他方法所不能给予的价值。这些技术的成功例子包括我们常见的 Google、Netflix 和 Amazon。这些应用的核心智能是由一些算法构成的，而本书要介绍的就是这些算法。

本书涵盖五类重要的算法：搜索、推荐、聚类、分类和分类器融合，其中的每一个算法都可以写成一本完整的书，所以面面俱到并不是本书的目的。本书只会对这五类算法做基本的介绍，我们的目的是展示智能应用中的基本算法，而不是覆盖计算智能中的所有算法。本书是为普通读者撰写的，所以尽可能地降低了对读者的知识背景的要求。

本书的一大特点是在每章结尾都有一个很特殊的小节，我们称为 To Do，其目的不仅仅是提供额外的参考资料，每个 To Do 小节还会指导读者更深入地了解这一章的主题，激起读者思考其他可能的好奇心，以及现实应用中可能要面对的挑战。

本书大量地使用了 BeanShell 脚本库，目的有两个：其一，让读者从更高的层次上理解算法，避免过早地陷入细节；其二，清楚地描述如何将这些算法整合到读者自己的应用中。在大部分情况下，读者只要写很少的几行代码就能使用本书附带的库。不仅如此，为了维护这些源代码，确保其时效性，我们还在 Google Code（http://code.google.com/p/yooreeka/）上专门建了一个项目。

全书结构

本书共分 7 章，第 1 章是简介，第 2～6 章分别介绍搜索、推荐、聚类、分类和分类器组合，第 7 章介绍如何把前几章中的算法整合到一个具体的应用中。

尽管章节之间有一些联系，但这并不会妨碍你单独阅读第 1～5 章中的任何一章。第 6 章是以第 5 章为基础的，如果单独阅读第 6 章，则可能有些难度。第 7 章则涉及本书所有的内容，单独阅读该章也会有些困难。

第 1 章介绍了智能应用的概况，并举例说明了智能应用的意义。这一章从实践的角度定义了智能 Web 应用和一些设计原则，接着介绍了六大类 Web 应用，这些应用都可以利用本书中介绍的智能算法加以改进。这一章还讲述了本书所涉及的算法的历史起源，及其与人工智能、机器学习、数据挖掘、软计算等领域的关系。最后还总结了八条具有重要的实践意义的设计原则。

第 2 章首先描述了依赖于传统信息获取技术的搜索方法。对传统方法稍作总结后，逐步转向不仅仅是索引的搜索，其中包括最负盛名的链接分析算法——PageRank。还有一节介绍了如何对用户的点击进行分析来提高搜索结果的质量。这项技术能获取用户对某个网站或话题的喜好，而且有很大的改进潜力，可以扩展出很多新的特性。

第 2 章还介绍了一个用于非网页文档搜索的新算法——DocRank。这个算法有一定的前景，但更重要的是这个算法说明了稍做改动，链接分析中的基本数学原理就能快速地扩展到其他应用中。另外还介绍了一些处理超大网络时有可能会遇到的挑战。最后，介绍了有关搜索结果的可信度和验证的问题。

第 3 章介绍了两个重要的概念，即距离和相似度。然后介绍两大类构建推荐系统的技术——协同过滤和基于内容的方法。该章以一个虚拟的网络音乐商店为例介绍了如何为其开发推荐系统，还介绍了两个更通用的例子，第一个例子是一个假想的网站，利用 Digg 的 API 获取用户感兴趣的内容，然后据此向用户推荐其没看过的文章。第二个例子是关于电影推荐的，引入了数据规范化（data normalization）的概念。本章还介绍了基于均方根误差的推荐系统精确性评价的方法。

第 4 章介绍了聚类算法。聚类有着广泛的应用领域，从理论上说，任何由多个对象组成的数据集都可以根据给定的属性进行聚类。在该章中，我们会介绍论坛帖子的分组，以及如何识别相似的网站用户。同时还介绍了不同类型的聚类算法，以及六种算法的完整实现：单链接（single link）、平均链接（average link）、最小生成

树单链接（minimum spanning tree single link）、k 均值（k-means）、ROCK 和 DBSCAN。

第 5 章介绍分类算法，这是智能应用所不可或缺的组件。该章首先描述了本体（ontology），它包含三个组成部分——概念（concept）、实例（instance）和属性（attribute）。所谓分类，就是将实例赋予最合适的概念。不同的分类器之间的差别就在于它们表示和衡量最优赋值方案的方法。该章简要介绍了分类问题，包括二分类和多分类、统计算法和结构算法。本章还介绍了使用分类器的三个步骤：训练、验证和生产阶段。

第 6 章介绍了分类器的组合——一种可以提升单个分类器准确性的高级技术。该章主要的例子是评价抵押申请的可靠性，同时会详细探讨 Bagging 和 boosting 两种技术。另外还介绍了 Breiman 的 arc-x4 boosting 算法的一个实现。

第 7 章举例介绍了这些智能算法在一个新闻门户网站中的应用。我们讨论了其中的技术问题，以及这些智能算法给应用所带来的新的业务价值。例如，聚类算法可以用于新闻的分组，还可以利用新闻之间的相互引用增加新闻的曝光度。在该章中，我们介绍了智能算法的实际应用，勾勒出了将各种智能算法组合在一起实现特定目标的大致框架。

有特色的 TO DO 小节

从第 2 章开始，每一章的最后一节会提供一些引导读者深入学习的内容。作为一个软件工程师，我们发现 To Do 这种形式非常有吸引力：带有祈使的语气，但又不像练习（exercise）那样正式。

有些 To Do 的内容是更加深入地探讨本章的内容，但有些是向读者展示与本章主题有关的其他内容。完成这些任务可以让读者更加深入和广泛地理解智能算法。

本书中所有标注了“TO DO”的代码都可以在各种 IDE 中查看，例如，在 Eclipse IDE 中可以单击 Tasks 面板。单击任何一个任务，都会显示与之相关的部分代码。

谁适合阅读本书

对于想学习在商业上取得巨大成功的算法的软件工程师和 Web 开发人员来说，本书正是为你们而写的。因为本书的源代码是基于 Java 编程语言的，所以本书对 Java 用户可能更具吸引力。尽管如此，使用其他编程语言的读者也能从本书中获益，或许还能将书中的代码转换成其他编程语言的版本。

本书中的例子和思想应用广泛，对于希望从业务角度更好地理解有关技术的技

术经理、产品经理和管理层来说，也有一定的价值。

最后，尽管在本书的标题中有 Web 一词，但本书中的内容也同样适用于其他类型的软件应用，包括移动应用，以及诸如文本编辑器和电子表格一类的传统桌面应用等。

代码约定

本书中所有的源代码都是等宽字体，并且是与上下文分开的。本书中的大部分代码清单都是用于说明代码中的关键概念的，而有些清单有时则是与代码有关的附加信息，某些很长的代码行会有行继续符号。

本书中所有的源代码都可以从 http://code.google.com/p/yooreeka/downloads/list 或出版社的网站 www.manning.com/AlgorithmsoftheIntelligentWeb 中获得。

我们假设读者使用的是微软 Windows 操作系统，否则，读者需要自行修改我们提供的脚本以适用于其他操作系统。将下载的文件解压到 C 盘。压缩文件的顶层目录的名字是 iWeb2，本书中所有的目录都是相对于这个顶层目录的。例如，如果说 data/ch02 目录，指的就是 C:\iWeb\Data\ch02 目录。

解压之后，就可以运行 Ant 构建脚本。很简单，切换到构建目录，然后运行 Ant。无论将文件解压在什么位置，Ant 脚本都能正常运行。现在就可以根据附录 A 的内容来运行 BeanShell 脚本了。

作者在线

购买本书的同时，读者也获得了免费访问 Manning Publications 论坛的权限，在这里，读者可以对本书进行评价、咨询技术问题，并从作者或其他读者那里获得帮助。在浏览器中输入 www.manning.com/AlgorithmsoftheIntelligentWeb，就能访问和订阅该论坛的内容，这个页面中说明了读者在注册后如何访问该论坛，可以获得哪些帮助以及论坛的规则，同时还有链接指向本书中例子的源代码、勘误表和其他下载。

Manning 出版社致力于提供一个用户之间以及用户和作者之间的交流平台。对作者参与该论坛的交流并没有强制要求，所有 Author Online 上的贡献都是自愿的（当然也没有报酬）。建议读者尝试问作者一些有挑战性的问题，作者对这样的问题会更有兴趣。

只要本书还在出售，读者就可以在出版商的网站上访问 Author Online 论坛和所有讨论的文档。

关于封面设计

本书的封面设计来自法国的一本服装设计书，即 J. G. St. Saveur 在 1796 年出版的 Encyclopedie des Voyages。旅游在当时还是一个比较新鲜的事物，诸如这样的旅行手册很受欢迎，无论是旅行者还是足不出户的读者，都能从书中了解到世界上其他地方的风土人情，以及法国和欧洲其他地区的特色服饰。

Encyclopedie des Voyages 一书中用丰富的图片生动地展示了 200 年前世界各地的特色。在那个时代，两个人即使是来自两个相隔不过十来英里的地方，也可以轻易通过着装区分出来。不仅如此，在那个时代，通过一个人的服饰还能轻易地判断出这个人的社会地位、行业和种族。

在那以后，不同地区之间服饰的差异逐步缩小。现在，仅仅根据服饰已经很难区分出来自不同大洲的人。或许，乐观地看，我们告别了一个文化和服饰极具特色的世界，换来了多姿多彩的个人生活，或者说得到了更丰富有趣的智能化高科技生活。

本书封面上两个世纪前极具地方特色的服饰就来自这本旅游手册，Manning 出版社以此来庆祝计算机产业的创造力、进取心和其中的乐趣。

目 录

前言 XV

致谢 XIX

关于本书 XXI

1 什么是智能 Web? 1

1.1 智能 Web 应用实例 3

1.2 智能应用的基本要素 4

1.3 什么应用会受益于智能? 5

1.3.1 社交网络 6

1.3.2 Mashup 7

1.3.3 门户网站 8

1.3.4 维基 9

1.3.5 文件分享网站 9

1.3.6 网络游戏 11

1.4 如何构建智能应用? 11

1.4.1 检查功能和数据 12

1.4.2 获取更多的数据 12

1.5 机器学习、数据挖掘及其他 16

1.6 智能应用中八个常见的误区 …… 17
1.6.1 误区 1：数据是可靠的 …… 18
1.6.2 误区 2：计算能马上完成 …… 19
1.6.3 误区 3：不用考虑数据规模 …… 19
1.6.4 误区 4：不考虑解决方案的可扩展性 …… 19
1.6.5 误区 5：随处使用同样的方法 …… 19
1.6.6 误区 6：总是能知道计算时间 …… 20
1.6.7 误区 7：复杂的模型更好 …… 20
1.6.8 误区 8：存在无偏见的模型 …… 20
1.7 小结 …… 20
1.8 参考资料 …… 21
2 搜索 …… 22
2.1 用 Lucene 实现搜索 …… 23
2.1.1 理解 Lucene 代码 …… 24
2.1.2 搜索的基本步骤 …… 31
2.2 为什么搜索不仅仅是索引？ …… 33
2.3 用链接分析改进搜索结果 …… 35
2.3.1 PageRank 简介 …… 35
2.3.2 计算 PageRank 向量 …… 37
2.3.3 alpha：网页间跳转的影响 …… 38
2.3.4 理解幂方法 …… 40
2.3.5 结合索引分值和 PageRank 分值 …… 45
2.4 根据用户点击改进搜索结果 …… 47
2.4.1 用户点击初探 …… 48
2.4.2 朴素贝叶斯分类器的使用 …… 50
2.4.3 整合 Lucene 索引、PageRank 和用户点击 …… 54
2.5 Word、PDF 等无链接文档的排序 …… 58
2.5.1 DocRank 算法简介 …… 58
2.5.2 DocRank 的原理 …… 60
2.6 大规模实现的有关问题 …… 65

2.7 用户得到了想要的结果吗？精确度和查全率 …… 67
2.8 总结 …… 69
2.9 To Do …… 70
2.10 参考资料 …… 72

3 推荐系统 …… 73

3.1 一个在线音乐商店：基本概念 …… 74
3.1.1 距离与相似度的概念 …… 75
3.1.2 走近相似度的计算 …… 80
3.1.3 什么才是最好的相似度计算公式？ …… 83
3.2 推荐引擎是怎么工作的 …… 84
3.2.1 基于相似用户的推荐 …… 85
3.2.2 基于相似条目的推荐 …… 94
3.2.3 基于内容的推荐 …… 98
3.3 推荐朋友、文章与新闻报道 …… 104
3.3.1 MyDiggSpace.com 简介 …… 105
3.3.2 发现朋友 …… 106
3.3.3 DiggDelphi 的内部工作机制 …… 108
3.4 像 Netflix.com 那样推荐电影 …… 114
3.4.1 电影数据集的介绍及推荐器 …… 114
3.4.2 数据标准化与相关系数 …… 117
3.5 大规模的实现与评估 …… 123
3.6 总结 …… 124
3.7 To Do …… 125
3.8 参考资料 …… 127

4 聚类：事物的分组 …… 128

4.1 聚类的需求 …… 129
4.1.1 网站中的用户组：案例研究 …… 129
4.1.2 用 SQL order by 子句分组 …… 131
4.1.3 用数组排序分组 …… 132

4.2 聚类算法概述 …… 135
4.2.1 基于分组结构的聚类算法分类 …… 136
4.2.2 基于数据类型和结构的聚类算法分类 …… 137
4.2.3 根据数据规模的聚类算法分类 …… 137
4.3 基于链接的算法 …… 138
4.3.1 树状图：基本的聚类数据结构 …… 139
4.3.2 基于链接的算法概况 …… 141
4.3.3 单链接算法 …… 142
4.3.4 平均链接算法 …… 144
4.3.5 最小生成树算法 …… 147
4.4 k-means 算法 …… 149
4.4.1 初识 k-means 算法 …… 150
4.4.2 k-means 的内部原理 …… 151
4.5 鲁棒的链接型聚类（ROCK） …… 153
4.5.1 ROCK 简介 …… 154
4.5.2 为什么 ROCK 这么强大？ …… 154
4.6 DBSCAN …… 159
4.6.1 基于密度的算法简介 …… 159
4.6.2 DBSCAN 的原理 …… 162
4.7 超大规模数据聚类 …… 165
4.7.1 计算复杂性 …… 166
4.7.2 高维度 …… 167
4.8 总结 …… 168
4.9 To Do …… 169
4.10 参考资料 …… 171

5 分类：把事物放到它该在的地方 …… 172
5.1 对分类的需求 …… 173
5.2 分类器的概述 …… 177
5.2.1 结构分类算法 …… 178
5.2.2 统计分类算法 …… 180

5.2.3 分类器的生命周期 …… 181
5.3 邮件的自动归类与垃圾邮件过滤 …… 182
5.3.1 朴素贝叶斯分类 …… 184
5.3.2 基于规则的分类 …… 197
5.4 用神经网络做欺诈检测 …… 210
5.4.1 交易数据中关于欺诈检测的一个用例 …… 210
5.4.2 神经网络概览 …… 212
5.4.3 一个可用的神经网络欺诈检测器 …… 214
5.4.4 神经网络欺诈检测器剖析 …… 218
5.4.5 创建通用神经网络的基类 …… 226
5.5 你的结果可信吗？ …… 232
5.6 大数据集的分类 …… 235
5.7 总结 …… 237
5.8 To Do …… 239
5.9 参考资料 …… 242

6 分类器组合 …… 244

6.1 信贷价值：分类器组合案例研究 …… 246
6.1.1 数据的简要说明 …… 247
6.1.2 为真实问题生成人工数据 …… 250
6.2 用单分类器做信用评估 …… 255
6.2.1 朴素贝叶斯的基准线 …… 255
6.2.2 决策树基准线 …… 258
6.2.3 神经网络的基准线 …… 260
6.3 在同一个数据集中比较多个分类器 …… 263
6.3.1 McNemar 检验 …… 264
6.3.2 差额比例检验 …… 266
6.3.3 Cochran Q 检验与 F 检验 …… 268
6.4 bagging: bootstrap 聚合（bootstrap aggregating） …… 270
6.4.1 bagging 实例 …… 272
6.4.2 bagging 分类器底层细节 …… 274

6.4.3 分类器集成 …… 276
6.5 boosting：一种迭代提高的方法 …… 279
6.5.1 boosting 分类器实例 …… 280
6.5.2 boosting 分类器底层细节 …… 282
6.6 总结 …… 286
6.7 To Do …… 288
6.8 参考资料 …… 292
7 智能技术大汇集：一个智能新闻门户 …… 293
7.1 功能概览 …… 295
7.2 获取并清洗内容 …… 296
7.2.1 各就位、预备、开抓！ …… 296
7.2.2 搜索预备知识回顾 …… 298
7.2.3 一个抓取并处理好的新闻数据集 …… 299
7.3 搜索新闻 …… 301
7.4 分配新闻类别 …… 304
7.4.1 顺序问题 …… 304
7.4.2 使用 NewsProcessor 类进行分类 …… 309
7.4.3 分类器 …… 310
7.4.4 分类策略：超越底层的分类 …… 313
7.5 用 NewsProcessor 类创建新闻分组 …… 316
7.5.1 聚类全部文章 …… 317
7.5.2 在一个新闻类别中聚类文章 …… 321
7.6 基于用户评分的动态内容展示 …… 325
7.7 总结 …… 328
7.8 To Do …… 329
7.9 参考资料 …… 333
附录 A BeanShell 简介 …… 334
A.1 什么是 BeanShell？ …… 334
A.2 为什么使用 BeanShell？ …… 335

A.3 运行 BeanShell ····· 335
A.4 参考资料 ····· 336
附录 B 网络采集 ····· 337
B.1 爬虫组件概况 ····· 337
B.1.1 采集的步骤 ····· 338
B.1.2 我们的简单爬虫 ····· 338
B.1.3 开源 Web 爬虫 ····· 339
B.2 参考资料 ····· 340
附录 C 数学知识回顾 ····· 341
C.1 向量和矩阵 ····· 341
C.2 距离的度量 ····· 342
C.3 高级矩阵方法 ····· 344
C.4 参考资料 ····· 344
附录 D 自然语言处理 ····· 345
D.1 参考资料 ····· 347
附录 E 神经网络 ····· 348
E.1 参考资料 ····· 349
索引 ····· 350

Download at http://www.pin5i.com/

1 什么是智能 Web?

本章内容提要：

- 智能 Web 应用的意义
- Web 应用的实例
- 让网站变得智能

这是一本什么书？先说说这不是一本什么样的书。本书不会讲如何设计绚丽的用户界面，也不会涉及如何使用 JSON 和 XPath，与 RESTful 架构也没什么关系。已经有很多书介绍了在 Web 2.0 应用中如何使用 AJAX 技术、如何设计丰富的用户界面(UI)。有关 XSL 转换(XSLT)、XML 路径语言(XPath)、可缩放矢量图(SVG)、XForms、XML 用户界面语言（XUL）以及 JavaScript 对象概念（JSON）的书籍也是数不胜数。

本书的出发点：是我们发现大部分传统的 Web 应用是非智能的，主要表现在系统做出反馈时不会考虑用户在此之前的输入和行为。我们在这里说的不是糟糕的 UI 设计，而是系统对给定的输入所做出的一成不变的反馈。我们想要构建的是这样一个 Web 应用：系统在做出反馈时会考虑到整个系统中所有的用户在不同时间的输入和行为，并对其他各种可能有用的信息加以利用。

假如你在某个网站上订购食物，并且每周三都会买鱼。如果网站能在周三早上

Download at http://www.pin5i.com/

问你："今天要买鱼吗？"，而不是死板地询问："今天想买什么？"，无疑能让你有更好的购物体验。前者说明网站通过某种方法知道了你在周三喜欢买鱼，而后者则表示网站完全没有意识到这个事实。也就是说，你跟网站之间的交互所产生的数据对于网站所提供的内容或内容的呈现方式是完全没有影响的。像这样根据用户之前的选择来提问，引入了一种全新的网站和用户之间的交互方式。可以说，这样的网站是具有"学习能力的"。

更进一步，智能 Web 应用和用户之间的交互还可以根据其他相关用户的行为进行调整。例如，你的饮食习惯与张三很相近，但张三经常购买的某些食物是你从没尝试过的，系统就可以将这些食品推荐给你。在第 3 章将会介绍如何构建推荐系统。

另一个例子是诸如 Facebook 一类的社交网站中具有事实核查（fact-checking）功能的聊天室和论坛。这里的事实核查指的是用户在输入消息的过程中，会有一个系统在后台对用户输入进行检查，确保其内容与事实相符，且与该用户之前的言论没有矛盾。这个功能有点类似于大家很熟悉的拼写检查，区别在于它所检查的不是语法规则，而是公认的事实（例如"日本于 1931 年入侵中国满洲地区"），或是个人所持的观点（例如，"减税有利于经济发展"），或是用户自己的具体信息（例如，"在 2008 年 11 月 11 日预约了医生"）。具有这类功能的网站就拥有了推测能力。在第 5 章将会介绍如何设计此类功能。

可以说，诸如 Google 一类的搜索引擎的诞生预示着智能 Web 应用时代的来临。你可能觉得奇怪：为什么是 Google？早在 Google 诞生之前，人类就已经知道如何获取所需的信息（搜索）。但是诸如 Google 一类的搜索引擎充分利用了互联网上的内容是相互链接的这一非常重要的特性。在 Google 看来，网页间的超链接实际上构成了一个可以用来确定网页重要性的系统。在第 2 章将会详细介绍实现这一功能的 PageRank 算法。

扩展一下我们的讨论，可以说智能 Web 应用基本的出发点就是世界是相互协作、相互联系的。智能 Web 应用能自动地训练自己去理解用户的输入和行为，并据此调整自己的反应。同事、朋友和亲友在社交网络上分享的个人信息，以及在新闻组和论坛里分享的新闻和观点，构成了新的联系。这种联系超越了普通的超链接，成了智能 Web 应用的新核心。

Download at http://www.pin5i.com/

1.1 智能 Web 应用实例

让我们回顾一下过去几年里利用了智能设计的 Web 应用。前面提到过，网络发展史上一个重要的转折点就是搜索引擎的诞生。在 1998 年之前，互联网的发展基本上是波澜不惊的，直到基于链接分析（link analysis，详见第 2 章）的搜索技术风卷残云般地占领了整个市场。Google 公司能在 10 年内从一个名不见经传的创业公司成长为技术王者，主要归功于基于链接分析的搜索技术，其次是它所提供的诸如 Google News 和 Google Finance 之类多种多样的服务。

然而，智能 Web 应用的发展已经不再局限于搜索引擎领域。在线零售商 Amazon 是第一个根据用户的消费习惯推荐商品的电子商务网站，你可能非常熟悉这个功能：假如用户购买了两本书，分别是关于 JavaServer Faces 和 Python 的。当用户将这两本书放入购物车时，Amazon 就会向用户推荐与这两本书相关的其他商品，例如，关于 AJAX 或者 Ruby on Rails 的书籍。此外，在用户下次访问 Amazon 时，Amazon 还会继续推荐有关的商品。

另一个智能 Web 应用的例子是世界上最大的在线电影出租网站 Netflix[1]，它现在为 700 万用户提供 90000 部 DVD，以及 5000 部可以随时在计算机上观看的电影和电视节目，这些数字还在不断地增长。根据 ForeSee Results and FGI Research 公司每半年一次的调查，Netflix 在 2005 年到 2007 年间连续 5 次被评为用户体验最佳的网站。

Netflix 的成功部分要归功于它能帮助用户在浩如烟海的电影库中便捷地挑选到中意的电影。实现这一功能的核心就是名为 Cinematch 的推荐系统，该系统的作用就是根据用户对其他电影的喜好来预测其是否会喜欢某部电影。这又是一个智能 Web 应用的绝佳实例。Netflix 在 2006 年 10 月宣布，如果有人能提高该系统的预测能力，将获得 100 万美元的奖励[2]，足见该系统的预测能力对 Netflix 的重要价值。到 2007 年 10 月，该比赛已经吸引了来自 165 个国家的 28845 位竞争者。在第 3 章将会详细介绍构建 Cinematch 这样的推荐系统所必需的算法。

利用收集的数据做出智能的预测并不仅仅局限于书籍和电影的推荐。Predict WallStreet 公司收集用户对股票或指数的预测，从中推测操盘手的观点，进而预测该

1 来源：Netflix 公司网站：http://www.netflix.com/MediaCenter?id=5379。
2 来源：http://www.netflixprize.com//rules。

Download at http://www.pin5i.com/

资产的价值。我们并不是建议你根据他们的预测进行股票交易，但这确实很有创意地利用了本书将要介绍的技术。

4 1.2 智能应用的基本要素

让我们再仔细地看看前一节介绍的所谓的智能应用到底有什么特点，尤其是如何区分协作与智能。试想，在某个网站上，用户可以合作编写文档。不论怎么定义“先进”，这样的网站都应该算得上先进了。它为用户提供了一个在线的协作平台，还有简单易用的富用户界面、流畅的工作流程等。但这样的应用算得上是智能 Web 应用吗？

相对于单个用户撰写的文档，多个用户在该网站上完成的文档的篇幅会更长，内容会更有深度，也有可能会更准确。从这个方面看，这份文档不仅是简单地把各个作者的知识汇集在一起，同时也受到用户之间相互交流的影响。用这种方式创建的文档是所有的作者相互协作的产物。

这并不是一个新概念。例如，在科学和工程领域，行业标准通常都是由一个技术委员会制定的。委员会首先会撰写一份草案，其中汇集了专家的知识和很多兴趣小组的观点，并试图满足整个行业的需求，而不是讨好某一个人或某一家企业的需求。接着，第一份草案就会公之于众，邀请所有的人提供意见和建议。这样做的目的是让最终的文件能代表整个业界的观点，从而满足各方面的要求。

回到我们的应用。到现在为止，这个应用只是获取了集体知识，是多人协作的结果，但这还不是智能的。集体智能（collective intelligence）是一个非常流行但常常被误解的术语。集体智能需要集体知识，而且是构建在相互协作的基础之上的，但这仅仅是该软件系统被视为智能的必要条件，而不是充分条件。

为了更好地理解到底什么是“智能”，不妨假设我们想象的网站有这样一个功能：用户在撰写文档时，系统会根据当前用户输入的内容搜索与之相关的文档，并将这些文档的摘要显示在侧栏中。这些文档可以是用户自己收集的文档，也可以是其他合作者所分享的进展中的文档，还可以是其他自由分享的文档。

用户可以在所处理的文档中标记出一段内容，然后要求系统在网络上发现摘要中有与此相关的文档时就通知用户。更有意思的功能是，根据用户设定的标准，当社区中对用户所标记的内容的观点有变化时提醒用户。

Download at http://www.pin5i.com/

创建有这些功能的应用所要求的不仅仅是漂亮的用户界面和一个协作平台。这样的应用需要理解用户输入的各种文本，要明白在特定语境中用户输入的内容的具体含义；要能自动地根据文档的相似性，分门别类地处理自然语言撰写的文档；需要对整个世界有系统的认识，至少也要有足够的关于文档所涉及的领域的知识；要能快速地处理符合用户要求的文档。

这样，我们就能将维基百科这样的应用与 Google 搜索、Google Ads、Netflix Cinematch 这样的智能应用区分开了。前一类应用是聚合并维护集体知识的协作平台。而后一类应用则能根据所收集的知识抽象出模式，进而挖掘出新的机会和价值。

下面总结一下构建智能 Web 应用所必需的基本要素。

- 内容聚合（aggregated content）：换句话说，就是与具体应用有关的大量数据。聚合的内容不是静态的，而是动态的，它的来源和存放的地理位置都是分散的，而且这些信息之间都是相互关联的。
- 参考结构（reference structures）：这些结构为内容提供了结构化、语义化的解释。例如，动态地用标签注释内容，并且持续地改进向用户展示集体知识的方式。有些人将此称为大众分类法（folksonomy）。无论是一般的参考结构或是针对某个具体领域的参考结构，都可以分为三种类型：字典、知识库和本体（ontology）（见本章的参考资料）。
- 算法：有了算法，应用才能获得数据中所隐藏的信息，并利用这些信息进行抽象（泛化），然后做出预测，最终提升用户的交互体验。算法的应用对象是聚合的内容，有时候也需要参考结构。

如图 1.1 所示，这三者都是智能 Web 应用所必需的基本元素。在本书中，我们将其称为智能三角（triangle of intelligence）。

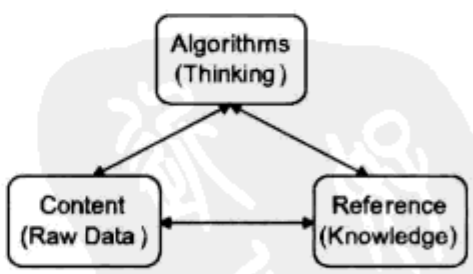


图 1.1 智能三角：构成智能应用的三个基本要素

要仔细区分这三个部分，并根据具体的需求构建最合适的三者间的交互模型。在本书的其他章节，尤其是第 7 章，将会继续讨论与此相关的架构设计。

1.3 什么应用会受益于智能?

如前所述，从社交网络到专业的反恐应用等各种应用中都能找到智能的影子。

Download at http://www.pin5i.com/

在本节中，我们会介绍几个不同类型的例子。当然，这份列表是不完全的，但已足以说明本书所介绍的技术有着广泛的应用，在某些情况下甚至是不可或缺的。

1.3.1 社交网络

近年来，互联网上最热门的网站就是社交网络。用户只需要一个浏览器，然后连接到互联网，这些 Web 应用就能让用户在互联网上展示自己。用户可以相互分享文件（演示文档、视频文件、音频文件）、对刚发生的事件或是其他人的页面发表评论、构建自己的社交网络或是根据自己的兴趣加入已有的社交网络。访问量最大[3]的两个社交网络网站是 MySpace 和 Facebook，两者分别有数亿和数千万注册用户。

这些网站本身是基于内容聚合的，所以自然就具备了智能应用的第一个要素，它们也有第二个要素。例如，在 MySpace 上，所有的内容都根据顶级标签在网站上清晰地分类成“Books”、“Movies”、“Schools”、“Jobs”等（见图 1.2）。

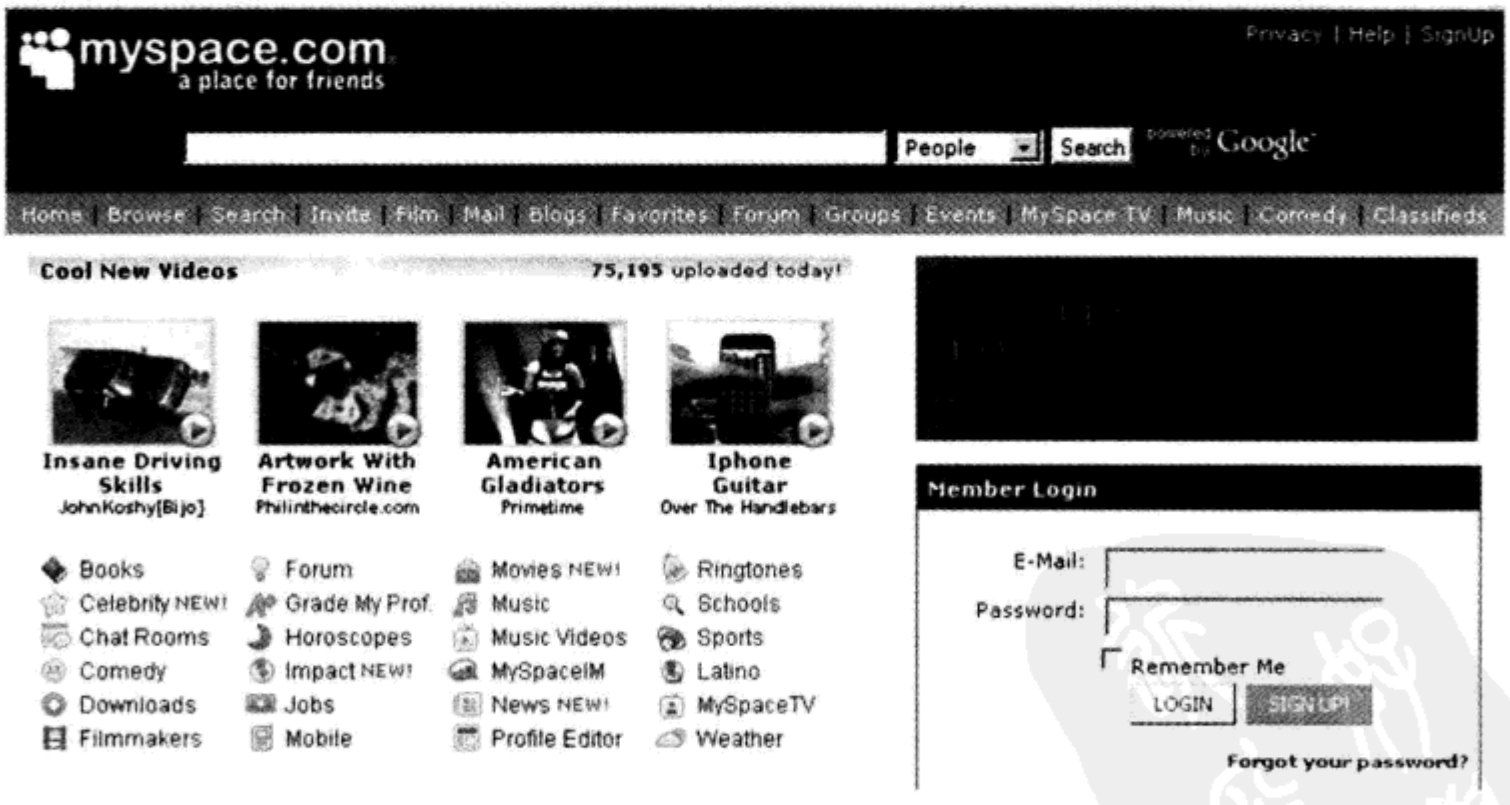


图 1.2 MySpace 网站上分类的截图

此外，这些大的分类还根据具体的结构进一步分成诸如“Polls”或“Weather”这样的小分类。最后，大部分社交网络还能向用户推荐新的好友和可能感兴趣的新帖子。要实现这个功能，这些网站就不得不利用各种高级的算法对收集的数据做出

7

3 根据 Alexa.com 在 2007 年 12 月采集的数据。

Download at http://www.pin5i.com/

预测，并从中抽象出所需的信息。这样就包含了智能的三个要素。

1.3.2 Mashup

在 IBM DeveloperWorks 网站（http://www.ibm.com/developerworks/spaces/mashups）上有一节是专门介绍 mashup（混搭）的，其中给出了一个合适的定义："Mashup 是一种激动人心的交互式 Web 应用，它从外部数据源获得内容，然后创建出全新的富有创意的服务。"换句话说，就是从其他网站"借用"内容和用户界面元素来构建自己的网站。ProgramableWeb（http://www.programmableweb.com）就是一个很有意思的 mashup 网站。下面以这个网站为起点来看看 mashup 的世界吧（见图 1.3）。

图 1.3 访问类似 ProgrammableWeb 的网站，更多地了解 mashup

在这里，mashup 的重要性在于尽管它们也是基于内容聚合的，但与社交网络的网站不一样，mashup 网站自身并不拥有这些内容——至少其中绝大部分是不属于其本身的。这些内容存放在不同的地方，从各种来源将其收集在一起，根据用户与应用之间的交互情况，以独特的方式呈现出来。

但并不是所有的 mashup 网站都是智能的。要构建智能化的 mashup 网站，我们

Download at http://www.pin5i.com/

需要协调采集的内容之间的差异，或是找到它们的相似之处。相应的，内容的整合和分类都需要参考结构来说明内容的含义，也需要用算法找到这些内容中所包含的参考结构元素，或是对这些来源各异的内容按照不同的显示需求进行分类。

1.3.3 门户网站

门户网站（尤其是新闻类门户网站）是另一类深受本书介绍的技术所影响的 Web 应用。从门户网站的定义可知，这类网站是通向散布在互联网上的各种内容的门户。如果是企业网站，则是通向企业内局域网上各种内容的门户。门户网站中所聚合的内容散布在互联网的各个角落，但却是可以访问的。

这类网站最典型的例子就是 Google News（http://news.google.com）。它从成千上万的网站收集新闻，并自动将相似的新闻放在同一个标题下。而且，每一组相似的新闻都被放入预先设置的新闻分类中，例如，Business、Health、World、Sci/Tech 等（见图 1.4）。

图 1.4 Google News 网站是一个智能的门户应用

用户甚至还可以定义自己的分类，把自己感兴趣的新闻放入其中。组成这一切

Download at http://www.pin5i.com/

的依然是聚合的内容和相关的参考结构，加上能自动或半自动地完成这些工作的算法。

OpenSocial（http://code.google.com/apis/opensocial/）是一个很有前途的，为门户网站尤其是社交类门户网站加入智能元素的项目。围绕着这个项目还诞生了其他的一些项目，例如 Apache 的 Shindig 项目。OpenSocial 的基本目标是构建统一的 API，使得开发出来的网站可以方便地跟其他网站进行交互，包括 Engage、Friendster、hi5、Hyves、imeem、LinkedIn、MySpace、Ning、Oracle、orkut、Plaxo、Salesforce、Six Apart、Tianji、Viadeo 和 XING 等诸多网站，这份列表还在快速地增长。

1.3.4 维基

维基百科（Wikipedia）就不需要做过多的介绍了，很多读者都应该访问过或听说过这个网站。这个维基网站常年位列访问量最大的十个网站之中。所谓维基（wiki），就是互联网上的知识库。互联网上的很多社区都在使用维基分享知识，很多企业也在内部使用维基。

这类网站显然是内容聚合器。此外，还有不少此类网站在页面的创建流程中提供了对内容进行注释的功能。在维基百科中，用户可以对文章进行分类，并且链接到同一个话题的其他文章。本书所介绍的技术在维基网站中也大有用武之地。例如，构建一个能对网页自动进行分类的维基网站。在维基中，如果有其他的维基页面能解释当前维基页面中的某个术语或主题，这两个页面就应该相互链接在一起。所以，维基网站要能告诉用户当前页面中的哪些术语可以链接到其他页面。最后，页面之间自然而然的链接为高级的搜索（第 2 章）、聚类（第 4 章）或其他的分析手段提供了丰富的素材。

1.3.5 文件分享网站

YouTube 无疑是互联网文件分享网站的典型，此外，同类网站如 RapidShare（http://www.rapidshare.com）和 MegaUpload（http://www.megaupload.com）也拥有不少用户。这些网站的一个共同特点就是它们的内容是二进制的视频或音频文件。在大部分情况下，相对于基于文本的网站，这类网站所提供的信息的体积都比较大，需要处理的数据量的显著增加也给智能应用带来了很大的挑战。

此外，智能应用中两个最难的问题（从业务的角度看，也是最有意思的问题）

Download at http://www.pin5i.com/

就与二进制信息密切相关，这两个难题就是音频处理和模式识别。例如，Clearspring（http://www.clearspring.com/） 和 ScanScout （http://www.scanscout.com/）两家公司帮助广告商提升品牌的曝光度，增加广告的受众。ScanScout 为广告商提供的服务是利用他们在 MySpace、Facebook、Google 和 Yahoo!等 25 个网站上的小程序智能化地发布广告。

前面提到的模式同样也能在这类网站中找到。内容聚合通常需要对内容进行分类，需要利用算法从内容中获取价值。我们也希望能根据预先定义的主题对二进制文件进行分类——“Autos & Vehi-cles”、“Education”、“Entertainment”、“Politics”等（见图 1.5）。

10

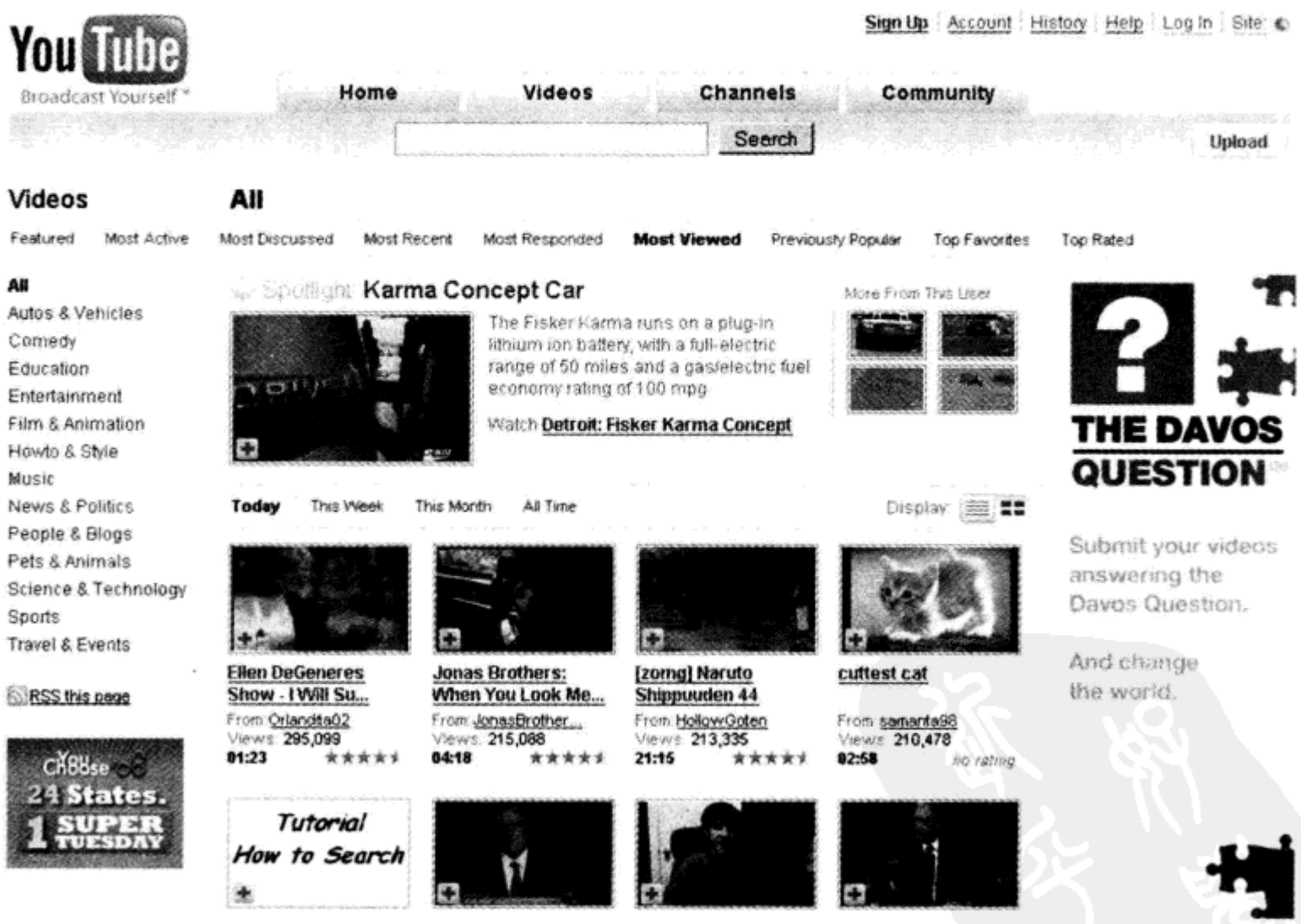


图 1.5 YouTube 对视频分类，在左侧的面板中显示了内容的分类方式

与其他的智能应用案例一样，这些分类也是层次化的。例如，“Autos & Vehicles”可以进一步地分为不同的子类：“Sedan”、“Trucks”、“Luxury”、“SUV”等。

Download at http://www.pin5i.com/

1.3.6 网络游戏

大型的多人在线游戏具有构建智能的所有要素。它们聚合了丰富的内容，以及能反应其中关联的参考结构，肯定也会用到本书介绍的算法，让游戏更加有深度，更加富有乐趣。能理解人类玩家所输入的信息的 NPC 角色可以让玩家获得更好的游戏体验，增加游戏的趣味性。

智能技术在网络游戏中大有用武之地，而且随着游戏可用的计算能力的增加和玩家对游戏复杂性和新创意的日益热衷，智能技术将成为在网络游戏竞争中获得差异化优势的关键。本书第 4～6 章中介绍的技术，以及附录中提供的资料都能直接应用到网络游戏中。

1.4 如何构建智能应用?

我们已经说明了在应用中嵌入智能会带来哪些好处，也介绍了受益于智能技术的各个领域，在这些领域中智能技术极大地提升了用户的使用体验，增加了用户从应用中获得的价值。现在，接下来的问题自然是“怎么才能构建自己的智能应用？”

本书都是在介绍智能组件的设计和实现，但为了能更好地利用本书，在构建智能应用之前，首先还要解决两个问题。

第一个问题是审查应用的功能。用户用这个应用干什么？该应用是怎样吸引客户和增加商业价值的？在这里，我们主要关注一些与本书将要介绍的算法有关的具体问题。对于不同的应用，这些问题的重要性也不尽相同。但无论如何，这些问题会帮助你找到应用中智能组件能发挥出最大价值的地方。

第二个问题是关于数据的。对于每个应用，数据可以来自应用内部（应用内随时可以使用的），也可以来自外部。首先检查内部数据，比较理想的情况是已经有了所需的数据，随时可以开工。反之，有可能还需要通过某种方法从用户那里获得更多的数据。例如，可能需要在网页的用户界面中加入“五星”评价功能，这样才能根据用户的评分构建推荐引擎。

此外，还有可能需要从外部来源获取更多的数据。有很多方法可以实现这个目的，在这里无法一一罗列，但我们会介绍四种广泛使用的方法，从技术角度来说，它们也是稳定性最好的。各种数据采集方法的细节则需要你查阅相关的文献。

Download at http://www.pin5i.com/

1.4.1 检查功能和数据

用户首先要找到一些受益于智能行为的用例，不同的应用显然会在不同的地方受益于智能行为。下面这些问题会帮助你找出这些用例。

- 应用所提供的内容是否来自不同的地方？
- 有基于向导的工作流程吗？
- 能否处理自由文本 ？
- 是否涉及某种类型的报表？
- 是否要处理诸如地图一类的地理信息？
- 有搜索功能吗？
- 用户之间分享内容吗？
- 欺诈检测是否很重要？
- 身份验证是否很重要？
- 需要根据规则自动做出决策吗？

当然，这并不是一份完整的列表，但其中已列出了一些可能的情况。只要有一个问题的答案是肯定的，应用就必然能从本书所涉及的技术中获益。

让我们用一个假想的应用来看看搜索应用中常见的用例。在几乎所有的应用中，用户都可以进行站内搜索。假设用户可以在我们的应用中购买不同类别的商品，也可以搜索他们想购买的商品。通常，这个功能都是直接利用 SQL 查询实现的。所有匹配用户查询的商品都会呈现到用户眼前。这没什么问题，但数据库服务器可能忽略了一个事实：在匹配用户查询的商品中，恰好有一种商品在降价促销。如果能够据此实现第 2 章介绍的排序方法和第 3 章介绍的推荐方法，无疑会提升用户的使用体验。

1.4.2 获取更多的数据

在很多情况下，已有的数据就足以为自己的应用构建出有价值的智能。但在某些情况下，在应用中构建有价值的智能组件可能需要访问外部的信息。图 1.6 是 mashup 网站 HousingMaps（http://www.housingmaps.com）的截图。结合 craigslist（http://www.craigslist.com）的房屋数据和 Google 的地图服务（http://code.google.com/apis/maps/index.html），用户可以在这个网站中查看任何地理区域内待售或待租的房屋。

Download at http://www.pin5i.com/

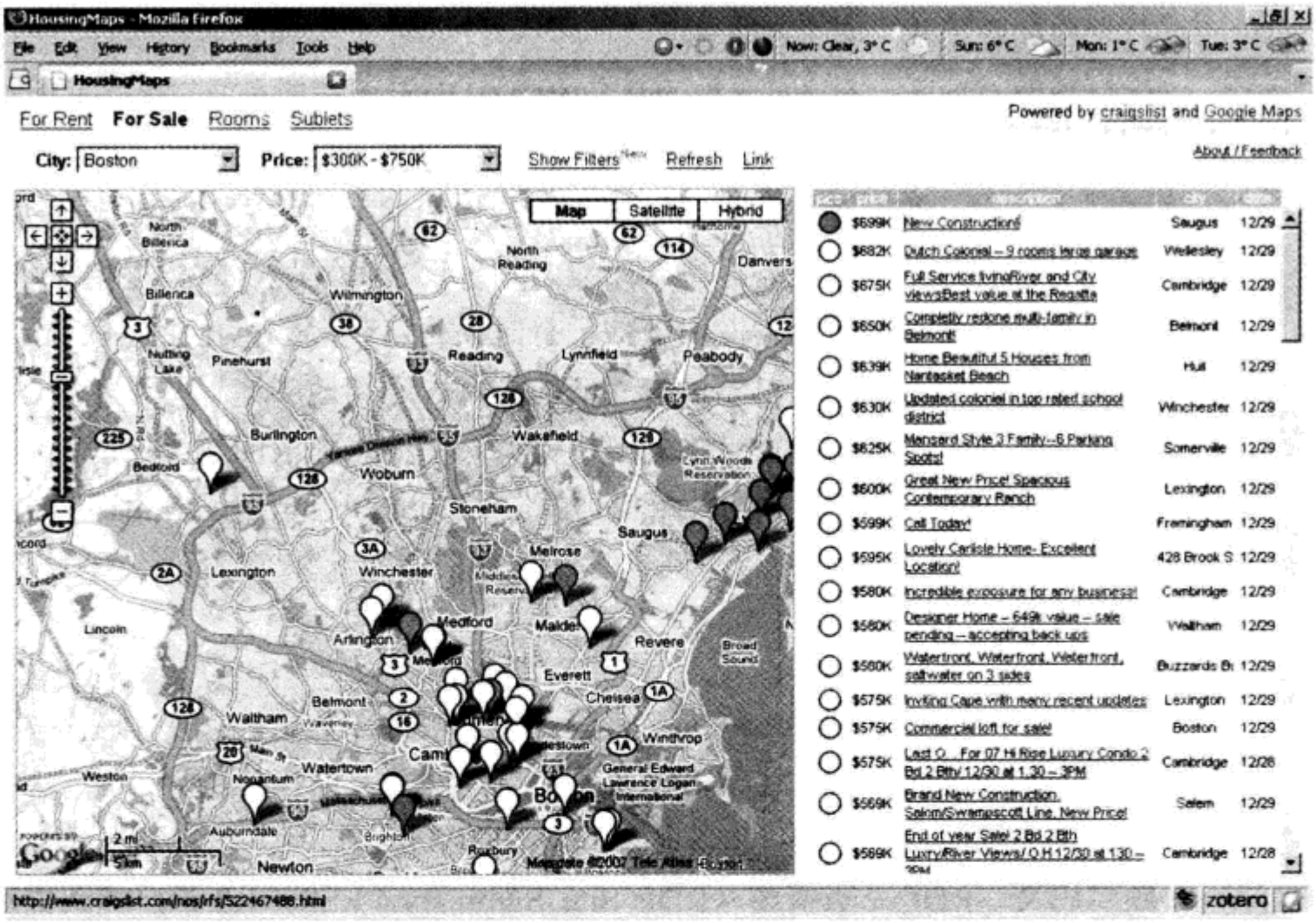


图 1.6 截图中是一份结合了 Google 地图服务的 craigslist 房屋信息列表（来自 http://www.housingmaps.com）

类似的，新闻网站也可以把新闻及事发地结合在一起，能从地图中获取位置信息，这对任何应用而言都是一种进步。但这并不意味着这是一个智能应用，除非能对这些从地图中获取的信息做一些智能的处理。

地图是一个很好的获取外部信息的例子，但在网络上还有很多其他的信息。让我们看看如何利用这些技术。

采集和信息抓取

爬虫（crawler）也被称为蜘蛛（spider），是用于从互联网上获取公开内容的程序。爬虫通常会访问一份 URL 列表，然后跟踪其中的每个链接。这个过程会不断地重复，重复次数被称为爬行深度（depth of crawling）。当爬虫访问完一个页面后，会将其内容存放到本地以做进一步的处理。这样可以收集大量的数据，但很快就会遇到存储和版权方面的问题。因此，在采集数据的时候一定要谨慎。在第 2 章中，我们会展示一个爬虫的实现。在附录中还有一个网络采集的整体介绍，其中总结了我

Download at http://www.pin5i.com/

们自己实现的爬虫和一些开源实现。

信息抓取（screen scraping）指的是从 HTML 网页中获取信息。这看上去很简单，但其实很烦琐。举个例子，要构建一个专用于外出就餐的搜索引擎（例如 http://www.foodiebytes.com），第一项工作就是要从各个餐厅的网页中获取菜单。信息抓取本身就能从本书介绍的技术中获益。在餐厅搜索引擎这个例子里，需要根据有用餐经验的食客们的评价对餐厅的优劣做出评估。有时候可以获得数值化的评分，但更多的情况是食客们的评论都是用自然语言写就的文章。逐篇地阅读这些评论，然后据此对餐厅进行排名显然是一个很糟糕的解决方案。在信息抓取的过程中应用智能技术可以实现评论的自动分类，并据此给餐厅打分，Boorah（http://www.boorah.com）就是一个例子。

RSS 源

网站聚合（website syndication）是另一种获取外部数据的方法，这种方法无须使用爬虫反复地访问网站。通常，相对于常见的网页，结构化的聚合内容对机器更友好。常见的源格式有三种：RSS 1.0、RSS 2.0 和 Atom。

顾名思义，RDF 站点摘要（RDF Site Summary，RSS）1.0 源自资源描述框架[4]（Resource Description Framework，RDF），主要目的是让机器和人都可以理解网络上的信息。但是，人类可以推测内容的语义（单词或短语在特定语境中的含义），而机器则很难做到这一点。引入 RDF 的目的就是要对网络上的内容做出语义化的解释，从而可以根据具体的需求，从中解析出有用的数据。RSS 1.0 的规范可在 http://web.resource.org/rss/1.0/站点找到。

14

简易资讯聚合（Really Simple Syndication，RSS 2.0）是基于 Netscape 的富站点摘要（Rich Site Summary）0.91 的，至少也是重载了后者的首字母缩写 RSS，它的目的是简化基于 RDF 的格式。它采用了一种基于 XML 格式的聚合专用语言，但没有 XML 命名空间和 RDF 引用。现在，几乎所有的主流网站都提供了 RSS 2.0，而且对个人用户和非盈利机构的非商业性使用都是免费的。Yahoo!的 RSS 源站点（http://developer.yahoo.com/rss）上有很多相关的介绍，你可以在 http://cyber.law.harvard.edu/rss 上找到 RSS 2.0 的规范和其他相关信息。

最后要介绍的是基于 Atom 的聚合。RSS 2.0 中存在的一些缺陷使得人们开始开发互联网工程任务组（Internet Engineering Task Force，IETF）的 RFC 4287

4 http://www.w3.org/RDF.

Download at http://www.pin5i.com/

（http://tools.ietf.org/html/rfc4287）中所描述的标准。Atom 并不是基于 RDF 的，它兼具 RSS 1.0 的灵活性和 RSS 2.0 的简洁。从本质上说，它是实现现有标准的功能和尽可能满足向后兼容性的妥协产物。然而，Atom 已经像 RSS 2.0 一样广受欢迎。大部分网络聚合器（例如 Yahoo!和 Google）都同时提供这两种格式的源。在 IBM Developer Works 上可以找到更多有关 Atom 聚合格式的信息：http://www.ibm.com/developerworks/xml/standards/x-atomspec.html。

RESTFUL 服务

REST（Representational State Transfer）是 Roy T. Fielding[5]在其博士论文中提出的。这是一种为分布式、超链接的媒介构建应用的软件架构风格。REST 是一种无状态的 C/S 架构，它将每个服务都映射成一个 URL。如果非功能性需求不是很复杂，而且不需要跟服务提供商签订正式的协议，REST 就是一个便捷地访问互联网上各种服务的不错选择。关于这种重要技术的更多信息，可以参考 Leonard Richardson 和 Sam Ruby 撰写的 *RESTful Web Services* 一书。

很多网站都提供了 RESTful 服务，你可以在自己的应用中使用这些服务。Digg 提供了能接受 REST 请求的 API（http://apidoc.digg.com/），并且提供 XML、JSON、JavaScript 和序列化 PHP 等多种不同类型的反馈。通过这个 API，可以获取符合各种要求的新闻、用户、好友或用户的粉丝等。

Facebook API 也是类 REST 风格的接口。所以，无论是什么编程语言，都可以与这个激动人心的平台进行交流。你要做的只是把 HTTP 的 GET 或 POST 请求发送给 Facebook API REST 服务器。Facebook API 有很好的文档，本书稍后也会用到这些 API。详细信息请参考 http://wiki.developers.facebook.com/index.php/API。

Web 服务

Web 服务是用于应用之间相互交流的 API，它有大量的 Web 服务框架，其中很大一部分是开源的。Apache Axis（http://ws.apache.org/axis/）是一个简单访问对象协议（Simple Access Object Protocol，SOAP）的开源实现，该协议“可以用于在无中心分布式环境中的节点间交换结构化、类型化的信息。”[6]Apache Axis 是一个很流行的框架，在第 2 版中做了全新的设计，Apache Axis2 支持 SOAP 1.1 和 SOAP 1.2，

5 http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
6 http://www.w3.org/TR/soap12-parto/_L1153.

Download at http://www.pin5i.com/

以及使用广泛的 REST 风格的 Web 服务，还有大量的其他功能。

另一个值得一提的 Apache 项目是 Apache CXF（http://incubator.apache.org/cxf/），这是 IONA 的 Celtix 和 Codehaus XFire 结合的产物。Apache CXF 支持的标准有：JAX-WS 2.0、JAX-WSA、JSR-181、SAAJ、SOAP 1.1 和 1.2、WS-I Basic Profile、WS-Security、WS-Addressing、WS-RM、WS-Policy、WSDL 1.1 和 2.0。它还支持多种传输机制、绑定和格式。如果考虑使用 Web 服务，就应该看看这个项目。

除了这些数量众多的 Web 服务框架，还有更多的 Web 服务提供商，几乎每个公司都在使用 Web 服务整合各种各样的应用，这些应用的功能各不相同，所使用的技术也千差万别。这种情况有可能是公司合并的结果，也有可能是因为大型公司内部的开发工作协调不到位。纵向地看，几乎所有的大型金融投资机构都在使用 Web 服务来实现无缝整合。Xignite（http://preview.xignite.com/Default.aspx）就提供了金融方面的各种 Web 服务，SAP、甲骨文、微软等软件巨头也为 Web 服务提供支持。总之，基于 Web 服务的整合无处不在，而且作为主要的整合技术，它在智能应用的设计中也是重要的基础构件。

至此，你一定已经对如何改进自己的应用有了一些想法，或者已经对下一次激动人心的创业有了新的创意。我们已经确保获得了所有必要的数据，至少我们能访问到这些数据。现在来看看这些我们将要添加到应用中的智能，以及它们跟其他已熟悉的概念之间的关系。

1.5 机器学习、数据挖掘及其他

在本书中，我们不断地提及“智能”，到底什么是“智能”？我们说的是人工智能吗？或者是机器学习？它跟数据挖掘和软计算有什么关系？在学术界，对于本书中所介绍的内容的精确定义也已经争论了好几年。从实践的角度看，这些概念并没有实质性的区别，更多的是指应用环境的不同。本书融合了上述所有领域的精华，让我们逐个来看看吧。

以其首字母缩写 AI 而广为人知的人工智能是一个起源于 20 世纪 50 年代的计算领域。最初，AI 的目标是极具野心的，试图开发出能像人类一样思考的机器（Russell and Norvig, 2002; Buchanan, 2005）。随着时间的推移，目标变得更加可行和具体。遥不可及的目标不得不屈服于残酷的现实，但是我们之前提及的很多领域都源自人工

Download at http://www.pin5i.com/

智能，比如机器学习、数据挖掘、软计算等。

现在，即使是最先进的计算智能系统也无法理解四岁小孩所阅读的小故事。所以，如果我们无法让计算机“思考”，是否能让计算机“学习”呢？可以教会计算机根据动物的特征来判断物种吗？识别不良的次级房贷呢？更复杂的事情，诸如语音识别并用自然语言答复，计算机能做到吗？所有这些问题的答案都是肯定的。然而你有可能会好奇，“这些问题到底说明了什么？”。解决上面这些问题，一个最简单的方法就是在计算机中构建一个庞大的数据表，把所有可能的问题的答案都存放在里面，然后在回答问题时只需在表中搜索现成的答案即可。

当然，这个查询数据表的方法是可行的，但其中也有一些问题。首先，在实际的产品系统中，包含所有问题和答案的表肯定是非常大的。所以，从效率的角度考虑，这肯定不是一个最优的解决方案。其次，如果数据库中没有某个问题的答案，就无法给出回答，如果真有用户问这些问题，你就只能用“敏感词”来搪塞他了。最后，还必须安排人来构建和维护这个查询表，而且随着表的增长，所需的人数也会不断增长，这估计会让公司的财务部门比较恼火。所以，查询表并不是一个好办法，我们需要更好的解决方案。

机器学习指的是软件系统能从已有的经验中抽象出普遍的规则，然后利用这些规则回答各种问题，包括曾经遇到过的和不曾见过的。有些算法对于人类是透明的，意思就是说，人类可以理解算法所抽象出来的规则。透明算法的典型例子有决策树，以及所有基于规则的学习方法。还有一类算法对人类是不透明的，例如，神经网络和支持向量机（SVM）就属于这一类算法。

时刻要记住，跟人类智能一样，机器智能也是不可靠的。在智能应用领域，你将学会如何处理不确定性和模糊性。就像真实世界一样，所有问题的答案都有一个可信度，而不是绝对可靠的。虽然在我们的日常生活中，我们总是简单地假设某些事情是一定会发生的。正因为如此，在使用智能应用时，我们需要解决可信度、有效性以及错误代价等方面的问题。

1.6 智能应用中八个常见的误区

到此，我们已经介绍了与智能应用相关的基础知识，现在你应该已经对什么是智能应用，以及应该如何使用智能应用有了一个整体的了解。你很有可能迫不及待

Download at http://www.pin5i.com/

地想要开始写代码，我们不会让你失望的。除了本章之外，每一章都会给你介绍新的有价值的代码。

但在开始进入激动人心且充满经济利益的智能应用世界之前，先来看一些带有智能功能的项目中常见的错误，或者说是误区。你可能熟悉分布式计算的八大误区（如果没有看过，可以参考 Van den Hoogen 的业界评论），其中列举了第一次开发分布式应用的程序员对分布式应用的常见误区。同样，根据这个传统，我们在这里也会介绍八个智能应用开发中常见的误区。

1.6.1 误区 1：数据是可靠的

很多因素会导致你获取的数据不可靠。在考虑具体的智能算法解决方案之前，首先就要判断数据是否可信。如果数据有问题，就算是最聪明的人，通常也会得出错误的结论。

可能导致数据出错的原因有很多，在此不可能一一列举，下面列出了部分有代表性的可能导致数据出错的因素：

- 在开发过程中所使用的数据不能代表产品环境中的数据。例如，对某个社交网络中的用户按身高分类“高”、“一般”和“矮”。如果开发阶段所使用的数据中最矮的用户的身高是 1 米 84，那就有可能会得出“你太矮了，才 1 米 84”的可笑结论。
- 数据中可能含有缺失值。实际上，除非数据是人造的，否则数据就肯定会含有缺失值。缺失值的处理非常需要技巧，通常，既可以保持缺失值状态不变，也可以用某种默认值或是计算得到的值来填充。两种情况都可能导致不稳定的实现。
- 数据可能会改变。数据库策略可能会变，或者数据库中数据的语义也有可能发生变化。
- 数据没有规范化。假如我们关注的是每个人的体重。为了能根据体重得到有意义的结论，所有的单位应该是统一的：磅或是公斤，两者只能取其一，而不能是两者的混合。
- 所想的算法可能不适合数据。数据有不同的形式，也就是数据类型。有些数据集是数值类型的，有些则不是；有些数据集可以排序，有些则不能；有些数值的数据集是离散的（例如，房间中的人数），有些则是连续的（例如气温）。

Download at http://www.pin5i.com/

1.6.2 误区 2：计算能马上完成

任何解决方案的计算都需要时间，应用的反馈速度对于业务在经济方面的成功是至关重要的。不能盲目地假设应用能在限定的反馈时间内在所有的数据集上完成所有的计算，需要仔细测试算法在各种操作中的性能。

1.6.3 误区 3：不用考虑数据规模

当我们讨论智能应用时，规模是很重要的。数据的规模会给整个应用带来两方面的影响。其一，是在误区 2 中提及的反馈速度问题。其二，则是如何从海量数据中获得有意义的结果。如果只有 100 个用户，系统可能给用户推荐非常靠谱的电影或音乐。但同样的算法在面对 100000 个用户时，就有可能变得非常糟糕。

反之，在某些情况下，数据越多，应用就会变得越智能。所以，数据规模带来的影响是多方面的，你应该经常问自己：我有足够的数据吗？如果数据增加 10 倍，会给我的智能应用带来什么影响？

1.6.4 误区 4：不考虑解决方案的可扩展性

另一个与误区 2、3 有关，但又有区别的误区是认为只需要增加计算机的数量就能不断地扩展智能应用解决方案。不要轻易地认为任何一个解决方案都是可扩展的，有些算法是可扩展的，有些则不行。例如，我们要把数十亿条新闻新标题按相似性分成若干组，并不是所有的聚类算法（见第 4 章）都可以并行。因此，在应用的设计阶段就应该考虑到可扩展性。在某些情况下，可以分割数据，然后将算法应用到分割后的较小的数据上，从而实现并行。运气好的话，在设计中所选择的算法可能是有并行版本的，但因为算法在智能应用中的重要性，以至于很多基础设施和业务逻辑都是围绕算法展开的，所以在设计的开始阶段就要关注所选择的算法的可扩展性。

1.6.5 误区 5：随处使用同样的方法

如果能反复用同样的成熟技术解决与应用中的智能行为有关的各种问题，这无疑充满了诱惑。要尽力拒绝这种诱惑！我曾经见过有人试图用 Lucene 搜索引擎解决世界上所有的问题。如果你也在做同样的事情，请记住这个经验：如果你手上握着

Download at http://www.pin5i.com/

锤子，所有的东西在你眼里都像是一颗钉子。

智能应用软件跟其他的软件类似——都有一定的应用领域，也有一定的局限性。在将你所喜爱的解决方案应用到新领域之前，要彻底地进行测试。此外，对于每一个问题都应该从新的角度去观察。不同的算法或许能更加高效地解决新问题。

19

1.6.6 误区 6：总是能知道计算时间

此类误区的典型例子是涉及优化的问题。在某些应用中，参数的细微变化都会带来计算时间的大幅变动。人们总是希望在改变问题的参数后依然能够在限定的反馈时间内解决问题。如果只是计算地球上两个地理位置间的距离，计算时间当然与具体的地理位置没有关系，但这并不适用于所有的问题。在某些情况下，数据的细微变化就会造成计算时间剧烈变化，有时候甚至是几秒钟与几个小时的差别。

1.6.7 误区 7：复杂的模型更好

任何事情都是过犹不及的。我们首先应该从最简单的模型开始。然后在解决方案中通过逐步添加其他智能元素改进效果。KISS（Keep It Simple, Stupid）原则永远都是软件工程师的好伙伴。

1.6.8 误区 8：存在无偏见的模型

如果有人说过这样的话，原因只有两个：无知或偏见！在选择所需的模型和选择用于训练算法的数据时，就已经引入了偏见。在这里，我们无法从科学的角度深入地讨论学习系统中的偏见，但我们应该注意解决方案中的偏见总是倾向于我们模型的描述和我们的数据。换句话说，偏见使我们的方案受到已知的事实或是我们获得这些事实的方法的限制，而泛化则是试图从已知的事实中推断出未知的事情。

1.7 小结

在本章中，我们用一些基于实际网站的例子简单介绍了智能 Web 应用，并且从实践的角度给出了智能 Web 应用的定义，这也能成为我们的设计原则。这个定义分三个部分：（1）数据聚合；（2）参考结构；（3）具有学习能力，且能处理不确定性

Download at http://www.pin5i.com/

的算法。

我们通过六类不同的 Web 应用验证了这个定义。接下来，我们介绍了能用于手机或访问数据收集平台的技术，还介绍了后面几章要介绍的各种技术的背景知识，尤其是人工智能、机器学习、数据挖掘和软计算等相关领域。

20

最后，我们列出了实践中常见的八个设计误区，这些更多的是经验性的原则，而不是严谨的事实。我们相信这八条指导原则能为你节约很多时间，减少你的咖啡消耗量。接下来，本书会逐一介绍能为我们的应用添加智能行为的各种技术，每种技术的相关资料都会在各章的“To Do”小节中给出。

1.8 参考资料

Buchanan, B.G. “A (Very) Brief History of Artificial Intelligence.” *AI Magazine*. Volume 26. issue 4, 2005.

Gómez-Pérez, A., M. Fernández-López, O. Corcho. *Ontological Engineering: With Examples from the Areas of Knowledge Management, E-commerce and the Semantic Web*. Springer, 2005.

Hart, P.E., N.J. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” *IEEE Transactions on Systems Science and Cybernetics*. Volume 4, issue 2, 1968.

Hart, P.E., N.J. Nilsson, and B. Raphael. Correction to “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” *SIGART Newsletter* 37, 1972.

Richardson, L. and S. Ruby. *RESTful Web Services*. O'Reilly Media, 2007.

Russell, S. and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2002.

Van Den Hoogen, I. “Deutsch's fallacies, 10 Years After.” *Java Developer's Journal*, 2004. http://java.sys-con.com/read/38665.htm.

Download at http://www.pin5i.com/

2 搜索

本章内容提要：

- 用 Lucene 实现搜索
- 计算 PageRank 向量
- 大规模计算的约束

假如现在有一份文档清单，我们对其中与短语“Armageddon is near（世界末日临近）”有关的文档感兴趣，当然也可以是其他没这么恐怖的事情。如何解决这个问题？简单、粗暴的方法是读取每个文档，然后找到其中含有“Armageddon is near”的文档，甚至还可以统计每个文档中搜索词出现的次数，然后据此按降序对文档排序。这种方法叫做信息获取（Information Retrieval，IR），也叫做简单搜索。搜索并不是一种新功能，几乎每个应用都具有某种搜索功能，但智能搜索不同于普通的老式搜索。

让我们通过例子来说明普通的 IR 解决方案存在的诸多问题。例如，只要增加文档的数量或者文档的体积，算法的效率就会急剧下降，最终不可接受。庆幸的是，关于 IR 已经积累了相当丰富的知识，而且有相当灵活和稳定的库，这些库的伸缩性和性能也都很好。在 Java 编程语言中，最成功的 IR 库就是大概 10 年前 Doug Cutting 创建的 Lucene 项目。Lucence 对所有的文档进行索引，从而可以快速地搜索所需的文档，这无疑帮读者解决了 IR 问题。如果想要知道如何索引数据，如何引入搜索，

Download at http://www.pin5i.com/

如何对搜索结果进行排序、过滤和高亮显示，Otis Gospodnetic 和 Erik Hatcher 撰写的 *Lucence in Action*（Manning 公司出版）一书是必读的。

现在的搜索技术已不仅仅是索引。搜索引擎公司之间最激烈的竞争不在于索引，而在于链接分析、用户点击分析和自然语言处理。这些技术能增强搜索功能，有时还能做成数十亿美元的大生意，例如 Google。

在本章中，我们总结 Lucene 库的特性并演示如何使用。我们会介绍迄今为止最成功的链接分析算法——PageRank 算法，还会介绍如何利用概率技术实现用户点击分析。我们会把所有的这些技术结合在一起来改进搜索结果，这些材料都是按先后顺序介绍的，如果现在没有足够的时间，可以先阅读一部分，以后再读剩余部分。废话不多说了，让我们先收集一些文档，然后用 Lucene 在其中搜索不同的词语。

2.1 用 Lucene 实现搜索

基于 Lucene 的搜索是本章其他内容的基础。所以，在学习更高级的智能算法前，先要学习传统的 IR 步骤。在这里，我们会演示如何使用 Lucene 搜索文档，并介绍 Lucene 内部的工作原理，此外，还会介绍构建搜索引擎的基本步骤。

所要搜索的数据可以在数据库中、互联网上，或是应用可以访问的任何其他网络中。可以用爬虫从互联网收集数据，有很多免费的爬虫，但在本书中我们会使用专门为本书编写的爬虫，所使用的数据则是我们在 2006 年 11 月 6 日收集的一些网页，这样我们就能自由地修改这些文档，并观察这些改动对算法结果的影响。

这些网页已经经过整理，相当于是一个简化版的互联网，你可以在 data/ch02/ 目录中找到它们。知道这些文档的内容后，才可能知道算法的工作方式和原理。我们的 19 篇文档分别是（内容是随机选取的）：

- 七篇与商业新闻有关的文档，其中三篇是关于 Google 进军报纸广告业的内容，另有三篇主要是讨论 NVidia 的股票，最后一篇宽泛地谈及了股票价格与指数变化。
- 三篇关于 Lance Armstrong 尝试参加纽约马拉松比赛的文档。
- 四篇关于美国政治（具体地说是国会选举）的文档。
- 五篇关于国际新闻的文档，其中四篇是 Ortega 赢得尼加拉瓜大选的文档，一篇是讨论全球变暖的文档。

Download at http://www.pin5i.com/

Lucene 可以帮助我们分析、索引并搜索这些文档，当然不仅限于网页，也可以用于其他各种能转换成文本的文档。用来快速读取和存储网页的类叫 FetchAndProcessCrawler，这个类还可以用于从互联网采集数据。其构造函数有三个参数：

- 存放所采集的数据的基目录。
- 访问的链接的深度。
- 采集的文件数量的上限。

代码清单 2.1 展示了如何在 BeanShell 中使用这个类。

代码清单 2.1 网页的读取、索引和搜索

```
FetchAndProcessCrawler crawler =
➥ new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);
crawler.setDefaultUrls();←加载文件
crawler.run(); ←收集并处理内容
LuceneIndexer luceneIndexer =
➥ new LuceneIndexer(crawler.getRootDir());
luceneIndexer.run(); ←索引目录中的内容
MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());
oracle.search("armstrong",5); ←基于所建立的索引进行搜索
```

爬取和预处理的过程应该只需要几秒钟，完成之后就会在基目录中出现一个新目录。在这个例子中，基目录是 C:/iWeb2/data/ch02。新目录的名字则以字符串“crawl-”开始，结尾则是爬取的时间戳，单位是毫秒，例如，crawl-1200697910111。

你可以修改这些文件中的内容，或是添加其他的文件，然后重复上述预处理和索引的过程，观察搜索结果的变化。图 2.1 是代码清单 2.1 中的代码在 BeanShell 的执行过程中的截图，其中包含了搜索词“Armstrong”的搜索结果。

24

搜索的基本过程包括：加载、索引、搜索。这个过程简单得不能再简单了！但它到底是怎么工作的？每一步中有哪些必不可少的元素？

2.1.1 理解 Lucene 代码

下面来看看搜索执行过程中所发生的事情。FetchAndProcessCrawler 的任务就是采集数据并进行解析，处理结果存放在名为 processed 的子目录中。稍微看一下这个目录，对于每一组处理过的文档，都有四个相应的子目录——fetched、knownurls、pagelinks 和 processed。注意，我们把网页中的元数据和核心内容分开了，同时还提

Download at http://www.pin5i.com/

取出了网页间的链接，即外链（outlink）。FetchAndProcessCrawler 类中没有任何来自 Lucene API 的代码。

```
bsh % FetchAndProcessCrawler c =
new FetchAndProcessCrawler("c:/iWeb2/data/ch02",5,200);
bsh % c.setDefaultUrls();
bsh % c.run();
There are no unprocessed urls.
Timer (s): [Crawler fetched data] -> 5.5
Timer (s): [Crawler processed data]-> 0.485
bsh %
bsh % LuceneIndexer lidx = new LuceneIndexer(c.getRootDir());
bsh % lidx.run();
Starting the indexing ... Indexing completed!

bsh % MySearcher oracle = new MySearcher(lidx.getLuceneDir());
bsh % oracle.search("armstrong",5);

Search results using Lucene index scores:
Query: armstrong

Document Title: Lance Armstrong meets goal in painful marathon
debut
Document URL: file:/c:/iWeb2/data/ch02/sport-01.html ->
Relevance Score: 0.397706508636475

---------------------------------
Document Title: New York 'tour' Lance's toughest
Document URL: file:/c:/iWeb2/data/ch02/sport-03.html ->
Relevance Score: 0.312822639942169

---------------------------------
Document Title: New York City Marathon
Document URL: file:/c:/iWeb2/data/ch02/sport-02.html->
Relevance Score: 0.226110160350800

---------------------------------
```

图 2.1 用数行代码实现的网页采集、解析、分析、索引和搜索

接下来创建一个 LuceneIndexer 类的实例，然后调用它的 run()方法。这里用 Lucene 对已经处理过的数据进行索引。Lucene 的索引文件会放在另一个名为 lucene-index 的目录中。LuceneIndexer 类实际上是一个为了便于从 BeanShell 中调用 LuceneIndexBuilder 类而设计的包装器。LuceneIndexBuilder 类才是真正用到 Lucene API 的地方。图 2.2 展示了用于采集和索引文档的几个主要类的全部 UML 图。

Download at http://www.pin5i.com/

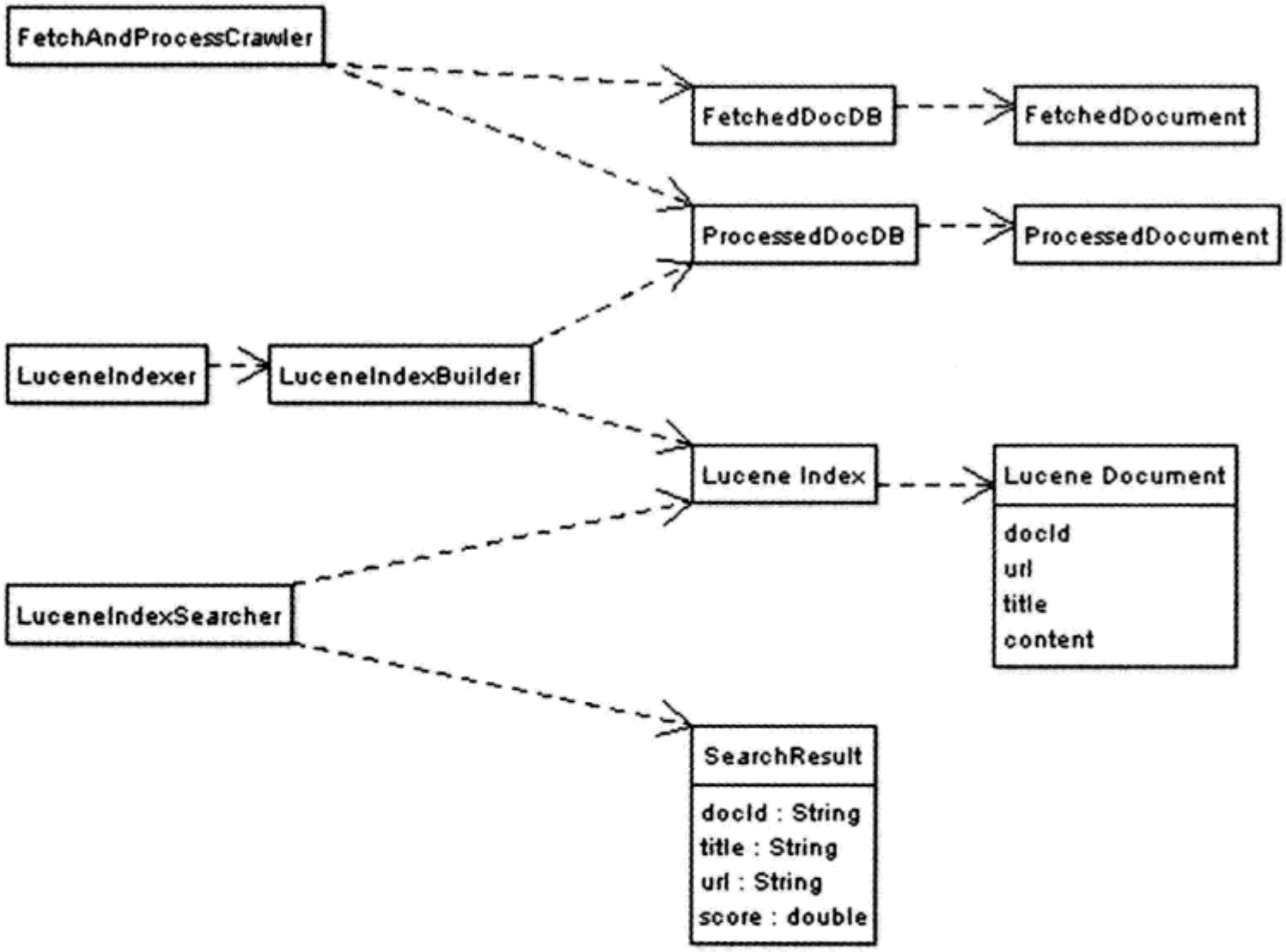


图 2.2 用于爬取、索引和搜索网页的类的 UML 图

代码清单 2.2 是 LuceneIndexBuilder 类的全部代码。

代码清单 2.2 LuceneIndexBuilder 创建 Lucene 索引

```
public class LuceneIndexBuilder implements CrawlDataProcessor {
  private File indexDir;
  public LuceneIndexBuilder(File indexDir) {
     this.indexDir = indexDir;
      try {
          IndexWriter indexWriter = ←创建 Lucene 索引
➥         new IndexWriter(indexDir, new StandardAnalyzer(), true);
          indexWriter.close();
} catch(IOException ioX) {
          throw new RuntimeException("Error: ", ioX);
      }
  }
```

Download at http://www.pin5i.com/


```
  public void run(CrawlData crawlData) {
      List<String> allGroups =
       crawlData.getProcessedDocsDB().getAllGroupIds();←获得所有文档的分组
      for(String groupId : allGroups) {
       buildLuceneIndex(groupId, crawlData.getProcessedDocsDB());
      }
  }
  private void buildLuceneIndex(String groupId,
➥ ProcessedDocsDB parsedDocsService) {
      try {
          List<String> docIdList =←获得分组的所有文档
parsedDocsService.getDocumentIds(groupId);
          IndexWriter indexWriter =
new IndexWriter(indexDir, new StandardAnalyzer(), false);
          for(String docId : docIdList) {←索引所有的文档
             indexDocument(indexWriter,
➥ parsedDocsService.loadDocument(docId));
          }
          indexWriter.close();
      } catch(IOException ioX) {
          throw new RuntimeException("Error: ", ioX);
      }
  }
  private void indexDocument(IndexWriter iw,
➥ ProcessedDocument parsedDoc) throws IOException {
      org.apache.lucene.document.Document doc =
➥ new org.apache.lucene.document.Document();
      doc.add(new Field("content", parsedDoc.getText(),
➥ Field.Store.NO, Field.Index.TOKENIZED));
      doc.add(new Field("url",
➥ parsedDoc.getDocumentURL().toExternalForm(),
➥ Field.Store.YES, Field.Index.NO));
      doc.add(new Field("docid", parsedDoc.getDocumentId(),
➥ Field.Store.YES, Field.Index.NO));
      doc.add(new Field("title", parsedDoc.getDocumentTitle(),
➥ Field.Store.YES, Field.Index.NO));
     doc.add(new Field("doctype", parsedDoc.getDocumentType(),
➥ Field.Store.YES,Field.Index.NO));
```

Download at http://www.pin5i.com/


```
        iw.addDocument(doc);
    }
}
```

27

Lucene 用 IndexWriter 类创建索引，这个类有很多构造函数，具体可以参考 Javadoc。我们的代码中所使用的构造函数有三个参数：

- 存放索引的目录。
- 所使用的分析器——本节稍后会介绍分析器。
- 布尔变量，决定是否要覆盖已有的目录。

如代码清单 2.2 所示，我们逐一处理爬虫所积攒的文档。第一组文档是初始 URL 清单中的网页。第二组文档则是在读取初始 URL 清单的内容时所发现的文档。第三组文档则是第二组文档中所发现的 URL 所指向的文档，以此类推。要注意，如果改变了 BasicWebCrawler 类的 maxBatchSize 参数，这些目录的结构也会随之发生变化。只有将该参数设成足够大的值，才能确保目录结构不发生变化，在这里我们将其设为 50。

在使用我们的爬虫从互联网采集海量数据时，这种目录结构会很有用。对于本书所使用的较为简单的网页结构，如果只用 FetchAndProcessCrawler 类的 addUrl 方法加入其中的部分 URL，让爬虫自己去发现其他文件的话，就能看出分组的效果。

我们会索引每组中各个文档的内容。如代码清单 2.2 底部所示，这是在 indexDocument 方法中完成的。Lucene 的 Document 类封装了所采集的文档，从而能将它们添加到索引中，这个类不仅能用于封装网页，还能用于电子邮件、PDF 文件以及其他所有能被解析并转换成纯文本的文档。每一个 Document 类的实例都是一个由若干个域组成的虚拟文档。现在用我们的分解方法处理所采集的文档，从而为每个文档创建若干个 Field 实例。

- content 域：文档的文本内容，删除了格式化标签和其他注释。这些文件位于 processed/1/txt 子目录中。
- url 域：用于获取文件的 URL。
- docid 域：文档独一无二的标识符。
- title 域：文件的标题。
- doctype 域：文件的类型，例如 HTML 或微软 Word 文档。

每个文档的 content 域都会被索引，但不会存放在索引文件中；其他域都会存放在索引文件中，但不会被索引。尽管我们的目的是对文档的内容进行查询，但我们

Download at http://www.pin5i.com/

需要从索引文件中获得文档的 URL、ID 和标题。

这是一种常用的方法。通常在索引文件中只需要存放一些能识别搜索结果的指针，如果没有特别的原因（例如，需要随时获得部分内容，而原始来源不能直接访问），不需要直接把内容放在索引文件中。在这里要注意索引过程中所创建的文件的大小。 <28

我们用 MySearcher 类搜索新建的索引。代码清单 2.3 展示了这个类的代码，其构造函数只有一个参数——存放 Lucene 索引的目录，接着就可以使用 search 方法进行搜索。search 方法有两个参数：

- 需要在索引中查询的字符串。
- 想要获取的文档数量的上限。

代码清单 2.3 MySearcher：根据 Lucene 的索引获得搜索结果

```
public class MySearcher {
  private static final Logger log =
➥ Logger.getLogger(MySearcher.class);
  private String indexDir;
  public MySearcher(String indexDir) {
    this.indexDir = indexDir;
  }
  public SearchResult[] search(String query, int numberOfMatches) {
    SearchResult[] docResults = new SearchResult[0];
    IndexSearcher is = null;
    try {
      is = new IndexSearcher(FSDirectory.getDirectory(indexDir));←打开 Lucene 索引
    } catch (IOException ioX) {
      log.error(ioX.getMessage());
    }
  QueryParser qp = new QueryParser("content",←创建查询解析器
                          new StandardAnalyzer());
    Query q = null;
    try {
      q = qp.parse(query); ←把文本查询转换成 Lucene 查询
    } catch (ParseException pX) {
      log.error(pX.getMessage());
    }
    Hits hits = null;
```

Download at http://www.pin5i.com/


```
try {
  hits = is.search(q); ←搜索索引
  int n = Math.min(hits.length(), numberOfMatches);
  docResults = new SearchResult[n];
29  for (int i = 0; i < n; i++) {←收集前 N 个搜索结果
    docResults[i] = new SearchResult(hits.doc(i).get("docid"),
                                     hits.doc(i).get("doctype"),
                                     hits.doc(i).get("title"),
                                     hits.doc(i).get("url"),
                                     hits.score(i)); ←第 i 个文档的分数
    // 报告结果
    System.out.println(docResults[i].print());
  }
  is.close();
 } catch (IOException ioX) {
  log.error(ioX.getMessage());
 }
 return docResults;
}
}
```

下面看看代码清单 2.3 中的各个步骤：

1．用 Lucene 的 IndexSearcher 类的实例打开要搜索的索引。

2．创建一个 Lucene 的 QueryParser 类的实例，参数是要搜索的域和分析查询文本所必需的分词器。

3．用 QueryParser 的 parse 方法将人类语言的查询转换成一个 Lucene 能理解的 Query 实例。

4．搜索索引，结果以 Lucene 的 Hits 对象的形式返回。

5．遍历前 *n* 个结果，将其收集在我们自己的 SearchResult 对象中。注意，Lucene 的 Hits 对象中只含有指向文档的引用。我们通过这些引用收集所需的域，例如，调用 hits.doc(i).get("url")会返回我们存放在索引中的 URL。

6．每个文档的相关性分数（relevance score）也会被记录，这个分数的范围是 0～1。

所有的这些东西构成了我们的实现。现在往回退一步，看看基于索引的搜索的全貌，这有助于理解基于索引的搜索引擎中各个元素的作用，也为进一步讨论高级

Download at http://www.pin5i.com/

的搜索功能做好准备。

2.1.2 搜索的基本步骤

如果我们能回到过去（1998 年），构建一个搜索引擎的基本步骤是什么？1998 年的基本步骤与现在没有什么区别，现在只是提高了每一步的效率和计算性能。图 2.3 描述了传统搜索的基本步骤：

- 爬取
- 解析
- 分析
- 索引
- 搜索

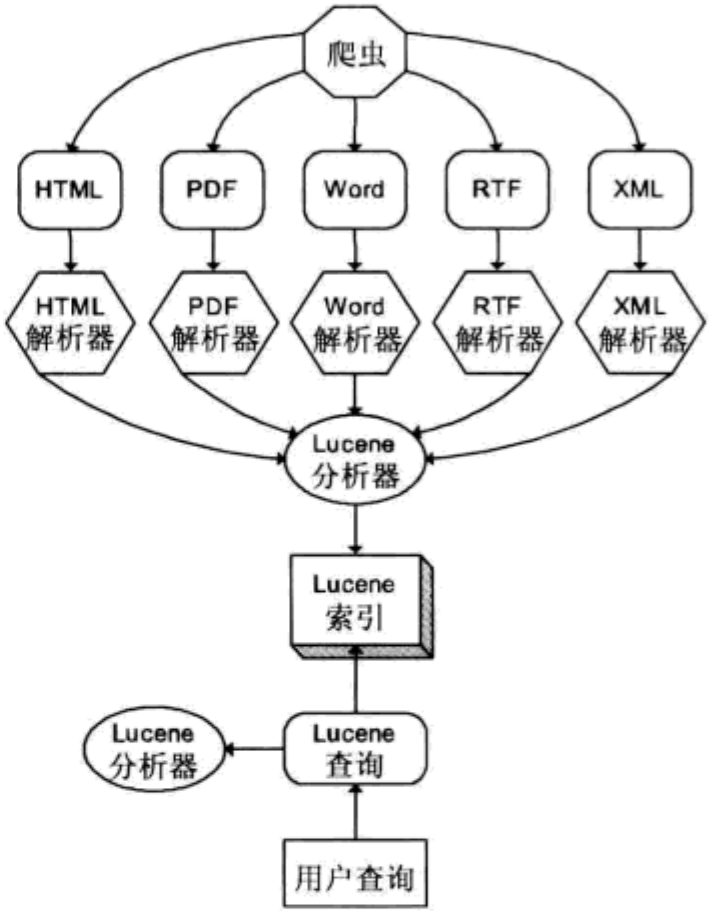


图 2.3 搜索不同格式的文档的基本步骤

爬取指的是收集要搜索的文档。如果文档已经收集好了，就可以略过这一步。解析是将各种文档（XML、HTML、Word、PDF）转换成统一的结构，即索引中的各个域，这些域都是纯文本形式的。在我们的例子中使用了 NekoHTML 项目的代码。NekoHTML 有一个简单的 HTML 解析器，能扫描 HTML 文件，修复 HTML 文件中的错误，添加缺失的元素，自动用结束标签关闭元素并且能处理缺失的行内元素标签。NekoHTML 稳定、速度快，适用于大部分的网站，但如果要爬取某些特殊的网站，可能还是需要自己编写的解析器。

如果要索引 PDF 文件，可以用 PDFBox 项目（http://www.pdfbox.org/）的代码。它是以 BSD 协议发布的，有丰富的文档。利用 PDFBox 中的 LucenePDFDocument 类，开发人员只需要编写一行代码就能得到一个 Lucene 的 Document 对象：

```
Document doc = LucenePDFDocument.convertDocument(File file)
```

在 Javadoc 中可以找到更多的信息。与 PDF 文档类似，也有针对 Word 文档的解析器。例如，Apache POI 项目（http://poi.apache.org/）所提供的 API 就可以用纯 Java 操作基于微软 OLE2 Compound Document 格式的文档。此外，TextMining 项目（http://www.textmining.org/）也提供了用于从微软 Word 97、2000、XP 和 2003 文档

Download at http://www.pin5i.com/

中提取文本的 Java 库。

文档分析这个步骤很重要。在代码清单 2.2 和代码清单 2.3 中都在两个关键之处用到了 Lucene 的 StandardAnalyzer 类，但之前我们并没有对此进行讨论。如图 2.3 所示，解析器能从各个文档中提取出文本内容，但在这些文本内容被索引之前还要经过 Lucene 分析器的处理。分析器的工作是非常关键的，因为它负责的是对所要索引的文本进行分词。换句话说，就是找到其中重要的单词，忽略不重要的。如果忽略了不该忽略的单词，那么无论索引算法有多高级，都不可能搜索到所需的结果。

31

当然，分析器并不能自动从索引文件中选择合适的域。举个例子，在代码清单 2.2 中，我们就明确定义了要搜索的域。StandardAnalyzer 会处理 content 域，这也是唯一被索引的域。这个默认的分析器是 Lucene 内置的分析器中适用范围最广的。在分词的过程中，它能智能地处理字母、数字、缩写、公司名称、电子邮件地址、计算机主机名，甚至连中文、日文、韩文都能处理。

在 Lucene 的最新版（本书写作时的最新版是 2.3）中用到了一个 Java 编写的词法分析器，名叫 JFlex（http://jflex.de/）。Lucene 中的 StandardTokenizer 就是以 JFlex 为基础构建的基于语法的分词器。StandardAnalyzer 类中就使用了这个分词器。为说明分析器的重要性，可以将 StandardAnalyzer 换成 WhitespaceAnalyzer，然后观察结果分数的变化。Lucene 分析器有丰富的功能，例如添加缩写、修改停用词（在索引前一定要删除的词）以及处理非英语语言。在本书中，我们会一直使用 Lucene 分析器，甚至在与搜索无关的几章中也会用到。在处理文档时，识别出一段文本描述中独一无二的特点是至关重要的。因此，分析器在诸如垃圾邮件过滤、基于文本的推荐系统、企业或税务服务应用等有关领域都有广泛的应用。

Lucene 的索引这一步是最终用户看不到的，但 Lucene 在这方面也很强大。在一次索引中，Lucene 的 Documents 对象对应于不同的实体（例如电子邮件、记录、法律文件），不同实体的各个域中的内容是不同的。可以删除或更新从索引中产生的 Documents 对象。Lucene 在索引方面另一个有趣的功能是 boosting。所谓 boosting，就是给特定的文档赋予特定的重要性，即或高或低。在代码清单 2.2 的 indexDocument 方法中，可以添加如下语句：

```
if ( parsedDoc.getDocumentId().equals("g1-d14")) {
        doc.setBoost(2);
}
```

Download at http://www.pin5i.com/

从代码中可以找到这段语句，但被标注成“To Do”注释掉了。如果删除注释，编译代码，然后重新运行代码清单 2.1 中的脚本，就会发现原本排名最后的文档现在排到第一了。实际上是 boosting 增加了该文档每个 Field 的分数，准确地说是翻倍了。想要更加细致地调整搜索结果，也可以提升单个 Field 的分数。

用 Lucene 实现搜索是很简单的。前面已经展示了，只要两行代码就能使用我们的 MySearcher 包装器。尽管在代码清单 2.1 的例子中是搜索了一个简单的单词，但实际上 Lucene 的 QueryParser 类可以处理非常灵活的搜索表达式。有时，在创建 Lucene 的 Query 时可能会遇到麻烦。例如，要搜索“nasdaq index”，同时还要考虑到“nasdaq composite index”[1]。在后一种情况下，“nadsaq”和“index”被分开了。遇到这种问题时，可以用 slope 参数控制“nasdaq”和“index”间隔的单词的数量上限。在这个例子中，将该参数设为 1，就能获得所需的结果。关于 Lucene 的这个功能，以及其他强大的功能，我们推荐读者浏览 Lucene 的 API 并阅读有关文档。 32

2.2 为什么搜索不仅仅是索引？

现在，我们已经演示了如何用 Lucene 快速地索引文档，并对索引进行搜索，你应该也能感觉到 Lucene 的使用非常方便，效果也很好。由此，你可能会觉得“Lucene 已经如此的灵活高效，还有什么可担心的？”本节我们会解释为什么搜索不仅仅是索引，为什么索引只是搜索的一个必要步骤。在第 1 章中，我们曾提及其中的原因，但在本节中会对此深入探讨。首先，让我们在种子 URL 清单中添加一个新文档。代码清单 2.4 与代码清单 2.1 很相似，但其中有一个含有垃圾信息的 URL。

代码清单 2.4 读取、索引和搜索网页，部分网页为垃圾网页

```
FetchAndProcessCrawler crawler =
➥ new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);
crawler.setDefaultUrls();
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-01.html");←添加垃圾网页
crawler.run();
LuceneIndexer luceneIndexer =
➥ new LuceneIndexer(crawler.getRootDir());←构建 Lucene 索引
luceneIndexer.run();
```

1 nasdaq index 是纳斯达克指数；nasdaq composite inder 是纳斯达克成分指数，与纳斯达克指数是同一个意思。——译者注

Download at http://www.pin5i.com/


```
MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());←构建一个普通的
                                                                  搜索引擎
oracle.search("armstrong",5);
```

图 2.4 是“Armstrong”的搜索结果。可以看到，垃圾网页竟然排名第一。甚至还可以在 URL 列表中添加两三个或者更多的垃圾网页，真正的相关网页很快就会淹没在垃圾网页的海洋中。

互联网上的网页不同于数据库或硬盘上的文档，它是不受控制的。所以，精心炮制的欺诈网页可以让传统的 IR 技术毫无用处。如果搜索引擎仅仅依赖于传统的 IR 技术，那么在线学习和娱乐（我们的全国性在线体育活动）都将会成为泡影。现在，让我们进入一个崭新的世界：链接分析。为了快速精确地搜索像网页这样相互链接的文件，链接分析是最早（也是最重要的）被引入的一项技术。

33

```
bsh % oracle.search("armstrong",5);

Search results using Lucene index scores:
Query: armstrong

Document Title: Cheap medicine--low interest loans
Document URL: file:/c:/iWeb2/data/ch02/spam-01.html --> Relevance
Score: 0.591894507408142

-----------------------------------------------------------
Document Title: Lance Armstrong meets goal in painful marathon
debut
Document URL: file:/c:/iWeb2/data/ch02/sport-01.html -->
Relevance Score: 0.370989531278610

-----------------------------------------------------------
Document Title: New York 'tour' Lance's toughest
Document URL: file:/c:/iWeb2/data/ch02/sport-03.html -->
Relevance Score: 0.291807949542999

-----------------------------------------------------------
Document Title: New York City Marathon
Document URL: file:/c:/iWeb2/data/ch02/sport-02.html -->
Relevance Score: 0.210920616984367

-----------------------------------------------------------

bsh %
```

图 2.4　一个伪造的网页就极大地改变了“Armstrong”搜索结果的排序

正是链接分析技术让 Google 从无名小辈变成全球网络搜索巨头，并且还推动了很多其他领域的研究和发展。

Download at http://www.pin5i.com/

链接分析是互联网的一个结构特点。互联网的另一个特点是用户点击分析（user click analysis），这是行为特点。简而言之，用户点击分析指的是记录用户在浏览搜索页面时的点击行为，然后据此改进该用户以后的搜索中搜索结果的排序。这么做的依据是，在搜索结果中，用户更有可能去点击他认为正确的搜索结果。反之，用户不会去点击那些与其搜索词和搜索意图无关的网页。我们之所以强调这个概念，是因为这是一个与传统应用很不同的地方，传统的应用只能根据用户直接的输入做出反应。如果应用能够发现用户的意图，无疑是在迈向智能的道路上前进了一大步，这意味着应用可以从用户的行为中学习知识，而无须程序员把所有的答案一个个地从“后门”塞进去。

2.3 用链接分析改进搜索结果

为了让我们的搜索不局限于索引，下面介绍 Google 发明的链接分析算法——PageRank。PageRank 算法最早出现在 Sergey Brin 和 Larry Page 于 1998 年在世界万维网大会（World Wide Web 会议，WWW98）上发表的题为“The anatomy of a large-scale hypertextual Web search engine”的论文中。几乎在同一时间，IBM Almaden 的 Jon Kleinberg 提出了 HITS（Hypertext Induced Topic Search）算法。这两个算法都是链接分析模型，只是 HITS 在商业上的成就远不及 PageRank。 34

在本节中，我们会介绍 PageRank 算法的基本概念和计算排名的原理。我们还会介绍传送机制（teleportation mechanism）以及 PageRank 算法的核心——幂方法（power method）的原理。最后，我们会演示如何整合索引分数和 PageRank 分数来改进搜索结果。

2.3.1 PageRank 简介

PageRank 的基本思路就是将网页间的超链接视为一种推荐或认可。所以，一个网页收到的推荐越多，该网页就应该越重要。换言之，若一个网页被很多重要的网页链接，那么这也应该是一个重要的网页。但如果要事先知道哪些网页重要才能判断网页的重要性，这岂不是自相矛盾？让我们来看一个例子，看看其中的细节。

Download at http://www.pin5i.com/

图 2.5 是所有以 biz 为前缀的网页构成的一个有向图，这些文档的标题和文件名如表 2.1 所示。

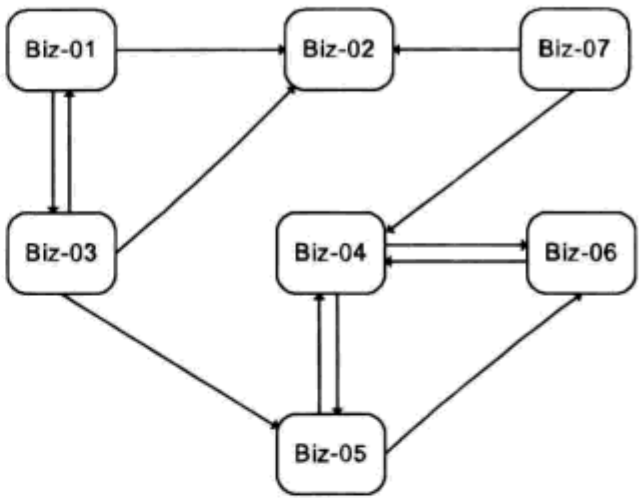


图 2.5 表示“biz”网页间链接的有向图

在图 2.5 中，如果网页 A 有一个链接指向网页 B，就会有一个 A 指向 B 的箭头。根据这个图，我们引入了超链接矩阵 $\boldsymbol{H}$ 和一个列向量 $\boldsymbol{p}$（PageRank 向量）。把矩阵想象成一个表格（二维数组），把向量看成 Java 中的一维数组。矩阵 $\boldsymbol{H}$ 中每一行的构成是：如果网页 P_i 有一个外链指向网页 P_j，网页 P_i 总共有 $N(i)$ 个外链，则矩阵 $\boldsymbol{H}$ 第 i 行 j 列的值是 $1/N(i)$；如果网页 P_i 没有指向网页 P_j 的外链，则该位置的值为 0。这样，就可以得到图 2.5 中有向图的矩阵 $\boldsymbol{H}$，如表 2.2 所示。

表 2.1 商业新闻文档及其链接（见图 2.5）

标　题	文 件 名	链 接 到
Google Expands into Newspaper Ads	biz-01.html	biz-02，biz-03
Google's Sales Pitch to Newspapers	biz-02.html	（没有外链，悬孤节点）
Google Sells Newspaper Ads	biz-03.html	biz-01，biz-02，biz-05
NVidia Now a Supplier for MP3 Players	biz-04.html	biz-05，biz-06
Nvidia Shares Up on PortalPlayer Buy	biz-05.html	biz-04，biz-06
Chips Snap: Nvidia, Altera Shares Jump	biz-06.html	biz-04
Economic Stimulus Plan Helps Stock Prices	biz-07.html	biz-02，biz-04

35

表 2.2 商业新闻网页的矩阵 $\boldsymbol{H}$ 及其链接（见图 2.5）

0	1/2	1/2	0	0	0	0
0	0	0	0	0	0	0
1//3	1/3	0	0	1/3	0	0
0	0	0	0	1/2	1/2	0
0	0	0	1/2	0	1/2	0
0	0	0	1	0	0	0
0	1/2	0	1/2	0	0	0

有以下两点需要注意：

- 矩阵中有很多 0——我们将这种矩阵称为稀疏矩阵。这并不是坏事，实际上

Download at http://www.pin5i.com/

这是好事。这是因为一个网页通常都只会链接很少的网页——这里的“少”是相对于互联网上的网页总数而言的。稀疏矩阵是有好处的，在实现过程中只要稍花心思就能节省大量的存储空间和计算时间。

- 矩阵中所有的值都小于或等于 1。这一点很重要，正是因为这一点，将 Brin 和 Page 提出的随机游走（见 2.3.2 小节）跟转移概率矩阵理论（即马尔科夫链理论）联系在一起。这种联系确保了数据能满足算法的需求。

2.3.2 计算 PageRank 向量

PageRank 用下面的迭代公式计算向量 $\boldsymbol{p}$：

$$p(k+1)=p(k)*H$$

p 的值就是图中每个网页的 PageRank 值。你可以从一组初始值开始，例如 $p(0)=1/n$，其中 n 是图中网页的数量。用这个公式就能得到 $p(1)$，然后是 $p(2)$，等等，直到两个相继 PageRank 向量之间的差别足够小，这个足够小的差别也被称为收敛判据或阈值。这种迭代方法就是把幂方法应用在矩阵 $\boldsymbol{H}$ 上。简而言之，这就是 PageRank 算法。

因为技术原因，为了能让该迭代过程收敛到某个 PageRank 向量，要用另一个有更好的数据特性的矩阵 $\boldsymbol{G}$（Google 矩阵）代替矩阵 $\boldsymbol{H}$。在此，我们不会深入讨论 PageRank 算法的数学细节，但为了让读者更好地理解这个算法，我们会讲解 PageRank 的基本原理，以及修改矩阵的原因。

36

PageRank 算法首先是模拟一个用户在随意浏览网页。这个过程可以从任意一个有外链的网页开始。通过网页中的某个链接，用户会跳转到另一个网页，然后点击一个新链接，再跳转，如此继续。经过若干次点击并浏览有向图中的各个网页后，每个网页的浏览次数就可以成为该网页相对重要的一个度量。如果浏览过程是完全随机的，没有任何偏好，我们就会更多地浏览那些被其他网页链接的网页，从而显示出这些网页的重要性。这很好，也很直观，但有两个问题。

第一个问题是，在互联网上有些网页中是没有指向其他网页的链接的。在我们这个例子中，图 2.5 中的 biz-02 这个网页就没有指向其他网页的链接。我们把这种网页称为图的悬孤节点（dangling node），这类节点会终止我们的浏览过程，没有了外链，就等于无路可走了。在 $\boldsymbol{H}$ 矩阵中，这类节点对应的就是全为 0 的行。为了解

Download at http://www.pin5i.com/

决这个问题，我们引入了随机跳转（jump），也就是当我们浏览网页遇到悬孤节点时，可以在浏览器的地址栏中输入另外一个网页的地址。对应到 ***H*** 矩阵中，这实际上就是把悬孤节点所对应的行中所有的 0 都改为 $1/n$，其中 n 是网页的总数。从技术上说，这种对 ***H*** 矩阵的修正叫做随机性调整（stochasticity adjustment）。

第二个问题是，上网也会有腻味的时候，或是停止上网，或者是跳转到另一个网页，而不是点击当前网页中的某个链接，就像是星际迷航电视剧里面的传送光束（teleportation beam）。考虑到这些随意的跳转，我们在代码中引入了一个新的参数——alpha，这个参数决定了访问网页中链接和随机跳转到其他网页的概率的比值。有时这个参数也被称为阻尼系数（damping factor）。从技术上说，这种对 ***H*** 矩阵的修正叫做原始性调整（primitivity adjustment）。

关于这两个问题，我们在代码中都加入了有关的注释。读者不需要担心其中的数学细节，但若是想了解它，Amy Langville 和 Carl Meyer 撰写的 *Google's PageRank and Beyond: The Science of Search Engine Ranking* 一书是最佳的参考资料。接下来，让我们开始运行一些代码来得到矩阵 ***H***。代码清单 2.5 中展示了如何加载商业新闻的网页，并计算它们的 PageRank。

代码清单 2.5 计算 PageRank 向量

```
FetchAndProcessCrawler crawler =
➥ new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);
crawler.setUrls("biz");←加载商业新闻网页
crawler.run();
PageRank pageRank = new PageRank(crawler.getCrawlData());←构建 PageRank 对象
pageRank.setAlpha(0.8);
pageRank.setEpsilon(0.0001);
pageRank.build();←得到 PageRank 值
```

37

图 2.6 是运行结果的截屏。从图中可看出，相关性最低的网页是 biz-07.html，而根据 PageRank，最重要的网页是 biz-04.html。我们计算出来的网页的相关性并不依赖于搜索词，我们从网页构成的网络中计算出了 PageRank 值。

38

2.3.3 alpha：网页间跳转的影响

下面来看看把 alpha 的值从 0.8 调整到其他介于 0 和 1 之间的值时，是如何影响网页的 PageRank 值的。随着 alpha 值趋向于 0，所有网页的 PageRank 值都会趋向于

Download at http://www.pin5i.com/

1/7（约等于 0.142857）。这完全符合我们的预期，因为浏览者会随机选择下一个访问目标，而不是根据链接访问网页。另一方面，随着 alpha 值趋向于 1，PageRank 的值会逐渐接近浏览者完全跟随链接访问网页所对应的 PageRank 向量。

```
Iteration: 8,   PageRank convergence error:
1.4462733376210263E-4
Index: 0-->  PageRank: 0.03944811976367004
Index: 1-->  PageRank: 0.09409188129468615
Index: 2-->  PageRank: 0.32404719855854225
Index: 3-->  PageRank: 0.24328037107628753
Index: 4-->  PageRank: 0.18555028886849476
Index: 5-->  PageRank: 0.05593157626783124
Index: 6-->  PageRank: 0.061816733771795335

 Iteration: 9,   PageRank convergence error:
5.2102415715682415E-5
Index: 0-->  PageRank: 0.039443819850858625
Index: 1-->  PageRank: 0.09407831778282823
Index: 2-->  PageRank: 0.3240636997004271
Index: 3-->  PageRank: 0.24328782624042117
Index: 4-->  PageRank: 0.18555238603685822
Index: 5-->  PageRank: 0.0559269660757835
Index: 6-->  PageRank: 0.06181315844717868

_____________ Calculation Results _____________
Page U RL: file:/c:/iWeb2/data/ch02/biz-04.html  -->  Rank:
0.324063699700427
Page URL: file:/c:/iWeb2/data/ch02/biz-06.html  -->  Rank:
0.243287826240421
Page URL: file:/c:/iWeb2/data/ch02/biz-05.html  -->  Rank:
0.185552386036858
Page URL: file:/c:/iWeb2/data/ch02/biz-02.html  -->  Rank:
0.094078317782828
Page URL: file:/c:/iWeb2/data/ch02/biz-03.html  -->  Rank:
0.061813158447179
Page URL: file:/c:/iWeb2/data/ch02/biz-01.html  -->  Rank:
0.055926966075784
Page URL: file:/c:/iWeb2/data/ch02/biz-07.html  -->  Rank:
0.039443819850859
```

图 2.6 计算一个由商业新闻网页构成的小型网络的 PageRank 向量

另一个能观察到的效果是随着 alpha 值接近 1，收敛所需的迭代次数会逐渐增加。实际上，对于我们的这个小网页网络，可以得到表 2.3（误差容忍度始终等于 10^{-10}）。

Download at http://www.pin5i.com/

表 2.3 在biz网页集合中，alpha值的增加对迭代次数的影响

Alpha	迭代次数	Alpha	迭代次数
0.50	13	0.85	23
0.60	15	0.95	29
0.75	19	0.99	32

如表 2.3 所示，随着 alpha 值的增长，迭代次数也迅速增加。如果是这七个网页，这个效果还不是很明显，但如果是 80 亿个网页（差不多是 Google 所使用的网页数），alpha 的选择就会变得非常重要。从本质上说，alpha 值的选择就是在 Web 的结构和计算效率之间的妥协。从现有资料可知，Google 使用的 alpha 的值是 0.85。根据网页构成的图的结构以及用户的浏览习惯，0.7～0.9 之间的值一般能做到兼顾应用的效果和效率。

有些技术可以加速幂方法的收敛，还有一些完全不依赖于幂方法的手段，称之为直接方法（direct method）。后者更适合于一些较小的网络（例如常见的局域网），有较高的 alpha 值（例如，0.99）。如果对这些方法感兴趣，可以参考本章最后提供的参考资料。

2.3.4 理解幂方法

让我们仔细看看计算 PageRank 值的代码。代码清单 2.6 展示了部分用于根据链接信息计算矩阵 ***H*** 的代码，它源自类 iweb2.ch2.ranking.PageRankMatrixH。

39

代码清单 2.6 根据网页间的链接计算矩阵 *H*

```
public void addLink(String pageUrl) {
  indexMapping.getIndex(pageUrl);
}
public void addLink(String fromPageUrl,
➥ String toPageUrl, double weight) {
  int i = indexMapping.getIndex(fromPageUrl);
  int j = indexMapping.getIndex(toPageUrl);
  try {
     matrix[i][j] = weight;
  } catch(ArrayIndexOutOfBoundsException e) {
   System.out.println("fromPageUrl:" + fromPageUrl
➥ + ", toPageUrl: " + toPageUrl);
```

❶赋初始值

Download at http://www.pin5i.com/


```
  }
 }
public void addLink(String fromPageUrl, String toPageUrl) {      ←❶赋初始值
      addLink(fromPageUrl, toPageUrl, 1);
  }
public void calculate() {←❷计算随机的矩阵版本
  for(int i = 0, n = matrix.length; i < n; i++) {
      double rowSum = 0;
      for(int j = 0, k = matrix.length; j < k; j++) {
        rowSum += matrix[i][j];
      }
      if( rowSum > 0 ) {
        for(int j = 0, k = matrix.length; j < k; j++) {
          if( matrix[i][j] > 0 ) {
            matrix[i][j] =
 (double)matrix[i][j] / (double) rowSum;
          }
        }
      } else {
        numberOfPagesWithNoLinks++;
      }
  }
}
/**
 * 悬孤节点对应的就是没有外链的网页。
 * 这类节点在矩阵 H 中就是全为 0 的行。
 */
public int[] getDangling() {←❸处理悬孤节点
  int n = getSize();
  int[] d = new int[n];
  boolean foundOne = false;
  for (int i=0; i < n; i++) {
     for (int j=0; j < n; j++) {
     if (matrix[i][j] > 0) {
       foundOne = true;
       break;
   }
     }
     if (foundOne) {
```

Download at http://www.pin5i.com/


```
        d[i] = 0;
      } else {
        d[i] = 1;
      }
      foundOne = false;
    }
    return d;
  }
```

❶ addLink 方法可以根据网页间的链接给 matrix 变量赋初值。

❷ calculate 方法求得每行的权重总和（外链），然后用相应的加权后的值替换现有的值。任何非悬孤节点所对应的行中所有元素的总和应该等于 1。这是原矩阵的随机版本。

❸ 由于悬孤节点没有外链，所以需要单独处理。getDangling()会处理所有对应悬孤节点的行，并返回悬孤向量。

之前我们把最终矩阵的组成分为三个部分：基本的链接的贡献、悬孤节点的贡献以及跳转的贡献。让我们看看如何合并这三者得到最终用于求 PageRank 的矩阵。代码清单 2.7 展示了用于组合各种贡献并执行幂方法的代码，在 iweb2.ch2.ranking.Rank 类中可以找到这些代码。

代码清单 2.7 用幂方法计算 PageRank

```
public void findPageRank(double alpha, double epsilon) {
  // 迭代的计数器
  int k = 0;
  // 辅助变量
  PageRankMatrixH matrixH = getH();
  // 矩阵 H 的大小是 n×n，PageRank 向量的大小是 n。
  int n = matrixH.getSize();
  // 辅助变量，n 的倒数
  double inv_n = (double)1/n;
  // 这是真正的双精度值的 n×n 矩阵
  double[][] H = matrixH.getMatrix();
  // 用于存放误差的虚拟变量，简单地设为 1
  // 这个变量存放 PageRank 向量的值
  pR = new double[n];
  // 前一次迭代所得到的 PageRank
```

Download at http://www.pin5i.com/


```
// 只在计算误差时才需要这个
double[] tmpPR = new double[n];
// 设置初始值（逐个设置）
for (int i=0; i < n; i++) {
 pR[i] = inv_n;
}
// 见本书 2.3 节——修改矩阵 H：悬孤节点
double[][] dNodes= getDanglingNodeMatrix();
// 见本书 2.3 节——修改矩阵 H：跳转
double tNodes=(1 - alpha) * inv_n;
// 用矩阵 G 替换矩阵 H
for (int i=0; i < n; i++) {
 for (int j=0; j < n; j++) {
  H[i][j] = alpha*H[i][j] + dNodes[i][j] + tNodes;
 }
}
// 迭代，直到收敛！
// 如果误差小于 epsilon，则得到 PageRank 值
while ( error >= epsilon) {
 // 在更新 PageRank 向量前，先复制一份
 for (int i=0; i < n; i++) {
  tmpPR[i] = pR[i];
 }
 double dummy =0;
 // 现在到了迭代中的下一个点
 for (int i=0; i < n; i++) {
  dummy =0;
  for (int j=0; j < n; j++) {
   dummy += pR[j]*H[j][i];
  }
  pR[i] = dummy;
 }
 // 得到误差，检查收敛情况
 error = norm(pR,tmpPR);
 // 计数器加 1
 k++;
}
// 报告最终的结果
System.out.println(
```

Download at http://www.pin5i.com/


```
➥ "\n_____________ Calculation Results _____________\n");
  for (int i=0; i < n; i++) {
   System.out.println("Page URL: "+
➥ matrixH.getIndexMapping().getValue(i)+" --> Rank: "+pR[i]);
  }
}
```

由于该方法的重要性，为了尽可能地提高可读性，这段代码写得比较长。我们删除了一些与 to-do 有关的 Javadoc 注释，但这段代码还是完整的。在这里，首先是根据网页间的链接得到矩阵 ***H*** 并初始化 PageRank 向量。接着，得到悬孤节点的贡献和跳转的贡献。注意，悬孤节点需要一个完整的二维数组，而跳转贡献则只需要一个双精度的变量。接着，将这三者加在一起。这是最高效的为幂方法准备数据的方法，但实际上还可以用稀疏矩阵代替完整的二维数组。我们会在本章末的 To Do 事项中介绍这种改进方法。

现在已经得到了新的 ***H*** 矩阵，接下来介绍幂方法，即 while 循环中的代码。如果两次计算得到的 PageRank 值之间的差别小于阈值 epsilon，就得到了所需的 PageRank 值。当然，你可能会好奇：如果修改 epsilon 会怎么样？PageRank 值会变吗？如果会变，将 epsilon 设为什么值才是合适的？让我们逐一解答这些问题。首先，误差就是逐项地计算新旧 PageRank 向量的绝对值之差。代码清单 2.8 展示了来自 iweb2.ch2.ranking.Rank 类中用于计算误差的 norm 方法。

代码清单 2.8　计算两个连续 PageRank 向量间的误差

```
private double norm(double[] a, double[] b) {
  double norm = 0;
  int n = a.length;
  for (int i=0; i < n; i++) {
    norm += Math.abs(a[i]-b[i]);
  }
  return norm;
}
```

如果反复多次运行这段代码，或是仔细观察图 2.6，你就会发现在收敛后各个网页的 PageRank 值的差别与 epsilon 的最小值是同一个数量级的。所以，要根据 PageRank 值将所有网页的重要性区分开，epsilon 要足够小才行。如果有 100 个网页，epsilon 要小于或等于 0.001 才行。整个互联网上有 10^{10} 个网页，epsilon 就需要小到 10^{-10}。

Download at http://www.pin5i.com/

2.3.5 结合索引分值和 PageRank 分值

前面演示了如何实现 PageRank 算法，现在我们学习如何将 Lucene 搜索分数和从 PageRank 算法得到的网页的相关性结合在一起。我们将继续使用这七个商业新闻的网页，但现在还要加入三个垃圾网页（名为 spam-biz-0x.html，其中 x 是一个数字）。这些垃圾网页可以瞒过基于索引的搜索，但却愚弄不了 PageRank。

下面运行这段代码，看看会发生什么。代码清单 2.9 中的代码演示了以下内容。

- 跟前面一样，加载商业新闻的网页。
- 添加分属三个类别的三个垃圾网页。
- 索引所有的网页。
- 构建 PageRank。
- 结合索引相关性分数（来自 Lucene）和 PageRank 分数计算出一个混合排序分数。

代码清单 2.9 整合 Lucene 和 PageRank 分数对网页进行排序

```
FetchAndProcessCrawler crawler =
➥ new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);
crawler.setUrls("biz");
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-01.html");←加入垃圾网页
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-02.html");
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-03.html");
crawler.run();
LuceneIndexer luceneIndexer =
➥ new LuceneIndexer(crawler.getRootDir());
luceneIndexer.run();←索引所有的网页
PageRank pageRank = new PageRank(crawler.getCrawlData());
pageRank.setAlpha(0.99);
pageRank.setEpsilon(0.00000001);
pageRank.build();←构建 PageRank
MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());
oracle.search("nvidia",5, pageRank);←用组合分数搜索
```

图 2.7 中是“nvidia”的搜索结果。第一部分结果是仅仅根据 Lucene 分数得到的，接下来是考虑了 PageRank 值的结果。从图中可以看到，我们成功地避免了垃圾网页的影响。如果只使用 Lucene，垃圾网页在结果中排名第一。但当我们使用混合

Download at http://www.pin5i.com/

排序时，相关性高的网页则排到了前面，垃圾网页回到了它应该待的位置。如此简单，我们就编写出了一个类似于 Google 的搜索引擎。可喜可贺！

```
bsh % oracle.search("nvidia",5,pr);

Search results using Lucene index scores:
Query: nvidia

Document Title: NVIDIA shares plummet into cheap medicine for
you!
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.html   ->
Relevance Score: 0.519243955612183

Document Title: Nvidia shares up on PortalPlayer buy
Document URL: file:/c:/iWeb2/data/ch02/biz-05.html
Relevance Score: 0.254376530647278

Document Title: NVidia Now a Supplier for MP3 Players
Document URL: file:/c:/iWeb2/data/ch02/biz-04.html   ->
Relevance Score: 0.190782397985458

Document Title: Chips Snap: Nvidia, Altera Shares Jump
Document URL: file:/c:/iWeb2/data/ch02/biz-06.html   ->
Relevance Score: 0.181735381484032

Document Title: Economic stimulus plan helps stock prices
Document URL: file:/c:/iWeb2/data/ch02/biz-07.html   ->
Relevance Score: 0.084792181849480

Search results using combined Lucene scores and page rank scores:
Query: nvidia

Document URL: file:/c:/iWeb2/data/ch02/biz-04.html   ->
Relevance Score: 0.087211910261991
Document URL: file:/c:/iWeb2/data/ch02/biz-06.html   ->

Document URL: file:/c:/iWeb2/data/ch02/biz-05.html   ->
Relevance Score: 0.062737066556678
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.html   ->

Document URL: file:/c:/iWeb2/data/ch02/biz-07.html   ->
Relevance Score: 0.000359708275446
```

图 2.7 整合 Lucene 和 PageRank 分数能消除垃圾网页

Download at http://www.pin5i.com/

在 MySearcher 类中重载的带 PageRank 类参数的 search 方法中可以找到合并这两种分值的代码。

代码清单 2.10 中的代码片段就来自这个结合了两种分数的方法。 44

代码清单 2.10 整合 Lucene 和 PageRank 分数 45

```
double m = 1 - (double) 1/pR.getH().getSize();←计算缩放因子
for (int i = 0; i < numberOfMatches; i++) {
  url = docResults[i].getUrl();
  double hScore =
➥ docResults[i].getScore() *Math.pow(pR.getPageRank(url),m);←计算混合分数
  docResults[i].setScore(hScore);
  urlScores.put(hScore, url);←构建分数和 URL 之间的映射
}
```

现在你脑子里面应该有一堆问题：为什么要引入变量 m？为什么不直接用两个分数的平均值？为什么不用更加复杂的公式组合索引分数和 PageRank 分数？这些都是很好的问题，答案会让您大感意外。我们设计这个公式的唯一原则就是让分数介于 0 和 1 之间，其他都是随意选择的，就算用两者的乘积来合并这两个分数也是可以的。

之所以要将 PageRank 扩大 m 次方，是因为我们索引的网页数量较少，这会导致垃圾网页的索引相关性分数较高，从而人为地降低了 PageRank 的效果。随着网页的增加，m 会快速接近 1，从而使缩放后的 PageRank 值（第二种混合分数）也趋向于原始的 PageRank 值。在小规模的网络中，幂次定律缩放能让链接结构比索引更重要。这个公式不仅适用于少量的文档，也适用于大量的文档。幂次定律和图与互联网的相似性之间有很深的数学联系，在此不做深入讨论（详见参考资料中 Adamic 的论文）。我们需要知道的推论是在少量网页中，如果搜索词在某个文档中大量出现（例如垃圾网页），该文档的索引分数就会接近 1（即 Lucene 返回的搜索结果的分数），此时就需要进行缩放来避免这种情况所带来的负面影响。

2.4 根据用户点击改进搜索结果

前面我们已经介绍了如何通过链接分析来利用互联网的结构信息。在本节中，我们会利用互联网的另一个特性：用户点击。大家都知道，用户每次执行查询时都

Download at http://www.pin5i.com/

会点击某个搜索结果，或是点击下一页结果（如果有的话）。在第一种情况中，用户发现了他感兴趣的东西并点击了该链接，这可能是因为用户找到了自己所要的东西，也有可能是对这个搜索结果感兴趣，想要进一步研究有关信息，从而最终决定这是否是他要找的。在第二种情况中，排在第一页的搜索结果都不是用户想要的，翻到下一页只是想在搜索引擎中碰碰运气而已。

46

实际上，评价相关性的难点就在于相关性是非常主观的。两个人同样是搜索“选举”，第一个人可能感兴趣的是美国选举，而第二个人感兴趣的则有可能是英国选举，甚至可能是他居住的小镇上的选举。如果没有额外的信息，搜索引擎不可能知道用户到底要搜索什么。所以，对于不同的人，即使是完全一样的搜索词，最相关的搜索结果也有可能是完全不一样的。

我们将要介绍的用户点击是一种针对单个用户改进搜索结果的方法。这种改进应该要归结到本书稍后会详细介绍的一个算法——朴素贝叶斯分类器。我们将把索引分数、PageRank 分数和用户点击分数结合在一起，改进我们的搜索结果。

2.4.1 用户点击初探

搜索引擎可以通过用户点击与用户的输入进行交互。亚里士多德说过：“我们每一个人都是由自己一再重复的行为所铸造的”，这就是用户点击分析的基础：用户与搜索引擎的交互行为定义了用户的兴趣和主观目的。这是我们介绍的第一个用于实现 Web 应用个性化的智能技术。当然，实现个性化的一个必要前提是搜索引擎能识别出查询请求是来自哪个用户的。换句话说，用户必须登录到应用中，或者是用其他方法构建一个与应用之间的会话。需要说明的是，这里介绍的用户点击分析方法并不是只适用于搜索应用，而是适用于所有能记录用户点击的应用。

现在，假设我们已经收集了一些用户的点击记录，并存放在 user-clicks.csv 文件中。在 data/ch02 目录中可以找到这个文件，以及本章用到的其他文件。PageRank 算法能帮我们利用网络的结构信息，而我们现在的目的就是写出能帮我们利用这些用户点击信息的程序。也就是说，要通过合理的方法利用用户点击信息，为不同的搜索用户提供不同的搜索结果排序，从而实现搜索结果的个性化。这个逗号分割文件含有以下三个字段：

- 一个用于标识用户的字符串。
- 表示用户的搜索请求的字符串。

Download at http://www.pin5i.com/

- 包含用户在查看搜索结果后选择的 URL 的字符串。

如果你不知道用户信息（没有登录或者没有会话机制），也可以用“anonymous”这样的默认值，当然，要确保这不是应用中真实存在的某个用户名。如果数据格式有区别，也没问题。把我们的代码移植到其他格式的数据上是完全没有问题的。为了个性化搜索结果，我们需要知道用户、用户的问题以及他过去针对这个问题所选择的链接。如果有了这些信息，就让我们开始行动吧！

你可能会注意到，在我们的数据中，同一个用户对同样的搜索会有多个选择，这是正常的，而且在你自己的数据中也要注意这点。文件中每个 URL 被点击的次数都会影响该 URL 在搜索结果中的表现。通常来说，同一个用户对同样的搜索会点击不同的链接，有可能是因为不同的时候有不同的需求，也有可能是想获得更多的有关信息。有一个有趣的属性是应该考虑的，那就是时间戳（timestamp）。时间信息可以帮助我们辨别出数据中的时间结构。有些用户的点击是周期性的，有些是事件驱动的，还有一些则是完全随机的。时间戳能帮我们找到用户点击的模式或是其与其他事件的关联。

47

首先，让我们看看如何得到个性化的结果。代码清单 2.11 中是我们编写的脚本，与代码清单 2.9 很像，但这次我们加载了用户点击的信息，并且会为两个用户 dmitry 和 babis 各执行一次“good ads”的查询。

代码清单 2.11 在搜索结果中考虑用户点击

```
FetchAndProcessCrawler crawler =
➥ new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);
crawler.setUrls("biz");
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-01.html");
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-02.html");
crawler.addUrl("file:///c:/iWeb2/data/ch02/spam-biz-03.html");
crawler.run();
LuceneIndexer luceneIndexer =
➥ new LuceneIndexer(crawler.getRootDir());
luceneIndexer.run();
MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());
PageRank pageRank = new PageRank(crawler.getCrawlData());
pageRank.setAlpha(0.9);
pageRank.setEpsilon(0.00000001);
pageRank.build();
```

Download at http://www.pin5i.com/


```
UserClick aux = new UserClick();
UserClick[] clicks =aux.load("C:/iWeb2/data/ch02/user-clicks.csv");←加载用户点击
TrainingSet tSet = new TrainingSet(clicks);←创建训练集
NaiveBayes naiveBayes = new NaiveBayes("Naïve Bayes", tSet);←定义分类器
naiveBayes.trainOnAttribute("UserName");←选择属性
naiveBayes.trainOnAttribute("QueryTerm_1");
naiveBayes.trainOnAttribute("QueryTerm_2");
naiveBayes.train();←训练分类器
oracle.setUserLearner(naiveBayes);
UserQuery dmitryQuery = new UserQuery("dmitry","google ads");
oracle.search(dmitryQuery,5, pageRank);
UserQuery babisQuery = new UserQuery("babis","google ads");
oracle.search(babisQuery,5, pageRank);
```

48

在代码清单 2.9 中已经见过该脚本的第一部分。首先，我们加载要搜索的网页。然后，用 Lucene 对其进行索引并构建与其结构相对应的 PageRank。其中新增加的代码是 UserClick 类，它表示某个用户对一个链接的点击。此外，还用到了 TrainingSet 类，该类用来存放所有的用户点击。当然，你可能会觉得奇怪，为什么不直接用 UserClick 的数组呢？为什么不能直接用这些对象？答案是：定义 TrainingSet 类是为了方便把用户点击加载到分类器（这里用的是 NaiveBayes 分类器）中，这样才能找到用户每次查询时最符合其需求的链接。

2.4.2 朴素贝叶斯分类器的使用

在第 5 章和第 6 章中我们会更全面地讨论分类问题，但为了能让读者更好地理解后面的内容，在这里先对有关的基础知识稍作介绍。分类就是依靠参考结构将空间中所有的数据点分成若干类（也称为类别（category）或概念（concept）），这些类之间通常是不重叠的。我们在日常生活中经常会遇到分类问题。根据日常经验，我们可以根据餐厅的菜单把食物分列成几类，例如沙拉、开胃菜、特色食品、意大利面、海鲜等。对于报纸或互联网上新闻组中的文章，我们也可以根据主题将其分类成政治新闻、体育新闻、商业新闻、国际新闻、娱乐新闻等。总之，分类算法可以自动地判断出对象所属的类别。

本节将用到一个名为朴素贝叶斯算法的概率分类器，它实现在 NaiveBayes 类中。分类器并不知道 UserClicks，它关注的是概念（concept）、实例（instance）和属性

Download at http://www.pin5i.com/

（attribute）。可以把概念、实例和属性类比成文件系统中的目录、文件和文件属性。

分类器的工作就是将概念赋给实例，仅此而已。为了能知道应该将哪个概念赋予各个实例，分类器会读取 TrainingSet，也就是一组已分配概念的实例。实例载入之后，分类器基于 TrainingSet 中的映射关系对自身进行训练，或者说是学习如何将概念映射到实例。不同的分类器有不同的训练方法。

我们试图用 NaiveBayes 分类器计算出 URL 对于某个用户及其所提交的查询的相关性分数。NaiveBayes 分类器的优点是能得到已知 Y 的条件下 X 的条件概率，即已经观察到了事件 Y 的情况下，事件 X 发生的概率。具体地说，该分类器的输入如下：

- 观察到概念 X 的概率，也被称为先验概率（prior probability），用 $p(X)$表示。
- 从属于概念 X 的实例中随机选择一个实例，选到 Y 的概率也被称为似然率（likelihood），用 $p(Y|X)$表示。
- 在一般情况下选到 Y 的概率也被称为证据（evidence），用 $p(Y)$表示。

分类器的核心部分就是计算实例 Y 属于概念 X 的概率，也被称为后验概率，用 $p(X|Y)$表示。这个计算是基于以下公式的（贝叶斯定理）： 49

$$p(X|Y)=p(Y|X)p(X)/p(Y)$$

NaiveBayes 分类器会计算出用户 A 在搜索 Q 时想要看到链接 X 的概率，在这里 $Y=A+Q$。换句话说，我们并不会用 NaiveBayes 分类器来进行分类。我们只是利用了它能计算相关性的特点，这完全符合我们的需求。代码清单 2.12 是 NaiveBayes 类中的代码，完整的描述见 5.3 节。

代码清单 2.12 用 NaiveBayes 分类器计算 URL 的相关性

```
public class NaiveBayes implements Classifier {
  private String name;❶
  private TrainingSet tSet;❷
  private HashMap<Concept,Double> conceptPriors;❸
  protected Map<Concept,Map<Attribute, AttributeValue>> p;❹
  private ArrayList<String> attributeList;❺
  public double getProbability(Concept c, Instance i) {
    double cP=0;
    if (tSet.getConceptSet().contains(c)) {
     cP = (getProbability(i,c)*getProbability(c))/getProbability(i);❻
```

Download at http://www.pin5i.com/


```
    } else {
     cP = 1/(tSet.getNumberOfConcepts()+1);❼
    }
    return cP;
  }
  public double getProbability(Instance i) {
    double cP=0;
    for (Concept c : getTset().getConceptSet()) {
     cP += getProbability(i,c)*getProbability(c);
    }
    return (cP == 0) ? (double)1/tSet.getSize() : cP;❽
  }
  public double getProbability(Concept c) {
    Double trInstanceCount = conceptPriors.get(c);
    if( trInstanceCount == null ) {
        trInstanceCount = 0.0;
    }
    return trInstanceCount/tSet.getSize();❾
  }
  public double getProbability(Instance i, Concept c) {
    double cP=1;
    for (Attribute a : i.getAtrributes()) {
50   if ( a != null && attributeList.contains(a.getName()) ) {
      Map<Attribute, AttributeValue> aMap = p.get(c);
      AttributeValue aV = aMap.get(a);
      if ( aV == null) {❿
          cP *= ((double) 1 / (tSet.getSize()+1));
      } else {
        cP *= (double)(aV.getCount()/conceptPriors.get(c));
      }
     }
    }
    return (cP == 1) ? (double)1/tSet.getNumberOfConcepts() : cP;
  }
}
```

首先，让我们看看这段代码的要点：

❶ NaiveBayes 分类器实例的名称。

Download at http://www.pin5i.com/

❷ 每个分类器都需要一个训练集，在构建阶段就要设好分类器和训练集的名称。NaiveBayes 分类器的实例在创建好之后，就不能再修改它的 TraningSet，但随时可以通过训练集的引用往其中添加新的实例。

❸ conceptPriors 表中存放的是训练集中属于各个概念的实例数量。也可以在其中存放先验概率，而不仅仅是数量。但我们要反复用到这些数量，所以出于计算效率的考虑，保存的只是数量，在此基础上，只要一个简单的除法就能得到先验概率。

❹ 变量 p 保存的是条件概率——观察到实例 Y，且其属于概念 X 的概率。具体到用户点击的案例中，就是用户 A 查询 Q 时想要看到 X 链接的概率。

❺ 这个列表中是分类器在训练时要考虑的属性。一个训练集的实例会有很多属性，但可能只有部分属性是与分类相关的（见第 5 章），所以我们需要把用到的属性记录下来。

❻ 如果在训练集中已经有了这个概念，就可以用之前提到的公式计算出后验概率。

❼ 也有可能会出现之前未曾遇到过的特殊实例，此时用 getProbability(i)方法就不起作用了。遇到这种情况时，可以设置一个合理的后验概率。如果没有信息表明要赋予某个概念较高的概率，可以将新概念的概率设为 1 除以已知概念的总数。这里我们还把总数加了 1，这个改动没有太多的理论依据，只是为了降低每个概念的概率，尤其是在观察到的概念数量较少的情况下。想一下为什么，以及在什么前提条件下这种做法会有用。

❽ NaiveBayes 类中的这个方法对纯粹的分类问题是不必要的，因为所有概念的这个值都是一样的。在现在这个例子中，我们决定保留它。可以修改这段代码，让程序只返回贝叶斯定理中的分子，看会是什么结果？

❾ 给定概念 c，其先验概率是根据训练集中该概念出现的次数得到的。对于未曾出现的概念，我们简单地将其概率设为 0。这有好，也有坏。如果有把握确保训练集中包含了所有可能的概念，选择这种方式就能消除数据中的特例。但更常见的情况是训练集中可能没有包含足够多的概念，这时就不能将此概率简单地设为 0，而需要一个更加合理的值——1 除以已知概念的总数。还有什么其他的合理选择吗？准确地估计这个数值很重要吗？无论答案是什么，最好让你的选择看起来是合理的，并努力把它调整到最好。 51

Download at http://www.pin5i.com/

⑩ 到了 NaiveBayes 类的核心。贝叶斯定理中所谓的“朴素”指的就是观察到实例 i 的似然率，等于观察到各个属性的概率的乘积。这里潜在的假设是各个属性是统计独立的。我们在“朴素”一词上加上引号是因为朴素贝叶斯算法非常鲁棒且应用广泛，甚至在属性相互独立的假设不成立的情况也大有用途。即使是在完全相反的情况下，属性完全相互依赖，依然可以证明朴素贝叶斯算法也是最优的（见 Rish 的论文）。

请你回忆一下代码清单 2.11 中的脚本，其中创建了一个训练集，并用这个训练集创建了一个分类器的实例，在将这个分类器赋予 MySearcher 实例前，我们还做了下面两件事情：

- 告诉分类器在训练中应该考虑哪些属性。
- 告诉分类器根据指定的属性和加载的用户点击数据对自身进行训练。

标签为 UserName 的属性表示用户。属性 QueryTerm_1 和 QueryTerm_2 分别对应查询的第一个和第二个搜索词，这些属性值都可以通过 Lucene 的 StandardAnalyzer 类得到。在训练过程中，我们会根据每个实例出现的频率，给其赋予一定的概率。这里最重要的方法是用于获得在用户执行特定查询（Instance）时产生的 URL（Concept）的相关性的 getProbability（Concept c，Instance i）方法。

2.4.3 整合 Lucene 索引、PageRank 和用户点击

得到用户在执行各种查询时对各个 URL 的偏爱概率后，我们可以将前面的三种技术整合在一起来改进我们的搜索结果。代码清单 2.13 中是有关的代码。

代码清单 2.13 Lucene 索引、PageRank 值和用户点击概率

```
public SearchResult[] search(UserQuery uQuery,
➥ int numberOfMatches, Rank pR) {
  SearchResult[] docResults =
➥ search(uQuery.getQuery(), numberOfMatches);←基于索引的结果
  String url;
  StringBuilder strB = new StringBuilder();
  int docN = docResults.length;
  if (docN > 0) {
   int loop = (docN<numberOfMatches) ? docN : numberOfMatches;←收集最多 numberOf
                                                              Matches 个文档
    for (int i = 0; i < loop; i++) {
```

52

Download at http://www.pin5i.com/


```
      url = docResults[i].getUrl();
      UserClick uClick = new UserClick(uQuery,url);←收集所有用户的点击分数
        double indexScore = docResults[i].getScore();
      double pageRankScore = pR.getPageRank(url);
      BaseConcept bC = new BaseConcept(url);
      double userClickScore = learner.getProbability(bC, uClick);
      double hScore;
      if (userClickScore == 0) {←计算最终（混合）分数
        hScore = indexScore * pageRankScore * EPSILON;
      } else {
          hScore = indexScore * pageRankScore * userClickScore;
        }
        docResults[i].setScore(hScore);
    strB.append("Document URL : ")
    .append(docResults[i].getUrl()).append(" --> ");
    strB.append("Relevance Score: ")
    .append(docResults[i].getScore()).append("\n");
      }
    }
    strB.append(PRETTY_LINE);
    System.out.println(strB.toString());
    return docResults;
  }
```

图 2.8 是用户 dmitry 的搜索结果。从图中可以看到，由于 dmitry 曾经多次点击 biz-03.html 页面，所以这个页面的相关性分数是最高的。第二个搜索结果 biz-01.html 也是用户曾经点击过的。垃圾网页能排在第三，只是因为网页的数量太少的缘故。为了说明缩放因子 m 的作用，我们在这里故意没有使用它。

图 2.9 中执行的是一样的搜索“google ads”，但这次是针对用户 babis。我们反转了 dimitry 的点击记录作为 babis 的点击记录。第一个搜索结果是 biz-01.html，第二个是 biz-03.html，其他的结果都是一样的。结果中的唯一差别就是因为搜索请求来自不同的用户，这种差别正表明了该应用从 user-clicks.csv 文件中“学到”了知识。

Download at http://www.pin5i.com/

53

```
bsh % UserQuery dQ = new UserQuery("dmitry", "google ads");
bsh % oracle.search(dQ,5,pr);

Search results using Lucene index scores:
Query: google ads

Document Title: Google Ads and the best drugs
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-01.html -->
Relevance Score: 0.788674294948578
_____________________________________________________
Document Title: Google Expands into Newspaper Ads
Document URL: file:/c:/iWeb2/data/ch02/biz-01.html -->
Relevance Score: 0.382
_____________________________________________________
Document Title: Google sells newspaper ads
Document URL: file:/c:/iWeb2/data/ch02/biz-03.html -->
Relevance Score: 0.317
_____________________________________________________
Document Title:Google's sales pitch to newspapers
Document URL: file:/c:/iWeb2/data/ch02/biz-02.html -->
Relevance Score: 0.291
_____________________________________________________
Document Title: Economic stimulus plan helps stock prices
Document URL: file:/c:/iWeb2/data/ch02/biz-07.html -->
Relevance Score: 0.031
_____________________________________________________

Search results using combined Lucene scores, page rank scores and
user clicks:
Query: user=dmitry, query text=google ads

Document URL: file:/c:/iWeb2/data/ch02/biz-03.html   -->
Relevance Score: 0.0057

Document URL: file:/c:/iWeb2/data/ch02/biz-01.html   -->
Relevance Score: 0.0044

Document URL: file:/c:/iWeb2/data/ch02/spam-biz-01.html-->
Relevance Score: 0.0040

Document URL: file:/c:/iWeb2/data/ch02/biz-02.html   -->
Relevance Score: 0.0012

Document URL: file:/c:/iWeb2/data/ch02/biz-07.html   -->
Relevance Score: 0.0002
_____________________________________________________
```

图 2.8 整合 Lucene、PageRank 和用户点击为用户 dmitry 产生的高度相关的搜索结果

Download at http://www.pin5i.com/

54

```
bsh % UserQuery bQ = new UserQuery("babis", "google ads");
bsh % oracle.search(bQ,5,pr);

Search results using Lucene index scores:
Query: google ads

Document Title: Google Ads and the best drugs
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-01.html -->
Relevance Score: 0.788674294948578
-----------------------------------------------------------
Document Title: Google Expands into Newspaper Ads
Document URL: file:/c:/iWeb2/data/ch02/biz-01.html -->
Relevance Score: 0.382
-----------------------------------------------------------
Documen t Title: Google sells newspaper ads
Document URL: file:/c:/iWeb2/data/ch02/biz-03.html -->
Relevance Score: 0.317
-----------------------------------------------------------
Document Title: Google's sales pitch to newspapers
Document URL: file:/c:/iWeb2/data/ch02/biz-02.html -->
Relevance Score: 0.291
-----------------------------------------------------------
Document Title: Economic stimulus plan helps stock prices
Document URL: file:/c:/iWeb2/data/ch02/biz-07.html -->
Relevance Score: 0.0314
-----------------------------------------------------------

Search results using combined Lucene scores, page rank scores
and user clicks:
Query: user=babis, query text=google ads

Document URL: file:/c:/iWeb2/data/ch02/biz-01.html    -->
Relevance Score: 0.00616

Document URL: file:/c:/iWeb2/data/ch02/biz-03.html    -->
Relevance Score: 0.00407

Document URL: file:/c:/iWeb2/data/ch02/spam-biz-01.html -->
Relevance Score: 0.00393

Document URL: file:/c:/iWeb2/data/ch02/biz-02.html    -->
Re levance Score: 0.00117
```

图 2.9 整合 Lucene、PageRank 和用户点击为用户 Babis 产生的高度相关的搜索结果

Download at http://www.pin5i.com/

55

太好了！现在我们已经极大地改进了纯粹的基于索引的搜索，我们考虑了文档间的超链接结构，还依据用户的点击记录考虑到了用户的喜好。但有很多应用需要搜索相互之间没有明显链接的文档。有什么办法可以在这种情况下改进搜索结果吗？这正是我们接下来要介绍的。

2.5 Word、PDF 等无链接文档的排序

假设想要在数十万个 Word 文档或 PDF 文档中进行搜索，或者还有其他类型的文档。咋一看，索引似乎是唯一的选择，最多还能考虑一下用户点击分析。但接下来，我们会演示如何把之前应用于网页的链接分析技术扩展到这类无链接的文档中。希望我们能抛砖引玉，引导读者想出更好的方法。顺便提一下，据我们所知，这里介绍的方法此前还从未公开发表过。

为了说明排序也适用于无链接文档，我们用前面用过的 HTML 文件的内容创建 Word 文档，所以两者的内容完全一样。这样我们就能与 2.3 节中的结果进行比较，找出两种方法的异同。用开源的 TextMining 库能很方便地解析 Word 文档，要注意其名字已改为 tm-extractor。从 1.0 版开始，该项目就使用 LGPL 协议，所以将其应用到商业项目中也是很方便的。你可以从 http://code.google.com/p/text-mining/source/checkout 中获得源代码。我们编写了一个名为 MSWordDocumentParser 的类来封装与 Word 文档解析有关的代码。

2.5.1 DocRank 算法简介

在代码清单 2.14 中，我们用读取 HTML 文件的同一个类（FetchAndProcessCrawler 类）来读取 Word 文档，然后用 Lucene 对这些文档内容进行索引。

代码清单 2.14 根据内容对文档排序

```
FetchAndProcessCrawler crawler =
➥ new FetchAndProcessCrawler("C:/iWeb2/data/ch02",5,200);
crawler.setUrls("biz-docs");←加载商业 Word 文档
crawler.addDocSpam();
crawler.run();
LuceneIndexer luceneIndexer =
➥ new LuceneIndexer(crawler.getRootDir());←构建 Lucene 索引
```

Download at http://www.pin5i.com/


```
luceneIndexer.run();
MySearcher oracle = new MySearcher(luceneIndexer.getLuceneDir());←构建普通的搜索引擎
oracle.search("nvidia",5);
DocRank docRank = new DocRank(luceneIndexer.getLuceneDir(),7);
docRank.setAlpha(0.9);
docRank.setEpsilon(0.00000001);
docRank.build();←创建 DocRank 引擎
oracle.search("nvidia",5, docRank);
```

56

图 2.10 中是“nvidia”的搜索结果，排名第一的是我们不想要的 spam-biz-02.doc 文档，这个结果与我们之前处理 HTML 文件时的结果类似。当然，在处理 Word、PDF 和其他类型的文本文档时，遇到垃圾文档的概率较低，但依然有可能遇到某些毫无意义的重复关键词的文档。

```
bsh % oracle.search("nvidia", 5);

Search results using Lucene index sco
Query: nvidia

Document Title: NVIDIA shares plummet into cheap medicine for
you!
Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.doc -->
Relevance Score: 0.458221405744553

---------------------------------------------------------------
Document Title: Nvidia shares up on PortalPlayer buy
Document URL: file:/c:/iWeb2/data/ch02/biz-05.doc  -->
Relevance Score: 0.324011474847794

---------------------------------------------------------------
Document Title: NVidia Now a Supplier for MP3 Players
Document URL: file:/c:/iWeb2/data/ch02/biz-04.doc  -->
Relevance Score: 0.194406896829605

---------------------------------------------------------------
Document Title: Nov. 6, 2006, 2:38PM?Chips Snap: Nvidia, Altera
Shares Jump
Document URL: file:/c:/iWeb2/data/ch02/biz-06.doc  -->
Relevance Score: 0.185187965631485

---------------------------------------------------------------
```

图 2.10 基于索引技术在含商业新闻和垃圾内容的 Word 文档中搜索“nvidia”的结果

目前，一切都与代码清单 2.9 相同。新的代码来自 DocRank 类，这个类负责计算文档间的相关性，等同于 PageRank 计算网页间的相关性。与 PageRank 类不一样，

Download at http://www.pin5i.com/

DocRank 还需要一个额外的参数，稍后会介绍。跟前面几节一样，这里也需要有一个矩阵来表示网页 Y 基于网页 X 的重要性。问题是，与网页不一样，这些文档之间并没有明显的链接。网页间的链接只是用来构建矩阵，告诉我们根据网页 X 得到的网页 Y 的重要性。如果能找到某种方法来衡量这种重要性，我们就能利用跟 PageRank 算法一样的数学原理实现搜索结果的排序。我们的代码就能得到这样一个矩阵。

57 2.5.2 DocRank 的原理

我们对重要性的衡量方法并不是很严谨，主要依赖于新的矩阵 ***H*** 中元素的两个性质，即：

- 都是正数。
- 各行元素的总和等于 1。

我们的方法能否成功取决于所处理的文档类型。代码清单 2.15 中的代码是用于构建 Word 文档的矩阵 ***H*** 的 DocRankMatrixBuilder 类。

代码清单 2.15 DocRankMatrixBuilder：根据内容对文本文档排序

```
public class DocRankMatrixBuilder implements CrawlDataProcessor {
  private final int TERMS_TO_KEEP = 3;
  private int termsToKeep=0;
  private String indexDir;
  private PageRankMatrixH matrixH;
  public void run() {
     try {
        IndexReader idxR =
➥ IndexReader.open(FSDirectory.getDirectory(indexDir));
        matrixH = buildMatrixH(idxR);
     }
     catch(Exception e) {
        throw new RuntimeException("Error: ", e);
     }
  }
  // 从文档索引中收集匹配的文档类型的文档 Id
  private List<Integer> getProcessedDocs(IndexReader idxR)
     throws IOException {
     List<Integer> docs = new ArrayList<Integer>();
     for(int i = 0, n = idxR.maxDoc(); i < n; i++) {
```

Download at http://www.pin5i.com/


```
        if( idxR.isDeleted(i) == false ) {
            Document doc = idxR.document(i);
            if( eligibleForDocRank(doc.get("doctype") ) ) {
                docs.add(i);
            }
        }
    }
    return docs;
  }
// 该索引条目符合要求吗?
  private boolean eligibleForDocRank(String doctype) {
    return ProcessedDocument.DOCUMENT_TYPE_MSWORD
➥   .equalsIgnoreCase(doctype);
  }
  private PageRankMatrixH buildMatrixH(IndexReader idxR)
    throws IOException {
  // 只考虑内容已经采集并经过处理的 URL
    List<Integer> allDocs = getProcessedDocs(idxR);
    PageRankMatrixH docMatrix =
➥ new PageRankMatrixH( allDocs.size() );
    for(int i = 0, n = allDocs.size(); i < n; i++) {
        for(int j = 0, k = allDocs.size(); j < k; j++) {
            double similarity = 0.0d;
          Document docX = idxR.document(i);
            String xURL= docX.get("url");
            if ( i == j ) {
              // 避免自吹自擂
              docMatrix.addLink(xURL, xURL, similarity);
            } else {
            TermFreqVector x =
➥ idxR.getTermFreqVector(i, "content");
            TermFreqVector y =
➥ idxR.getTermFreqVector(j, "content");
            similarity = getImportance(x.getTerms(),
➥ x.getTermFrequencies(), y.getTerms(), y.getTermFrequencies());
            // 添加从 docX 到 docY 的链接
            Document docY = idxR.document(j);
            String yURL = docY.get("url");
```

Download at http://www.pin5i.com/


```
                docMatrix.addLink(xURL, yURL, similarity);
            }
        }
    }
    docMatrix.calculate();
    return docMatrix;
}
// 计算文档 Y 对于文档 X 的重要性
private double getImportance(String[] xTerms, int[] xTermFreq,
                             String[] yTerms, int[] yTermFreq){
// xTerms 数组是第一个文档中的最频繁词汇
    Map<String, Integer> xFreqMap =
➥ buildFreqMap(xTerms, xTermFreq);
   // yTerms 数组是第二个文档中的最频繁词汇
    Map<String, Integer> yFreqMap =
➥ buildFreqMap(yTerms, yTermFreq);
    // sharedTerms 是两个集合的交集
    Set<String> sharedTerms =
➥ new HashSet<String>(xFreqMap.keySet());
    sharedTerms.retainAll(yFreqMap.keySet());
    double sharedTermsSum = 0.0;
    // 注意，这个不是对称的。
    // 如果交换 X 和 Y，就能得到不一样的值；
    // 当然，若两者的频次相同，则不会有变化！
    double xF, yF;
    for(String term : sharedTerms) {
        xF = xFreqMap.get(term).doubleValue();
        yF = yFreqMap.get(term).doubleValue();
        sharedTermsSum += Math.round(Math.tanh(yF/xF));
    }
    return sharedTermsSum;
}
private Map<String, Integer> buildFreqMap(String[] terms, int[] freq) {
int topNTermsToKeep = (termsToKeep == 0)? TERMS_TO_KEEP: termsToKeep;
Map<String, Integer> freqMap =
➥ TermFreqMapUtils.getTopNTermFreqMap(terms, freq, topNTermsToKeep);
    return freqMap;
}
```

59

Download at http://www.pin5i.com/

}

我们的解决方案有两个基本要素。第一，用到了 Lucene 的搜索词向量（term vector），其中是搜索词和它们的出现频率。在讨论用 Lucene 索引文件时曾提到，首先要解析文档中的文本，然后在索引之前对其进行分析。在分析阶段，要把文本分解成词汇（token 或 term），具体的分词方式取决于所使用的分析方法。有了 Lucene，就能方便地获得和使用这些信息。此外，Lucene 不仅仅提供文本的分词结果，还会提供每个词汇在文档中出现的次数。这就是我们从 Lucene 得到的信息：一组词汇，以及它们在文档中出现的次数。

解决方案的第二个要素是赋予每个文档一定的重要性。在代码清单 2.15 的 getImportance 方法中，对于文档 X，计算文档 Y 的重要性分为两步：（1）找到文档 X 和 Y 中出现最频繁的词汇的交集；（2）对于交集中的每个词，计算该词在 X 和 Y 两个文档中出现次数的比值。对该比值取双曲正切值（Math.tanh）后再四舍五入 (Math.round)，最后求出所有比值的总和，就得到了文档 Y 在文档 X 中的重要性。这些运算的最终结果就是矩阵 $\boldsymbol{H}$ 中第 X 行第 Y 列的元素。

我们使用双曲正切函数是为了估计在两个文档中都出现的一个词是否是合适的重要性指标。我们并不关注具体的值，我们要做的只是将重要性因子控制在合理的范围内。双曲正切函数得到的值介于 0 和 1 之间，所以最终四舍五入的结果就是一个词或是被忽略或是有 1 个单位的重要性。这就是用这些函数构建公式的原因。 60

图 2.11 中是“nvidia”的搜索结果，biz-05.doc 排名最高，这是一个合理的文档（不是垃圾），而且确实是和 nvidia 有关的。垃圾网页之所以还能排在第二，是因为我们的文档数量太少了，但我们确实也看到了 DocRank 的效果。Lucene 所使用的信息是完全一样的，但它所得到的相关性却被垃圾文档歪曲了。DocRank 帮我们提高了 biz-05.doc 文档的相关性，在实际的应用中，它能帮我们从一堆文档中找到最相关的那个。与 PageRank 一样，DocRank 的优点在于只需要计算一次，得到的结果就能反复使用。 61

还有其他方法改进普通文件搜索的方法，在本章末提供了有关的参考资料。DocRank 在处理来自关系数据库的数据时会变得更加强大。下面看一个常见的情况，假设有两个表——表 A 和表 B，两者通过表 C 联系在一起。例如，一个表中存放的是用户，另一个表中存放的是用户分组，再有一个表用两者的 ID 保存着用户和分组之间的关系。因此，可以根据用户的分组构建一个图，也可以根据组内的用户构建

Download at http://www.pin5i.com/

一个图。只要能够用这样相互链接的形式表示出来，那就可以试试 DocRank 或是类似的算法。不要害怕尝试！这类问题并没有唯一的答案，有些答案甚至会让你大吃一惊。

```
bsh % oracle.search("nvidia",5,dr);

Search results using Lucene index scores:
Query: nvidia

Document Title: NVIDIA shares plummet into cheap medicine for
you!

Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.doc  ->
Relevance Score: 0.4582
_____________________________________________________________
Document Title: Nvidia shares up on PortalPlayer buy

Document URL: file:/c:/iWeb2/data/ch02/biz-05.doc  ->
Relevance Score: 0.3240
_____________________________________________________________
Document Title: NVidia Now a Supplier for MP3 Players

Document URL: file:/c:/iWeb2/data/ch02/biz-04.doc  ->
Relevance Score: 0.1944
_____________________________________________________________
Document Title: Chips Snap: Nvidia, Altera Shares Jump

Document URL: file:/c:/iWeb2/data/ch02/biz-06.doc  ->
Relevance Score: 0.1852
_____________________________________________________________

Search results using combined Lucene scores and page rank scores:
Query: nvidia

Document URL: file:/c:/iWeb2/data/ch02/biz-05.doc  ->
Relevance Score: 0.03858

Document URL: file:/c:/iWeb2/data/ch02/spam-biz-02.doc  ->
Relevance Score: 0.03515

Document URL: file:/c:/iWeb2/data/ch02/biz-04.doc  ->
Relevance Score: 0.02925

Document URL: file:/c:/iWeb2/data/ch02/biz-06.doc  ->
Relevance Score: 0.02233
_____________________________________________________________
```

图 2.11 Word 文档中“nvidia”基于索引和排序的搜索结果

Download at http://www.pin5i.com/

2.6 大规模实现的有关问题

我们目前所探讨的内容可以用于不同功能、不同领域的 Web 应用。但如果有足够的计算资源要处理海量数据，就会遇到两大类问题。第一类问题是与算法的数学性质有关的；第二类问题则是关于软件工程中如何处理 TB 级甚至 PB 级的海量数据。

大规模计算中最常见的限制就是可分配内存不足。换句话说，因数据太大，无法将相应的数据结构加载到内存中。尤其是像 Java 这样的解释型语言，即使能将数据塞到内存中，还要考虑垃圾回收的问题。在大规模计算中，有两种处理这个问题的基本策略。第一种是构建更有效的数据结构，使得数据能放入内存；第二种是构建高效的分布式的 I/O 基础设施，从而实现海量数据的就地读取和操作。对于像 Google 处理的这种大规模数据，两种策略都要实施，因为任何一点点效率的改进都会给整个系统带来莫大的好处。

若要更有效地表示数据，可以考虑我们用来存储矩阵 $\boldsymbol{H}$ 的结构。原始的链接结构需要一个 double[n][n]的二维数组，而悬孤节点还需要一个 double[n][n]的二维数组，其中 n 是网页的数量（在 DocRank 中是文件的数量）。仔细想一下，当 n 很大时，这实际上是对资源的巨大浪费，因为大部分的 double 元素都是 0。存储这种信息的更有效的方法是用邻接表（adjacency list）。在 Java 中可以更方便地用 HashSets 组成 Hashtable 来实现邻接表。因此，PageRankMatrixH 类的变量 matrix 可以定义为：

62

```
Hashtable<Integer, HashSet<Integer,Double>> matrix;
```

本章的一个习题就是用这类更高效的结构重写我们的算法，甚至还可以用参考编码（reference encoding）或是其他技术（见 Boldi 和 Vigna 的论文）进一步压缩邻接表。参考编码依赖于网页间的相似性，提高了内存效率，但牺牲了实现的简洁性。

在大规模搜索的实现中，另一个要考虑的方面是对 PageRank 值的精度要求（或者是实现其他任何基于 Rank 的类）。要用 PageRank 值区分 N 个网页中的任意两个网页，数值计算的精度就至少要到 $1/N$。因此，如果要处理 N=1000 个网页，10^{-4} 应该足够了；如果要处理几十亿个网页，PageRank 值的精度就要达到 10^{-10}。

考虑这样一种情形：所获取的网页中大部分是悬孤节点。如果是为 Apache 项目集合或是稍小一点的 Jakarta 项目这样中心型的网站构建搜索引擎，就有可能遇到这种情况。Brin 和 Page 发现处理大量的悬孤节点是非常低效的，因为它们在矩阵 $\boldsymbol{H}$

Download at http://www.pin5i.com/

中所对应的元素并不能反映网络的链接结构，只是让矩阵满足某些数学性质而已。他们建议在计算 PageRank 时先删除悬孤节点，在 PageRank 值足够收敛后再把这些节点加回去。

当然，我们并不知道 Google 搜索引擎到底是如何实现的（这是绝对的机密），但我们可以肯定地说，要公平地处理所有的网页，就必须在计算 PageRank 的整个过程中都把悬孤节点考虑进来。为了兼顾公平和效率，我们可以使用基于矩阵 $\boldsymbol{H}$ 的对称性的方法。这些方法跟原来的 PageRank 算法有一样的收敛速度，但该问题的规模更小，能显著地节约计算时间。详细信息可参考 *Google's PageRank and Beyond: The Science of Search Engine Rankings* 一书。

所有与大规模搜索的计算有关的讨论关注的都是内存和速度。影响速度的一个因素是幂方法的迭代次数，这取决于 alpha 值和链接的网页数量。不过，在其他类似的书中，我们发现有人断言 PageRank 向量初始值是没有意义的，完全可以都设为 1。严格地说，这并不属实，在处理定期改变的大规模数据集时，PageRank 向量的初始值有很大的影响。初始值越接近最终的 PageRank 值，所需的迭代次数就越少。已经有很多被称为近似聚合的技术（approximate aggregation techniques）可以根据一个较小的矩阵计算出 PageRank 向量，用来估计真正的更新过的 PageRank 向量的分布。反之，估计的结果就会被用来当做 PageRank 的初始值计算最终的结果。本书不会探讨其中的数据原理，与此技术有关的细节可以参考本章末的参考资料。

63

既然是讨论计算 PageRank 向量的加速技术，就不能不提到 Kamvar 等人提出的二次外插技术——Aitken 外插法（extrapolation），以及更加高级的以切比雪夫（Chebyshev）多项式谱方法为代表的谱方法（spectral method）等。这些方法的目的是在两次迭代之间得到一个更好的 PageRank 的近似。你可以在自己的应用中使用这些方法，也可能值得去实现，具体详见参考资料。

在大规模计算实现的软件方面，我们要提到 Hadoop（http://hadoop.apache.org/）。Hadoop 是 Apache 软件基金会的一个成熟的高水平项目，它提供了一个经济、高效、可靠、可扩展的开源软件平台。Hadoop 基于自己的分布式文件系统（HDFS）实现了 MapReduce（见 Dean 和 Ghemawat 的论文）。MapReduce 可以将应用分解成小块的工作。为了可靠，HDFS 会创建数据块的多份副本，并将它们放在计算集群的各个计算节点上（见图 2.12），然后 MapReduce 开始就地处理这些数据。Hadoop 已经成功地运行在有 2000 个节点的集群上，Hadoop 现在的设计目标是要在 10000 个节

Download at http://www.pin5i.com/

点的集群上运行。

在现实的产品系统中，海量数据的处理能力无疑是非常重要的。我们非常粗略地介绍了这些有可能出现的问题，提供了一些相关的项目和参考资料。在设计搜索引擎时，需要考虑的不仅仅是对海量数据的处理，还要考虑到搜索结果的质量。综合考虑，用户需要的是快而好的结果。所以，下面介绍几个量化地评价搜索结果质量的方法。

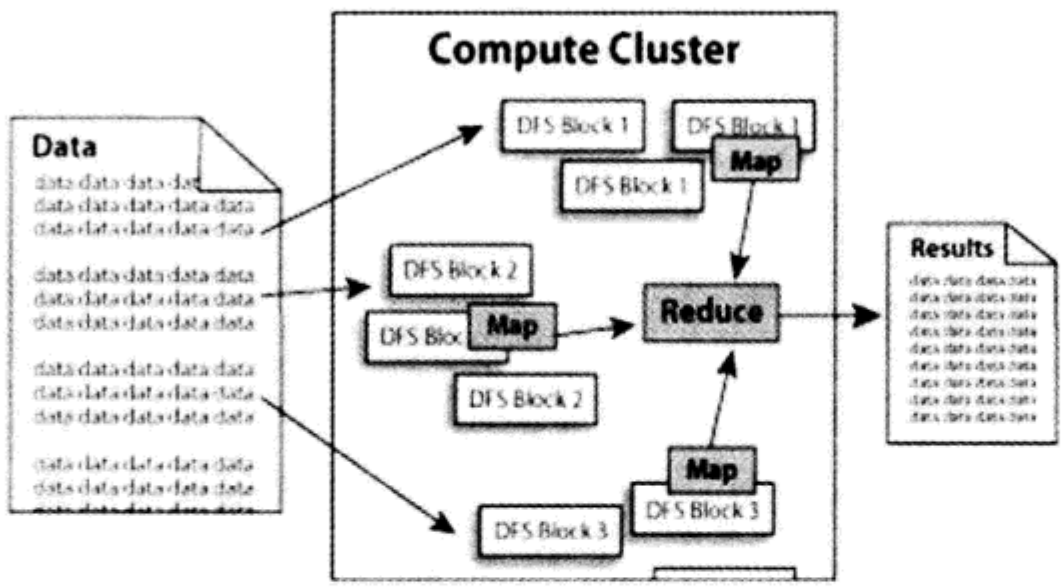


图 2.12 Hadoop 实现的 MapReduce 使用了分布式文件系统

2.7 用户得到了想要的结果吗？精确度和查全率

64

Google 和 Yahoo！都投入了大量的时间研究搜索引擎的质量。跟软件系统的验证和确认过程（QA）一样，搜索质量也是搜索引擎获得成功的一个关键环节。用搜索引擎搜信息时，得到的结果有可能是你想要的，也有可能不是。有很多量化搜索引擎成功程度的方法，其中最常用也是比较容易实现和量化地理解的两种是精确度（precision）和查全率（recall）。

图 2.13 中是一个典型查询的可能结果。给定一个含有若干文件的集合，与查询有关的文件构成一个子集，而查询结果中的文件构成另一个子集。显然，我们的目标是要获得所有的相关文件，但这很难。所以，我们的注意力很快就会转移到图 2.13 中这两个集合的交集上。

在获取信息时，精确度是所获取的相关文件的数量（RR）除以所获取的文件总

Download at http://www.pin5i.com/

数（Rd），即精确度=*RR/Rd*。在图 2.13 中，精确度约为 1/5 或 0.2。这只是用眼睛大致估计的，不是准确值，毕竟我们是工程师。而查全率则是获取的相关文件的数量除以相关文件的总数（Rt），即查全率=*RR/Rt*。

定性地说，这两个评估回答的是不同的问题。精确度回答的是“我得到的结果中间有多少是我想要的？”查全率回答的是“我得到了所有我能得到的吗？”显然，提高精确度比提高查全率容易，因为提高查全率意味着要知道给定查询的所有相关的文件。而事实上，这几乎是不可能的。我们把这两种衡量方法放在一起，看看好结果和坏结果是如何混在一起的。如果我得到的都是对的，没有任何错误，那么精确度和查全率都应该接近 1。

在测试算法和搜索引擎优化措施时，应该针对你的用户想要解决的各类问题的典型查询画出这两个量化指标的图。图 2.14 就是一个典型的这两个指标的图。对于每个查询，都可以用一个点表示这个查询的精确度和查全率。如果执行很多查询，并将这些点画出来，就得到了类似于图 2.14 中的线条。如果查询的数量较少，把这些点连起来就不是一个好主意。保留点的形式，而不要把它们连成线。

65

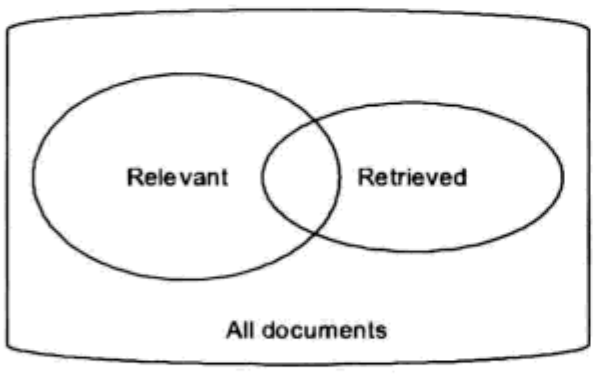


图 2.13　相关文件和所获取的文件，它们的交集用来定义搜索的精确度和查全率

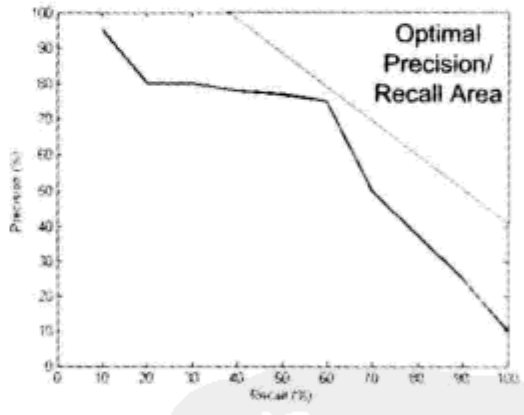


图 2.14　典型的搜索引擎精确度/查全率图

好的精确度/查全率点应该位于图 2.14 的右上角，因为我们希望既有较高的精确度，又有较高的查全率。这些图可以帮助你找到算法中需要改进的地方，或是比较不同算法的优劣。它还可以帮助你劝说那些满腹疑惑的高层管理团队使用本书中的算法。你可以用本章介绍的三种方法（基于 Lucene 的搜索；基于 Lucene 和 PageRank 的搜索；基于 Lucene、PageRank 和用户点击的搜索）做练习，可以将这些方法应用到我们提供的数据集，或者你自己创建的数据集中，然后就可以画出含有 10～20 个查询结果的精确度/查全率图。

Download at http://www.pin5i.com/

在 5.5 节中，我们将会探讨如何评价算法的可靠性，以及如何比较两个算法。还会谈及如何对实验进行验证才能增强我们对结果的信心。精确度和查全率只是评价搜索结果质量的两个基本方法而已。在介绍完所有要介绍的基本智能算法之后，我们会更详细地介绍如何分析算法的可靠性。该方法使我们可以用一个通用的方法去评价智能的质量。

2.8 总结

在 2000 年的时候，就有很多网络新闻预言“搜索为王！”，在 20 世纪能得到这样的结论还算是有先见之明的，但现在，这已经成为一个广为接受的事实。如果有人不相信，可以去 Google 一下！

从本章可以看出，如何智能地满足用户查询遍布全球的富内容材料的需求已经获得了广泛的关注，所涉及的技术也不仅仅是索引。为了讲解搜索策略，我们首先基于 Lucene 库实现了传统的信息获取技术。我们介绍了如何从互联网采集内容（Web 爬取），并且提供了我们自己的爬虫实现。我们还用到了数个文件解析器，例如 NekoHTML 和 TextMining 库（tm-extractor），然后将解析的内容传递给 Lucene 分析器。标准的 Lucene 分析器强大灵活，可以满足大部分需求。如果它们还是不能满足您的需求，我们还讨论一些可能会用到的扩展和修改方法。此外，还提及了 Lucene 查询框架的功能及其可扩展性和灵活性。

更重要的是，我们详细介绍了最负盛名的链接分析算法——PageRank，还提供了一个不依赖其他任何软件的完整实现，并修改 ***G***（oogle）矩阵的公式，使之适用于稀疏矩阵的大规模实现。我们提供了完成该步骤的基本思路，如果你能自己完成，无疑会很有成就感。我们还详细介绍了算法中的一些难点，例如跳转和幂方法等。

我们还展示了用户点击分析，引入了智能概率技术，例如文中实现的 NaiveBayes 分类器。我们提供的包装类揭示了所有相关的重要步骤，同时文中还对其中的大部分代码进行了分析。这种技术让我们获得了用户对某个网站或话题的个人喜好，并还有很大的改进空间，而且可以扩展到其他的用途。 66

没有万能的方法，为了处理非网页的文件，我们还介绍了一种称为 DocRank 的新算法。这种算法很有前景，但更重要的是它说明了经过精心改进后，PageRank 中的数学理论可以方便地扩展到其他领域。最后，我们介绍了在处理大规模网络中可

Download at http://www.pin5i.com/

能会遇到的挑战，以及评测搜索结果和提升搜索引擎可信度的方法。

“搜索为王”的说法可能是对的，但推荐系统也不乏王者血统！下一章关注的是如何生成建议和推荐。把这两者加到你自己的应用中可以给用户带来完全不同的使用体验。在进入下一章之前，务必要阅读以下关于搜索的 To Do 项目，其中充满了有趣和有价值的信息。

2.9 To Do

本书每一章的最后一节都会为读者提供一些 To Do 项目，引导读者探索各种问题。我们的代码中已经在必要的地方注释了“TO DO”标签，这可以在 Eclipse IDE 的 Tasks 面板中看到。随意点击一个任务，该任务的链接就会显示与之相关联的代码。如果读者使用的不是 Eclipse，也可以在代码中搜索“TO DO”。

有些 To Do 项目是更深入地研究主要内容中涉及的话题，有些则是探讨一些跟主要内容中某些话题相关的内容。完成这些任务可以加深和拓宽读者对智能算法的认识。我们强烈建议读者完成这些任务。

下面是第 2 章的 To Do 清单。

1. 构建你自己的搜索引擎。用自己选择的爬虫爬取喜欢的网站，例如http://jakarta.apache.org/，然后用我们的爬虫处理所获取的数据，构建索引，最后在其中进行搜索。

如果加入 PageRank 结果会有什么变化？

加入用户点击呢？

基于本章的知识，你应该能编写出自己的小型搜索引擎。尝试一下，然后告诉我们结果。

2. 尝试 boosting。反注释掉 LuceneIndexBuilder 类的第 83～85 行代码，看看 Lucene 排序结果会有什么变化。根据应用的具体领域，可以采取不同的措施 boosting 你的文件。

67

3. 缩放 PageRank 值。在结合 Lucene（索引）和 PageRank（排序）的搜索例子中，我们用一个缩放因子放大了 PageRank 的作用。选择的指数函数只有一个参数——$m=(1-1/n)$，其中 n 是矩阵 $\boldsymbol{H}$ 的大小。所以在大型网络中，我们的缩放因子会趋向于 1，而在小型网络中则会介于 0 和 1 之间。实际上，当且仅当只有一个网页时，该缩放因子才会等于 0，这样的网络没有实际意义。

尝试不用缩放因子，观察它对排序的影响。还可以修改公式，增加 n 的影响，

Download at http://www.pin5i.com/

例如 m=(1−1/Math.pow(n,k))，当 k 的取值大于 1 时，随着网络增大，最终的 PageRank 值会更快地趋近之前计算得到的值。

4．修改 $\boldsymbol{G}$ 矩阵：悬孤节点。对于悬孤节点在矩阵中对应的行，我们给其中每个元素的赋值是 $1/n$。因为没有关于用户浏览习惯的额外信息，或是假设有足够的用户能覆盖所有的浏览习惯，这种赋值是合理的。如果有其他不同的，但同样合理的假设，整个系统会怎么工作？

假设用户遇到了悬孤节点，可以很自然地假设用户的下一个目标是搜索引擎，或是跟悬孤节点类似的网页，而不会是跟悬孤节点的内容完全无关的网站。基于这种假设，我们可以对悬孤节点的值进行调整：给搜索引擎和有相似内容的网页赋较高的值，而给其他网页赋较低的值。这会给 PageRank 值带来什么影响？给查询结果带来什么影响？精确度/查全率图会有什么变化？

5．修改 $\boldsymbol{G}$ 矩阵：跳转。在我们原来的实现中，跳转的贡献是以平均的方法分配的——所有网页的跳转概率都是(1-*alpha*)/n，其中 n 是网页的数量。但还有很多其他的方法可以用来解决这个问题。如果选择合适，就能反映出各网页的特点；如果从用户的角度选择，就可以像用户点击一样让我们照顾到各个用户的喜好。后者正是跳转贡献也被称为个性化向量（personalization vector）的原因。

尝试修改它，赋给某些网页较高的权重。有效果吗？这些网页的 PageRank 值是不是比较高？这样的实现有什么问题吗？假设我们给每个用户分配一个个性化的向量，从计算量来说这意味着什么？值得吗？可行吗？Haveliwala、Jeh、Widom，以及 Richardson 和 Domingos 的论文与此相关，给你提供了更多有关这个重要话题的信息和观点。 68

6．合并不同的分数。在 2.4.3 节中，为了能得到查询结果的最终排序，我们演示了一种结合三种不同分数的方法，但这种方法并不是唯一的。你可以根据自己的需求调整这三种分数间的关系。这里有一个想法：给三种分数引入权重的概念，然后尝试给每种分数赋予不同的权重。

如果考虑一个给定的网页或文件网络，权重系数的变化会给结果带来什么影响？尝试三组不同的权重组合，画出 20 个不同查询的精确度/查全率图，例如权重组合可以是(0.6, 0.2, 0.2)、(0.2, 0.6, 0.2)、(0.2, 0.2, 0.6)。得到了什么结果？图中的点跟等权重为（1,1,1）得到的点有什么异同？可否提出其他的公式来合理地衡量不同类型的分数的权重？

Download at http://www.pin5i.com/

2.10 参考资料

Adamic, L.A., R.M. Lukose, A.R. Puniyani, and B.A. Huberman. "Search in power-law networks." *Physical Review E*, vol. 64, 046135. 2001.

Boldi, P., and S. Vigna. "The WebGraph Framework I: Compression Techniques." *WWW 2004*, New York.

Dean, J. and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." *Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004. http://labs.google.com/papers/mapreduce-osdi04.pdf.

Haveliwala, T.H. "Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search." *IEEE transactions on Knowledge and Data Engineering*, 15 (4): 784. 2004. http://www-cs-students.stanford.edu/~taherh/papers/topic-sensitive-pagerank-tkde.pdf.

Jeh, G. and J. Widom. "Scaling personalized web search." Technical report, Stanford University, 2002. http://infolab.stanford.edu/~glenj/spws.pdf.

Kamvar, S.D., T.H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation Methods for Accelerating PageRank Computations. WWW 2003. http://www.kamvar.org/code/paper-server.php?filename=extrapolation.pdf.

Langville, A.N. and C.D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.

Richardson, M. and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in PageRank. *Advances in Neural Information Processing Systems*, 14:1441, 2002. http://research.microsoft.com/users/mattri/papers/nips2002/qd-pagerank.pdf.

Rish, I. An empirical study of the naïve Bayes classifier." *IBM Research Report*, RC22230 (W0111-014), 2001. http://www.cc.gatech.edu/~isbell/classes/reading/papers/Rish.pdf.

Download at http://www.pin5i.com/

3 推荐系统

本章内容提要：

- 计算对象之间的距离与相似度
- 了解基于用户、条目与内容的推荐引擎
- 朋友、文章与新闻的推荐
- 像 Netflix 那样的网站是怎样推荐的

在如今生活的世界里，我们被淹没在大量的选择之中，因为我们生活中方方面面的事情几乎都具有不止一个的可选项。我们日常需要做出的决定，包括从汽车到家庭影院，从寻找自己的另一半到选择律师或者会计，从书籍、报纸到 wiki 和博客，从电影到音乐，如此等等。此外，我们每天都得忍受持续地被信息轰炸（有时还是垃圾信息）的事情。在这样“恶劣”的环境下，要是谁能给你推荐一个合适的选择，该是多么有用啊！何况，如果这个推荐还比较切合你的偏好，这种能力更是弥足珍贵。

总有那么一些公司在孜孜不倦地影响你的选择，对于怎样给你一个好的结果这件事情上，广告公司无疑显得特别用心。这些公司存在的理由就是要说服你是真的喜欢 X 商品或者 Y 服务的。但如果你对这些东西没有丝毫的兴趣，那么不单是你感觉到被骚扰，他们的努力也只不过是在浪费时间而已。传统广告的“广播”方式（比如广告牌、电视广告、电台广告）都存在着这样的问题，因为这种“广播”只是希

Download at http://www.pin5i.com/

望通过持续不断地重复相同的信息来改变你的爱好。而另一种更友好、更有效的方式是：定位你的偏爱，然后诱导你去选择符合你个人的愿望与需要的产品。这就是单纯的网络世界与基于网络的智能广告商业之间的区别所在。Google 之所以闻名遐迩，也许是因为它在搜索领域方面的能力，但 Google 之所以富有，却是因为它们的广告系统。

在本章，我们会告诉你关于构建一个推荐引擎的所有知识，你会学习到协同过滤（collaborative filtering）与基于内容（content-based）的推荐引擎，同时还会学习到怎样优化并扩展经典的算法，以使它们能应用到更多的实际应用领域。我们会以一个在线音乐商店中的推荐问题作为引子，然后把我们的解决方案推广到其他的应用环境中。这个在线的音乐商店只是一个简单的例子，但它足够具体、详细，有助于我们理解在构建一个推荐引擎过程中所涉及的所有的基本概念。

后面我们会通过展示一些更复杂的例子来让学习更加有趣。在谨遵商业宣传的重要原则的前提下，例子的内容涵盖在线电影租赁、网络书店和普通网上商店中至关重要的各种推荐引擎。

3.1 一个在线音乐商店：基本概念

假设你有一个专门提供音乐下载销售的在线商店，典型的场景是：注册用户登录到你的在线商店后，可以试听某些歌曲，如果用户喜欢某首歌曲，可以把这首歌曲放到购物车，如此循环，在完成全部的选购之后，下单购买购物车里的所有歌曲。通常来说，当用户在完成购买过程后，或者是当用户刚刚来到这个假想的应用时，我们很希望能给他们推荐更多的歌曲。因为我们拥有数以百万计的可供选择的歌曲，不计其数的歌手，数十种音乐的风格——经典、宗教、流行、重金属、乡村，以及很多其他可以进一步细分的风格，能把它们推销出去当然是一桩更好的买卖。但是，人们通常对那些他们不喜欢的音乐极为敏感，就我个人而言，如果你展示给我的都是与说唱乐（rap）相关的歌曲，那还不如雷死我，而对于另一些人来说，他们会对经典音乐表现出极不感兴趣，如此等等，不一而足。

以上例子的寓意在于，当你给一个用户展示音乐时，你希望这些内容推荐能够命中用户的偏好，同时避开用户不喜欢的内容。这个看起来有点困难？不要害怕，推荐引擎做的正是这样的事情，它会帮助你把正确的内容传递到你的用户面前。

Download at http://www.pin5i.com/

71

一个推荐引擎会仔细审查一个用户以前所做出的选择，然后识别出他对某个自己还没接触过的条目的喜好程度。也即是说，它可以识别出你的用户喜欢什么类型的音乐，进而判定他会购买什么样的音乐，判定的方法是在音乐类型特征上比较他偏好的歌曲与其他歌曲的“相似度”。进而发挥一下想象力——也许我们还可以依据用户对音乐喜好的差异，帮助用户在网站上建立一个关于音乐喜好的社会网络。所以，显而易见，推荐引擎的核心功能在于计算任意两个用户或两个条目之间的相似程度，这种相似度即可用于后续的推荐。

3.1.1 距离与相似度的概念

现在让我们从一些数据开始理解推荐引擎的这些概念的细节，最基本的概念是：条目、用户与评分。在推荐引擎的范畴里，相似度是一个用于比较两个条目相近程度的度量，大致上与我们习惯的用于表征两个城市相近程度的地理距离是一回事。在度量两个城市之间的距离时，我们会倾向于使用它们的经纬度坐标，而在条目或用户这个虚拟空间里，我们则使用“评分”作为这个空间的坐标来度量任意两个节点的距离。下面具体介绍这些概念，我们会从一个名为 MusicUsers 的评分数据库里选取三名用户，然后计算他们对数据库里所有歌曲（条目）的可能评分。

在互联网里，评分通常介于 1 到 5 分之间（含 1 和 5 分），对于选定的一些歌曲，我们令前两名用户（Frank 与 Constantine）给它们打 4 或 5 分（喜欢这些歌曲），同时令第三名用户（Catherine）给它们的评分为 1～3 分。很显然，我们会期待前两名用户彼此是相似的，并同时与第三名用户不相似。当我们通过脚本（代码清单 3.1 第 1 行）载入样例数据后，三名用户及其对条目的评分如表 3.1 所示。

表 3.1 用户的评分显示Frank与Constantine的一致程度要高于Frank与Catherine的一致程度（见图 3.2）

用户名	歌曲名	评分
Frank	Tears In Heaven	5
	La Bamba	4
	Mrs. Robinson	5
	Yesterday	4
	Wizard of Oz	5
	Mozart: Symphony #41 (Jupiter)	4
	Beethoven: Symphony No. 9 in D	5

Download at http://www.pin5i.com/

72

续表

用户名	歌曲名	评分
Constantine	Tears in Heaven	5
	Fiddler on the Roof	5
	Mrs. Robinson	5
	What a Wonderful World	4
	Wizard of Oz	4
	Let It Be	5
	Mozart: Symphony #41 (Jupiter)	5
Catherine	Tears in Heaven	1
	Mrs. Robinson	2
	Yesterday	2
	Beethoven: Symphony No. 9 in D	3
	Sunday, Bloody Sunday	1
	Yesterday	1
	Let It Be	2

下面通过代码清单 3.1 的脚本演示整个过程。

代码清单 3.1　几个 MusicUsers 及其对 MusicItems 的评分

```
MusicUser[] mu = MusicData.loadExample();
mu[0].getSimilarity(mu[1],0);
mu[0].getSimilarity(mu[1],1);
mu[0].getSimilarity(mu[2],0);
mu[1].getSimilarity(mu[2],0); ←相似性是对称的
mu[2].getSimilarity(mu[1],0);
mu[0].getSimilarity(mu[0],0); ←一个用户与其自身的相似性
mu[0].getSimilarity(mu[0],1);
```

这里提供了两个相似性的定义，通过类 MusicUser 的成员函数 getSimilarity 的第二个参数来区别调用。后面会简单描述这些代码的实现细节，但首先让我们看看图 3.1 中三名用户的相似性比较结果。

73

图 3.1 显示的是我们计算的结果：与 Catherine 相比，Frank 与 Constantine 对歌曲的偏好更相似；两名用户的相似性与他们作为参数传入 getSimilarity 函数的顺序无关；Frank 与他自身的相似度为 1.0，这也是任意两个实体之间相似度的最大值。之所以存在这些特性，是因为很多相似性度量是基于距离进行计算的，类似于我们

Download at http://www.pin5i.com/

在中学学过的平面上两个点的几何距离。

```
bsh % MusicUser[] mu = MusicData.loadExample();

bsh % mu[0].getSimilarity(mu[1],0);

 User Similarity between Frank and Constantine is equal to
0.3911406349860862

bsh % mu[0].getSimilarity(mu[1],1);

 User Similarity between Frank and Constantine is equal to
0.22350893427776353

bsh % mu[0].getSimilarity(mu[2],0);

 User Similarity between Frank and Catherine is equal to 0.
004197074413470947

bsh % mu[1].getSimilarity(mu[2],0);

 User Similarity between Constantine and Catherine is equal to
0.0023790682635077554

bsh % mu[2].getSimilarity(mu[1],0);

 User Similarity between Catherine and Constantine is equal to
0.0023790682635077554

bsh % mu[0].getSimilarity(mu[0],0);

 User Similarity between Frank and Frank is equal to 1.0

bsh % mu[0].getSimilarity(mu[0],1);

User Similarity between Frank and Frank is equal to 1.0
```

图 3.1 计算表 3.1 数据中两个用户间的相似度。Frank 与 Constantine 的相似性明显高于 Frank 与 Catherine 的相似性（参看表 3.1）

一般而言，数学上的距离具有如下四个重要的性质：

- 所有的距离都大于或等于 0。与此类似，我们在大部分情况下会把相似度限定为非负值，就像刚刚看到的 MusicUser 的例子那样。实际上，我们限定了相似度的取值范围为区间[0, 1]。
- 任意两点 A 与 B，当且仅当它们是同一个点时，其距离为 0。在我们的例子

Download at http://www.pin5i.com/

中，相似度的性质体现在这样的事实里：当两个用户具有一样的评分记录，他们的相似度等于 1.0。作为佐证，在图 3.1 中，我们计算用户与其自身的相似度，得到的结果为 1.0。如果你创建第四个用户，使他与某一个已有用户的记录一致，也可以得到他们的相似度为 1.0。

- 距离的第三个性质是对称性：A 与 B 的距离等于 B 与 A 的距离。这个性质意味着如果 Catherine 的音乐兴趣与 Constantine 是相似的，那么反之也成立，并且两个相似度的取值是一样的。所以，我们时常希望定义的相似性度量对于传入的参数能够保留距离的这种对称性。
- 数学距离的第四个性质叫三角不等式，因为它关系到三个点之间的距离。按数学术语来描述，如果 $d(A,B)$表示 A 与 B 的距离，那么三角不等式可以表示为：对于任意第三个点 C 来说，$d(A,B) <= d(A,C)+d(C,B)$。在图 3.1 中，Frank 与 Constantine 的相似度为 0.391，Constantine 与 Catherine 的相似度是 0.002，而 Frank 与 Catherine 的相似度是 0.004，这个值小于前两个值的和。不过，这只是碰巧而已，在相似度的定义里，这个性质通常并不能得到保持。

当从距离过渡到相似度时，放松对第四个基本性质的支持是更好的选择，因为并没有义务把距离的所有性质都带到相似度中来。同时应该时刻保持警惕的是：数学只有当我们认为它是合理时才应当被引进来。当我们开始谈论相似度时，关于三角不等式的反例已经有一个世纪之久，这归功于 William James 所说的[1]："火焰与月亮相似是因为它们都是发光体，月亮与一个球相似是因为它们都是圆形的，但与三角不等式相矛盾的是，火焰与球并不相似。"[2]对相似度与认知的关系感兴趣的读者，我们推荐阅读 W.K. Estes 的 *Classification and Cognition*。

在图 3.2 的上图中，我们把两个用户对相同歌曲的评分记录画在一张图上，可视化地展现了 Frank 与 Constantine 的相似性，两条评分曲线相距越近，他们的相似性就越高；反之，相距越远，则相似性越低。图 3.2 的下图是通过同样的方法展示 Frank 与 Catherine 的评分曲线，这两条曲线相距甚远，这一点与我们之前在计算中得到的低相似性的结论是一致的。

Frank 与 Constantine 评分曲线的相近表示他们是相似的，如果查看一下 MusicUser 类中成员函数 plot 的实现，你会看到对评分的差值进行过排序，以使差

1 来自 scholarPedia (http://www.scholarpedia.org/article/similarity_measures)。
2 我们不能认为火焰与球的相似性小于或等于之前两个相似性之和。——译者注

Download at http://www.pin5i.com/

值越来越大。实际效果是，在图中画出来时，从左到右，两条曲线各个点间的差距越来越大。

75

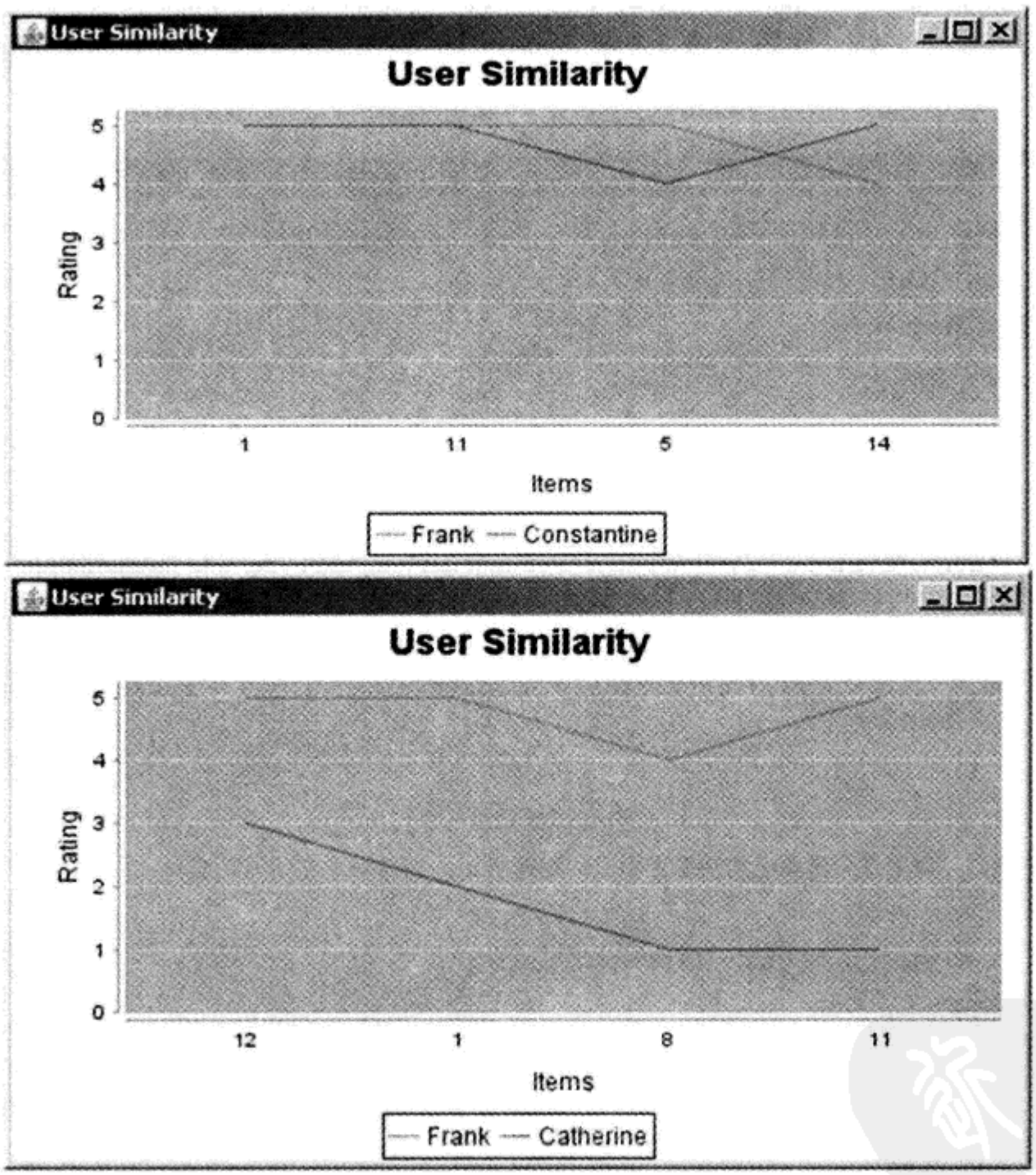


图 3.2 两个用户间的相似度可以通过图中两条曲线间重叠的程度进行度量，所以 Frank 与 Constantine（上）的相似度大于 Frank 与 Catherine（下）的相似度

图 3.2 中关于分值的曲线很明显地展示了距离与相似度天然的互反性。两条曲线的距离越大，意味着两个用户的相似度越小；两条曲线的距离越小，表示两个用户的相似度越大。正如我们在下一节里所看到的那样，尽管并非是必需的，但相似度的计算经常会涉及某种距离的计算，距离与相似度都是度量这个概念里的特殊例子，而距离的概念对于我们来说更为熟悉一些。

76

Download at http://www.pin5i.com/

3.1.2 走近相似度的计算

现在，让我们看看那些计算用户间相似度的代码，以便真正知道这些相似度的值是怎么计算出来的。代码清单 3.2 是 getSimilarity 方法的代码细节，它接受两个参数，第一个为待比较用户的引用，第二个指定要使用哪一种相似性。

代码清单 3.2 MusicUser 类的方法 getSimilarity 中的两种相似性度量

```
public double getSimilarity(MusicUser u, int simType) {
  double sim=0.0d;
  int commonItems=0;
  switch(simType) {
  case 0:
     for (Rating r : this.ratingsByItemId.values()) {←辨识所有共同的条目
        for (Rating r2 : u.ratingsByItemId.values()) {
        //找到相同条目
        if ( r.getItemId() == r2.getItemId() ) {
         commonItems++;
         sim += Math.pow((r.getRating()-r2.getRating()),2); ←对评分的差的平方求和
        }
      }
    }
// 若无相同条目，就无法分辨两个用户是否相似，则返回 0
if (commonItems > 0) {
   sim = Math.sqrt(sim/(double)commonItems);
   // 相似度介于 0～1 之间，
   // 0 表示两用户完全不相似
   // 1 表示两用户爱好一致
  //
   sim = 1.0d - Math.tanh(sim);
}
break;
case 1:
  for (Rating r : this.ratingsByItemId.values()) {←辨识所有共同的条目
   for (Rating r2 : u.ratingsByItemId.values()) {
       //找到相同条目
       if ( r.getItemId() == r2.getItemId() ) {
        commonItems++;
```

Download at http://www.pin5i.com/


```
            sim += Math.pow((r.getRating()-r2.getRating()),2); ←对评分的差的平方求和
          }
        }
      }
// 若无相同条目，就无法分辨两用户是否相似，则返回 0
if (commonItems > 0) {
    sim = Math.sqrt(sim/(double)commonItems);
    // 相似度介于 0～1 之间
    // 0 表示两用户完全不相似
    // 1 表示两用户的爱好一致
    //
    sim = 1.0d - Math.tanh(sim);
    // 算出两用户间最大可能相同的条目
    int maxCommonItems =
➥ Math.min(this.ratingsByItemId.size(), u.ratingsByItemId.size());
    // 考虑相同条目的重要性，以相同条目与最大可能相同条目的比值作为相似度
    sim = sim * ((double)commonItems/(double)maxCommonItems);
}
break;
} //switch 块结束
// 打印结果信息
System.out.print("\n"); // 只是为了使 Shell 输出格式更美
System.out.print(" User Similarity between");
System.out.print(" "+this.getName());
System.out.print(" and "+u.getName());
System.out.println(" is equal to "+sim);
System.out.print("\n"); // 只是为了使 Shell 输出格式更美
return sim;
}
```

这段代码里包含了两个相似度计算的公式，可见相似性的概念是相当灵活和可扩展的。在相似度计算的过程中，首先对于两个用户共同打过分的所有条目，计算他们对每个条目评分的差值，对差值求平方、求和，再对结果求平方根。这样得到的值称为欧氏距离（也叫欧几里德距离），但这并不足以作为相似性的度量值。如我们之前所说，相似度与距离的概念从某种程度上说是互反的，就其意义而言，欧氏距离越小，两个用户的相似度就越大。相似度与距离的这种反序的关系很容易就可

Download at http://www.pin5i.com/

以调整过来，比如，只要定义相似度为欧氏距离加 1，再取倒数即可。

乍看之下，相似度定义为距离的倒数（加上常数 1 之后，避免分母为 0）是可行的，但是这个看似细小的修改其实也是有缺陷的。如果两个用户都对同一首歌曲打过分，其中一个打 1 分，另一个打 4 分，他们评分差值的平方为 9，按以上定义的朴素相似度公式，他们的相似度为 0.25。另一种情况也能得到这个相似度的值：两个用户共同对三首歌曲评分，并且每首歌曲评分的差值都是 1，根据朴素相似性计算公式得到他们的相似度也是 0.25。但是直观地说，我们希望后两个用户比前两个用户更相似，因为前两个用户只有一首共同的歌曲，就这仅有的一个爱好，他们的评分还相差了 3 分（总分才 5 分。）。

78

另一方面，朴素相似度公式削弱了取值小的距离的效应（因为我们加了 1），而对取值大的距离几乎没什么影响（距离的值远大于 1）。如果加的是其他的值呢？朴素相似度公式的通式为：$y=beta/(beta+x)$，其中 *beta* 是自由参数，x 是欧氏距离。图 3.3 显示了当自由参数 *beta* 取值从 1 到 2 时，朴素相似度取值所表示出来的不同形式。

让我们先把朴素相似度的不足缓存在大脑里，再来看看代码清单 3.2 中计算两个用户相似度的第一种定义方式（即 case 0 代码块）。如果两个用户有一些共同评分的歌曲，那么把他们对其中每首歌曲评分差值的平方和除以这部分歌曲的数量，再对商取平方根，然后把结果输入到一个特定的函数中（我们之前见过这个函数：双曲正切函数），最后用 1 减去这个双曲正切函数的返回值，这样最后的相似度取值范围从 0 到 1，0 表示完全不同，1 表示最大的相似度。这样就得到了基于评分的用户相似度的第一种定义。

代码清单 3.2 中相似度的第二种定义（即 case 1 代码块）对第一种进行了一些改进，在于它考虑了两个用户共有条目的量与两人所有可能共有条目的量的比率。这个改进从直观上说是有意义的。如果我听过 30 首歌曲，而你听过 20 首，我们至多可能有 20 首共同的歌曲。现在假定我们实际上有 5 首共同歌曲，并且我们对它们的评分都很相近，这挺好，但为什么我们没有更多的共同歌曲呢？这是否也在某种程度上反映着我们的相似度？这正是第二种相似度公式里力求要反映的因素。换句话说，我们听过的共同歌曲的数量在一定程度上也反映了我们作为一个听众的相似性。

79

Download at http://www.pin5i.com/

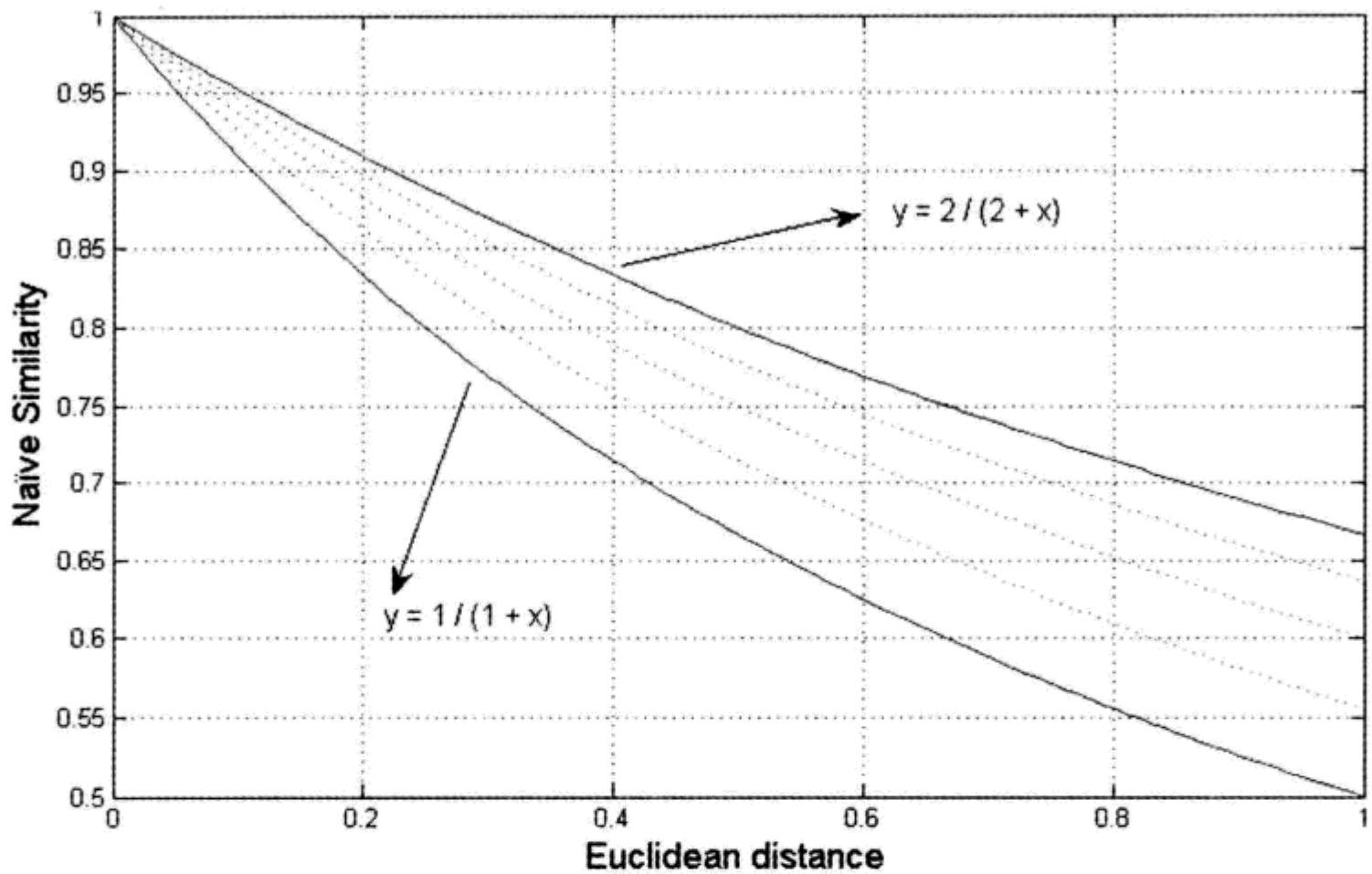


图 3.3 以欧几里德距离为自变量，朴素相似度为因变量的函数曲线

3.1.3 什么才是最好的相似度计算公式？

目前为止，你已很清楚，两个用户或两个条目间的相似度是可以通过不同的计算公式来确定的。除了上面介绍的 MusicUser 类中的两种相似度之外，我们还可以使用雅可比相似度，它定义为两个用户评分条目的交集与这些条目并集之比（如果是计算条目相似度，则是收藏这两个条目的用户的交集与这些用户的并集之比）。换言之，集合 A 与 B 的雅可比相似度用伪代码定义为：*Jaccard* = *intersection*(*A*, *B*) / *union*(*A*, *B*)。下一节我们会使用雅可比相似度，在本章最后的 To Do 事项一节里也会列出更多的相似性计算公式。

既然有那么多的相似度公式，你自然会好奇："哪一种才是更合适的？"这个问题的答案通常需要具体问题具体分析。在这里，答案取决于你的数据。尽管已有更精巧且直观感觉表现会更好的算法，但在为数不多的相似性度量的大规模比较中，由 Spertus、Sahami 和 Buyukkokten 主导的研究结果却表明在七种相似性度量方式中，简单的基于欧氏距离的相似度的实践效果依然是最好的。他们的测量值是基于

Download at http://www.pin5i.com/

1 279 266 次对“相关社区推荐”这个功能的点击中得到的，数据来源是社交网站 Orkut(http://www.orkut.com)自 2004 年 9 月 22 日到 2004 年 10 月 21 日间的点击。

我并不建议你随机地从那些相似性度量中选择一个来用，但是如果你急于使用，不妨选择一个类似于 MusicUser 中介绍的相似度（使用欧氏距离或者雅可比距离），它应该能给你一个不错的结果，你也应当尝试去理解你的数据的本质，以及两个用户或者两个条目相似到底意味着什么。如果你不能理解为什么某一特定的相似性度量是好还是坏，那就比较麻烦。为了说明这个问题，考虑一个我们常有的错觉“两点之间最短的距离是直线”。这个论断只有当它定义在平面几何的时候才成立，而像在足球表面的情况，这个论断则是错的。为了说服你自己，比较一下这两个距离：翻过一座很高但并不宽的山到达对面，以及绕着山脚走过去。这个时候“直线”就不是最短的距离了。

总之，推荐的基石在于度量任意两个用户或两个条目之间相似度的能力，我们已经提供了大量现成的可用的相似性度量方式，同时音乐商店的例子也展示了当你要做推荐时所面临的典型的数据结构。下面我们要深入一点，转向探究推荐引擎的类型，以及它们是怎么工作的。

80 3.2 推荐引擎是怎么工作的

深入理解了两个用户或两个条目之间相似性的意义之后，下面继续探讨推荐引擎。一般来说，推荐引擎可以分为两类，第一类称为协同过滤（collaborative filtering，缩写为 CF），CF 的第一个实例是大约在 1992 年诞生于施乐帕洛阿尔托研究中心的实验邮件系统，由 Goldberg 等人所构建。CF 依赖于用户与系统交互时所留下的蛛丝马迹来工作，这些零碎的足迹通常是用户的评分，就像上一节我们描述的歌曲评分那样的东西。CF 并不只限于一维的或是离散的变量，它的主要特点在于它是依据用户已往的行为做出推荐，而非兴趣集里各个条目内容的差异，所以 CF 不需要领域知识，在进行推荐前也不需要对内容进行事先收集与分析。

推荐引擎第二个常见的类别是基于内容分析的推荐（content-based）——与条目或用户（或两者）相关的内容。基于内容的方法的主要特点在于与用户和条目相关的内容的积累与分析。这个过程可以通过系统或者外部资源来完成，系统收集信息主要通过用户显式和隐式的行为来完成，前者如用户对调查问卷的反馈，后者如对

Download at http://www.pin5i.com/

用户介绍、新闻阅读、电子邮件、博客等内容的分析。

对于 CF 类别，我们会描述基于用户相似度与基于条目相似度的推荐系统。我们也会介绍基于内容的推荐系统，这样就覆盖了目前所有的推荐引擎类别。

3.2.1 基于相似用户的推荐

古希腊有句谚语（世界上几乎每个民族都有类似的说法）是这么说的："观其友，知其人。"以相似用户为邻居进行推荐的协同过滤算法或多或少都是这个谚语的现实案例。为了评估某一用户对某一条目的可能评分，我们需要找到与他相似的用户（称为邻居或者朋友）对这个条目的评分，然后，把每个朋友的评分乘以其自身的权重，最后把朋友们的加权评分都加起来就算完成。原理上说，就是这么简单！

代码清单 3.3 展示了怎么创建并使用一个名为 Delphi 的推荐引擎的全过程。首先，我们需要创建算法使用的数据集，样例数据集通过给所有的用户分配评分来产生。对于每个用户，我们从在线音乐商店里随机抽取 80%的歌曲，如果这个用户的名字首字母为 A 至 D（含），我们令他对这些歌曲的评分为 4 分或 5 分，如果这个用户的名字首字母为其他值，则评分被随机地分配为 1、2 或 3 分。

这样，就建立两个组内具有相似偏好的大组别，以便我们快速地评价推荐引擎的结果。

代码清单 3.3 基于相似用户产生推荐 81

```
BaseDataset ds = MusicData.createDataset();←创建音乐数据集
ds.save("C:/iWeb2/deploy/data/ch3_2_dataset.ser");←保存它以备后用
Delphi delphi = new Delphi(ds,RecommendationType.USER_BASED); ←创建推荐引擎
delphi.setVerbose(true);
MusicUser mu1 = ds.pickUser("Babis");
delphi.findSimilarUsers(mu1);
MusicUser mu2 = ds.pickUser("Lukas");
delphi.findSimilarUsers(mu2);
delphi.recommend(mu1); ←推荐一些歌曲
```

找到相似用户

上述代码中，第一行代码用于创建用户对条目评分的数据集，产生的规则如前所述。代码很直截了当，你也可以在 MusicData 类中按你所想的去修改它。第二行代码用于存储数据集，以便在以后的实验中继续使用。第三行代码用于创建推荐引擎 Delphi 的一个实例，第四行设置它为记录模式，这样就可以看到结果的详细信息。

Download at http://www.pin5i.com/

需要注意的是，Delphi 的构造函数使用的是 Dataset 接口，而不是我们之前的样例数据。你可以自己实现相应的代码，或直接用 JAR 包中提供的。

图 3.4 显示了 findSimilarUsers 方法的结果。在第一种情况里，用户名首字母为 B，所以返回他所有相似朋友的名字都是以 A 到 D 的字母作为首字母的。第二种情况是，用户名首字母为 J，返回所有相似朋友的名字首字母都是 E 到 Z 中的一个。两种情况的结果与我们预期的一致。

看起来推荐引擎工作得不错！但我们还注意到第一种情况中的相似度取值要大于第二种情况的取值，这是因为前者的评分只有两个值（4 和 5），而后者则有三个值（1、2 和 3），且分布得更离散一些。这类健全性的检查是很有用的，你也应当对智能算法返回的结果始终保持警惕，因为如果它们的结果不符合我们的常识标准，又谈何智能？

此外，图 3.4 还显示了对其中一名用户的歌曲推荐结果，以及对每个推荐的预测评分。需要注意的是，虽然用户的评分是整数，但推荐引擎会对它的预测评分使用双精度浮点数 double，因为预测值反映的只是关于评分的可信赖度，而非实际的评分值。你也许会问为什么网站不允许打一个非整数的分值，或者说，为什么不提供更大的评分范围，比如从 1 到 10，甚至到 100。这是很好的问题，我们会在本章末的 To Do 事项里再度谈及。

可以看到推荐引擎计算的预测评分值准确地落在 4～5 之间，这是因为名字首字母为 A 到 D 的用户的评分值都是 4 或 5。

82

Delphi 引擎是怎样得到这样的效果的？它是怎么根据每一个用户找到他的朋友的呢？它又怎么能从这个用户压根就没听过的歌曲列表中推荐出他喜欢的？那就让我们细细检查一下其中的基本过程，看看到底发生了什么吧。协同过滤型的推荐引擎有两个步骤：首先，计算用户或者条目间的相似度；然后，根据其他用户评分的加权平均值来计算某个用户对一个他未评分条目的预测评分。

计算用户相似度

因为要进行的是基于用户相似度的推荐，所以 Delphi 首先要做的是计算用户之间的相似度，这部分代码定义在代码清单 3.4 的 UserBasedSimilarity 类（Delphi 的辅助类）的方法 calculate 中。因为考虑到相似度矩阵的对称性，所以其中两层循环内的计算被优化过，在这段代码之后，我们会对此进行讨论，并介绍更多的优化技巧。

Download at http://www.pin5i.com/


```
bsh % MusicUser mu1 = ds.pickUser("Bob");
bsh % delphi.findSimilarUsers(mu1);

Top Friends for user Bob:

name: Babis                                , similarity: 0.692308
name: Alexandra                            , similarity: 0.666667
name: Bill                                 , similarity: 0.636364
name: Aurora                               , similarity: 0.583333
name: Charlie                              , similarity: 0.583333

bsh % MusicUser mu2 = ds.pickUser("John");
bsh % delphi.findSimilarUsers(mu2);

Top Friends for user John:

name: George                               , similarity: 0.545455
name: Jack                                 , similarity: 0.500000
name: Elena                                , similarity: 0.461538
name: Lukas                                , similarity: 0.454545
name: Frank                                , similarity: 0.416667

bsh % delphi.recommend(mu1);

Recommendations for user Bob:

Item: I Love Rock And Roll          , predicted rating: 4.922400
Item: La Bamba                      , predicted rating: 4.758600
Item: Wind Beneath My Wings         , predicted rating: 4.540900
Item: Sunday, Bloody Sunday         , predicted rating: 4.526800
```

图 3.4 使用基于用户相似度的 Delphi 推荐引擎发现朋友及提供推荐

代码清单 3.4 UserBasedSimilarity:计算用户相似度 83

```
protected void calculate(Dataset dataSet) {
  int nUsers = dataSet.getUserCount();←定义相似度矩阵的规模
  int nRatingValues = 5; ←定义评分计数矩阵的规模
  similarityValues = new double[nUsers][nUsers];
  if( keepRatingCountMatrix ) {
    ratingCountMatrix = new RatingCountMatrix[nUsers][nUsers];
  }
  // 如果需要从用户 Id 到下标索引的映射，则为每个用户生成索引
  if( useObjIdToIndexMapping ) {
    for(User u : dataSet.getUsers() ) {
      idMapping.getIndex(String.valueOf(u.getId()));
    }
```

Download at http://www.pin5i.com/


```
}
for (int u = 0; u < nUsers; u++ ) {
  int userAId = getObjIdFromIndex(u);
  User userA = dataSet.getUser(userAId);
  for (int v = u + 1; v < nUsers; v++) {←❶相似度矩阵
    int userBId = getObjIdFromIndex(v);
    User userB = dataSet.getUser(userBId);
    RatingCountMatrix rcm =
➥   new RatingCountMatrix(userA, userB, nRatingValues); ←两个用户评分的一致性矩阵
    int totalCount = rcm.getTotalCount();
    int agreementCount = rcm.getAgreementCount();
    if (agreementCount > 0) {←计算相似度或把它置为零
      similarityValues[u][v] =
➥   (double) agreementCount / (double) totalCount;
    } else {
      similarityValues[u][v] = 0.0;
    }
    // 对大数据集的处理
    if( keepRatingCountMatrix ) {
      ratingCountMatrix[u][v] = rcm;
    }
  }
  // 对于 u == v 的情况，赋值为 1。
  // 此时不会创建 RatingCountMatrix
  similarityValues[u][u] = 1.0; ←❶相似度矩阵
 }
}
```

❶ 这就是之前所说的计算优化。正常的思路是第一层循环选定第一个用户，接着第二层循环遍历其他所有的用户。但在这里，根据相似度矩阵的对称性，即是说，如果用户 A 与用户 B 的相似度为 X，那么 B 与 A 的相似度也是 X，可以在第二层循环中避免对同一组相似度进行重复计算。另外，代码里也避免了计算用户与其自身的相似度，因为这个值总为 1。这两处的代码优化只是反映了每种相似性度量中都会遵守的基本特性，就像 3.1.1 节所述的那样。

84

如你所见，这里的相似性是通过雅可比度量来定义的，评分上的一致性代表两个用户评分集的交集，评分总数代表两个评分集的并集。相似度被放在一个两维的 double 型的数组中。但相似度是对称的，这意味着不管怎么定义相似性，只要我与

Download at http://www.pin5i.com/

你相似，那么你也与我相似。所以，很显然的是，我们可以通过稀疏矩阵或者其他的一些特殊结构来使得对它们的存储更有效，后者比如只存储矩阵一半值的上三角矩阵。从计算角度来说，代码清单 3.4 的代码已经利用了这个事实。这里再强调一次，第二层循环并没有遍历所有的用户，而是从外层循环中的用户在用户列表中的位置作为遍历的起点。

对每一次用户的相似度计算依赖于另一个附加的类 RatingCountMatrix，这个类的目的是把一个用户与另一个用户的评分关系存储成表格形式，以便后续平滑地计算相似度。代码清单 3.5 包含了 RatingCountMatrix 的代码。

代码清单 3.5 把两个用户在评分上的一致信息保存在表格中

```
public class RatingCountMatrix implements Serializable {
  private int matrix[][] = null;
  public RatingCountMatrix(Item itemA, Item itemB,
➥ int nRatingValues) {
    init(nRatingValues);
    calculate(itemA, itemB); ←计算基于条目的相似度
  }
  public RatingCountMatrix(User userA, User userB,
➥ int nRatingValues) {
    init(nRatingValues);
    calculate(userA, userB); ←计算基于用户的相似度
  }
  private void init(int nSize) {
    // 开始点——所有的元素都为 0
    matrix = new int[nSize][nSize];
  }
  private void calculate(Item itemA, Item itemB) {←计算基于条目的相似度
    for (Rating ratingForA : itemA.getAllRatings()) {
      // 检查是否有相同的用户给条目 A 和条目 B 都打了分
      Rating ratingForB =
➥ itemB.getUserRating(ratingForA.getUserId());
      if (ratingForB != null) {
        int i = ratingForA.getRating() - 1;
        int j = ratingForB.getRating() - 1;
        matrix[i][j]++;
      }
```

初始化评分计数矩阵

Download at http://www.pin5i.com/


```
    }
  }
  private void calculate(User userA, User userB) {←计算基于用户的相似度
    for (Rating ratingByA : userA.getAllRatings()) {
      Rating ratingByB =
➥ userB.getItemRating(ratingByA.getItemId());
      if (ratingByB != null) {
        int i = ratingByA.getRating() - 1;
        int j = ratingByB.getRating() - 1;
        matrix[i][j]++;
      }
    }
  }
  public int getTotalCount() {
    int ratingCount = 0;
    int n = matrix.length;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        ratingCount += matrix[i][j];
      }
    }
    return ratingCount;
  }
  public int getAgreementCount() {                    ← 不同计数器的辅助方法
    int ratingCount = 0;
    for (int i = 0, n = matrix.length; i < n; i++) {
      ratingCount += matrix[i][i];
    }
    return ratingCount;
  }
  public int getBandCount(int bandId) {
    int bandCount = 0;
    for (int i = 0, n = matrix.length; (i + bandId) < n; i++) {
      bandCount += matrix[i][i + bandId];
      bandCount += matrix[i + bandId][i];
    }
    return bandCount;
  }
}
```

86

Download at http://www.pin5i.com/

这个类的核心是那个二维的 int 型数组（这里的维度是 5×5），它存储了两个用户在评分上的一致性。我们假定用户 A 与 B 都听了 10 首歌，其中对 6 首的评分一致，而其余的都不同。这个矩阵所有的单元都被初始化为 0；对于两个用户对同一条目的评分，在分值对应的行与列中加 1。所以，如果三个一致性的评分是 4 分，另三个是 5 分，就会得到 matrix[3][3]与 matrix[4][4]都是 3。所以，只要把 matrix 数组对角线的元素加起来，就能得到两个用户评分一致的次数。

这种保存分值的方式有三个好处：第一，如果分值范围是从 1 到 10（或者 1 到 100），它可以继续沿用这里的处理方法；第二，就像下面将要看到的那样，它允许你以此衍生出更多精巧的相似性度量方式，比如，除了考虑用户间评分一致的次数之外，还可以考虑用户间评分不一致的次数，或由此得到的其他信息；第三，这个矩阵形式还可以泛化为更一般的对象，它可能不再是一个二维数组，而是一个更复杂的结构，比如，当你需要一个比简单的评分更复杂的评估方式时，构造一个更复杂的数据结构就变得必要了。

Delphi 内部工作机制

现在代码清单 3.4 的代码已经解释过了，这里，两个用户 A 与 B 的相似度就简单地等于两者一致性评分的次数除以两者共同评分过条目的数量。所以，距离创建我们的推荐引擎已经只有一步之遥了。

代码清单 3.6 Delphi: 基于用户相似度产生推荐

```
public List<PredictedItemRating> recommend(User user, int topN) {
  List<PredictedItemRating> recommendations =
➥ new ArrayList<PredictedItemRating>();
  for (Item item : dataSet.getItems()) {←遍历所有的条目
     // 只考虑该用户还未评分的条目
     if (user.getItemRating(item.getId()) == null) {
       double predictedRating = predictRating(user, item);←为这名用户预测评分
  if (!Double.isNaN(predictedRating)) {
     recommendations.add(new PredictedItemRating(user.getId(),←把候选推荐的预测分
                                                                 值累加起来
                         item.getId(), predictedRating));
  }
     }
  }
```

Download at http://www.pin5i.com/

87

```
  Collections.sort(recommendations);
  Collections.reverse(recommendations);  对候选推荐进行排序
  List<PredictedItemRating> topRecommendations =
➡ new ArrayList<PredictedItemRating>();
  for(PredictedItemRating r : recommendations) {←选出前 N 个推荐
     if( topRecommendations.size() >= topN ) {
            // 已有足够的推荐
            break;
     }
     topRecommendations.add(r);
  }
  return recommendations;
}
```

代码清单 3.6 展示了 Delphi 的高级方法 recommend，正如我们在代码清单 3.3 中所见，它是用于产生推荐结果的。这个方法忽略了用户已经打过分的条目，此功能并非是必需的，可根据实际的应用而定。如果你需要更改它，可以在这个方法中进行修改，例如，可以给第一个 if 语句添加一个 else 子句。

recommend 方法通过调用 predictRating(user, item)方法来计算一个用户（第一个参数）对所有条目的预测评分。代码清单 3.7 展示了 predictRating 方法的实现，它又通过调用 estimateUserBasedRating 方法来计算具体的加权分值。recommend 方法存在的意义在于隐藏了各种不同的相似度计算方式（比如 user-based，item-based 等）的细节，其中有些逻辑分支实现了，有些还没有，所以你有一定的空间可以去完善它们。

代码清单 3.7 预测一个用户对一个条目的评分

```
public double predictRating(User user, Item item) {
  switch (type) {
    case USER_BASED:
      return estimateUserBasedRating(user, item);
    case ITEM_BASED:
      return estimateItemBasedRating(user, item);
    case USER_CONTENT_BASED:
      throw new IllegalStateException(
➡ "Not implemented similarity type:" + type);
    case ITEM_CONTENT_BASED:
```

Download at http://www.pin5i.com/


```
      throw new IllegalStateException(
➥ "Not implemented similarity type:" + type);
    case USER_ITEM_CONTENT_BASED:
      return MAX_RATING * similarityMatrix
➥ .getValue(user.getId(), item.getId());
    }
    throw new RuntimeException("Unknown type:" + type);
  }
```

88

代码清单 3.8 的 estimateUserBasedRating 方法是基于用户相似度的实现。如果已经存在该用户对该条目的评分，则直接返回这个分值。这里的控制流与代码清单 3.6 稍有不同，因为虽然我们只会针对未评分的条目进行计算，但这里的写法可以使其适用于需求更广泛的外部调用。

代码清单 3.8 基于用户间相似度预测评分

```
private double estimateUserBasedRating(User user, Item item) {
  double estimatedRating = Double.NaN;
  int itemId = item.getId();
  int userId = user.getId();
  double similaritySum = 0.0;
  double weightedRatingSum = 0.0;
  // 检查用户是否已经给该条目评过分
  Rating existingRatingByUser = user.getItemRating(item.getId());
  if (existingRatingByUser != null) {
    estimatedRating = existingRatingByUser.getRating();
  } else {
     for (User anotherUser : dataSet.getUsers()) {←遍历其他所有的用户
       Rating itemRating = anotherUser.getItemRating(itemId);←为同一个条目计算分值
       // 只考虑该条目打过分的用户
       if (itemRating != null) {
         double similarityBetweenUsers =
          similarityMatrix.getValue(userId, anotherUser.getId());  计算两个用户的相似度
         double ratingByNeighbor = itemRating.getRating();
         double weightedRating =
➥ similarityBetweenUsers * ratingByNeighbor;  根据相似度对分值加权
         weightedRatingSum += weightedRating;
         similaritySum += similarityBetweenUsers;
       }
```

Download at http://www.pin5i.com/


```
        }
        if (similaritySum > 0.0) {
           estimatedRating = weightedRatingSum / similaritySum;←以加权和与直接和之比
                                                                 作为预测分值
        }
    }
    return estimatedRating;
}
```

89

更值得关注的是，用户未曾对该条目评分的情况，我们通过一个循环遍历所有对该条目有过评分记录的用户，每一个用户的贡献值等于他的评分乘以他对待计算用户的影响权值（相似度）。变量 similaritySum 用于标准化——所有权值的和标准化为 1。

从代码清单 3.4 到代码清单 3.6 中的实现可知，如果系统里的用户量比较大，计算他们的推荐将是一件棘手的事情，而这种情况在大型的在线商店里都是司空见惯的。实际应用中的这些代码还有很大的可优化的空间，之前我们已经提及一些存储方面的优化技巧，但我们还可以通过实施另一种结构上的变化，从而使运行时的空间与时间效率都能得到了极大的改善，这就是 kNN。当计算用户之间的相似度时，我们可以只存储前 k 个最相似的用户，而且只基于这 k 个相似用户来计算预测分值，而不是像之前那样遍历全部对某条目打过分的用户来计算。kNN 这个缩写中 NN 表示最近的邻居（nearest neighbors），而 k 表示我们需要考虑邻居的数量。基于用户相似度的推荐是一个可信赖的技术，但如果用户量很大，即使实施 kNN 的优化技巧，它的效率还是会有问题，这种情况下，使用基于条目的相似度会是一个更好的选择。

3.2.2 基于相似条目的推荐

基于条目（item-based）的协同过滤与基于用户（user-based）的情况很类似，只是把用户相似度置换成条目相似度罢了。下面我们通过配置 Delphi 使它能依据条目相似度做出推荐，代码清单 3.9 的代码就是为了实现这个目的。首先载入在代码清单 3.3 中保存好的数据集，为了与之前的结果进行比较，我们对同一个用户进行了推荐。此外，代码还计算了与歌曲“La Bamba”相似的歌曲列表。

代码清单 3.9 基于相似条目进行推荐

```
BaseDataset ds = BaseDataset
➥ .load("C:/iWeb2/deploy/data/ch3_2_dataset.ser");←载入代码清单 3.3 中使用的数据
```

Download at http://www.pin5i.com/


```
Delphi delphi = new Delphi(ds,RecommendationType.ITEM_BASED);←创建基于条目的推荐引擎
delphi.setVerbose(true);
MusicUser mu1 = ds.pickUser("Bob");
delphi.recommend(mu1);                    给 Bob 推荐一些条目
MusicItem mi = ds.pickItem("La Bamba");
delphi.findSimilarItems(mi);              找出与 La Bamba 相似的条目
```

图 3.5 显示了代码清单 3.9 中的计算结果。如果把这些结果与图 3.4 进行比较，你会发现推荐结果是一致的，只是顺序变了。当然没有人能保证 user-based 的推荐与 item-based 是一致的，此外，预测的分值可以肯定是不同的。另一个有趣的地方在于，对这个人为的数据集产生的推荐与之前的结果在顺序上刚好相反。这并不会是常态，这里只是意外地发生了。在其他情况，特别是真实的数据集中，结果可能是其他的顺序。你可以试试，当产生不一样的数据集时，运行一下这段代码看看顺序怎么变化。

90

```
bsh % MusicUser mu1 = ds.pickUser("Bob");
bsh % delphi.recommend(mu1);

Recommendations for user Bob:

    Item: Sunday, Bloody Sunday  , predicted rating: 4.483900
    Item: La Bamba               , predicted rating: 4.396600
    Item: I Love Rock And Roll   , predicted rating: 4.000000
    Item: Wind Beneath My Wings  , predicted rating: 4.000000

bsh % MusicItem mi = ds.pickItem("La Bamba");
bsh % delphi.findSimilarItems(mi);

Items like item La Bamba:

    name: Yesterday                    , similarity: 0.615385
name: Fiddler On The Roof              , similarity: 0.588235
name: Vivaldi: Four Seasons            , similarity: 0.555556
name: Singing In The Rain              , similarity: 0.529412
name: You've Lost That Lovin' Feelin'  , similarity: 0.529412
```

图 3.5 使用基于条目相似度的 Dephi 引擎发现相似条目并给出推荐

基于条目相似度的推荐与我们之前看到的很类似，当然，这次我们会使用条目来计算。计算代码在 ItemBasedSimilarity 类的方法 calculate 中实现。

代码清单 3.10 计算条目间的相似度

```
protected void calculate(Dataset dataSet) {
  int nItems = dataSet.getItemCount();←定义相似度矩阵的规模
```

Download at http://www.pin5i.com/


```
int nRatingValues = 5;←定义评分计数矩阵的规模
similarityValues = new double[nItems][nItems];
if( keepRatingCountMatrix ) {
  ratingCountMatrix = new RatingCountMatrix[nItems][nItems];
}
// 若需要从条目 Id 到下标索引的映射，则为每个条目生成索引
if( useObjIdToIndexMapping ) {
  for(Item item : dataSet.getItems() ) {
     idMapping.getIndex(String.valueOf(item.getId()));
  }
}
for (int u = 0; u < nItems; u++) {
  int itemAId = getObjIdFromIndex(u);
Item itemA = dataSet.getItem(itemAId);
// 只需要计算主对角线上方的元素
for (int v = u + 1; v < nItems; v++) {          ❶
  int itemBId = getObjIdFromIndex(v);
  Item itemB = dataSet.getItem(itemBId);
  RatingCountMatrix rcm =
➡ new RatingCountMatrix(itemA, itemB, nRatingValues);←两个条目间评分的一致性矩阵
  int totalCount = rcm.getTotalCount();
  int agreementCount = rcm.getAgreementCount();
  if (agreementCount > 0) {←计算相似度或置为零
     similarityValues[u][v] =
➡    (double) agreementCount / (double) totalCount;
  } else {
     similarityValues[u][v] = 0.0;
  }
  if( keepRatingCountMatrix ) {
     ratingCountMatrix[u][v] = rcm;
  }
 }
 // 对于 u == v 的情况，赋值为 1
 similarityValues[u][u] = 1.0;❶
}
}
```

91

❶ 这个优化与在代码清单 3.4 中计算用户相似度时所看到的相同。

RatingCountMatrix 类再一次被用于记录评分间的一致性与差异性，不同的是，

Download at http://www.pin5i.com/

之前的评分一致/差异的比值是针对两个不同的用户，现在则是针对两个不同的条目而言。代码中穷举了所有不同的条目并计算它们的雅可比相似度。Delphi 类的这部分用于 item-based 推荐的代码基本上跟 user-based 推荐时差不多。同样情况的还有代码清单 3.11 中基于条目相似度的推荐结果的计算，你可以把它与清单 3.8 中的代码进行比较。代码清单 3.6 与代码清单 3.7 的代码则不会因不同的相似度类型而改变，这里不再重复。

代码清单 3.11 Delphi：基于条目相似度生成推荐

```
private double estimateItemBasedRating(User user, Item item) {
  double estimatedRating = Double.NaN;
  int itemId = item.getId();
  int userId = user.getId();
  double similaritySum = 0.0;
  double weightedRatingSum = 0.0;
  // 检查用户是否已经对该条目评过分
  Rating existingRatingByUser = user.getItemRating(item.getId());
  if (existingRatingByUser != null) {
   estimatedRating = existingRatingByUser.getRating();
  } else {
   double similarityBetweenItems = 0;
   double weightedRating = 0;
   for (Item anotherItem : dataSet.getItems()) {←遍历其他所有的条目
    // 只考虑该用户打过分的条目
    Rating anotherItemRating = anotherItem.getUserRating(userId);←为同一个用户计算评分
    if (anotherItemRating != null) {
     similarityBetweenItems =
➥ similarityMatrix.getValue(itemId, anotherItem.getId());←计算两个条目间的相似度
     if (similarityBetweenItems > similarityThreshold) {
      weightedRating =
➥ similarityBetweenItems * anotherItemRating.getRating();←根据相似度加权分值
      weightedRatingSum += weightedRating;
      similaritySum += similarityBetweenItems;
     }
    }
   }
   if (similaritySum > 0.0) {
    estimatedRating = weightedRatingSum / similaritySum;←以加权和与直接和之比作为预测分值
```

Download at http://www.pin5i.com/


```
    }
  }
  return estimatedRating;
}
```

在这些代码清单的帮助下，我们对协同过滤推荐有了初步的认识，实际上即是基于用户与条目的推荐。一般情况下，由于用户的数量如此庞大（数百万甚至更多），基于条目相似度的协同过滤都会是一个更可取的方法。但有时为了得到更好的推荐，两种方法也可以组合起来使用。在后面的章节里，我们会定制一个类似于 Amazon.com（http://www.amazon.com，提供了 item-based 的推荐）的网站，引入这两种方法的组合，然后提供类似于 Netflix.com（http://www.netflix.com）的推荐。

3.2.3 基于内容的推荐

创建一个基于内容的推荐引擎需要依赖用户间、条目间或者用户与条目间内容的相似性。这样即使没有评分数据，也可以度量两个文档到底有多接近了。第 2 章已经讨论过查询与文档间的相关性，文档间距离的概念可视为对那部分内容的一般化，因为总可以认为一篇文档是查询而另一篇是引用。当然，只要考虑每篇文档中有意义的部分即可，否则文档中真正的信息将陷入一片混乱之中。

93

样例设置

我们的文本源是在第 2 章中用到的文档集，通过类似于上述把音乐分配给用户的做法，给每个用户分配一部分网页。对于每个用户，从全部文档的一个候选子集中挑选 80%分配给他，而每个用户的候选子集蕴含着强烈的偏好性。

- 如果用户名首字母为 A 到 D（含），候选子集为商业或体育类。
- 否则，候选子集为美国或国际类。

因而，我们建立起两组各自具有相似偏好用户的组别（虽说是人工生成的），以便快捷地评估结果。接下来，我们开始创建 Delphi 推荐引擎基于内容推荐的模块。代码清单 3.12 的代码用于准备数据，并识别相似用户与相似条目，其中还混合了基于用户-条目内容相似度的条目推荐。

代码清单 3.12 基于内容相似度的推荐引擎

```
BaseDataset ds = NewsData.createDataset();
Delphi delphiUC = new Delphi(ds,RecommendationType.USER_CONTENT_BASED);
delphiUC.setVerbose(true);
```

创建基于用户-内容的引擎

Download at http://www.pin5i.com/


```
NewsUser nu1 = ds.pickUser("Bob");
delphiUC.findSimilarUsers(nu1);
NewsUser nu2 = ds.pickUser("John");
delphiUC.findSimilarUsers(nu2);
Delphi delphiIC = new Delphi(ds,RecommendationType.ITEM_CONTENT_BASED);
delphiIC.setVerbose(true);
ContentItem i = ds.pickContentItem("biz-05.html");
delphiIC.findSimilarItems(i);
Delphi delphiUIC =
new Delphi(ds,RecommendationType.USER_ITEM_CONTENT_BASED);
delphiUIC.setVerbose(true);
delphiUIC.recommend(nu1);
```

创建基于条目-内容的引擎

创建基于用户-条目-内容的引擎

第一行脚本依据刚刚描述的原理创建了一个测试数据集。然后创建一个 Delphi 的实例，它依赖一个由类 UserContentBasedSimilarity 计算生成的用户间相似度矩阵。因为每个用户拥有的文档不止一个，所以我们必须把一个用户所有的文档与其他每一个用户的所有文档都进行比较。有很多方法都可以实现这点，在代码清单 3.13 中，对于每一对用户（即用户 A 与用户 B），遍历 A 的所有文档，并找到 B 中与之相似度最高的文档作为最佳相似度，然后对每个文档的这个最佳相似度取平均，并把这个平均值作为 A 与 B 的相似度。 94

代码清单 3.13 基于文本计算用户相似度

```
protected void calculate(Dataset dataSet) {
  int nUsers = dataSet.getUserCount();
  similarityValues = new double[nUsers][nUsers];
  // 若需要从用户 Id 到下标索引的映射，则为每个用户生成索引
  if( useObjIdToIndexMapping ) {
    for(User u : dataSet.getUsers() ) {
        idMapping.getIndex(String.valueOf(u.getId()));
    }
  }
  CosineSimilarityMeasure cosineMeasure =
➡ new CosineSimilarityMeasure();←生成余弦相似性度量
  for (int u = 0; u < nUsers; u++ ) {
    int userAId = getObjIdFromIndex(u);
    User userA = dataSet.getUser(userAId);
```

Download at http://www.pin5i.com/


```
    for (int v = u + 1; v < nUsers; v++) {❶
      int userBId = getObjIdFromIndex(v);
      User userB = dataSet.getUser(userBId);
      double similarity = 0.0;
      for(Content userAContent : userA.getUserContent() ) {←对用户 A 的所有评分条目进行迭代
        double bestCosineSimValue = 0.0;
        for(Content userBContent : userB.getUserContent() ) {←对用户 B 的所有评分条目进行迭代
          double cosineSimValue = cosineMeasure
➥ .calculate(userAContent.getTFMap(), userBContent.getTFMap());
          bestCosineSimValue =
➥ Math.max(bestCosineSimValue, cosineSimValue);
        }
        similarity += bestCosineSimValue;←累加所有文档的最佳相似度
      }
      similarityValues[u][v] = similarity /
➥ userA.getUserContent().size();←通过简单的取平均得到相似度
    }
    // 对于 u == v 的情况，则赋值为 1.
    similarityValues[u][u] = 1.0;❶
  }
}
```

❶ 这是与代码清单 3.4 中计算用户相似度时相同的代码优化。

95

基于内容的相似度的要点

所有基于内容的方法的关键在于把文本信息表示为数值量，一个简单的方法是找出每个文档中前 N 个出现最频繁的单词，然后用所有文档中的这些频繁词构成坐标空间。我们可以利用 Lucene 的 StandardAnalyzer 类来过滤停用词（stop word）以及把单词退化为词根，从而在增强有意义单词重要性的同时显著地消减噪声。为了达到这样的效果，我们创建了类 CustomAnalyzer，它继承自 StandardAnalyzer，用于去除某些会给文档向量带来极大噪声的常见词。

现在让我们稍稍离题一会，以便有时间把这些要点具体化。为了方便论证，假设 N=4，现有三个文档，它们的频繁词如下：

- D1 = {Google, shares, advertisement, president}
- D2 = {Google, advertisement, stock, expansion}
- D3 = {NVidia, stock, semiconductor, graphics}

Download at http://www.pin5i.com/

每个文档都可以用数学的方法表示为一个 9 维的向量，用于标识该文档是否包含 9 个单词 {Google, shares, advertisement, president, stock, expansion, Nvidia, semiconductor, graphics} 中的一个。所以，这三个文档可以表示为如下所示的三个向量。

- D1 = {1,1,1,1,0,0,0,0,0}
- D2 = {1,0,1,0,1,1,0,0,0}
- D3 = {0,0,0,0,1,0,1,1,1}

这样，我们就创建了三个纯粹的数值量，接下来可以量化地对文档进行比较，这里要使用的相似性度量是余弦相似度。目前为止，我们已经见过很多相似性公式了，现在这个也是常见的相似性公式。为方便起见，我们在代码清单 3.14 中以 CosineSimilarityMeasure 类的方式列出封装了该定义的代码。

代码清单 3.14 计算词向量间的余弦相似度

```
public class CosineSimilarityMeasure {
  public double calculate(double[] v1, double[] v2) {
      double a = getDotProduct(v1, v2);
      double b = getNorm(v1) * getNorm(v2);←标准化两个向量并计算它们的乘积
      return a / b;←得到余弦相似度
  }
  private double getDotProduct(double[] v1, double[] v2) {
      double sum = 0.0;
      for(int i = 0, n = v1.length; i < n; i++) {
        sum += v1[i] * v2[i];
      }
      return sum;
  }
  private double getNorm(double[] v) {←计算一个向量的欧几里德模
      double sum = 0.0;
      for( int i = 0, n = v.length; i < n; i++) {
        sum += v[i] * v[i];
      }
      return Math.sqrt(sum);
  }
}
```

计算点积

96

正如你所看到的，首先我们得到两个向量间的点积（内积），将其存储在双精度变量 *a* 中。然后计算每一个向量的模（向量长度），把它们的乘积存储在双精度变量

Download at http://www.pin5i.com/

b 中，余弦距离就是简单地取 a/b。如果把文档 X 与 Y 的余弦距离表示为 CosSim(X,Y)，在这个例子中可以得到如下的相似度：

- $CosSim(D1,D2) = 2 / (2*2) = 0.5$
- $CosSim(D1,D3) = 0 / (2*2) = 0$
- $CosSim(D2,D3) = 1 / (2*2) = 0.25$

在信息检索领域，基于词来对文档进行表示是基本的技术。但我们必须指出，正确地识别词是关键的一步，对于一般语料库中的文档来说，这一步要做好是有难度的。例如，如果把代码里的 StandardAnalyzer 置换成我们自己的 CustomAnalyzer，看看会发生什么？咋看之下，我们自定义的类中并没有什么太多特别的东西，但结果却发生了显著的变化。这个小实例也许能让你相信基于内容的方法对于词法分析这一步是非常敏感的。

三类基于内容的推荐引擎

回到我们的例子里看看结果。图 3.6 是执行代码清单 3.12 的代码后的部分结果，即用于寻找相似用户的那部分。

```
bsh % BaseDataset ds = NewsData.createDataset();
bsh % Delphi delphiUC =
new Delphi(ds,RecommendationType.USER_CONTENT_BASED);

bsh % delphiUC.setVerbose(true);
bsh % NewsUser nu1 = ds.pickUser("Bob");
bsh % delphiUC.findSimilarUsers(nu1);

Top Friends for user Bob:

    name: Albert        , similarity: 0.950000
    name: Catherine     , similarity: 0.937500
    name: Carl          , similarity: 0.937500
    name: Alexandra     , similarity: 0.925000
    name: Constantine   , similarity: 0.925000

bsh % NewsUser nu2 = ds.pickUser("John");
bsh % delphiUC.findSimilarUsers(nu2);

Top Friends for user John:

    name: George        , similarity: 0.928571
    name: Lukas         , similarity: 0.914286
    name: Eric          , similarity: 0.900000
    name: Nick          , similarity: 0.900000
    name: Frank         , similarity: 0.900000
```

图 3.6　与 Bob 相似的用户名字首字母在 A 到 D 之间。算法成功地识别出两组相似用户

Download at http://www.pin5i.com/

很明显，算法是成功的，因为它正确地识别出两个不同的用户组——名字首字母为 A 到 D 的用户组与名字首字母为 E 到 Z 的用户组——内部的相似性。需要注意的是，相似性的值差异并不大，看来当对用户进行比较时，基于内容的方法并不能产生很好的分离效果。图 3.7 显示了负责寻找相似条目的代码的执行结果，正如你所看到的，一些相关条目被辨识出来，但也有一些条目在人看来却并非那么相似。

```
bsh % Delphi delphiIC =
new Delphi(ds,RecommendationType.ITEM_CONTENT_BASED);

bsh % delphiIC.setVerbose(true);
bsh % ContentItem biz1 = ds.pickContentItem("biz-01.html");
bsh % delphiIC.findSimilarItems(biz1);

Items like item biz-01.html:

    name: biz-03.html    , similarity: 0.600000
    name: biz-02.html    , similarity: 0.600000
    name: biz-04.html    , similarity: 0.100000
    name: biz-07.html    , similarity: 0.100000

bsh % ContentItem usa1 = ds.pickContentItem("usa-01.html");
bsh % delphiIC.findSimilarItems(usa1);

Items like item usa-01.html:

    name: usa-02.html    , similarity: 0.300000
    name: usa-03.html    , similarity: 0.300000
    name: world-03.html  , similarity: 0.100000
    name: world-05.html  , similarity: 0.100000
    name: usa-04.html    , similarity: 0.100000

bsh % ContentItem sport1 = ds.pickContentItem("sport-01.html");
bsh % delphiIC.findSimilarItems(sport1);

Items like item sport-01.html:

    name: sport-03.html  , similarity: 0.400000
    name: sport-02.html  , similarity: 0.300000
```

97

图 3.7 与被查询条目来自同一个类别的条目被正确地识别为相似

你会再一次看到这些相似值的差异并不大，这一点导致算法很难给出优质的推荐，这种模糊的区分能力源于词法分析的缺乏。自然语言处理（NLP）是一个非常广袤而又很有难度的领域，然而，它在过去的二十年里已经取得了长足的发展。本书虽然不会深入探讨这个激动人心的主题，但我们会把 NLP 系统的各个组成部件放

Download at http://www.pin5i.com/

在附录 D 中做一个概述。

图 3.8 给出的是基于用户–条目相似度的推荐。虽然 CF 通常用于处理用户间或条目间的相似度，但基于内容的方法的一个优点就是可以基于用户–条目相似度来生成推荐。然而，词法分析的问题仍旧存在，如果不对 NLP 进行漫长而专注的努力，结果很难令人满意。如果你扩大数据集，对不同的用户多运行几次脚本，你会发现大量的推荐都有相同的分值，并且这些预测分值的差别也不会显著。

总的来说，推荐系统可以基于用户间、条目间、内容间的相似度而建立起来。基于用户相似度建立的推荐系统是可靠的，但在面对大用户量的场景可能会是低效的。在这种情况下，基于条目相似度的协同过滤是一种更好的选择，因为用户（数百万甚至数千万）的数量级比条目的数量级要大。基于内容的方法用得并不多，但它确实有自身的优点，把它与协同过滤的方法组合在一起使用可以提高推荐的质量。通常，生产环境中的系统会把这些技术组合起来使用，下面我们看看组合推荐引擎的概念。

98

```
bsh % Delphi delphiUIC = new Delphi(
    ds,RecommendationType.USER_ITEM_CONTENT_BASED);
bsh % delphiUIC.setVerbose(true);
bsh % delphiUIC.recommend(nul);

Recommendations for user Bob:

    Item: biz-06.html       , predicted rating: 2.500000
    Item: biz-04.html       , predicted rating: 1.500000
    Item: usa-02.html       , predicted rating: 0.500000
    Item: world-03.html     , predicted rating: 0.500000
    Item: world-05.html     , predicted rating: 0.500000
```

图 3.8　基于与用户 Bob 相关的内容，我们得到条目的推荐结果

99

3.3　推荐朋友、文章与新闻报道

本节我们会提出一个更现实的例子，便于阐明怎么把目前为止所讨论过的技术组合起来工作。我们的对象是一个假想的网站，它的目的在于鉴别出具有相似观点的个人、有相似评论的文章以及相似内容的新闻报道，我们称这个网站为 MyDiggSpace.com。顾名思义，这个网站会使用 Digg API 来获取你通过自己的 Digg 账号提交的文章（注册之后可以得到 Digg 账号的信息）。这样它就可以识别出与你

Download at http://www.pin5i.com/

"dig"[3]的文章相似的文章并推送给你。此外，还可以对你读过的文章评分，以便系统将来根据你的反馈在推荐文章中做出更切合你心意的选择。如果这样还不够，那么这个网站还能给你推荐一些有着共同兴趣的组，如果你感兴趣，就可以加入它们，这样就促进了具有相似思想的个体之间的社交互动。

3.3.1 MyDiggSpace.com 简介

下面让我们看看建立这个网站的具体流程。我们会忠于自己在第 1 章里的承诺，不会把议题专注于类似于 UI 的设计、持久化以及其他重要的工程的组件。为了使整个过程更有趣，我们会使用 Digg API 来获取数据，也会使我们的例子尽量真实。首先，需要解释一下，Digg 是一个网站（http://digg.com/），用户在那里可以分享他们在任一个网站发现的内容。它的核心在于内容的收集并不是由一个很懂（或者不懂）你的编辑完成，而是由用户自发完成的。不管你想谈论的内容是来自高知名度的商业新闻门户网站，还是一个毫无名气的博客，Digg 都会让你发布自己的选择，然后通过参与用户的投票使得最好的内容能被挖掘出来。

Digg API 允许第三方通过程序与 Digg 进行交互，存储于 Digg 的大部分数据都可以通过 API 获取，你可以依据热门程度、时间或者类别（讨论的主题）来获取文章的列表。我们写了一组类封装了 Digg API 的使用，以后你也可以根据自己的需要来扩展它们。

我们会通过几个简单的步骤来构筑 MyDiggSpace.com 的数据集。首先，对于 Digg 中的每一个类别，我们都把排名最高的文章收集回来，这样可以同时得到一组用户以及每个用户的条目列表。

对于每个用户的每个条目，我们会基于条目内容的相似性找出 10 个由其他用户提交的条目。也就是说，我们会创建一个基于条目内容的推荐引擎，并找到最相似的 10 个条目。

为了数据集的完整性，我们会假定用户为这些条目都贡献了分值，所以我们需要为这些分值分配一个随机的数值。分值的分配遵循之前的例子中所使用的惯例：用户名字首字母为 A 到 D 之间的分配分值为 4 或 5，其他用户分配分值为 1、2 和 3。 100

本例的目的在于给读者介绍一个概念：组合不同的推荐引擎，以谋求得到比任

3 Digg 的一种类似于"顶"的功能。——译者注

Download at http://www.pin5i.com/

何一个单一的引擎都更好的效果。即使跳出推荐引擎这个小圈子，对于其他各种应用而言，这也是一个明智的做法。在本书后面的章节，我们还会讨论融合分类器的结果。这里的例子只是深入到一个广阔而有前景的领域的前奏，同时它也传递了我在本书中一直试图传达的一个更大的信息——各种智能技术的协同工作在为实际应用提供更高质量结果中的重要性。

3.3.2 发现朋友

下面运行代码清单 3.15 的脚本，然后实际操作这些假想的 MyDiggSpace. com 的数据[4]。

代码清单 3.15 MyDiggSpace.com: 一个组合推荐引擎的例子

```
BaseDataset ds = DiggData
➥ .loadDataFromDigg("C:/iWeb2/data/ch03/digg_stories.csv");←把来自 Digg 的数据保存下来
// BaseDataset ds = DiggData
[CA].loadData("C:/iWeb2/data/ch03/digg_stories.csv");←或者从本地载入数据
iweb2.ch3.collaborative.model.User user = ds.getUser(1);←选取一名用户
DiggDelphi delphi = new DiggDelphi(ds);←创建推荐器的实例
delphi.findSimilarUsers(user);←找到相似的用户
delphi.recommend(user);←推荐文章
```

实际的网站中，相似用户会被放在侧栏，推荐的文章也可以放在一个特定的模块里。为了提高对每一个用户的推荐质量，我们可以使用类似于第 2 章的基于点击的方法，甚至可以让用户对每个推荐进行评分，这样对用户的偏好有一个更高层面的置信度。我们会对这些改进进行简单的讨论，但首先看看在写本书时我们的脚本运行的结果。

我们收集了来自 7 个类别 33 个用户的 146 个条目（文章），你可以在类 iweb2. ch3.content.digg.DiggCategory 中控制条目及类别的数量。我们为这些用户分配了 811 个条目评分。对于每个用户，条目的选择与分值都是随机的，只是仍然会遵循我们之前所使用的惯例，依据用户名字首字母来分配分值的组别。在这个数据集中，用户打过的最小分值是 7，最大分值是 31，中值是 26。

4 声明：脚本中你能收集到的数据是公开的，我们并不对这些内容负责，我们的目的是提供一个通过 Digg API 获得的工作样例，然后可以通过它来做一些有用的事情。

Download at http://www.pin5i.com/

101

三角效应

图 3.9 显示了列表里的一号用户 adamfishercox 的相似用户集，接着是他最相似的用户 adrian67 的相似用户集，然后是 adrian67 的相似用户（虽然不是最相似）DetroitThang1 的相似用户集。从图 3.9 的数据中我们可以观察到一个有趣的结果，很难说这是否明显：用户 amipress 在 adamfishercox 的前 5 个相似用户之列，但不在 adrian67 的前 5 个相似用户之列。然而 amipress 又以 0.7 的相似度位居 DetroitThang1 的前 5 个相似用户之列，这个值与 amipress 和 adamfishercox 之间的相似度几乎相等。是不是很有趣？我们称为三角效应（triangulation effect），它告诉我们可以利用二阶效应来提高推荐的精度和效果。

```
bsh % delphi.findSimilarUsers(user);
Top Friends for user adamfishercox:

    name: adrian67        , similarity: 1.000000
    name: amipress        , similarity: 0.666667
    name: dvallone        , similarity: 0.500000
    name: cosmikdebris    , similarity: 0.500000
    name: cruelsommer     , similarity: 0.500000

bsh % iweb2.ch3.collaborative.model.User u2 =
ds.findUserByName("adrian67");

bsh % delphi.findSimilarUsers(u2);

Top Friends for user adrian67:

    name: adamfishercox   , similarity: 1.000000
    name: dvallone        , similarity: 1.000000
    name: ambermacbook    , similarity: 1.000000
    name: DetroitThang1   , similarity: 0.800000
    name: cruelsommer     , similarity: 0.750000

bsh % iweb2.ch3.collaborative.model.User u3 =
ds.findUserByName("DetroitThang1");

bsh % delphi.findSimilarUsers(u3);

Top Friends for user DetroitThang1:

    name: adrian67        , similarity: 0.800000
    name: cosmikdebris    , similarity: 0.750000
    name: amipress        , similarity: 0.700000
```

图 3.9 在一个 digg 数据集中发现相似用户及他们间的三角效应

Download at http://www.pin5i.com/

102

让我们利用图 3.9 的数据来深入探讨这种效应。用户 adamfishercox 在 adrian67 相似列表中排第 1，相似值为 1；用户 amipress 在 adamfishercox 相似列表中排第 2，相似值约为 0.67。用户 amipress 在 adrian67 相似列表中排第 7，相似值为 0.57。通过图 3.10 来勾画出这种关系，其中 User1 是 adamfishercox，User2 是 amipress，User3 是 adrian67。

圆括号中的数字代表位次，方括号中的数字代表相似值；箭头的根部指代要为该用户寻找相似用户。连接 User3 与 User2 的箭头是一根虚线，用于说明这是一个可以利用其他关系信息（实线的箭头）来提升的关系。

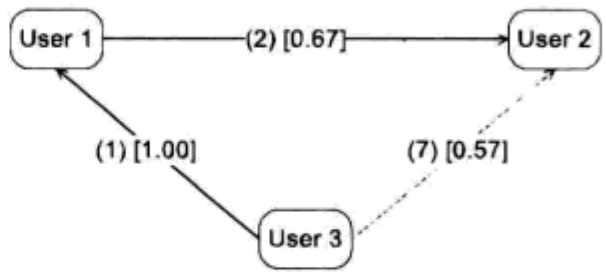


图 3.10 三角效应及提升相关位次的机会

3.3.3 DiggDelphi 的内部工作机制

现在我们看看生成的这些推荐的代码，清单 3.16 展示的是 DiggDelphi 的代码。

代码清单 3.16 MyDiggSpace.com 网站的组合推荐引擎系统

```
public class DiggDelphi {
  private Dataset ds;
  private Delphi delphiUC;
  private Delphi delphiUIC;
  private Delphi delphiUR;
  private Delphi delphiIR;
  private boolean verbose = true;
  public DiggDelphi(Dataset ds) {←初始化不同的推荐引擎
     this.ds = ds;
     delphiUC =
➥ new Delphi(ds,RecommendationType.USER_CONTENT_BASED);
     delphiUIC =
➥ new Delphi(ds,RecommendationType.USER_ITEM_CONTENT_BASED);
     delphiUR = new Delphi(ds,RecommendationType.USER_BASED);
     delphiIR = new Delphi(ds,RecommendationType.ITEM_BASED);
  }
public SimilarUser[] findSimilarUsers(User user, int topN) {❶
       List<SimilarUser> similarUsers =
➥ new ArrayList<SimilarUser>();
       similarUsers.addAll(
```

Download at http://www.pin5i.com/


```
➥ Arrays.asList(delphiUC.findSimilarUsers(user, topN)));
        similarUsers.addAll(
➥ Arrays.asList(delphiUR.findSimilarUsers(user, topN)));
        return SimilarUser.getTopNFriends(similarUsers, topN);
  }
  public List<PredictedItemRating> recommend(User user, int topN) {❷
     List<PredictedItemRating> recommendations =
➥ new ArrayList<PredictedItemRating>();
     recommendations.addAll(delphiUIC.recommend(user, topN));
     recommendations.addAll(delphiUR.recommend(user, topN));
     recommendations.addAll(delphiIR.recommend(user, topN));
     return PredictedItemRating
➥ .getTopNRecommendations(recommendations, topN);
  }
}
```

103

❶ 表示我们希望通过用户间相似度及用户的内容相似度来寻找相似用户；❷表示通过用户-条目-内容、用户、条目相似度来做出推荐。

如你所见，在方法 findSimilarUsers 中，我们用了简单的方法来组合相似用户列表——把所有的结果放在同一个列表里，并以相似值对结果进行排序（在 getTopNFriends 方法中实现）。这个方法中使用了基于内容的方法（通过 delphiUC 实例调用），以及基于用户间相似度的协同过滤方法（通过 delphiUR 实例调用）。需要注意的是，这两个推荐引擎得到的相似性结果并没有做过任何标准化的处理，这意味着尽管我们进行了排序，但结果不可避免是混乱的。

为了更深入地理解这一点，考虑一个由 20 个银行账户构成的列表，如果其中 10 个是美元账户，另外 10 个是欧元账户，那么对这些混合的账户以金额总量作为排序依据得到的结果并不具有太大的意义，除非我们把它们都表示为美元或是欧元。但无论如何，钱少的账户总是排在底部，而钱多的账户则会排在前面，只是排序不太精确而已。

我们用货币做这个类比虽然是为了阐明问题，但也未免过于简化这两种情况的主要差异。货币间的标准化很直观，也很好理解，如果我想把 100 美元兑换成欧元，那么我只需要知道这两种货币间的汇率，然后就可以得到这 100 美元转换成欧元时的面值。实际生活中，如果你想拿到这些欧元（或者转到你的银行账户），你还需要额外支付给银行一部分佣金，但你的标准化公式仍然是极其简单的。很不幸的是，

Download at http://www.pin5i.com/

用户相似度与推荐分值并不能同样简单地被标准化，融合不同推荐引擎的分值，与其说是一项科学任务，还不如说是一项艺术任务。对此，我们经常需要使用一些巧妙的先验知识，同时在初步的推荐结果之上构造一个信息处理层，而机器学习算法会在其中扮演重要的角色。

图 3.11 显示的是通过简单地组合三个不同的推荐引擎，对那三位我们一直在讨论的用户给出的推荐结果。如代码清单 3.16 中 recommend 方法所示，我们所得到的推荐列表是用户–条目内容推荐器、用户间协同过滤推荐器以及条目间协同过滤推荐器组合而成的结果。

104

```
bsh % delphi.recommend(user);

Recommendations for user adamfishercox:

Item: Lumeneo Smera: French Concept of Car and MotorCycle,
predicted rating: 5.0
Item: Bill Gates to Congress: Let us hire more foreigners -
CNET N, predicted rating: 5.0
Item: The Best Tools for Visualization, predicted rating: 5.0
Item: Coolest Cubicle Contest, Part Three, predicted rating: 5.0
Item: Bush: Telecoms Should Be Thanked For Their Patriotic
Service, predicted rating: 5.0

bsh % delphi.recommend(u2);

Recommendations for user adrian67:

Item: Can women parallel park on Mars?, predicted rating: 5.0
Item: Coast Guard loses a few flares and ..., predicted rating:
5.0
Item: 10.5.2 released, predicted rating: 5.0
Item: They are all hot!, predicted rating: 5.0
Item: 11 Greatest Basketball Commercials Ever Made, predicted
rating: 5.0

bsh % delphi.recommend(u3);

Recommendations for user DetroitThang1:

Item: The Best Tools for Visualization, predicted rating: 5.0
Item: Coolest Cubicle Contest, Part Three, predicted rating:
5.000000
Item: Stink Films comes correct with 3 Adidas Original Films,
predicted rating: 5.0
Item: The Power Rangers Meet The Teenage Mutant Ninja Turtles,
predicted rating: 5.0
```

图 3.11 组合三个不同的推荐引擎得到的一些结果显示

Download at http://www.pin5i.com/

从推荐的分值都是 5 分这点来看，这些推荐结果都很好，因为按照对这个数据集的有偏设计（名字首字母为 A 到 D 的用户总是给出 4 或 5 分），这正符合我们的预期。别忘了我们提醒过大家，像这样缺乏标准化的组合是有可能导致相似性的最终结果更倾向于某个推荐器的。所以，我们需要一种机制来保证来自不同推荐器产生的结果都是处于平等的地位进行融合。

代码清单 3.17 中 recommend 函数的实现就关注到了这一层因素。所以一开始就以几个推荐引擎得到的最大预测分值作为参考，把所有预测的分值标准化。同时我们引入了一个临时的阈值，用于过滤预测分值低于该阈值的推荐。这算是你与“糟糕推荐处理”这一有趣主题的第一次亲密接触。换句话说，我们这个阈值（虽然是人工的）设置了一道关卡，只有预测分值高于这个值的推荐才会被后续处理正式地考虑。 105

这个实现的最后部分会对某个条目的多个预测分值取平均，以便得到一个唯一的预测值。这样做的有效性是建立在我们已经对分值做了标准化的基础上的，如果没有之前的步骤，这样的平均没有任何意义。如果某个推荐引擎没有对该条目进行评分，其预测值设为 0，全部分值取平均之后，这个条目往往就已经被丢到推荐列表很靠后的地方。也就是说，这个方法综合了不同推荐器预测分值的平均影响与投票影响。一旦算出这个综合分值，推荐结果会被加入列表里，并按照这个新的预测值对列表进行排序。

代码清单 3.17 组合推荐器推荐的改良实现

```
public List<PredictedItemRating> recommend(User user, int topN) {
    List<PredictedItemRating> recommendations =
➥ new ArrayList<PredictedItemRating>();
  double maxR=-1.0d;
  double maxRatingDelphiUIC =
➥ delphiUIC.getMaxPredictedRating(user.getId());
  double maxRatingDelphiUR =
➥ delphiUR.getMaxPredictedRating(user.getId());      各推荐器计算其最大预测评分
  double maxRatingDelphiIR =
➥ delphiIR.getMaxPredictedRating(user.getId());
  double[] sortedMaxR =
➥ {maxRatingDelphiUIC, maxRatingDelphiUR, maxRatingDelphiIR};←各推荐器的最大预测评分
  Arrays.sort(sortedMaxR);
  maxR = sortedMaxR[2];←maxR 是最大的预测评分
```

Download at http://www.pin5i.com/


```
// 辅助变量
double scaledRating = 1.0d;
// 推荐器 1——用户-条目内容推荐器
double scaling = maxR/maxRatingDelphiUIC;←为每个推荐引擎计算比例因子
// 设置并缩放一个临时阈值
double scaledThreshold = 0.5 * scaling;
List<PredictedItemRating> uicList =
➥ new ArrayList<PredictedItemRating>(topN);
uicList = delphiUIC.recommend(user, topN);←从各个推荐引擎中获得推荐
for (PredictedItemRating pR : uicList) {
  scaledRating = pR.getRating(6) * scaling;
  if (scaledRating < scaledThreshold) {
    uicList.remove(pR);
  } else {
    pR.setRating(scaledRating);
  }
}
// 推荐器 2——基于用户的协同过滤
scaling = maxR/maxRatingDelphiUR;
scaledThreshold = 0.5 * scaling;
List<PredictedItemRating> urList =
➥ new ArrayList<PredictedItemRating>(topN);
urList = delphiUR.recommend(user, topN);
for (PredictedItemRating pR : urList) {
  scaledRating = pR.getRating(6) * scaling;
        if (scaledRating < scaledThreshold) {
            urList.remove(pR);
        } else {
            pR.setRating(scaledRating);
        }
  }
  // 推荐器 3——基于条目的协同过滤
  scaling = maxR/maxRatingDelphiIR;
  scaledThreshold = 0.5 * scaling;
  List<PredictedItemRating> irList =
➥ new ArrayList<PredictedItemRating>(topN);
  irList = delphiIR.recommend(user, topN);
  for (PredictedItemRating pR : irList) {
```

106

为每个推荐引擎计算比例因子

加权评分应大于阈值

从各个推荐引擎中获得推荐

Download at http://www.pin5i.com/


```
        scaledRating = pR.getRating(6) * scaling;
        if (scaledRating < scaledThreshold) {
            irList.remove(pR);
        } else {
            pR.setRating(scaledRating);
        }
}
double urRating=0;
double irRating=0;
double vote=0;
for (PredictedItemRating uic : uicList) {
    //初始化
    urRating=0; irRating=0; vote=0;
  for (PredictedItemRating ur : urList) {
    if (uic.getItemId() == ur.getItemId()) {
      urRating = ur.getRating(6);
    }
    }
    for (PredictedItemRating ir : irList) {
      if (uic.getItemId() == ir.getItemId()) {
        irRating = ir.getRating(6);
      }
    }
    vote = (uic.getRating(6)+urRating+irRating)/3.0d;
  recommendations.add(
  new PredictedItemRating(user.getId(), uic.getItemId(), vote));
  }
  rescale(recommendations,maxR);
  return PredictedItemRating
  .getTopNRecommendations(recommendations, topN);
}
```

加权评分应大于阈值

计算平均值并适当缩放

107

你还可以通过定位 MyDiggSpace.com 上每位独立用户的偏好来进一步改良你的推荐器，这可以通过组合类 DiggDelphi 与第 2 章中的 NaiveBayes 分类器来实现。欲知这个方法更多的细节，可以参看本章后面的 To Do 事项。你也可以通过引入多种学习机制（第 5 章会介绍一部分）及最优化技术来增强基础推荐器的结果，像这样把多种技术的组合封装到学习层里的方式，无论是在工业界的前沿应用还是在学术

Download at http://www.pin5i.com/

界中，都得到了支持并正在普及应用。

现在你应该对如下事项有了一定的认识：如何组合推荐系统，系统如何帮助你给网站上的用户推荐朋友与感兴趣的文章。下一节会关注另一个实例：在像 Netflix 那样的网站上做电影推荐，这个例子的主要特点在于其数据集的大规模性。

3.4 像 Netflix.com 那样推荐电影

在简介里，我们已经提及过 Netflix 公司——世界上最大的在线电影租赁服务提供商，该公司为超过 700 万的用户提供 9 万部 DVD，以及 5000 部可以随时在你的计算机上观看的完整的电影与电视节目，这些数字还在不断地增长。如果你还记得，Netflix 的线上成功部分基于它能帮助用户快捷地从浩如烟海的电影库中挑选到喜欢的电影。它这个能力的核心是一个名为 Cinematch 的推荐系统，其工作是根据某个用户喜欢或不喜欢其他的一些电影来预测他是否喜欢某部电影。

3.4.1 电影数据集的介绍及推荐器

在这一节里，我们会描述一个与 Cinematch 功能类似的推荐系统的实现。数据来源于公开可用的 MovieLens 项目的电影数据集，MovieLens 项目是明尼苏达州立大学 GroupLens 研究组提供的一个免费服务，这个计划包括一个提供电影推荐的网站。你可以通过http://www.movielens.org/quickpick来尝试一下它的服务，这个网站有两个 MovieLens 数据集。

108

第一个数据集[5]由 943 个用户对 1 682 部电影的 10 万个评分构成，第二个数据集[6]由 6 040 个用户对 3 900 部电影的 100 万个评分构成。第一个数据集随本书一起发行，请确认你读过它的许可与使用条款。两个数据集的格式稍有区别，我们发现第二个数据集（100 万个评分）更合适且更方便，它只包括三个文件：movies.dat、ratings.dat 与 users.dat。不过因为效率的缘故，我们还是选择了规模更小的第一个数据集。同时，为了方便起见，我们需要把那个小数据集（10 万个评分）转换成大数据集的格式。两个数据集都可以从 GroupLens 网站上下载，把数据解压到 C:/iWeb2/data/ch03/MovieLens/ 目录中，如果你放的不是这个目录，需要修改代码清单 3.18 中

5 原数据链接地址 http://www.grouplens.org/system/files/ml-data.tar__0.gz。
6 原数据链接地址 http://www.grouplens.org/system/files/million-ml-data.tar__0.gz。

Download at http://www.pin5i.com/

createDataset 方法的调用，把数据目录作为参数传递进去。

像 Netflix 和 Amazon.com 这样的大型推荐系统，都很依赖于基于条目的协同过滤（参考 Linden、Smith 和 York 的论文），这个方法已经在 3.2.1 节与 3.2.2 节中描述过，但在这里又对三个主要部件进行了改进。

第一个是数据标准化，这是一个可顾名思义的精巧术语。如果一个用户倾向于给所有的电影都打高分（在此前的章节里，我们构造对条目的人工评分时所使用的评分模式），那么考虑这名用户的相对评分比考虑他的绝对评分会更有意义。

第二个主要部件是邻居的选择。在协同过滤中，我们会找出一批条目（或用户）作为邻居，它们的评分会被用于推导未评分条目的分值。那么很自然的两个问题就是：需要几个邻居？怎样选择最好的邻居（即能提供最佳预测精度的邻居）？

协同过滤中的第三个主要部件是决定邻居的权重——每个邻居的评分有多重要？（Bell 与 Koren 曾论证过数据标准化与邻居权值选择是提高协同过滤算法精度的两个最重要的组件）[7]。让我们通过描述 Bean Shell 脚本来启动这个例子。代码清单 3.18 展示了怎么载入数据，并创建推荐器的实例（名为 MovieLensDelphi），选取用户，然后为每一个用户计算出推荐。

代码清单 3.18 MovieLensDelphi：用 MovieLens 数据集做推荐

```
MovieLensDataset ds = MovieLensData.createDataset();←载入 MovieLens 数据集
MovieLensDelphi delphi = new MovieLensDelphi(ds);←创建推荐器
iweb2.ch3.collaborative.model.User u1 = ds.getUser(1);←选取用户并获得推荐
delphi.recommend(u1);
iweb2.ch3.collaborative.model.User u155 = ds.getUser(155);
delphi.recommend(u155);
iweb2.ch3.collaborative.model.User u876 = ds.getUser(876);
delphi.recommend(u876);
```

109

第一个用户可以是任何人，所以我们取 ID 为 1 的那名用户，另外两个用户是通过执行命令 Delphi.findSimilarUsers(u1)来找到的，我们这样做的原因在于希望能够快速检查推荐结果是否是有意义的。因为如能出现如下情况，则是合理的：如果两个用户相似，并且两人都没看过某部电影，如果这部电影被推荐给其中一个人，那么另一个用户也很有可能会被推荐这部电影。图 3.12 显示了运行脚本之后的结果，这个

7 在本节，这体现在相似性的计算上。——译者注

Download at http://www.pin5i.com/

合情合理的假设也得到了证实。

```
bsh % iweb2.ch3.collaborative.model.User u1 = ds.getUser(1);
bsh % delphi.recommend(u1);

Recommendations for user 1:

Item: Yojimbo (1961)                , predicted rating: 5.000000
Item: Loves of Carmen, The (1948) , predicted rating: 4.303400
Item: Voyage to
the Beginning of the World (1997) , predicted rating: 4.303400
Item: Baby, The (1973)              , predicted rating: 4.303400
Item: Cat from Outer Space,
The (1978)                          , predicted rating: 4.123200

bsh % iweb2.ch3.collaborative.model.User u155 = ds.getUser(155);
bsh % delphi.recommend(u155);

Recommendations for user 155:

Item: Persuasion (1995)               , predicted rating: 5.000000
Item: Close Shave, A (1995)           , predicted rating: 4.373000
Item: Notorious (1946)                , predicted rating: 4.181900
Item: Shadow of a Doubt (1943)        , predicted rating: 4.101800
Item: Crimes and Misdemeanors (1989) , predicted rating: 4.061700

bsh % iweb2.ch3.collaborative.model.User u876 = ds.getUser(876);
bsh % delphi.recommend(u876);

Recommendations for user 876:

Item: Third Man, The (1949)       , predicted rating: 5.000000
Item: Bicycle Thief,
The (Ladri di biciclette)(1948)   , predicted rating: 4.841200
Item: Thin Blue Line, The (1988) , predicted rating: 4.685600
Item: Loves of Carmen, The (1948), predicted rating: 4.600200
Item: Heaven's Burning (1997)     , predicted rating: 4.600200
```

图 3.12　MovieLensDelphi 推荐器在 MovieLens 数据集上的推荐

这个数据集并不像 Amazon.com 或者 Netflix 的应用环境中的数据那么庞大，但它的规模显然已经比之前所有用做运算的数据都大，也已接近于现实的应用。在 MovieLens 的小数据集（10 万个评分数据）中运行脚本，仅创建推荐器也要耗费 30 秒到 1 分钟的时间。在这段时间里，你将会看到推荐器做了很多处理工作，而推荐本身会相当快，一般都小于 1 秒钟。

110

Download at http://www.pin5i.com/

3.4.2 数据标准化与相关系数

正如所承诺的，在本节的例子里，我们会通过引入两个新工具来丰富协同过滤算法，一个是数据标准化，另一个是条目相关性这一相似性度量。这个新的相似性度量方式被称为线性相关系数（也被称为积差相关系数，或者皮尔逊 r 系数）。计算两个数组 x 和 y 的 r 系数相当简单，代码清单 3.19 列出了负责计算这个统计量的三个函数。

代码清单 3.19 线性相关系数的计算（皮尔逊 r 系数）

```
public double calculate() {
  if( n == 0) {
     return 0.0;
  }
  double rho=0.0d;
  double avgX = getAverage(x);
  double avgY = getAverage(y);
  double sX = getStdDev(avgX,x);
  double sY = getStdDev(avgY,y);
  double xy=0;
  for (int i=0; i < n; i++) {
     xy += (x[i]-avgX)*(y[i]-avgY);❶
  }
   if( sX == ZERO || sY == ZERO) {❷
     double indX = ZERO;
     double indY = ZERO;
   for (int i=1; i < n; i++) {
     indX += (x[0]-x[i]);
     indY += (y[0]-y[i]);
   }
     if (indX == ZERO && indY == ZERO) {
     // 所有的点都取相同的值，这是相关性的退化形式
     return 1.0;
   } else {
     // x 或 y 的取值有差异
     if (sX == ZERO) {
        sX = sY;
        } else {
```

为每个向量计算平均值

为每个向量计算标准差

Download at http://www.pin5i.com/


```
            sY = sX;
            }
        }
    }
    rho = xy / ((double)n*(sX*sY));←皮尔逊 r 值
    return rho;
}
private double getAverage(double[] v) {
  double avg=0;
  for (double xi : v ) {
      avg += xi;
  }
  return (avg/(double)v.length);
}
private double getStdDev(double m, double[] v) {
  double sigma=0;
  for (double xi : v ) {
      sigma += (xi - m)*(xi - m);
  }
  return Math.sqrt(sigma / (double)v.length);
}
```

❶ 是计算两向量在各维度计算平均偏差后的点积。❷ 是一种特殊（奇异）情况，发生在 X 或 Y 或两者中的点都相同时，这种情况会导致 0 为分母，所以要单独处理。

getAverage 方法的功能不言自明，用于计算作为参数传入的向量的均值。getStdDev 方法计算作为第二个参数传入的数据向量的标准差，第一个参数必须是这个向量的均值。有一个更好的方法可以实现这个功能，并解决数值计算中的舍入误差问题，即修正双行程（corrected two-pass）的算法，该算法在 Chan、Golub 和 LeVeque 的书中有叙述。

基于皮尔逊相关系数的相似性是一个被广泛使用的度量，它有如下性质。

- 当相似度为 0 时，两个条目是（统计）不相关的。
- 当相似度为 1 时，两个条目的评分精确地拟合在一条正斜率的直线上，例如，（1，2），（3，4），（4，5），（4，5），圆括号中第一个数字表示第一个条目的评分，第二个数字表示第二个条目的评分，这称为完全正相关。换句话说，

Download at http://www.pin5i.com/

如果我们知道一个条目的评分，就能以极大的概率推测出另一个条目的评分。

- 当相似度为-1 时，两个条目的评分精确地拟合在一条负斜率的直线上，例如（1，5），（2，4），（3，3），（4，2），这称为完全负相关。在这种情况中，我们同样可以通过一个条目的评分推测出另一个条目的评分，不过现在是只要第一个条目评分上升，第二个条目评分就会下降。

如果条目间是线性相关的，线性相关系数会是度量这种相关性强度的一个绝好工具。事实上，如果用一条直线来拟合你的数据集，线性相关系数可以反映你的评分与这条直线拟合的程度。但很不幸的是，如果不存在什么关联，这实在是一种极其糟糕的相关性度量方式，开什么玩笑？真的，如假包换！ 112

一个著名的反例被称为安斯康比四重奏（Anscombe's quartet）。图 3.13 描绘了安斯康比四重奏的四个不同的数据对，这个图的 SVG 格式在维基百科中有（http://en.wikipedia.org/wiki/Image:Anscombe.svg）。

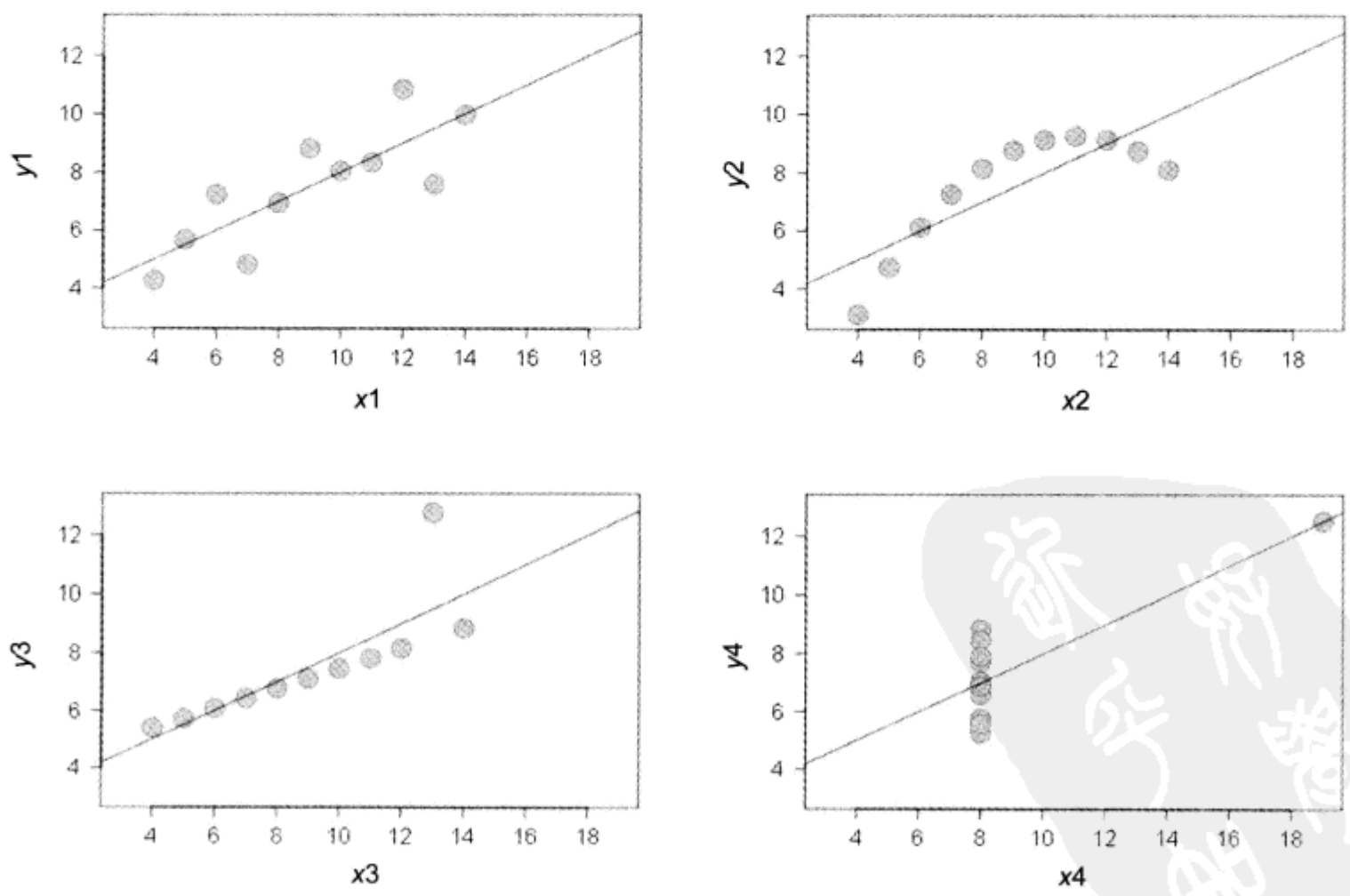


图 3.13 安斯康比四重奏：四个数据集具有相同的皮尔逊相关性，却是不同的分布

简单地说，如果你把两个条目对应位置的评分画在图中，而且它比较类似于图 3.13 中左上图的分布，线性相似系数就是一个有意义的度量。在其他的图形中，皮

Download at http://www.pin5i.com/

尔逊 r 系数有同样的取值，但它的意义是成疑的。这四个数据集是精心制作的，目的是使得它们具有相同的均值、方差与一样的线性拟合（y=3+0.5*x）。由于不能确定线性相关系数的意义，这促使人们去尝试一种不一样的相似性度量方式：非参数相关性（nonparametric correlation）。有两个著名的非参数相似系数：斯皮尔曼秩序相关系数（Spearman rank-order correlation coefficient）rs 与肯得尔 tau（τ）系数。这些度量方式以损耗部分信息为代价来保证检测到的相关性是真实存在于数据中的，就如度量值所显示出来的那样。我们会在 To Do 列表里再讨论非参数相关性，因为在很多情况下评分的分布就像图 3.13 右下角的图那样。然而，从现在开始，我们假定只要条目评分是相关的，它们就是线性相关的，这样就可以很安全地使用皮尔逊相关系数。你可以在本章参考资料一节里找到更多关于非参数相关的内容。

113

在讨论过采用线性系数（皮尔逊 r 系数）与非参数系数作为评估相似性的可能性之后，我们继续介绍一种数据标准化方法，代码清单 3.20 的代码做的正是这么一件事情，它们是类 PearsonCorrelation 的一个构造函数。第一个参数提供了到源数据集的引用，其他两个参数分别为待计算相关性的两个条目的引用。如你所见，被构造用于进行皮尔逊相关计算的数组并不是数据集中记录的每个用户的评分，而是一个新的数据序列，它是由原用户评分减去条目的平均分值而得到的。很明显，这不是唯一进行数据标准化的方法，Bell 与 Koren 介绍了更复杂的应用于 Netflix 竞赛数据集的数据标准化技术。

代码清单 3.20 对条目平均分值的数据标准化

```
public PearsonCorrelation(Dataset ds, Item iA, Item iB) {
  double aAvgR = iA.getAverageRating();
  double bAvgR = iB.getAverageRating();
  Integer[] uid = Item.getSharedUserIds(iA, iB);
  n = uid.length;
  x = new double[n];
  y = new double[n];
  User u;
  double urA=0;
  double urB=0;
  for (int i=0; i<n; i++) {
    u = ds.getUser(uid[i]);
    urA = (double) u.getItemRating(iA.getId()).getRating();
    urB = (double) u.getItemRating(iB.getId()).getRating();
```

Download at http://www.pin5i.com/


```
    x[i] = urA - aAvgR;
    y[i] = urB - bAvgR;
  }
}
```

数据标准化与皮尔逊相关都包含在 PersonCorrelation 类中，而它们的使用则封装在类 MovieLensItemSimilarity 中。因为做了数据标准化，MovieLensDelphi 类与其他的 Delphi 形式的类不太一样，代码清单 3.21 中的代码强调了这些差异。

代码清单 3.21 包含重标准化与重新标度的评分计算

114

```
private double estimateItemBasedRating(User user, Item item) {
  double itemRating = item.getAverageRating();
  int itemId = item.getId();
  int userId = user.getId();
  double itemAvgRating = item.getAverageRating();
  double weightedDeltaSum = 0.0;
  int sumN=0;
  // 检查用户是否已给该条目评过分
  Rating existingRatingByUser = user.getItemRating(item.getId());
  if (existingRatingByUser != null) {
    itemRating = existingRatingByUser.getRating();
  } else {
        double similarityBetweenItems = 0;
        double weightedDelta = 0;
        double delta = 0;
        for (Item anotherItem : dataSet.getItems()) {←遍历所有的条目
      // 只考虑该用户评过分的条目
       Rating anotherItemRating =
➥ anotherItem.getUserRating(userId);
       if (anotherItemRating != null) {
          delta = itemAvgRating - anotherItemRating.getRating();←数据重标准化
        similarityBetweenItems =
➥ itemSimilarityMatrix.getValue(itemId, anotherItem.getId());←计算两个条目的相似度
          if (Math.abs(similarityBetweenItems) >
➥ similarityThreshold) {❶
          weightedDelta = similarityBetweenItems * delta;
          weightedDeltaSum += weightedDelta;
          sumN++;
```

Download at http://www.pin5i.com/


```
            }
        }
        }
        if (sumN > 0) {
         itemRating = itemAvgRating -
➥ (weightedDeltaSum/(double) sumN)❷
        }
    }
    return itemRating;
}
public List<PredictedItemRating> getTopNRecommendations(
➥ List<PredictedItemRating> recommendations, int topN) {
  PredictedItemRating.sort(recommendations);
  double maxR = recommendations.get(0).getRating();
  double scaledR;
  List<PredictedItemRating> topRecommendations =
➥ new ArrayList<PredictedItemRating>();
  for(PredictedItemRating r : recommendations) {
  if( topRecommendations.size() >= topN ) {
   // 推荐量已足够
   break;
  }
  scaledR = r.getRating() * (5/maxR);
  r.setRating(scaledR);
  topRecommendations.add(r);
 }
 return topRecommendations;
}
```

115

❶ 是我们根据两个条目的相似度对分值和均值的偏差做了加权；❷ 是基于条目的均值与加权偏差的和计算得到评分。

数据重标准化源于我们的相似度是围绕着条目平均评分而建立起来的，所以为了计算条目的预测分值，我们需要把差异值（delta）重标准化为真实评分值。这种数据标准化的缺点在于预测评分的最大值有可能会落到可接受分值值域之外，所以还需要对预测评分做一个重新标度，如 getTopNRecommendations 方法中所显示的那样。

Download at http://www.pin5i.com/

3.5 大规模的实现与评估

商用推荐系统面临着苛刻的条件，用户数通常在数百万量级，条目也有几十万量级。还有一个额外的要求就是在不牺牲推荐质量的前提下具有进行实时推荐（通常要求一秒以内的响应时间）的能力。此外，像我们之前看到的，通过时间来积累更多用户的评分可以提高系统预测的精度。但在实际应用中，必须做到对于显然没有更多评分记录的新用户也能给出优秀的推荐结果。对于先进的推荐系统的另一个苛刻要求是具有根据新增评分进行增量式更新的能力。在大型的商业网站中，在短短几个小时之内就会产生数千条评分或购买记录，一天的记录量可能会数以万计。根据这些额外的信息对推荐系统不停机地进行在线更新的能力就显得尤为重要了。

假定你写了一个推荐器，并且你对它的速度以及它所能处理的数据量都颇为满意，那它就是一个好的推荐器吗？如果它只是一个速度极快、扩展性极好但只能产生很烂的推荐结果的推荐器，那它一无是处！既然如此，让我们接下来谈谈怎么评估一个推荐系统的精度。如果你搜索相关的文献，你会发现有大量的定量指标以及若干个定性的方法用于评估推荐系统的结果。那不计其数的指标反映了一个事实：要对推荐结果得到一个有意义的、公平的与精确的评估是一件多么具有挑战性的工作。Herlocker、Konstan、Terveen 与 Riedl 的综述文章包含了丰富的信息，如果你对这个主题感兴趣，可参阅这些文章。 116

我们已经写了一个类，通过计算预测评分的均方根误差（RMSE）来评估我们的推荐在 MovieLens 数据上的效果。RMSE 是一个简单却鲁棒的用于评估推荐精度的技术，这个指标有两个主要特点：（1）它反映的是误差总体的累加效应（仅优化个别预测的效果不大）。（2）通过对误差计算平方，大的误差值（>1）会被放大，而不管它是正的还是负的误差值。

我们可以争辩说 RMSE 很可能落后了。让我们考虑两种情况：第一种情况推荐给用户一部预测评分为 4 分的电影，而他实际并不喜欢（只打了 2 分）；第二种情况推荐给用户一部预测评分为 3 分的电影，事实上他喜欢这部电影（打了 5 分）。在这两种情况中，对于 RMSE 的贡献都是一样的，但很有可能用户对第一种情况的不爽程度会高于第二种情况，因为我们对于自己的不满总是特别敏感的。

你可以在 RMSEEstimator 类中找到 RMSE 的计算代码，代码清单 3.22 展示了怎

Download at http://www.pin5i.com/

么计算 MovieLensDelphi 推荐器的精度。

代码清单 3.22 计算一个推荐器的均方根误差

```
MovieLensDataset ds = MovieLensData.createDataset(100000);←❶创建数据集并保留
                                                          100 000 个评分记录作为测试
MovieLensDelphi delphi = new MovieLensDelphi(ds);
RMSEEstimator rmseEstimator = new RMSEEstimator();
rmseEstimator.calculateRMSE(delphi);
```

❶ 表示我们从拥有 100 万条记录的 MovieLens 大数据集中剔除 10 万条记录，得到一个数据集。推荐器会基于剩下的 90 万条数据做训练，并在那 10 万条数据中进行评估，其余脚本的功能都是不言而喻的。如果你没有改动过本节的代码，运行这些脚本得到的 RMSE 值会是 1.0256。这并不是一个糟糕的值，但也算不上有多好。我们强烈建议你改进这个结果，并以把 RMSE 做到小于 1 为你的目标。作为一个相对意义上的成功，我们必须要提及 Netflix 竞赛中最好的组能够把 RMSE 做到 0.86 与 0.88 之间[8]。所以，尽管数据集不同，但如果你的改进能把 RMSE 做到 0.9 左右的话，不要感到失望，这对于你和我们来说都是一个很了不起的成功。

117 3.6 总结

在本章中，你已经学习了用户间与条目间的距离与相似性的概念。我们也已经看到单一的尺度并不处处适用，所以在选择相似性度量时必须小心。本章我们接触了几个度量方式：雅可比度量、皮尔逊相关系数，以及这些度量的变体。相似性公式得到的结果必须与一些基本准则保持一致，除此之外，我们可以自由地选择能为我们的目的产生最好结果的那一种相似度。

我们还讨论了两大类推荐技术——协同过滤与基于内容的方法。我们通过构建一个在线音乐商店来详细而清晰地阐明其内部的原理。在建构这些例子的同时，我们还创建了一系列基础架构，以便你构建自己的推荐系统并用于实际的应用中。

最后，我们处理了两个更普通的例子。第一个例子是一个假想的网站，它使用 Digg API 并获取用户的内容用于深入分析用户间的相似性，最终把他们没看过的文章推荐给他们。在这个例子里，我们指出了二阶效应的存在，并拓展出高阶效应，

8 到竞赛结束时，优胜者的 RMSE 值已经做到低于 0.86。——译者注

Download at http://www.pin5i.com/

我们也建议利用这种效应来提高推荐的精度。第二个例子处理的是电影推荐，并介绍了数据标准化的概念，以及著名的线性（皮尔逊）相关系数。在后面的内容里，我们还介绍了一个基于均方根误差指标评估推荐精度的类。

在上述例子中，我们论述了随着问题复杂性与规模的增长，利用多种技术的组合对于提升推荐的效率与质量会变得非常必要。所以，我们在 MyDiggSpace.com 的例子中讨论了反复利用用户点击记录的可能性。这是一个贯穿全书的主题：能抓住问题不同方面的多种技术的组合，并且常常会让我们的推荐获得更高的准确性。

在下一章，我们会面对智能算法的另一个家族：聚类算法。但如果你还没做过 To Do 里的主题，或者你现在想先看看它们，正好在所有与推荐相关的信息还回响在你大脑里的时候，趁热打铁。

3.7 To Do

1．相似性度量：为 MusicUsers 实现雅可比相似度，你能看到结果有什么不一样？雅可比度量的一个变体是 Tanimoto 度量，这是一种更适合于连续值的方法。Tanimoto 度量等于两个集合的交（$Ni=|X\cap Y|$）除以两个集合并[9]（$Nu=|X|+|Y|$）与集合交之差，即 $T=Ni/(Nu-Ni)$。

例如，如果 X={baseball, basketball, volleyball, tennis, golf}，Y={baseball, basketball, cricket, running}，那么 Tanimoto 度量等于 2/((5+4)−2)，约为 0.2857。当它们以向量形式（Java 的数组 double[]x 与 double []y）表示时，尝试按这个公式算出来。提示：交集相当于两向量的内积，并集相当于两者数量之和。 118

另一个有趣的相似性度量是城区距离（city block），它的名字源于这样一个事实：向量 $\boldsymbol{X}$ 与 $\boldsymbol{Y}$ 的值可看做多维正交网格上的坐标值。当向量是二维时，这类似于一个的士司机在城市里给你指路的方式："帝国大厦从这里要往南走两个街区，往东走三个街区。"如果你喜欢这种度量方式或者想从它最常用的案例里进行学习，Eugene F. Krause 的 *Taxicab Geometry: An Adventure in Non-Euclidean Geometry* 一书有详细的剖析。

2．改变预测的范围：你有没有好奇过为什么那么多的网站都希望你给电影、歌曲或其他商品打 1～5（含）的正整数分值？为什么不是 1～10 分？或者甚至是 1～100 分？这难道不是能让你更灵活地表达自己对商品的满意程度吗？还可以从这一步走得更远，为什么不可能对一个商品的不同方面进行评分？对于电影的情况，我

9 实际的定义是两个集合长度之和。——译者注

Download at http://www.pin5i.com/

们可以对情节、演员的表演、原声音乐与视觉效果进行评分。你可以扩展本章的代码并试验这一切疑问。你还能找到什么潜在的议题么？

3．通过集成的方法改善推荐效果：通过组合不同的技术以提高推荐精度的方法已经越来越流行。有很多理论上的原因促使了对这种集成技术的追捧，如果你对这方面的主题感兴趣，可以阅读 Dietterich 的文章。抛开理论上的原因，还有很多实验性的证据可以说明集成方法能得到比独立使用某种技术更好的结果。在 Netflix 竞赛中独领风骚（到本书撰写为止）的 Bell 与 Koren，他们的观点是："我们发现没有完美的模型，而我们最好的结果是来自对具有互补作用的模型预测结果的组合。"

如果把我们在本章提供给你的这些推荐器和你自己发明的推荐器组合起来，把它们得到的结果与每个单独的推荐器相比，会是怎样的结果？如果组合的效果更好，这样的组合就给力了。否则，研究一下你所使用的推荐器，看看它们在多大程度上抓住了问题的不同方面。

4．最小化舍入误差：如你所知，一般数值在 Java 和大部分其他语言都是以有限的精度来存储的。整型（int）与长整型（long）数值是精确的，尽管它们的表示范围是有限的，这由与它们类型相关的二进制位数来决定。但涉及浮点算法（float 与 double）时，由于数值表示的非精确性，一大堆的问题会突然出现。最好的情况是你根本无须理会它，最坏的情况是你可以由始至终都使用 double 型。

119

然而，在智能应用中，由于对数值运算的重度使用，你需要清楚实数值的有限精度会对计算结果造成的潜在影响，特别是非常小值或非常大值因积累与相乘而产生的结果。让我们考虑在 PearsonCorrelation 类中计算标准差时提及的舍入误差。与 1.0 相加，并使其计算机表示不再是 1.0 的最小的浮点数，这就叫做机器精度（ε）。几乎所有浮点数间的运算都会产生与ε同量级的相对误差，这种误差就叫舍入误差。

阅读 Chan、Golub 与 LeVeque 关于修正双行程的算法，并依此实现标准差的计算。你还可以在经典之作 *Numerical Recipes: The Art of Scientific Computing* 中找到关于这个算法的一个简要说明。你能否在这样的算法所生成的结果里看到一些细微的差别？如果使用的数据集比本书所考虑的更大，你认为会发生什么？提醒一下，这个算法同样适用于作为评估推荐精度所使用的 RMSE 的计算。

5．非参数（秩序）相关：当你有理由去质疑皮尔逊相关度量中线性假设的可靠性时，这一类的相关性就特别有用。你可以依据这种类型的度量创建一个新的相似性的类，即使丢失部分数据信息，也要保证两个数据集（在我们的例子里，是两个

Download at http://www.pin5i.com/

评分集）的相关性是真实存在的。非参数相关的要点在于把变量的值用该值在数据集中的排列位置来取代。最著名的非参数相关系数是斯皮尔曼秩序相关系数（rs）与肯得尔 tau（τ）系数。你可以通过这本专著 *Numerical Recipes: The Art of Scientific Computing* 来获得这些系数的相关内容。

在电影的案例里，评分是从 1 到 5 的，在排序上会有很多同位冲突，比如会有大量电影的评分值都为 4。但这种情况也提供了一个让我们创造性地去使用这些系数的机会。比如是否可以使用评分的时间来打乱取值的顺序呢？试着实现这样的方法，并把它的结果与单纯的皮尔逊相关得到的结果进行比较。

3.8 参考资料

Bell, R.M., and Y. Koren. "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights." IEEE International Conference on Data Mining (ICDM'07), 2007. http://www.research.att.com/~yehuda/pubs/BellKorIcdm07.pdf.

Chan, T.F., G.H. Golub, and R.J. LeVeque. "Algorithms for computing the sample variance: Algorithms and recommendations." *American Statistician*, vol. 37, pp. 242-247, 1983. 120

Dietterich, T.G., "Ensemble methods in machine learning." *Multiple Classifier Systems*, (Editors: J. Kittler and F.Roli) volume 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy. Springer, pp.1-15, 2000. http://citeseer.ist.psu.edu/dietterich00ensemble.html.

Estes, W.K. *Classification and Cognition*. Oxford University Press, 1996.

Herlocker, J.L., J.A. Konstan, L.G. Terveen, and J.T. Riedl (2004). "Evaluating Collaborative Filtering Recommender Systems." *ACM Transactions on Information Systems*, Vol 22, 5-53. ACM Press, 2004. http://web.engr.oregonstate.edu/~herlock/papers/eval_tois.pdf.

James, W. *The Principles of Psychology*. Henry Holt and Company, 1918.

Krause, E.F. *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*. Dover Publications, Inc. 1986.

Linden, G., B. Smith, and J. York. "Amazon.com recommendations: Item-to-item collaborative filtering." *IEEE Internet Computing*, January-February 2003, pp.76-80.

Press, W.H., S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes: The Art of Scientific Computing* (3rd Edition). Cambridge University Press, 1997.

Download at http://www.pin5i.com/

聚类：事物的分组

4

本章内容提要：

- 聚类的需求和意义
- 发掘典型网站中的用户分组以及同类新闻、博客和文件的分组
- 基于链接的聚类算法和快速 k-means 算法

人类积累和获取信息的能力在很大程度上依赖于利用各种方法对所接受的各种信息进行结构化的能力，例如感官知觉、推理、语言和情感。如果缺少了某些参考结构，信息就会泛滥成灾。在大脑中有序地整理所获得的数据，能帮助我们把握住这些数据的本质，并理解我们身边的这个世界。

通常，我们将自己感知到的信息分成不同的种类。智能应用也会对数据做同样的处理，这是通过两大类算法实现的——聚类和分类。本章会介绍聚类算法，下一章则会介绍分类。

从广义上说，聚类这个词指的是分组时，将类似的事物放在一起。例如，现在有一个记录书籍信息的数据库。具体来说，每本书有一个 ID、一个 ISBN 号、一个指向作者表的外键（author_ID）及其他有关的字段。如果执行带 ORDER BY author_ID 子句的 SQL SELECT 查询，就能获得按作者 ID 排序的书籍列表。在这份列表中，

Download at http://www.pin5i.com/

首先是第一个作者的书，然后是第二个作者，等等。实际上，书是按照作者分组的。从聚类的意义上说，书的分组就被称为书的聚类，而我们所描述的这个过程就是一种直观的书籍聚类算法，尽管其有很大的局限性。

聚类有广泛的用途，但通常都不是简单的 SQL 查询就能达到目的的。在很多情况下都无法仅仅根据一个元素就识别出数据的分组，这就需要我们开发出能处理各类数据的技术。

在前面几章中我们看到，定义不同对象的距离和相似性是可能的。在本章中还会定义任意两个对象间的距离，只有当两个对象足够相似时，它们才属于同一类，同样，只有两者的差异足够大时才会分属于不同的类别。

我们首先会用一个例子说明为什么要使用聚类。因为聚类的对象千差万别，我们不可能一一涵盖，我们会根据聚类结构、聚类数据类型和数据大小对聚类算法的概况做一个介绍。本章其他部分会详细介绍几个具体的算法，还有一节会介绍一些有关聚类的高级话题，例如，计算复杂性和高维度数据的问题。

4.1 聚类的需求

本节所举的例子是在 Web 应用中识别用户的分组，这是一个常见的用例。该技术可以用来发布有针对性的广告、通过显示同类用户喜欢的内容来提升用户体验、在网站中构建社交网络等。识别用户分组时自然要用到各种聚类方法。

首先，假设我们完全没听说过聚类这个概念，如何才能合理地解决诸如用户分组这样的问题。换言之，在介绍聚类之前，我们先介绍一些能用于解决此类问题的简单方法。可以将聚类视为对有多个属性的记录进行排序，当然基于这些属性的排序指标也是多种多样的。

前面我们已经说明了基于 SQL 语句的直观方法只能用于少数情况，并解释了为什么在一般情况下基于普通的 SQL 查询的方法是有缺陷和不实际的。我们之所以依靠排序，是因为在各种排序指标的使用上具有一定的灵活性，但它依然无法有效地处理多个属性的情况。因此，我们需要的是通用的聚类技术。

4.1.1 网站中的用户组：案例研究

现在我们要引入一个简单的案例研究，本节将用这个案例介绍聚类。假设我们

Download at http://www.pin5i.com/

要开发一个类似 SourceForge.net 的大型开源社区软件平台，我们想知道为什么人们会参与开源项目。我们可以根据用户的资料进行聚类分析，找出用户的分组。为了方便讨论，假设我们只考虑以下属性：

- 用户的年龄，以年为单位。
- 用户的收入，分为若干个区间。例如，年收入 65 000～80 000 美元对应的收入范围为 0、80 000～95 000 美元对应的收入范围为 1，等等。可以在 data/ch04 目录下的 README 文件中找到各个范围的具体信息。
- 用户的教育水平：高中、大学、研究生等。
- 在多大程度上认为参与开源项目有助于自己专业技能的进步。分为 1～5，共五个等级。
- 在多大程度上认为参与开源项目有助于与有同样兴趣的人建立社交关系。还是分为 1～5 共五个等级。
- 是否是有偿参与，就是说，能否从参与开源项目获得报酬。我们可以用一个布尔变量来表示，或者也可以更详细地记录其有偿贡献所占的时间比例。

可以通过引入符合具体情况的属性，将这个例子扩展到任何涉及社交网络的 Web 应用。为了让例子更加具体，我们编造了表 4.1 中的数据，其中的表头是我们刚刚介绍的六个属性，总共 20 个用户，表中的各行代表各个用户的这六个属性值。

表 4.1　用于聚类分析的人造网络社区用户数据

用户名	年龄	收入范围	教育	技能	社交	有偿工作
Albert	23	0	0	3	3	0
Alexandra	25	1	2	4	2	0
Athena	24	0	1	3	4	0
Aurora	23	1	2	5	2	0
Babls	21	0	0	3	4	0
Bill	31	1	2	4	2	0
Bob	32	1	1	3	1	1
Carl	30	0	2	4	2	0
Catherine	31	1	1	3	3	0
Charlie	30	1	2	3	2	0
Constantine	37	1	1	3	2	0
Dmitry	35	2	2	1	1	1
Elena	38	1	1	3	2	0

124

Download at http://www.pin5i.com/

续表

用户名	年龄	收入范围	教育	技能	社交	有偿工作
Eric	37	2	2	2	2	0
Frank	39	3	1	3	1	1
George	42	2	2	2	1	1
Jack	43	3	1	1	1	1
John	45	4	2	1	1	1
Maria	43	2	1	3	1	0
Lukas	45	3	2	1	1	1

我们的目的很明确：尽可能地根据这些属性的值，对参与开源项目的用户进行分组。接下来的两节会按照先易后难，从低级到高级的顺序介绍两个能实现该目标的直观方法。

4.1.2 用 SQL order by 子句分组

实现上述目标的最简单的方法就是把我们的数据加载到数据库的表中（如果它还不在数据库中的话），然后写一个 SQL 查询找到感兴趣的用户组。这里我们把数据加载到 MySQL 中，当然也可以用其他任何你所选择的数据库，结果会是一样的。将数据加载到 MySQL 中的 SQL 命令在 data/ch04 目录下的 README.txt 文件中。

图 4.1 是查询 select * from sf_users order by IncomeRange, Education 的执行结果。从图中可以看到，普通的 SQL 查询对单个属性的效果非常好。我们可以根据 Income-Range 属性方便地得到五个分组，分别是（Albert, Babis, Athena, Bill, Carl），（Elena, Constantine, Catherine, Bob, Charlie, Aurora, Alexandra），（Maria, Dmitry, Eric, George）等。对于其他的属性，我们也能得到类似的结果。但要注意，在 order by 子句中加入更多的属性是无法改变分组结果的。第一个属性就决定了查询的结果，而其他的属性只能对 SQL 子句中第一个属性的查询结果做进一步细分。

假设我们能用肉眼识别出分组，我们需要回答一个问题：什么属性最适合用来识别有效分组？这不是一个简单的问题。如果有几千条记录该怎么办？如果要考虑十几个甚至更多的属性，而不仅仅是两三个属性，怎么办？在这些情况下，除非我们有关于数据的先验知识，否则就不可能完成这些任务，至少也是非常困难的。仔细思考一下，你就会清晰地感觉到 SQL 查询并不能带我们走多远。

SQL 方法的基本问题有两个：其一，难以自动发现聚类；第二，不能实现针对

Download at http://www.pin5i.com/

多个属性的聚类。对枚举类型的数据进行聚类是比较简单的，但如果是连续变量，事情就会变得更复杂；如果是未经整理的文本数据，就几乎是不可能的任务了。更重要的是，很难根据多个属性进行分组，因为结果在很大程度上取决于查询中属性的排列顺序。用普通的 SQL 方法进行聚类有非常大的局限性。

```
mysql> select * from sf_users order by IncomeRange, Education;
+-------------+-----+--------+-----+--------+--------+--------+
| Name        | Age | Income | Edu | Skills | Social | isPaid |
|             |     | Range  |     |        |        |        |
+-------------+-----+--------+-----+--------+--------+--------+
| Albert      |  23 |    0   |  0  |    3   |    3   |    0   |
| Babis       |  21 |    0   |  0  |    3   |    4   |    0   |
| Athena      |  24 |    0   |  1  |    3   |    4   |    0   |
| Carl        |  30 |    0   |  2  |    4   |    2   |    0   |
| Elena       |  38 |    1   |  1  |    3   |    2   |    0   |
| Constantine |  37 |    1   |  1  |    3   |    2   |    0   |
| Catherine   |  31 |    1   |  1  |    3   |    3   |    0   |
| Bob         |  32 |    1   |  1  |    3   |    1   |    1   |
| Bill        |  31 |    1   |  2  |    4   |    2   |    0   |
| Charlie     |  30 |    1   |  2  |    3   |    2   |    0   |
| Aurora      |  23 |    1   |  2  |    5   |    2   |    0   |
| Alexandra   |  25 |    1   |  2  |    4   |    2   |    0   |
| Maria       |  43 |    2   |  1  |    3   |    1   |    0   |
| Dmitry      |  35 |    2   |  2  |    1   |    1   |    1   |
| George      |  42 |    2   |  2  |    2   |    1   |    1   |
| Eric        |  37 |    2   |  2  |    2   |    2   |    0   |
| Frank       |  39 |    3   |  1  |    3   |    1   |    1   |
| Jack        |  43 |    3   |  1  |    1   |    1   |    1   |
| Lukas       |  45 |    3   |  2  |    1   |    1   |    1   |
| John        |  45 |    4   |  2  |    1   |    1   |    1   |
+-------------+-----+--------+-----+--------+--------+--------+
20 rows in set (0.03 sec)
```

图 4.1　用 SQL 查询识别分组

然而，将 SQL 与其他更先进的算法结合在一起可以实现真正的聚类，因为在大规模数据中，可以用 SQL 命令高效地完成很多操作。详见 To Do 一节中关于 SQLEM 算法的描述和参考资料。

4.1.3　用数组排序分组

你可能会想到一个办法来解决 SQL 方法中的问题：把数据加载到 Java 代码中，然后用自定义的比较操作得到原始的多维数据的有意义的排序。如果我们的数据是某种能自定义比较的数组，那就应该能够对其进行排序，并得到可能的聚类，对吗？

126

Download at http://www.pin5i.com/

让我们试试，看看结果如何。图 4.2 是从自定义排序的数组中得到的结果。

结果看上去不错，但如何定义群组间的边界依然是有待算法的使用者去解决。我们可以添加几行代码让每四个用户分成一组，但这可能并不是真实的分组。同属于一个年龄组的人会有一定的相似性，但如果就此认为这些结果反映了普遍情况，就未免有些武断了。图 4.2 中就是来自代码清单 4.1 中的两行代码运行的结果。第一句命令是加载表 4.1 中的数据。第二句命令是用 SortedArrayClustering 类识别数据中的群组。

代码清单 4.1 通过对 DataPoints 数组排序识别群组

```
SFDataset ds = SFData.createDataset();
SortedArrayClustering.cluster(ds.getData());
```

代码清单 4.2 中是 SortedArrayClustering 类的代码。实质上，这个类跟 SQL 方法没有本质区别，因为我们所做的只是对数据进行排序，然后显示到屏幕上。唯一的区别只是把数据排序的任务从 SQL order by 语句转移到自己定义的 Comparator 中。所以，有些基本的东西是这两种方法都没有捕获到的。

```
bsh % SortedArrayClustering.cluster(ds.getData());
John        ([45.0, 4.0, 2.0, 1.0, 1.0, 1.0])
Lukas       ([45.0, 3.0, 2.0, 1.0, 1.0, 1.0])
Maria       ([43.0, 2.0, 1.0, 3.0, 1.0, 0.0])
Jack        ([43.0, 3.0, 1.0, 1.0, 1.0, 1.0])
George      ([42.0, 2.0, 2.0, 2.0, 1.0, 1.0])
Frank       ([39.0, 3.0, 1.0, 3.0, 1.0, 1.0])
Elena       ([38.0, 1.0, 1.0, 3.0, 2.0, 0.0])
Eric        ([37.0, 2.0, 2.0, 2.0, 2.0, 0.0])
Constantine([37.0, 1.0, 1.0, 3.0, 2.0, 0.0])
Dmitry      ([35.0, 2.0, 2.0, 1.0, 1.0, 1.0])
Bob         ([32.0, 1.0, 1.0, 3.0, 1.0, 1.0])
Bill        ([31.0, 1.0, 2.0, 4.0, 2.0, 0.0])
Catherine   ([31.0, 1.0, 1.0, 3.0, 3.0, 0.0])
Carl        ([30.0, 0.0, 2.0, 4.0, 2.0, 0.0])
Charlie     ([30.0, 1.0, 2.0, 3.0, 2.0, 0.0])
Alexandra   ([25.0, 1.0, 2.0, 4.0, 2.0, 0.0])
Athena      ([24.0, 0.0, 1.0, 3.0, 4.0, 0.0])
Aurora      ([23.0, 1.0, 2.0, 5.0, 2.0, 0.0])
Albert      ([23.0, 0.0, 0.0, 3.0, 3.0, 0.0])
Babis       ([21.0, 0.0, 0.0, 3.0, 4.0, 0.0])
```

图 4.2 用自定义的 Comparator 类对数组元素排序，从而实现聚类

在处理一维数据时，排序是一种高效、合理的聚类技术。在 getR()方法中，数据的多维性被忽略了。该方法计算每个元素到所有元素的原点的距离。可以将此想 127

Download at http://www.pin5i.com/

象成从坐标原点（该点所有的属性值均等于 0）指向各个数据点的箭头。实际的值可以通过 EuclideanDistance 类得到，顾名思义，这个类实现了第 3 章中介绍的欧几里德距离。

代码清单 4.2 SortedArrayClustering：对数据点排序并打印

```
public class SortedArrayClustering {
  public static void cluster(DataPoint[] points) {
    Arrays.sort(points, new Comparator<DataPoint>() {←按降序排列
    public int compare(DataPoint p1, DataPoint p2) {
        int result = 0;
      // 根据分值排序
      if (p1.getR() < p2.getR()) {
        result = 1;
      } else if (p1.getR() > p2.getR()) {
        result = -1;
      } else {
        result = 0;
      }
      return result;
    }
    });
    for (int i=0; i < points.length; i++) {
      System.out.println(points[i].toString());
    }
  }
}
```

若各种属性消失了，那么所有的东西都变成一样的了，我们不得不面对两个严重的问题。其一，依然要判断有多少群组，并找出各组的成员。其二，由于数据没有规范化，在计算欧几里德距离时，用户的年龄对最终结果的影响远超其他属性。在这个数据中，对属性的值进行规范化就能削弱这种不利影响，但如果是其他类型的数据集，就没这么简单了。

我们要的是具有智能特征的聚类算法。人类是智能的，那么，如果人类看到这份数据集会怎么想呢？人类会识别出哪些群组？尤其是，我们把注意力集中到 40 岁以上的人群时。George、Jack、John、Maria 和 Lukas 都是 40 出头的人，他们的大部分属性都是一样的，或是类似的。但是，所有的人中间只有 Maria 能从开源项目

Download at http://www.pin5i.com/

的贡献中获得报酬，而且他参与开源项目的主要动力是提高自己的技术水平，从而增加收入。从这些方面看，Maria 就不应该与 George、Jack、John 和 Lukas 等人分在同一组。但如果根据排序结果分组，这是不可能实现的！

你可以通过调整我们之前用到的距离将 Maria 从这个分组排除出去。但如果这么做，无疑就破坏了该方法的简单性，而且修改后的算法很可能无法成功地应用到其他的数据集上。 128

对于给定的数据集，什么样的聚类结果才是正确的？这是一个亟待解决的难题，该领域的研究也非常活跃。这两种朴素方法——SQL 和数组排序，以及接下来要介绍的所有算法都有各自的优势和缺点。说到底，数据的特性决定了算法能否成功。

这个结论并不仅仅适用于聚类，在设计任何智能应用时都应该牢记这点。但因为机器无法直接衡量聚类的成功与否，所以聚类是最难的。聚类没有任何先验知识，如果有先验知识，就不需要从聚类算法开始了。正是因为这个原因，在机器学习中，聚类属于无监督学习（unsupervised learning）。

下一节是聚类的概述，以及聚类算法的各种分类依据，例如聚类的结构、所研究的数据结构和类型、要聚类的数据规模大小等。

4.2 聚类算法概述

SQL 方法和数组排序是我们介绍聚类的前奏，你可以将聚类视为广义上的一种排序。对，就是这样！实际上，在对一组对象排序时，我们首先是把所有的对象都列好，用某种比较方法从中选出第一个对象，然后是第二个，如此反复。该过程的最终结果就是找到排序结果中每个对象的直接邻居。

在聚类一组对象时，实际上也是要找到对象的直接邻居，但对象可以同时保持它们的多维性。它们可以是平面或三维空间中的点，或者根据要考虑的属性数量和所使用距离的计算方法，它们可以是更加广义的几何结构中的点。

聚类算法的目的就是识别分组，同时避免 SQL 方法或是简单的数组排序方法的缺点，而且能扩展到任何类型的多维对象空间。同一分组的成员相互之间（邻居）应该非常相似，同时不同于任何其他分组的成员。聚类在大量的问题中都有广泛的应用，包括生物、医学、金融、营销等。

Download at http://www.pin5i.com/

聚类算法形式多样，很难根据单一的标准对其进行分类。因此，为了避免缺乏对聚类的整体了解，在讲解具体的实现之前，我们先对聚类算法做一个概述。我们要介绍的聚类算法的第一种分类方法会基于分组的天然结构，即算法是寻找数据点之间的层次化结构还是简单地将空间分割成不同的区域？第二种分类方法是基于数据的类型和结构。有些聚类算法在数值型数据上表现很好，而有些算法则专注于类别型数据。第三种分类方法的依据是算法是否是为处理大规模数据而设计的。接下来，让我们从这些方面对聚类算法做一个概述。

129

4.2.1 基于分组结构的聚类算法分类

图 4.3 是根据聚类结果的结构对各种聚类算法的分类。

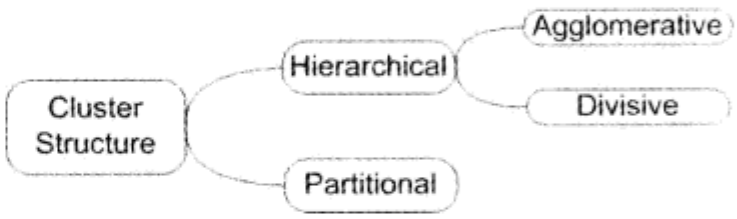


图 4.3 根据分组结构对聚类算法分类

层次聚类的结果就是在分组中识别分组。例如，一个针对新闻的层次聚类算法会先将新闻分成四个大类，例如政治、体育、商业和科技。然后将每个大类分成小类，例如，在体育新闻中，进一步分出篮球新闻、棒球新闻等。在本节的例子中，层次聚类算法可以将开源项目的参与者分成两个大组：有偿参与者和无偿参与者。然后可以根据年龄或收入进一步分解这两个大组。

大部分层次聚类算法都有一个阈值参数，用于控制算法应停止继续搜索更小分组的深度。这个参数不仅能保持数据分组的最终结构的合理性，还能在一定程度上避免不必要的计算消耗。当然，最终的分组数事先是不知道的，它取决于算法中决定分组层次终止条件的参数设置。

凝聚型层次聚类算法遵循的是自底向上的思路——首先从各个单独的元素开始，然后将各个元素关联在一起构成分组，逐步形成全局聚类（超级聚类）。而分裂型层次聚类算法则是从上至下进行的——从全局聚类（超级聚类）开始，将数据逐步划分成更小的聚类。

划分算法所创建的聚类数量是固定的。本章稍后将用到的 k-means 聚类算法就属于这一类。最小生成树（MST）和最近邻居算法也有划分版。这类算法有两种基本思路：概念建模和迭代优化。前者的代表是基于概率模型的算法；后者的代表则是 k-means 算法。

Download at http://www.pin5i.com/

130

4.2.2 基于数据类型和结构的聚类算法分类

图 4.4 是根据数据类型和数据结构的聚类算法分类。如果只处理数值数据，例如，地图上的地理坐标或股票价格的历史数据，基于网格的算法（grid-based algorithms）就比较合适。在这类算法中，基于谱和小波的方法有显著的优势。Gholamhosein Sheikholeslami 等人的 WaveCluster 算法能得到高质量的聚类结果，同时还具有最低的计算复杂性。

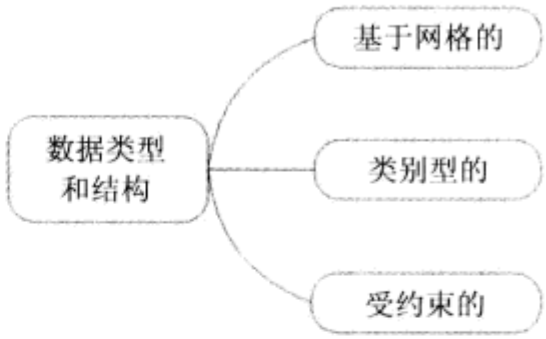


图 4.4 根据数据类型和数据结构对聚类算法分类

另一种聚类算法则专注于处理类别型数据。这些算法的主要特点是使用基于集合关系的度量，例如雅可比系数。通常，类别数据是无序的，很难找到合适的数值表示方法。如何用数值表示一串姓名？只能根据具体的情况寻找合适的方法，而不可能找到一个适用于所有情况的神奇算法。对于人的姓名，字典排序或许就足够了，但如果是企业团体的名称，字典排序就有可能把毫不相关的公司混在一起。因此，很多在数值数据上表现优异的算法在处理类别数据时往往一败涂地。

为了进一步说明这一点，让我们回顾一下 2.5 节中所使用的方法。当时我们根据一组能刻画新闻要点的单词对新闻进行排序，而没有使用它们之间的链接。那些数据是类别型，而不是数值型的。接着，我们用文件间的共有单词的数量来判断两个文件是否应该链接在一起。本章将会介绍的算法就类似于 2.5 节中所使用的技术，该算法在类别数据上有很好的表现。算法的名字叫 ROCK，是一种用雅可比相似性度量定义新闻间相似度的凝聚层次算法。

在聚类要满足特定条件时会用到受约束的聚类算法。典型的情况是在有障碍物的二维平面上对数据点进行聚类。显然，聚类应该绕过障碍物。在这类情况下，常用的欧几里德距离就行不通了，需要其他更有意义的度量来衡量两点间的距离。计算能绕过障碍物的最短路径无疑是一个选择，Tung 等人提出了能圆满地处理该问题的算法。但本书不会介绍受约束的聚类算法。

4.2.3 根据数据规模的聚类算法分类

131

图 4.5 中是针对大数据集设计的聚类算法的分类。对这类聚类算法要特殊对待，很多聚类算法的空间和时间复杂性跟要聚类的数据大小都是平方阶的关系。稍不注意，就可能出现内存不足，或是聚类分析永远都无法结束的情况。

Download at http://www.pin5i.com/

正因为如此，Paul S. Bradley、Usama M. Fayyad 和 Cory A. Reina 提出的框架要求网络应用中处理大型数据库的聚类算法应具有以下特性：

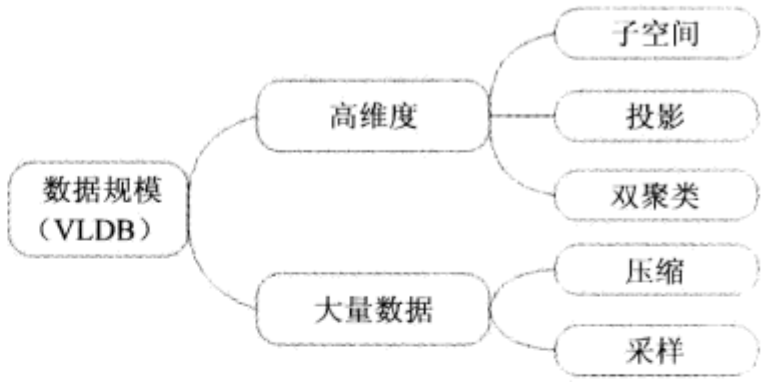


图 4.5 根据数据规模对聚类算法分类

- 如果可能，只需要扫描一次数据库。
- 支持实时行为——随时都能给出一个好结果。
- 算法的活动应能被暂停、终止和恢复。
- 如果有必要，要满足 RAM 限制。
- 在可能的情况下，应该利用各种扫描模式，例如序列化的扫描、基于索引的扫描，以及采样。
- 应该选择能处理数据库视图上的前置指针（forward-only cursor）的算法，因为这些视图通常是计算代价很高的连接操作的结果。

不同的算法有不同的特性，现在的趋势是将众所周知的基本算法与启发式、压缩、采样等技术融合在一起，从而实现效率和性能的平衡。

可以想象，有些算法可以满足多个标准，一个算法也有多个分类。例如，BIRCH（balanced iterative reducing and clustering using hierarchies）算法既是一个适用于超大型数据库（VLDB）的聚类算法，又是一个层次聚类算法。

这个概述有点冗长，但这说明“将类似的事物分组”这样一个看似简单直观的问题其实是很有深度的，吸引了很多人来研究。有各种各样的聚类算法可供选择，而我们的选择也取决于很多因素，例如数据的特点、所需结果的类型以及计算上的各种限制。接下来的几节会介绍本节中提及的大部分聚类算法，并会详细介绍如何聚类超大规模数据集。下面做好准备，开工！

132 4.3 基于链接的算法

本节会继续使用 4.1 节中的数据，看看从这些虚构的开源项目参与人员中能找出什么样的用户分组。首先会介绍在聚类中很有用的树状图（dendrogram）数据结构，本章所有的代码都会用到该数据结构。接着会介绍基于链接的算法的核心思想，

Download at http://www.pin5i.com/

并详细介绍三种基于链接的算法：单链接（single link）算法、平均链接（average link）算法和最小生成树（minimum spanning tree）算法。

4.3.1 树状图：基本的聚类数据结构

我们将把聚类中用的基本数据结构封装在 Dendrogram 类中。树状图的结构如图 4.6 所示，这是一种树状数据结构[1]，它可以帮助我们获得聚类的层次信息。我们可以将其视为一个有序三元组的集合：[d, k, {…}]，其中第一个元素是相似度阈值（d），第二个元素是聚类的数量（k），第三个元素是聚类的集合。

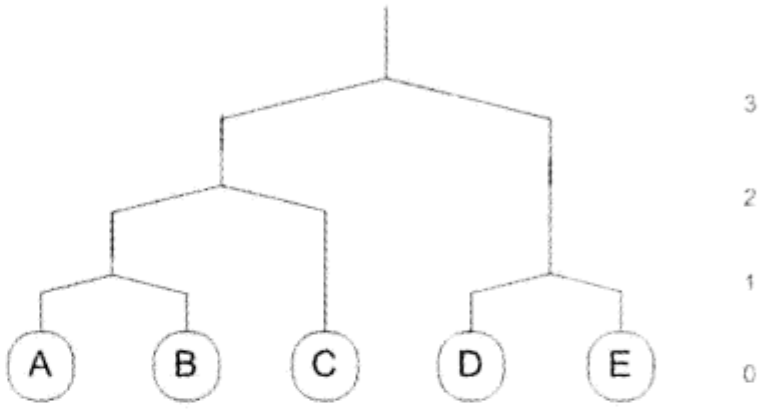


图 4.6 可视化层次聚类：一个简单的树状图

图 4.6 中展示的是一个四层的树状结构，该结构可以用下面的有序集合表示：{[0,5,{{A},{B},{C},{D},{E}}], [1, 3, {{A,B},{C},{D,E}}], [2, 2, {{A,B,C}, {D,E}}], [3,1,{A,B,C,D,E}]}。因此，树状图等价于聚类的集合，而不是一个聚类。这样我们就可以描述一组聚类是如何从单个元素逐步构造出来的。所有的层次凝聚算法都有以下步骤：

1. 定义初始的树状图，所有的元素都独自构成一个单元素聚类。
2. 将距离阈值提高一个等级，然后判断哪些元素会构成新的聚类。
3. 用所有的新聚类给树状图添加一层。
4. 继续执行第 2 步和第 3 步，直到所有的元素都属于一个大的聚类。

从实现的角度看，可以用两个链接哈希图表示树状图，见代码清单 4.3。该代码清单省略了两个辅助性的信息输出方法。

代码清单 4.3 Dendrogram：用于封装层次聚类的基础类

133

```
public class Dendrogram {
  private Map<Integer, ClusterSet> entryMap;
  private Map<Integer, String> levelLabels;
  private Integer nextLevel;
  private String levelLabelName;
```

1 Dendro 在希腊语中是“树”的意思。

Download at http://www.pin5i.com/


```
public Dendrogram(String levelLabelName) {
  entryMap = new LinkedHashMap<Integer, ClusterSet>();
  levelLabels = new LinkedHashMap<Integer, String>();
  nextLevel = 1;
  this.levelLabelName = levelLabelName;
}
public int addLevel(String label, Cluster cluster) {
  List<Cluster> values = new ArrayList<Cluster>();
  values.add(cluster);
  return addLevel(label, values);
}
public int addLevel(String label, Collection<Cluster> clusters) {
  ClusterSet clusterSet = new ClusterSet();
  for(Cluster c : clusters) {
    // 在添加之前复制聚类——随着时间的推移，聚类中的元素可能会发生变化
    // 但对于树状图，我们希望能保持当前的状态
    clusterSet.add(c.copy());
    }
    int level = nextLevel;
     entryMap.put(level, clusterSet);
     levelLabels.put(level, label);
     nextLevel++;
     return level;
    }
public void setLevel(int level, String label,
➥ Collection<Cluster> clusters) {
  ClusterSet clusterSet = new ClusterSet();
  for(Cluster c : clusters) {
    clusterSet.add(c.copy());
  }
  System.out.println("Setting cluster level: "+level);
  entryMap.put(level, clusterSet);
  levelLabels.put(level, label);
  if( level >= nextLevel ) {
    nextLevel = level + 1;
  }
}
```

总之，无论聚类结果是否是层次化的，树状图数据结构都能将其展示出来。这

Download at http://www.pin5i.com/

134

种数据结构非常适合用于表示聚类结果。下面继续看看基于链接的算法。

4.3.2 基于链接的算法概况

代码清单 4.4 中的脚本首先加载前面介绍的类似于 SourceForge 的数据，然后调用有关算法。

代码清单 4.4 层次凝聚聚类算法

```
SFDataset ds = SFData.createDataset();
DataPoint[] dps = ds.getData();                          加载数据
double[][] adjMatrix = ds.getAdjacencyMatrix();
SingleLinkAlgorithm sla = new SingleLinkAlgorithm(dps,adjMatrix);   单链接聚类
Dendrogram dendroSLA = sla.cluster();
dendroSLA.print(4);
MSTSingleLinkAlgorithm sla2 =
➥ new MSTSingleLinkAlgorithm(dps,adjMatrix);   带 MST 的单链接聚类
Dendrogram dendroSLA2 = sla2.cluster();
dendroSLA2.print(4);
AverageLinkAlgorithm ala = new AverageLinkAlgorithm(dps,adjMatrix);   平均链接聚类
Dendrogram dendroALA = ala.cluster();
dendroALA.print(4);
```

SFDataset 类表示 4.1 节中的数据。按出现的先后顺序，SingleLinkAlgorithm、MSTSingleLinkAlgorithm 和 AverageLinkAlgorithm 三个类分别表示相应的算法。为了消除数据表示和缺乏规范化的影响，所有的算法都使用相同的数据——表 4.1 中的原始数据（Datapoint[] dps 数组）和保存有每个用户跟数据集中其他用户的相似性的邻接矩阵（double[][] adjMatrix）。

所有基于链接的算法都会用三元组[0, N, {{$X1$}, {$X2$}, …, {XN}}]初始化它们的树状图。当相似度阈值（三元组中的第一个元素）设为 0 时，对所有的元素而言，唯一能满足该阈值的就是其自身。因此，所有的元素都独自构成一个聚类。

前面提到，所有的算法都用 double 类型的数组表示邻接矩阵。矩阵中存放着集合中每两个元素间的距离，这有些像第 3 章中存放用户和商品的相似性矩阵。邻接矩阵的用途是根据相似性阈值判断哪两个元素应该合并成新的聚类，哪些元素应该继续独自构成一个聚类。在阈值逐步变大的过程中，这些比较要反复进行。在经过有限步之后，所有的元素都会合并成一个聚类，算法到此终止。

135

Download at http://www.pin5i.com/

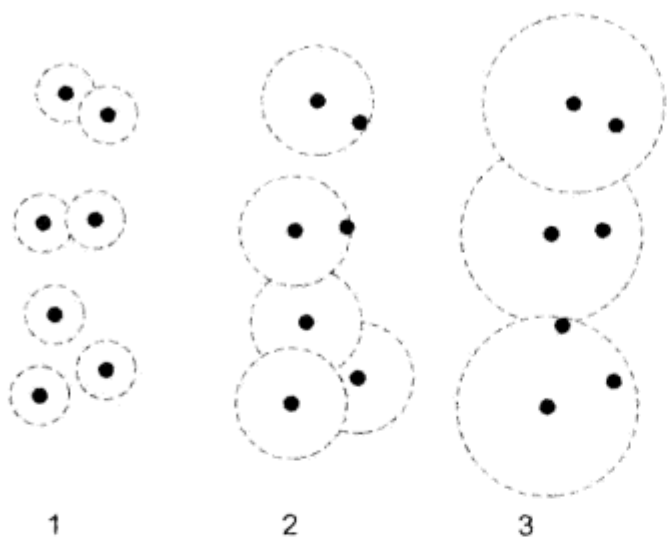


图 4.7 单链接算法的运行（三次迭代）

图 4.7 演示了三次迭代的过程。为了方便描述，我们仅限于二维平面。对于更高的维度，基本原则是完全一样的，只是高维度空间很难可视化。图中黑色圆点是要聚类的数据点，虚线圆圈是每个数据点在每次迭代中的相似度阈值。

在第一次迭代中，所有的数据点都单独成为一个聚类，因此，有 7 个聚类。在第二次迭代中，在图的上方产生了两个聚类，由此有 5 个聚类。在第三次迭代中，图下方的三个数据点合并成一个聚类，我们总共还有 3 个聚类。随着一次次成功的迭代，圆圈变得越来越大，直到中间的数据点的相似度圆圈的半径大到能包含整个数据集。到这时，迭代停止。

凝聚算法之间的差别主要有两个方面：

- 每次迭代中合并聚类的方法。
- 邻接矩阵的定义。

单链接、平均链接和最小生成树算法是三种著名的基于图论概念的凝聚层次聚类算法。接下来的三个小节会逐一介绍这三个算法。

4.3.3 单链接算法

单链接算法（如图 4.5 所示）的目的是尽可能多地找出图中的连通分量。如果两个聚类之间有一条边相连，该算法就会将两者合并，因此，被称为单链接。换句话说，如果两点间的最小距离小于或等于相似性阈值，从图中看就是数据点在虚线圆圈内，那么两个聚类就应该合并。具体的步骤在代码清单 4.5 的 cluster 方法和 buildClusters 方法中。

代码清单 4.5 只要聚类间有一条链接，就将两者合并

```
public Dendrogram cluster() {
  Dendrogram dnd = new Dendrogram("Distance");
  double d = 0;
  List<Cluster> initialClusters = new ArrayList<Cluster>();
  for(DataPoint e : elements) {        ❶
```

Download at http://www.pin5i.com/

136

```
  Cluster c = new Cluster(e);
  initialClusters.add(c);
}
dnd.addLevel(String.valueOf(d), initialClusters);
d = 1.0;
int k = initialClusters.size();
while( k > 1 ) {      ❷
  int oldK = k;
  List<Cluster> clusters = buildClusters(d);
  k = clusters.size();
  if( oldK != k ) {
    dnd.addLevel(String.valueOf(d), clusters);
  }
  d = d + 1;
}
return dnd;
}
private List<Cluster> buildClusters(double distanceThreshold) {
  boolean[] usedElementFlags = new boolean[elements.length];
  List<Cluster> clusters = new ArrayList<Cluster>();
  for(int i = 0, n = a.length; i < n; i++) {
    List<DataPoint> clusterPoints = new ArrayList<DataPoint>();
    for(int j = i, k = a.length; j < k; j++) {  ❸
      if( a[i][j] <= distanceThreshold && usedElementFlags[j] == false ) {
        clusterPoints.add(elements[j]);
        usedElementFlags[j] = true;
      }
    }
    if( clusterPoints.size() > 0 ) {
      Cluster c = new Cluster(clusterPoints);
      clusters.add(c);
    }
  }
  return clusters;
}
```

首先把每个数据点都加载到各自的聚类中❶。然后开始迭代，直到只剩下一个包含所有数据点的聚类❷。在每次迭代中，聚类都是在 buildClusters 方法中发生的。

Download at http://www.pin5i.com/

每次迭代时，距离的阈值都增加一个单位。

注意，尽管我们利用了邻接矩阵的对称性（第二次循环不是从 0 开始的，而是从 i 开始的）❸，但算法所需操作次数的增长速度依然是所要聚类的元素的数量的平方阶。我们说该算法的时间和空间复杂度都是 $O(N^2)$。对于小规模的数据集，这没什么影响，但真正要处理真实的数据集时，这就会变得至关重要。我们会在 4.4 节中探讨有关实际应用的话题。图 4.8 是我们用单链接算法从表 4.1 的数据集中得到的结果。这里是第 4 层的聚类结果。

137

{Dmitry} {Frank} {Alexandra, Aurora}
{John, Lukas} {George, Jack, Maria}
{Albert, Athena, Babis}
{Constantine, Elena, Eric}
{Bill, Bob, Carl, Catherine, Charlie}

图 4.8 根据单链接算法得到的第 4 层聚类结果

4.3.4 平均链接算法

代码清单 4.6 是平均链接算法的代码，与单链接算法很类似，但合并聚类的条件有所不同。平均链接算法会检查分属两个目标聚类的两个数据点之间的平均距离是否小于相似度阈值。在这个算法中要注意，相似度阈值每次只增加半个单位（0.5），而不是一个单位。这只是一个随意的选择，你也可以选择其他值，看看对结果有什么影响。

代码清单 4.6 根据平均距离合并聚类

```
public Dendrogram cluster() {
  Dendrogram dnd = new Dendrogram("Distance");←初始化
  double d = 0.0;
  for(DataPoint e : elements) {
    Cluster c = new Cluster(e);
    allClusters.add(c);
  }
  dnd.addLevel(String.valueOf(d), allClusters.getAllClusters());
  d = 1.0;
  while( allClusters.size() > 1 ) {←构建层次结构的顶层循环
    int K = allClusters.size();
    mergeClusters(d);
    // 对于当前的 d，有可能没有聚类能合并
```

Download at http://www.pin5i.com/


```
    if( K > allClusters.size() ) {
      dnd.addLevel(String.valueOf(d),
➥ allClusters.getAllClusters());
      K = allClusters.size();
    }
    d = d + 0.5;
  }
  return dnd;
}
private void mergeClusters(double distanceThreshold) {
  int nClusters = allClusters.size();
  ObjectToIndexMapping<Cluster> idxMapping =
➥ new ObjectToIndexMapping<Cluster>();
  double[][] clusterDistances = new double[nClusters][nClusters];
  for(int i = 0, n = a.length; i < n; i++) {←给所有聚类中的所有链接加上距离
    for(int j = i + 1, k = a.length; j < k; j++) {
      double d = a[i][j];
      if( d > 0 ) {
        DataPoint e1 = elements[i];
        DataPoint e2 = elements[j];
        Cluster c1 = allClusters.findClusterByElement(e1);
        Cluster c2 = allClusters.findClusterByElement(e2);
        if( !c1.equals(c2) ) {
          int ci = idxMapping.getIndex(c1);
          int cj = idxMapping.getIndex(c2);
          clusterDistances[ci][cj] += d;
          clusterDistances[cj][ci] += d;
        }
      }
    }
  }
  boolean[] merged = new boolean[clusterDistances.length];
  for(int i = 0, n = clusterDistances.length; i < n; i++) {
    for(int j = i+1, k = clusterDistances.length; j < k; j++) {
      Cluster ci = idxMapping.getObject(i);
      Cluster cj = idxMapping.getObject(j);
      int ni = ci.size();
      int nj = cj.size();
```

138

Download at http://www.pin5i.com/


```
        clusterDistances[i][j] =
➥ clusterDistances[i][j] / (ni * nj);←聚类间的平均距离
        clusterDistances[j][i] = clusterDistances[i][j];
        // 当距离小于阈值时，合并聚类
        if( merged[i] == false && merged[j] == false ) {
          if( clusterDistances[i][j] <= distanceThreshold) {
             allClusters.remove(ci);
             allClusters.remove(cj);
             Cluster mergedCluster = new Cluster(ci, cj);
             allClusters.add(mergedCluster);
             merged[i] = true;
             merged[j] = true;
          }
        }
      }
    }
  }
```

跟前面一样，首先将每个元素都设为单独的聚类来初始化树状图，然后产生新的聚类，直到所有的元素都合并成一个聚类。与单链接算法不同的是，我们需要找出两个聚类间的所有链接。平均链接算法需要更多的计算。代码清单中 mergeClusters 方法的第一个循环是将数据集中每两个元素间的距离加到两者分别所属的聚类的总距离中。同一方法中的第二个循环则是将总距离除以链接数，然后将得到的平均距离与阈值进行比较。如果平均距离低于阈值，就将两个聚类合并。跟单链接算法一样，平均链接算法在完成树状图中一层所有的合并得到新的一层后，就接着用同样的方法处理下一层。

139

图 4.9 是平均链接算法在表 4.1 的数据中得到的结果。跟图 4.8 一样，这也是第四层的聚类结果。注意，现在聚类变少了。你认为是什么原因？为什么结果会有如此大的差异？尽管层数是一样的，但每个算法计算距离的方法是不同的。这样，同一层的相似度阈值是不一样的。换句话说，经过四次迭代后，单链接算法的相似度圆圈（见图 4.5）要比平均链接算法的相似度圆圈大。所以，平均链接算法的聚类结果自然就要少一

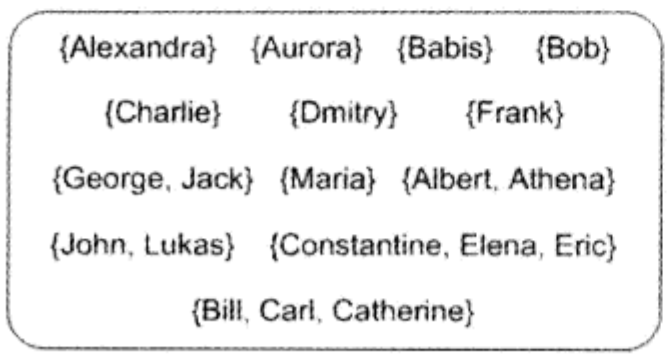


图 4.9 平均链接算法的聚类结果

Download at http://www.pin5i.com/

些。由此可以看出，在比较这类算法时，应该控制相似度圆圈的大小，而不是迭代的次数。

4.3.5 最小生成树算法

为了帮助你理解第三种凝聚算法，我们首先介绍一下最小生成树（MST）的概念。通常，给定一组元素，只要将任意两点仅用一条边（链接）连在一起就构成了一棵树。生成树则要连接集合中所有的顶点。显然，构造生成树的方法不是唯一的，但 MST 不仅要求连接所有的顶点，同时还要求所连接的顶点间的边的权重总和最小。对于给定的邻接矩阵，我们利用 Prim-Jarník 算法找出最小生成树，操作次数是 $O(N^2)$。还有其他接近线性效率的算法，操作次数是 $O(N \log(N))$。如果想了解更多有关于最小生成树的图论算法的知识，可以参考本章的 To Do 和参考资料。

顾名思义，MST 单链接算法是单链接算法基于最小生成树的变体。最小生成树来自邻接矩阵，自然而然地产生了集合中元素的一个排序。如果邻接矩阵是一个 5×5 的数组，相应的 MST 也可以表示成一个 5×5 的二维矩阵，而且肯定是对称的。我们将这两个矩阵的对角线元素都设为-1，因为我们对自链接不感兴趣。这是最小生成树算法跟前面两个算法唯一的不同之处。该算法利用了 MST 中的信息，根据树中元素的增长顺序合并聚类。 140

看一下代码清单 4.4 中脚本的运行结果。图 4.8 是原来的单链接算法运行在迭代四次后得到的聚类结果。图 4.10 中是 MST 单链接算法在表 4.1 的数据中得到的结果。与图 4.8 的单链接算法和图 4.9 中平均链接算法的结果一样，这也是迭代四次后的结果。

MST 单链接算法所产生的聚类也比单链接算法产生的聚类少一些，原因跟平均链接算法一样：在第四次迭代时的相似度圆圈要少于单链接算法中的圆圈。如果逐步增加迭代次数，各自独立的聚类就会逐步合并成大的聚类。跟前面的算法一样，当所有的元素都合并成一个聚类时，算法就结束。让我们看看代码，代码清单 4.7 中是 MST 类，用来根据邻接矩阵构建最小生成树。

{Alexandra} {Aurora} {Babis} {Bob}
{Dmitry} {Frank} {George} {Jack}
{Maria} {Albert, Athena} {John, Lukas}
{Constantine, Elena, Eric}
{Bill, Carl, Charlie, Catherine}

图 4.10 MST 链接算法的聚类结果

Download at http://www.pin5i.com/

代码清单 4.7 根据邻接矩阵构建最小生成树

```
public class MST {
  public double[][] buildMST(double[][] adjM) {
    boolean[] allV = new boolean[a.length];      ←初始化存放 MST 节点的向量
    allV[0] = true;
    double[][] mst = new double[adjM.length][adjM.length];   ←初始化 MST 矩阵
    for(int i = 0, n = mst.length; i < n; i++) {
      for(int j = 0; j < n; j++) {
        mst[i][j] = -1;
      }
    }
    Edge e = null;
    while( (e = findMinimumEdge(allV, adjM)) != null ) {←迭代，直到找到最小值
      allV[e.getJ()] = true;
      mst[e.getI()][e.getJ()] = e.getW();
      mst[e.getJ()][e.getI()] = e.getW();
    }
    return mst;
  }
  private Edge findMinimumEdge(boolean[] mstV, double[][] a) {
    Edge e = null;
    double minW = Double.POSITIVE_INFINITY;
    int minI = -1;
    int minJ = -1;
    for( int i = 0, n = a.length; i < n; i++ ) {
      if( mstV[i] == true ) {
        for(int j = 0, k = a.length; j < k; j++) {
          if( mstV[j] == false ) {
            if( minW > a[i][j]) {
              minW = a[i][j];
              minI = i;
              minJ = j;
            }
          }
        }
      }
    }
    if( minI > -1 ) {
```

141

Download at http://www.pin5i.com/


```
      e = new Edge(minI, minJ, minW);
    }
    return e;
  }
}
```

为了节省篇幅，我们省略了一个内部类 Edge，这是一个用于封装图的边及其权重的基础类，细节请参看完整的源代码。你可以看出，这是一个简单的寻找最小生成树的算法，被称为 Prim-Jarník 算法。该算法可以总结成以下步骤：

1. 初始化用于表示各个元素是否属于 MST 的向量（allV）。
2. 用默认的负数（例如-1）初始化 MST 矩阵（变量 mst）。
3. 任选一个节点开始，找到从该节点出发的最小边。
4. 将权重最小的边的另一个节点添加到 MST 节点中。
5. 重复第 3 步和第 4 步，直到包括所有的点，树必须涵盖整个图。

换句话说，Prim 算法从任意一个点开始构造生成树，逐一将 MST 部分和非 MST 部分之间权重最小的边添加进来，当所有的节点都加入 MST 后算法结束。每次执行 Prim 算法得到的结果可能会有些许差异。在某些情况下，无论从哪个节点开始，最终都得到一样的 MST。什么情况下才会如此？当然，这并不是唯一一种寻找最小生成树的算法，还有两个有名的算法——Kruskal 算法和 Boruvka 算法。

MST 链接算法的时间复杂度是 $O(N^2)$，因为寻找 MST 需要二阶的复杂度。看一下 findMinimumEdge 方法的源代码，注意其中大小为 N 的二重循环就明白了。这是整个算法中最主要的操作。利用哈希表，以及像我们那样保存每个节点权重最小的边可以改进这部分的效率。 142

最后，我们不得不提一下使单链接算法臭名昭著的连锁效应（chain effect），即两个聚类中如果有两个点之间的距离非常近，那么无论其他的点之间的差别有多大，两个聚类都会被合并。单链接算法对此无能为力，但下面将要讨论的其他算法就没有这个缺陷。

4.4 k-means 算法

前面介绍的三种基于链接的算法都是层次化凝聚聚类算法。k-means 算法是我们介绍的第一种分割算法，其出色的性能使其成为实际应用中最广的聚类算法。

Download at http://www.pin5i.com/

4.4.1 初识 k-means 算法

首先运行 k-means 算法得到聚类结果。在代码清单 4.8 中，首先加载表 4.1 中的数据，然后运行 k-means 实现。前面的单链接算法得到了 8 个聚类，为了将 k-means 算法与之进行比较，我们选择 k=8。

代码清单 4.8 尝试 k-means 算法

```
SFDataset ds = SFData.createDataset();
DataPoint[] dps = ds.getData();                    ←加载数据
KMeansAlgorithm kMeans = new KMeansAlgorithm(8, dps);   ←初始化 k-means 算法
kMeans.cluster();    ←开始聚类
kMeans.print();
```

143

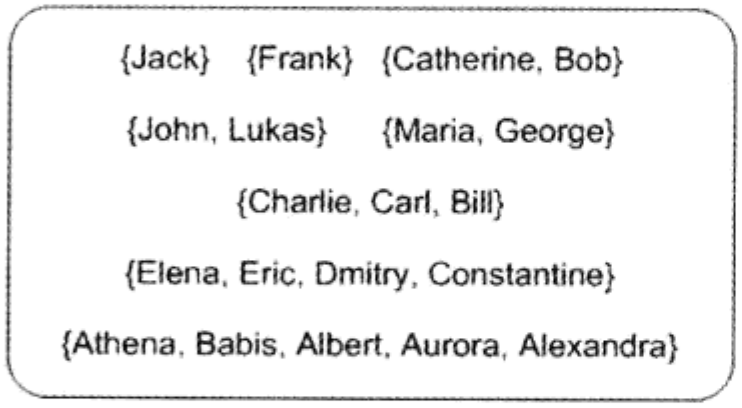


图 4.11 k-means 算法的聚类结果（k=8）

图 4.11 中是根据 k-means 算法得到的候选聚类，将这些聚类跟其他算法的结果（尤其是图 4.8 中的 8 个聚类）进行比较。k-means 算法思想的核心是聚类的中心点（centroid），也叫做中心或平均值。将组成聚类的元素想象成一块有质量的物体，聚类的中心点就应等同于这块物体的质量中心。

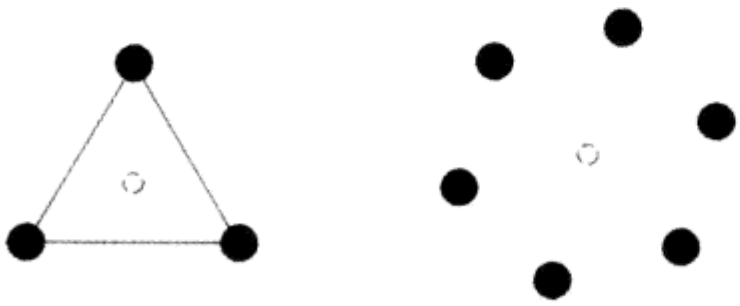


图 4.12 三角形和六边形的数据点（黑点）聚类的中心点（虚线圆圈）

图 4.12 显示了一个数据点（黑色圆点）分布呈六边形的聚类的中心点。这个聚类的中心点就在六边形的中心（因为对称性），即虚线圆点。中心点本身并不一定是要聚类的数据点。实际上，如图 4.12 所示，大部分情况下，中心点都不是数据点。中心点的意义只是一个能代表聚类中所有数据点的点。

运行代码清单 4.8 中的脚本得到的候选聚类可能会与图 4.11 中的结果有些差异，差异的原因是因为中心点的初始位置不同，下一节会对此详细介绍。

Download at http://www.pin5i.com/

4.4.2 k-means 的内部原理

为了更好地理解 k-means 算法的内部原理，让我们看一下它的实现，见代码清单 4.9。

代码清单 4.9 KMeansAlgorithm：k-means 算法的核心方法

```
public void cluster() {
  boolean centroidsChanged = true;
  while (centroidsChanged == true) {
    List<Set<DataPoint>> clusters =
➥ new ArrayList<Set<DataPoint>>(k);
    for (int i = 0; i < k; i++) {
      clusters.add(new HashSet<DataPoint>());←创建每个聚类的数据点集合
    }
    for (DataPoint p : allDataPoints) {
      int i = findClosestCentroid(allCentroids, p);←根据聚类分配数据点
      clusters.get(i).add(p);
    }
    for (int i = 0; i < k; i++) {
      allClusters[i] = new Cluster(clusters.get(i));←创建聚类
    }
    centroidsChanged = false;
    for (int i = 0; i < allClusters.length; i++) {
    if (clusters.get(i).size() > 0) {←计算新的中心点
      double[] newCentroidValues = findCentroid(allClusters[i]);
      double[] oldCentroidValues =
➥ allCentroids[i].getNumericAttrValues();
      if (!Arrays.equals(oldCentroidValues, newCentroidValues)) {
        allCentroids[i] =
➥ new DataPoint(allCentroids[i].getLabel(), newCentroidValues);
        centroidsChanged = true;
      }
    } else {
      // 若聚类没有元素，保持中心点不变
      }
    }
  }
}
public static DataPoint[] pickInitialCentroids(int k,
```

144

Download at http://www.pin5i.com/


```
» DataPoint[] data) {
  Random randGen = new Random();
  DataPoint[] centroids = new DataPoint[k];
  Set<Integer> previouslyUsedIds = new HashSet<Integer>();
for (int i = 0; i < k; i++) {
   // 选出还未用到的数据点的索引
   int centroidId;
   do {
    centroidId = randGen.nextInt(data.length);
   }
   while( previouslyUsedIds.add(centroidId) == false );
   String label = "Mean-"+i+"("+data[centroidId].getLabel()+")";
   double[] values = data[centroidId].getNumericAttrValues();
   String[] attrNames = data[centroidId].getAttributeNames();
   centroids[i] = new DataPoint(label,
» Attributes.createAttributes(attrNames, values));
  }
  return centroids;
}
```

k-means 算法随机选择 k 个点（见 pickInitialMeanValues 方法）表示候选聚类的初始中心点。接下来就会计算这些中心点和各个数据点之间的距离，然后将各个数据点添加到距离最近的中心点的聚类。这样做的结果是，聚类的中心点发生了改变，所以需要重新计算中心点，直到其位置不再发生变化。k-means 这个特别的算法要归功于 E.W. Forgy 和 S.P. Lloyd，它具有以下优势：

145

- 兼容各种度量。
- 可以方便地实现算法的并行——将数据分割成 N 份，每份数据都可以并行地在 N 个计算单元上各自进行聚类。
- 对数据的顺序不敏感。

你可能会好奇，如果算法不结束会发生什么情况？别担心！算法保证会在有限次的迭代后结束。在实际使用中，算法收敛速度很快（这是数学术语）。当然，如果使用的是该算法的变种，要时刻小心。假如度量不是欧几里德距离，就有可能出问题。Inderjit S. Dhillon、James Fan 和 Yuqiang Guan 在对海量的文件进行聚类时发现余弦相似性的表现比欧几里德距离差，详细情况可以参考他们的论文。在另一些情况下，他们发现用 Kullback-Leibler 发散度（这甚至不是距离！）代替平方欧几里德

Download at http://www.pin5i.com/

距离能取得更好的效果。

与其他的聚类算法相比，k-means 算法速度很快，其计算复杂度通常是 $O(N)$，其中 N 是要聚类的数据点数量。k-means 算法在商业软件 SAS 中的名字是 FASTCLUS（意思是快速聚类），这个名字显然是当之无愧的。

要注意的是，跟凝聚算法不一样，k-means 要求必须在输入中指定聚类的个数，这样问题就出来了：k 的值应该是多少？答案取决于数据（又来了）：用不同的 k 值运行 k-means 算法，然后检查聚类结果。有时候，因为要处理超大规模数据，或是需要使用层次聚类，就可以先用 k 值较小的 k-means 算法将数据集分成若干个大的分类，然后在各个大类中使用层次聚类。这个方法显然是可以并行的，可以充分利用手头的计算资源。

还要注意，k-means 算法适用于属性是数值的数据点。用 k-means 算法处理分类数据（例如字符串）要面对的挑战是如何找到非数值属性值的数值表达方式。在这里，度量的选择非常重要。

你应当知道，初始中心点的选择对迭代的快速停止和聚类结果的质量都是至关重要的。从数学的角度看，k-means 算法试图最小化同一聚类中数据点之间距离的平方的均值。因此，如果选择的初始中心点周围的数据点密度很高，那么就有可能更快地得到高质量的聚类结果。这就是 David Arthur 和 Sergei Vassilvitskii 最近在一篇论文中描述的 k-means++算法。

总之，前面几节介绍的算法可以识别出网站用户的分组。当然，发挥一下创造性，你可以将同样的算法应用到不同的环境中，还可以将不同的算法组合在一起，通常也推荐这么做。业界常用的算法是 k-means 及其变体，k-means 算法的简单、快速和可并行使其备受欢迎。 146

4.5 鲁棒的链接型聚类（ROCK）

本节将会介绍与之前两种聚类算法都不相同的一种聚类算法。首先，这个算法非常适用于类别型数据，例如关键字、布尔属性、枚举值等。其次，该算法能很好地处理大型数据集。我们的例子数据是从 Digg.com 中收集来的。

为了演示算法，我们使用了保存在 data/ch4 目录下的 ch4_digg_ stories.csv 文件中固定的数据集，这些数据是利用第 3 章中介绍的 Digg API 收集的。数据中含有 49

Download at http://www.pin5i.com/

篇 Digg 中来自 10 个随机用户的新闻，每篇新闻都有若干个属性。人们可以轻易地将数据中的这 49 篇新闻分成 8 组。你可以用你喜爱的文本编辑器打开并查看这份数据。有可能用算法而不是我们的眼睛来找到其中的聚类吗？下面我们来看看。

4.5.1 ROCK 简介

代码清单 4.10 中的代码加载了数据，初始化 ROCKAlgorithm，并且用你已经熟悉的 Dendrogram 类表现聚类的结构。

代码清单 4.10 用 ROCK 对大量的网络新闻进行聚类

```
MyDiggSpaceDataset ds = MyDiggSpaceData.createDataset(15); ←加载 Digg 新闻，只是用
                                                            前 15 条
DataPoint[] dps = ds.getData();
ROCKAlgorithm rock = new ROCKAlgorithm(dps, 5, 0.2);←初始化 ROCK，寻找 5 个聚类
Dendrogram dnd = rock.cluster();
dnd.print(21);
```

在 Dendrogram 类的 print 方法中，我们只输出含有两个或两个以上元素的聚类。换句话说，为了保证聚类结果的显示效果，我们不会显示单个的元素，它们在业界也被称为单例（singleton）。图 4.13 是代码清单 4.10 中代码的执行结果。

要注意，被算法用来判断新闻相似性的不仅仅是标题，还包括描述。从语法的观点来看，这些描述是千差万别的（例如关于献血和 Facebook 的新闻），所以直接对其内容做字符串比较是行不通的。其中的关键就是雅可比系数不依赖于文字中单词的语法，它比较的是描述相同的词汇数量。从图中可以看出，在第 21 层时 8 个聚类中已有 6 个被正确地识别出来。你可以用自己的数据，例如文件、新闻、文章等，看看会得到什么类型的聚类。只要能将数据放在 DataPoint 类的数组中，就可以实现聚类。

147

4.5.2 为什么 ROCK 这么强大？

让我们仔细看看 ROCK 算法的原理。代码清单 2.11 中是 ROCKAlgorithm 类的构造函数和核心方法 cluster。ROCK 的核心思想就是用链接作为相似性的度量，而不是仅仅依赖于距离。当然，为了判断哪些点之间有链接，我们依然要用到我们熟悉的独立度量，目标则是将有大量共同链接的点聚在一起。

Download at http://www.pin5i.com/


```
bsh % dnd.print(21);
Clusters for: level=21, Goodness=1.044451296741812

{5619782:Lack Of Democracy on Digg and Wikipedia?,
 5611165:Lack Of Democracy on Digg and Wikipedia?}

{5571841:Lack Of Democracy on Digg and Wikipedia?,
 5543643:Lack Of Democracy on Digg and Wikipedia?}

{5142233:The Confederacy's Special Agent,
 5620839:The Confederacy's Special Agent,
 5586183:The Confederacy's Special Agent,
 5610584:The Confederacy's Special Agent,
 5598008:The Confederacy's Special Agent,
 5613383:The Confederacy's Special Agent,
 5613380:The Confederacy's Special Agent}

{5585930:Microsoft, The Jekyll And Hyde Of Companies,
 5524441:Microsoft, The Jekyll And Hyde Of Companies,
 5609070:Microsoft, The Jekyll And Hyde Of Companies,
 5618201:Microsoft, The Jekyll And Hyde Of Companies,
 5620878:Microsoft, The Jekyll And Hyde Of Companies,
 5609797:Microsoft, The Jekyll And Hyde Of Companies}

{5607788:Recycle or go to Hell, warns Vatican -- part I,
 5592940:Recycle or go to Hell, warns Vatican -- part II,
 5618262:Recycle or go to Hell, warns Vatican -- part III,
 5595841:Recycle or go to Hell, warns Vatican --- part IV}

{5608052:Contract Free on AT&T,
 5620493:Contract Free on AT&T,
 5621623:Contract Free on AT&T,
 4955184:Contract Free on AT&T,
 5594161:Contract Free on AT&T}
```

图 4.13 代码清单 4.10 中代码的聚类结果

代码清单 4.11 ROCKAlgorithm：ROCK 的 cluster 方法

```
public ROCKAlgorithm(DataPoint[] points, int k, double th) {
  this.points = points; ←要聚类的数据点
  this.k = k;  ←聚类的最小数量
  this.th = th; ←创建链接的阈值
  this.similarityMeasure = new JaccardCoefficient(); ←相似度矩阵
  this.linkMatrix =
  new LinkMatrix(points, similarityMeasure, th);  ←链接矩阵
}
public Dendrogram cluster() {
```

148

Download at http://www.pin5i.com/


```
    List<Cluster> initialClusters = new ArrayList<Cluster>();
    for(int i = 0, n = points.length; i < n; i++) {❶
        Cluster cluster = new Cluster(points[i]);
        initialClusters.add(cluster);
    }
    double g = Double.POSITIVE_INFINITY;
    Dendrogram dnd = new Dendrogram("Goodness");
    dnd.addLevel(String.valueOf(g), initialClusters);
    MergeGoodnessMeasure goodnessMeasure =
➥ new MergeGoodnessMeasure(th);❷
    ROCKClusters allClusters = new ROCKClusters(initialClusters,
➥ linkMatrix, goodnessMeasure);❸
    int nClusters = allClusters.size();
  while( nClusters > k ) {❹
        int nClustersBeforeMerge = nClusters;
        g = allClusters.mergeBestCandidates();
        nClusters = allClusters.size();
        if( nClusters == nClustersBeforeMerge ) {
          // 没有相互链接的聚类要合并
          break;
        }
        dnd.addLevel(String.valueOf(g),
➥ allClusters.getAllClusters());
    }
    return dnd;
  }
```

构造函数的参数如下：

- 待聚类的数据点。
- 我们想要的最少聚类数量。ROCK 是自底向上的层次凝聚算法——最初，每个点都是单独的聚类，然后进行合并，直到所有的点都合并成一个聚类。这个参数控制我们想要得到的聚类数量的下限，让算法可以在所有的点合并成一个聚类之前就停止。
- 确定两点间是否有链接的相似度阈值。

149

在构造函数中，我们创建了一个雅可比相似度的实例（JaccardCoefficient），以及一个用于封装数据点之间链接的类（LinkMatrix）的实例。你也可以使用其他的距离度量，例如 CosineSimilarity，看看得到的结果是好是坏，或者是否相同。你能解

Download at http://www.pin5i.com/

释相似度及其导致结果的差异吗？通过试验，对于不同的距离度量，阈值会不一样，但最终结果都会很相似。这就是为什么我们说这个算法是“鲁棒”的。

当然，这个类不可能独自完成 ROCK 算法。它需要将整个任务分解，委托给其他几个我们马上要介绍的类。下面是代码清单 4.11 中 cluster 方法涉及的步骤：

❶ 这是初始化步骤，为每个数据点创建一个新的聚类。

❷ 这一步创建一个“goodness 度量”，其用途是判断两个聚类是否应该合并。在每个聚类算法中，一个基本的问题就是“什么是最好的聚类？”如果我们能定义“最好”的聚类，那就能发明一个以此为目标的算法。ROCK 算法中，最好的聚类就是 goodness 度量最大的聚类。

❸ ROCKClusters 类中封装了所有根据 goodness 度量识别最佳聚类所需的数据和算法。

❹ 这一步反复执行查找最佳聚类的过程，并检查两个终止条件。第一个终止条件是：如果聚类的数量等于所需的聚类数的最小值，算法停止。如果没有这个终止条件，算法会一直运行到所有的数据点都成为一个聚类，这没有意义。第二个终止条件是：如果两次迭代中聚类的数量没有变化，就没必要继续了，算法停止。

让我们更近距离地看看 MergeGoodnessMeasure 类。正如已经提到的，这个类中封装了评判聚类优劣的依据。基于相似度距离的算法自身是无法区分两个分割不彻底的聚类的，因为两个数据点有可能分属于两个聚类，但两者却是距离非常近的邻居。所以，其他的算法有可能会因为两个聚类中有两个元素相距很近而将两者合并，甚至在这两个数据点只有很少的公共邻居时也会如此。

因此，我们的首要任务就是要确保我们的聚类结果评价标准能帮助我们有效地处理这类情况。顾名思义，为了实现这个目标，ROCK 算法用到了链接。什么是链接？我们说，如果两个数据点有共同邻居，则这两个数据点之间有链接。当我们考虑是否要合并聚类 X 和聚类 Y 时，我们感兴趣的是两个聚类中数据点之间的链接数量，即一个点来自聚类 X，而另一个点来自聚类 Y 的链接数量。这样的链接越多，这两个聚类就更有可能属于同一个聚类，从而就能得到更好的聚类结果。让我们看看这种机制，代码清单 4.12 中是有关的代码。

代码清单 4.12 MergeGoodnessMeasure：识别最优聚类的依据 150

```
public class MergeGoodnessMeasure {
  private double th;
  private double p;
```

Download at http://www.pin5i.com/


```
    public MergeGoodnessMeasure(double th) {
        this.th = th;
        this.p = 1.0 + 2.0 * f(th);
    }
    public double g(int nLinks, int nX, int nY) {
        double a = Math.pow(nX + nY, p);
        double b = Math.pow(nX, p);
        double c = Math.pow(nY, p);
       return (double)nLinks / (a - b - c);
    }
    private double f(double th) {
        return (1.0 - th) / (1.0 + th);
    }
}
```

最核心的方法调用是 g(int nLinks, int nX, int nY)，其中 nLinks 是聚类 X、Y 之间的链接数量。你应该能想到（根据前面提到的共同邻居和链接），g 的返回值取决于两个聚类中数据点之间的链接数量。但其他的参数是什么意思？为什么这个公式如此复杂？首先来回答第一个问题。nX 和 nY 两个参数分别是聚类 X 和聚类 Y 中数据点的数量。第二个问题的答案比较烦琐，但也更加有趣。

你可能会想到，最大化聚类中数据点之间的链接数可能是判断两个聚类是否应该合并的好方法。但要记住，聚类实际上有两重目标。我们不仅要把属于同一个聚类的数据点放在一起，还要把不属于同一个聚类的数据点分开。取链接数量的最大值能保证相互之间有大量链接的数据点能被放在同一个聚类中，但却不能阻止算法将所有的数据点都放在一个聚类中。换言之，只考虑链接的数量不能有效地将相互链接较少的点分割到不同的聚类中。

ROCK 公式能用变量 *b* 和 *c* 分别估计聚类 X 和聚类 Y 中的链接总数。*f* 方法中的函数展示了这样一个重要的性质：聚类 X 中的每个数据点在聚类 X 中大概有 Math.pow(nX,f(th))个邻居。因此，聚类好坏程度的计算就是用数据点之间的链接数除以期望的链接数，即(*a*-*b*-*c*)。这样就能避免将相互之间链接很少的数据点放到同一个聚类中。变量 *th* 取决于 *f* 的具体实现，它的值大于或等于 0，小于或等于 1。如果它的值是 0，则 *f* 恒等于 1，所有的数据点相互之间都是邻居。如果它的值等于 1，则 *f* 等于 0，那么数据点就只会是其自身的邻居。*f* 的这种实现在实际的市场消费数据中有很好的表现，但对其他情况则有可能不太适用。当不同的数据点的某个属性

Download at http://www.pin5i.com/

的值比较统一时，就可能需要调整。你应该用我们的数据或是自己的数据尝试各种 f 的实现。在 To Do 一节中有更多的细节。

4.6 DBSCAN

本节将介绍一种先进的聚类算法，这是一种基于密度的空间聚类算法（Density-Based Spatial Clustering of Applications with Noise，即 DBSCAN），适用于含噪声的数据。该算法既不是基于链接的概念，也不是基于数据点之间的直接距离，而是引入了一个新的概念，即点密度（point density）。为了说明这个想法，假设有一个盛满清水的浅碟，在其中滴入几滴墨水，你立即就能看到哪些地方有墨水。这是因为由于密度的差别，墨水和清水对光线的反射是不一样的。这对聚类有什么意义？

4.6.1 基于密度的算法简介

有很多聚类算法都是基于简单的日常经验的。尤其是在实际生活中，我们根据密度的差异识别物体的边界，从而实现视觉上的模式识别，这就是基于密度的算法（density-based algorithm）的基本思路。如此将同样的原则扩展到二维、三维乃至多维空间，我们就可以根据空间中特定区域的数据点的密度来识别聚类（物体）。在图 4.14 左图中，大部分人都可以观察到如图 4.14 右图所示的三个聚类。通常，我们可以认为不属于这三个聚类的点（七个白色圆点）是噪声。

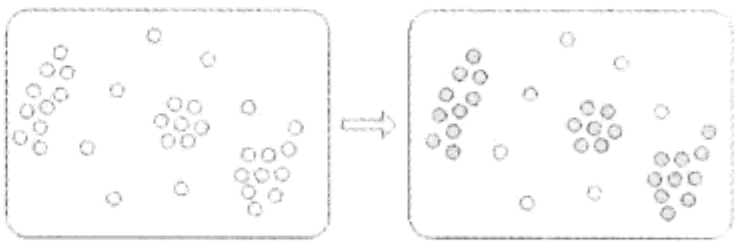


图 4.14 基于密度的聚类源自我们用眼睛识别形状和形式的能力

DBSCAN 算法是由 Martin Ester 等人提出的，目的是发现数据集中的聚类和噪声。在我们深入算法的细节之前，先运行代码清单 4.13 中的脚本得到聚类结果，所使用的数据与前面 ROCK 算法所使用的数据相同。

代码清单 4.13 DBSCAN 算法的使用

```
MyDiggSpaceDataset ds = MyDiggSpaceData.createDataset(15);←加载前 15 个 Digg 的新闻
DataPoint[] dps = ds.getData();
CosineDistance cosD = new CosineDistance();
DBSCANAlgorithm dbscan = new DBSCANAlgorithm(dps,cosD,0.8,2,true);←初始化 DBSCAN 算法
dbscan.cluster();
```

Download at http://www.pin5i.com/

152

可以看出，DBSCAN 算法的使用与前面介绍的算法一样简单。除了加载数据，在实例化 DBSCANAlgorithm 类之前唯一要做的事情就是定义合适的距离度量。在这里，我们使用了 CosineDistance，但实际上只要是实现了 distance 接口的类都可以在这里使用。

当然，我们没有将距离度量整合到算法的实现中，但距离度量的选择是非常重要的。因此，在构造函数中以参数的形式明确地定义距离度量应该是更好的选择。距离度量的选择实际上定义了“邻居”的形状（外观或体积），换言之，邻居定义了各种跟密度有关的参数，我们马上就会介绍这些参数。下面先来看看聚类结果，代码清单 4.13 中脚本的运行结果如图 4.15 所示。

```
bsh % dbscan.cluster();
DBSCAN Clustering with NeighborThreshold=0.8 minPoints=2
Clusters:
1:
{5605887:A Facebook Application To Find Blood Donors Fast,
5611687:A Facebook Application To Find Blood Donors Fast,
5608576:A Facebook Application To Find Blood Donors Fast}
-----------------------------------------------------------
2:
{5142233:The Confederacy's Special Agent,
5613383:The Confederacy's Special Agent,
5620839:The Confederacy's Special Agent,
5598008:The Confederacy's Special Agent,
5586183:The Confederacy's Special Agent,
5610584:The Confederacy's Special Agent,
5613380:The Confederacy's Special Agent}
-----------------------------------------------------------
3:
{5620878:Microsoft, The Jekyll And Hyde Of Companies,
5618201:Microsoft, The Jekyll And Hyde Of Companies,
5585930:Microsoft, The Jekyll And Hyde Of Companies,
5609797:Microsoft, The Jekyll And Hyde Of Companies,
5609070:Microsoft, The Jekyll And Hyde Of Companies,
5524441:Microsoft, The Jekyll And Hyde Of Companies}
-----------------------------------------------------------
4:
{5594161:Contract Free on AT&T,
4955184:Contract Free on AT&T,
5608052:Contract Free on AT&T,
5621623:Contract Free on AT&T,
5579109:Contract Free on AT&T,
5620493:Contract Free on AT&T}
-----------------------------------------------------------
```

图 4.15(a)　运行代码清单 4.13 中脚本得到的聚类结果

Download at http://www.pin5i.com/


```
5:
{5607863:Lack Of Democracy on Digg and Wikipedia?,
 5571841:Lack Of Democracy on Digg and Wikipedia?,
 5619782:Lack Of Democracy on Digg and Wikipedia?,
 5611165:Lack Of Democracy on Digg and Wikipedia?,
 5543643:Lack Of Democracy on Digg and Wikipedia?}
_____________________________________________________
6:
{5481876:How Traffic Jams Occur : Simulation,
 5613023:How Traffic Jams Occur : Simulation}
_____________________________________________________
7:
{5617459:Robotic drumstick keeps novices on the beat,
 5619693:Robotic drumstick keeps novices on the beat}
_____________________________________________________
8:
{5617998:Obama: ""I Am NOT Running for Vice President"",
 5625315:Obama Accuses Clinton of Using ""Republican Tactics""}
_____________________________________________________
9:
{5607788:Recycle or go to Hell, warns Vatican -- part I,
 5595841:Recycle or go to Hell, warns Vatican --- part IV,
 5618262:Recycle or go to Hell, warns Vatican -- part III,
 5592940:Recycle or go to Hell, warns Vatican -- part II}
_____________________________________________________

Noise Elements:
 {5610213:Senate panel critiques prewar claims by White House,
  5619818:A Facebook Application To Find Blood Donors Fast,
  5612810:Super Mario Bros Inspired Wii with USB base [ Pics ],
  5522983:Smoking Monkey[flash],
  5609833:NSA's Domestic Spying Grows As Agency Sweeps Up Data,
  5625339:Lawmaker's Attempt to Criminalize Anonymous Posting
Doomed,
  5610081:Digg's Algo Change Cut Promotions by 38%,
  5604438:Archaeologists Unveil Finds in Rome Digs,
  5614085:House Files Contempt Lawsuit Against Bush Officials,
  5592473:Maryland police officers refuse to pay speeding
tickets,
  5622802:House Democrats Defy White House on Spying Program}
```

图 4.15(b) 运行代码清单 4.13 中脚本得到的聚类结果（续）

结果很不错！该算法不仅识别出了几个比较明显的聚类，还识别出了不太明显的文章聚类（聚类 8）。而且该算法还能识别出不属于任何聚类的数据点，这些数据点被视为噪声。要注意的一个小问题是，噪声中的一个元素，即 ID 为 5619818 的文章应该属于第一个聚类。

Download at http://www.pin5i.com/

4.6.2 DBSCAN 的原理

现在来看看细节。首先，需要定义构建 DBSCANAlgorithm 类的参数。构造函数的声明如下：

```
public DBSCANAlgorithm(DataPoint[] points,
                       Distance distance,
                       double eps,
                       int minPoints,
                       boolean useTermFrequencies)
```

154

你现在应该很熟悉 DataPoint 数组了，它是用来存放数据的。通过 Distance 接口，我们可以使用适合自己的数据的距离度量。有趣的是，*eps* 变量通常表示希腊字母ε，而这个希腊字母经常用来表示很小的正数。这个ε的值有助于我们确定数据点p在 DataPoints 集合中的ε邻居（epsilon neighborhood），即距离小于或等于ε的数据点。所以，ε邻居的定义是很直观的，非常符合我们的直觉。还有一些更加复杂的定义，让我们来看看图 4.16，这幅图与 Ester 等人论文中的图类似。

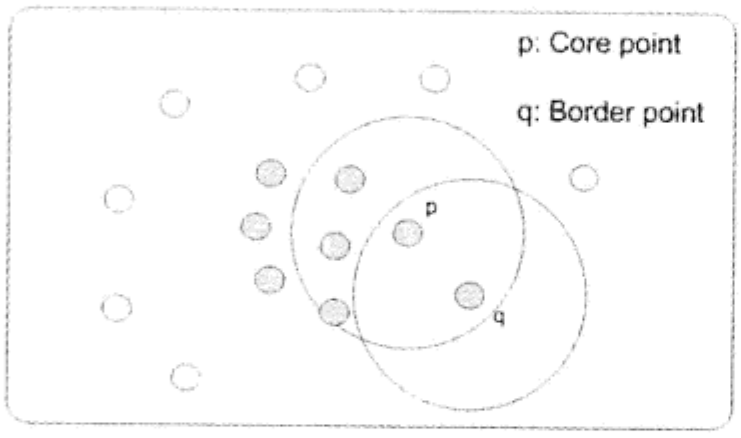


图 4.16 DBSCAN 中的核心点和边界点

图 4.16 中的大圆圈是数据点 p 和 q 的ε邻域，一个圈的圆心是 p，另一个的圆心是 q。这些圆的半径等于ε（*eps*），而 *minPoints* 变量则指定了能成为核心点（core point）的数据点的圆圈中所必须包含的数据点的数量。属于某个聚类，但不是核心点的数据点被称为边界点（border point）。如果满足下面两个条件，我们就说对于 *eps* 和 *minPoints*，数据点 q 基于密度可直接达到数据点 p：

- p 在 q 的ε邻域内。
- 在 q 的ε邻域内的数据点数量多于 minPoints。

在图 4.16 中，从 p 基于密度可直接到达 q，而从 q 则不能基于密度直接到达 q。这些都是理解 DBSCANAlgorithm 代码所必需的基本概念，其代码如代码清单 4.14 所示。

Download at http://www.pin5i.com/

代码清单 4.14 DBSCANAlgorithm：我们的实现中的两个核心方法

```
public List<Cluster> cluster() {
  int clusterId = getNextClusterId();
  for(DataPoint p : points) {
    if( isUnclassified(p) ) {
    boolean isClusterCreated = createCluster(p, clusterId);❶
    if( isClusterCreated ) {
       clusterId = getNextClusterId();
    }
   }
  }
  List<Cluster> allClusters = new ArrayList<Cluster>();
  for(Map.Entry<Integer, Set<DataPoint>> e : clusters.entrySet()) {
    String label = String.valueOf(e.getKey());
    Set<DataPoint> points = e.getValue();
    if( points != null && !points.isEmpty() ) {
       Cluster cluster = new Cluster(label, e.getValue());
       allClusters.add(cluster);
    }
  }
  return allClusters;
}
private boolean createCluster(DataPoint p, Integer clusterId) {
  boolean isClusterCreated = false;
  Set<DataPoint> nPoints = findNeighbors(p, eps);❷
  if( nPoints.size() < minPoints ) {❸
    assignPointToCluster(p, CLUSTER_ID_NOISE);
    isClusterCreated = false;
   } else {
    assignPointToCluster(nPoints, clusterId);❹
    nPoints.remove(p);
    while( nPoints.size() > 0 ) {❺
     DataPoint nPoint = nPoints.iterator().next();
     Set<DataPoint> nnPoints = findNeighbors(nPoint, eps);❻
     if( nnPoints.size() >= minPoints ) {
       for(DataPoint nnPoint : nnPoints ) {
          if( isNoise(nnPoint) ) {
            assignPointToCluster(nnPoint, clusterId);❼
```

155

Download at http://www.pin5i.com/


```
            } else if( isUnclassified(nnPoint) ){
              nPoints.add(nnPoint);❽
              assignPointToCluster(nnPoint, clusterId);
            }
          }
        }
      nPoints.remove(nPoint);❾
    }
    isClusterCreated = true;
  }
  return isClusterCreated;
}
```

❶ 对于每个还没有被聚类的数据点，创建一个包含它的聚类。cluster 方法中的其他内容都是一些细节处理。

❷ 根据给定的 *eps* 参数，找到数据点 *p* 的ε邻居。

156

❸ 如果 *p* 的ε邻域中没有足够的数据点，那么 *p* 就是噪声或边界点。暂时先将其视为噪声，就算它是边界点，也等稍后再来修改它的标签。在这一步，还没有给 *p* 创建聚类。

❹ 如果ε邻域中的数据点数量大于或等于数据点数量的下限 *minPoints*，就可以继续。但我们首先要将给定的数据点（*p*）从它自己的聚类中删除。

❺ 处理除 *p* 外的所有的数据点。

❻ 找到ε邻域，判断数据点是否基于密度可直接达到。

❼ 如果是边界点，将其分配给相应的聚类。

❽ 如果数据点不是噪声，我们就要判断它是否是核心点，所以要把它添加到原来的列表中以找到它的ε邻居，然后将其分配给我们合适的聚类。

❾ 在继续检查*p*的邻域中的下一个数据点之前，先将已经检查完的数据点移除。

从这段介绍中应该能看出，*eps* 和 *minPoints* 的选择是非常重要的。找到这两个变量的最优值的最佳方法就是根据已知的聚类形式在数据上尝试不同的值。如果处理的是二维数据集（数据点有两个属性），那么 4 就是 *minPoints* 的一个合理选择。可以从这个值开始，然后尝试其他不同的值，看看聚类结果是否有改进。由于多维空间的属性不同于低维度空间，所以多维数据集要更困难一些，我们会在附录 C 中对此进行探讨。如果能利用以下因素，则 *eps* 的选择相对会简单一些。

Download at http://www.pin5i.com/

- 数据规范化。首先在条件允许的情况下将数据规范化，这对于维度高于二维的数据尤其重要。
- 对数据点之间的距离做统计分析。
- 数据集维度——数据点有多少属性？

首先考虑同一个聚类中数据点的距离应该较近，从而得到一个 *eps* 的初始值。然后按几何级数递增，例如以 2 为因子，重新做聚类。*eps* 的变化会给结果带来什么影响？如果结果没有明显的变化，选用平均值就可以了。如果聚类结果有明显变化，则可以分为两种情况：

- 随着值的增加，结果持续地变好或是变差。
- 结果好坏的变化并不一致，例如 *eps* 的值翻倍后，结果变差了，但再次翻番后结果却变好了。

第一种情况比较简单。如果随着值的增加，聚类结果不断变好，此时选用最大值就可以了，或是在时间和资源允许的前提下不断地增加 *eps* 的值，直到结果不再变好。

第二种情况比较少见，通常出现在高维聚类且距离度量不理想的情况下。因此，首先要做的事情就是检查是不是真的需要所有的属性。其次，尝试其他可能更加适合数据的距离度量。这就是 DBSCAN 算法的核心思想，以及实现的基本步骤。Ester 等人的论文中更加具体地介绍了这种算法，在学习完本章后再阅读这篇论文应该不会有困难。 157

目前所介绍的算法已经能够处理大部分的数据集。前面说过，在不同类型的数据中，各个算法的表现也不尽相同。不过，聚类结果的好坏不是我们唯一要考量的因素。在下一节中，我们会介绍一些对大规模数据集进行聚类分析时的常见问题。

4.7 超大规模数据聚类

在超大规模数据中常见的问题可以分为两大类。第一个问题是要聚类的数据点的数量。这就需要考虑到聚类算法的计算复杂性，及其对性能的影响。换言之，我们需要知道计算的数量和要聚类的数据点数量之间的函数关系。第二个问题则是属性（维度）的数量对算法的影响。我们最熟悉的世界是三维的，所以我们的直觉只

Download at http://www.pin5i.com/

适用于三维或更低的维度。对于高于三维的维度，集合对象和相似性概念的性质都是不一样的，这会带来两方面的影响。首先，维度越高，需要的计算量越大，这就降低了处理速度。其次，会出现很多涉及高维度特性的问题，这些问题被统称为“维度灾难”（curse of dimensionality）。下面让我们逐一看看这些问题。

4.7.1 计算复杂性

聚类算法的性能与待聚类的数据点的数量之间是有函数关系的，知道这一点很重要。比较对 MySpace 的用户（有 $O(10^8)$个注册用户）进行聚类分析和对本地报纸（几百个订阅者）或社区大学（几千名学生）数据库中的信息进行聚类分析，这两者间的差别是巨大的。如果要量化数据点数量（n）的影响，就需要理解算法的时间复杂性和空间复杂性。其实就是执行一个聚类算法所需的内存大小和所需进行的操作次数。表 4.2 中列出了本章中实现的各种算法的时空复杂性，这里 k 表示聚类的数量，t 表示迭代次数（在 k-means 算法中）。

注意 n^2 因子的重要影响。这种与数据点数量的平方关系给很多聚类算法带来了麻烦。

158

表 4.2 聚类算法的时空复杂性

算法名称	空间复杂性	时间复杂性
单链接	$O(n^2)$	$O(kn^2)$
平均链接	$O(n^2)$	$O(kn^2)$
MST 单链接	$O(n^2)$	$O(n^2)$
k-means 算法	$O(n)$	$O(t\text{kn})$
ROCK 算法	$O(n^2)$	$O(n^2\log(n))$或使用空间索引是 $O(n\log(n))$
DBSCAN 算法	$O(n^2)$	$O(n^2)$

在 Margaret Dunham 的 *Data Mining* 一书（见第 5 章参考资料）中有一张表格更加详细地比较了各种聚类算法。从性能方面看，k-means 算法无疑是赢家，而且在实际中广泛应用也很有可能是因为它的高效。但要记住，k-means 算法无法处理类别型数据。另外，如果对数据点用空间索引，还可以进一步降低 DBSCAN 算法的时间复杂性。我们要在度量空间中处理密度，所以很自然地可以将属性的值视为数据点在空间中的坐标。通常用 R-tree 实现空间索引，绝大部分的商业数据库都提供了空间数据的 R-tree 实现。但是，你应当意识到索引高维空间数据时会遇到的各种困难。

Download at http://www.pin5i.com/

尽管人们已经发表不少解决问题的思路，但这依然是一个非常活跃的研究领域，最终问题（高效索引高维度数据）依然未解决。

当然，还有其他一些算法致力于解决这些效率问题。BIRCH 就是一个非常流行的专为大规模数据集设计的聚类算法，目前已经有人对其做了深入研究。它的时空复杂性都是线性的——$O(n)$，而且它只需要扫描一次数据库。这个算法属于基于数据挤压（data squashing）的一类算法。这类算法已经超出了本书的范畴，对数据挤压算法有兴趣的读者请自行查阅参考资料。

4.7.2 高维度

处理超大规模数据集容易遇到的第二类问题就是数据的高维度。在超大规模数据集中，数据点可能会有很多属性，如果我们不舍弃其中的部分属性，度量空间的维度就会很高，甚至会有数百维！前面已经提到了有关高维度的缺点，但我们还没有具体说明到底多高才算“高”。通常来说，“高维度”表示数据点的属性超过 16 个（我们喜欢用 2 的幂），当数据维度达到 $O(10)$时，就会给我们解决问题带来麻烦。你可能会好奇，这到底有多大影响？我们前面看到的公式并没有局限于低维空间，高维度到底会带来什么影响？ 159

对于聚类来说，高维度主要会带来两个难题。实际上，接下来的大部分内容对分类算法也同样成立。第一个问题是高维度扩大了数据点所分布的空间。换言之，如果保持数据点数量不变，但增加用于描述这些数据点的属性，空间中的数据点密度就会成指数减少。这样，就很难判断出聚类的优劣。

第二个基本问题有一个恐怖的名字，叫做维度灾难（curse of dimensionality）。简单地说，在高维度空间中，任何数据集的数据点，无论是用何种距离度量，数据点之间的差异都会变小。为了说明维度的这种重要影响，让我们看看下面这个简单的例子，如图 4.17 所示。

如果从左到右看图 4.17，每幅图的维度都增加了 1。我们首先把 8 个点均匀地排布在一维空间（x 轴）中，例如 0 到 1 之间。两个数据点之间的最小距离 $\min(D)=0.125$，而两个点之间的最大距离 $\max(D)=1$。这样 $\min(D)$和 $\max(D)$之间的比值等于 0.125。在二维空间中，我们再将这 8 个数据点均匀排布，但现在 $\min(D)=0.5$，而 $\max(D)=1.414$（沿主对角线）。这样，$\min(D)$和 $\max(D)$的比值就等于 0.354。在三维空间中，$\min(D)=1$，而 $\max(D)=1.732$。这样，$\min(D)$和 $\max(D)$的比值等于 0.577。

Download at http://www.pin5i.com/

160

换言之，随着维度的增加，最小距离和最大距离之间的比值会逐渐逼近 1。也就是说，无论从什么角度用什么度量去看，所有的数据点都变成一样的。

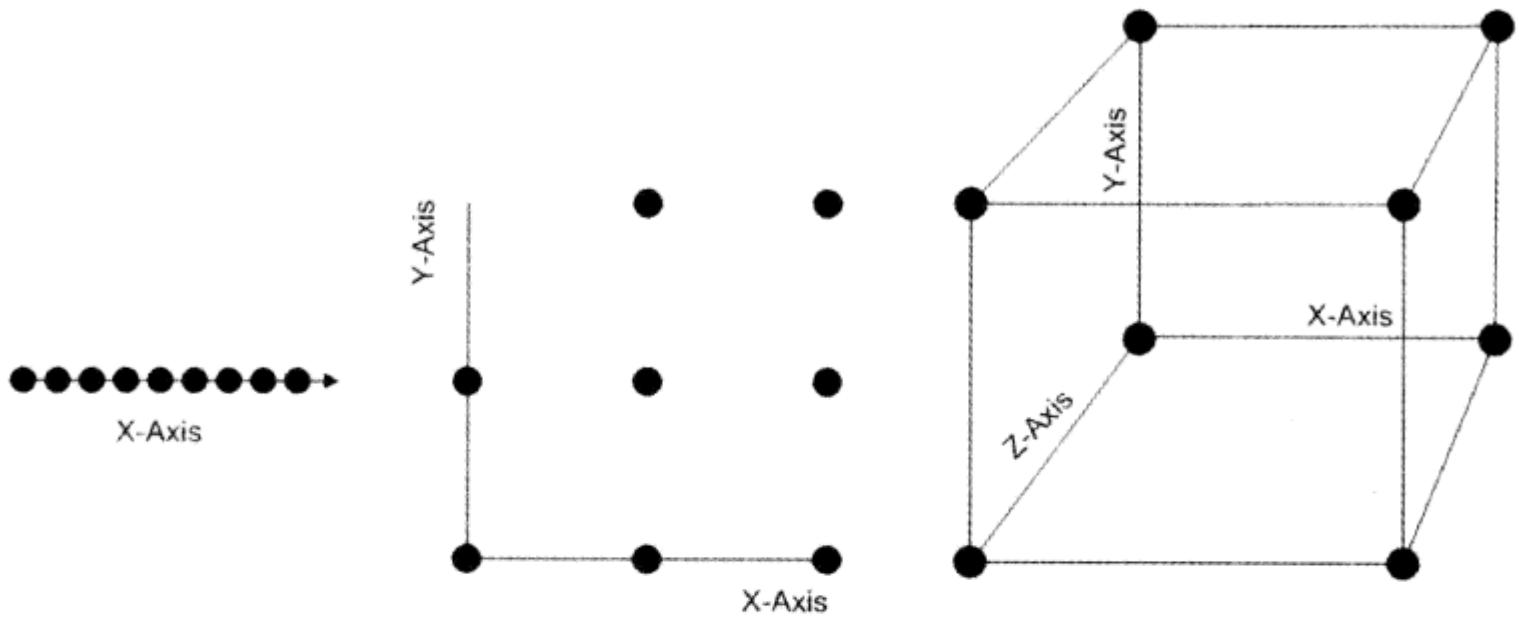


图 4.17 维度灾难：每个数据点跟其他数据点的距离都是一样的

最终的结果就是，随着维度的增加，所有依赖于距离计算的算法都会很快滑入这个深渊。我们在这个例子中使用的是标准的欧几里德距离，但其实无论是用什么距离度量（合适的），都会遇到同样的问题。已经有人（Kevin Beyer 等人）质疑，在高维度空间中讨论最近邻居是否有意义，对此问题有兴趣的读者可以参考 C. C. Aggarwal 的论文。

4.8 总结

聚类算法是非常有用的数据探索工具。我们可以构建一个多层的聚类结构，或者也可以事先确定数据的聚类数量。聚类有广泛的应用领域，从理论上说，聚类适用于所有可用属性值定义的对象组成的数据集。但要注意的是，对象间距离的计算方法和选择合适的算法。

在本章中，我们介绍了如何分组论坛帖子和识别相似的网站用户。从算法的复杂性来看，既有简单的 SQL 语句，也有一些相当高级的数学方法。我们概述了各种类型的聚类算法和 6 个聚类算法的完整实现：单链接、平均链接、MST 单链接、k-means、ROCK 和 DBSCAN。

单链接、平均链接和 MST 单链接算法都是凝聚层次算法，在计算时需要所有的

Download at http://www.pin5i.com/

数据。它们的计算复杂性，无论是空间复杂性还是时间复杂性，都不是很理想，基本上都是数据点数量的平方。因此，尽管这些算法容易实现，但在大规模数据集上的表现并不尽如人意。这里要特别提一下基于 MST 的单链接算法，我们可以改进该算法的时间复杂性，使其接近数据对象的数量。

k-means 算法是一个高效的迭代分割算法，效果通常也不错。但因为它依赖于几何意义上的中心点，而在处理分类数据时无法找出这样的中心点，所以 k-means 算法无法处理分类数据。这个算法的另一个不足就是无法处理异常点，也就是那些远离主要聚类的数据点。

ROCK 算法非常适合于布尔数据和分类数据，因为它利用的是对象间的链接数，而不是距离。这是一个层次凝聚算法，其空间复杂性并不理想，是 $O(n^2)$，而时间复杂性更差，是 $O(\log(n)n^2)$。

DBSCAN 算法引入了密度的概念，而且区分了聚类中的核心点和边界点。尽管空间和时间复杂性都是 $O(n^2)$，但它能很好地处理异常点，而且还能跟 k-means 算法组合在一起构成一个不错的快速聚类算法，具体内容详见下一节的 To Do 事项。 161

本章对聚类算法的覆盖既不深入，也不够全面。在文献中还有更多的算法，就是本章介绍的算法也还有很多衍生算法。不可能用本书的一章内容把它们都囊括进来。但你现在已经掌握了基本知识，实现了几个聚类算法，可以尝试实现这些算法，也可以根据自己的需求扩展它们。

4.9 To Do

1．SQLEM 算法。这是一个基于 SQL 的最大期望（Expectation-Maximization，EM）算法，也是统计领域里一个非常有名的算法。它分为两个步骤：E-step 和 M-step。这个算法蕴含着高深的数学知识，但我们并不要求你掌握这些知识。读者可以将 k-means 算法（4.4 节）视为一个 EM 算法，E-step 就是初始分配中心点，而 M-step 就是更新中心点的值。

根据不同的应用，在关系数据库中有很多方法可以实现这个算法，你可以将结果与本章中的结果进行比较。关于此算法的具体描述，可以在 Carlos Ordonez 和 Paul Cereghini 最初的论文中找到。

2．最小生成树（MST）算法。本书没有提供 Prim-Jarník 算法的实现。最小生成树并不是只用于聚类，在网络组织和旅行问题中也适用。在 1926 年，Otakar

Download at http://www.pin5i.com/

Boruvka 展示了第一个知名的计算 MST 的算法，其用途是评价高效电气系统在摩拉维亚的覆盖率。

在互联网上一份自由传播的教程中（见本章参考资料一节），Jason Eisner 解释了经典的算法，以及 Harold Gabow、Zvi Galil 和 Thomas H. Spencer 提出的改进方法，还介绍了 David R. Karger、Philip N. Klein 和 Robert E. Tarjan 的随机算法。最后一个算法只需要 $O(m)$次计算，其中 m 是边的数量。阅读 Eisner 的这本教程，并根据书中的介绍扩展 MST 类以支持更多的高效算法。

3．ROCK：计算链接的期望数。前面提到，ROCK 算法用链接的概念替代了直接的距离比较，因此，擅长处理分类属性和布尔属性。它通过最大化优度（goodness measure）寻找“最佳”聚类。优度是很随意的，其基本思想是用链接比较最佳聚类，但具体的实现会有一些差异。我们的实现中所使用的启发式算法是 Ramanathan V. Guha 等人最早提出的。在这个算法中的优度是两个聚类间的链接数量和聚类间的链接期望数之间的比值。

162 这其中蕴涵的基本原理是什么？还能构建其他的估计值来推测两个聚类间链接的期望数吗？尝试用不同的思路，并比较结果。如果我们简单地用两个聚类间交叉链接的比例来计算优度，会有什么结果？

4．大规模数据集聚类。在处理大规模数据集时，我们经常要整合本章所介绍的各种算法，从而做到效率和聚类结果质量之间的平衡。例如，将 k-means 算法和 ROCK 算法（如果数据主要是分类属性或布尔属性）或 DBSCAN 算法（如果数据能对应到空间或是其他度量距离有意义且有效的度量坐标）整合在一起，就是一种可行的策略。

这样会有什么效果？回忆一下，k-means 算法在空间和时间上都是最高效的。而且，如果有大量的处理器，还可以采取并行的方法。也就是说，可以用 k-means 算法做若干次迭代，得到一些顶层的聚类，然后可以用 ROCK 或 DBSCAN 算法对这些聚类进行处理。实现这个想法，然后用一个大规模数据集（成百上千个数据点）测试效果。你可以使用个人计算机上的文档或是工作中使用的大型数据库。

另一个思路是对大规模数据集进行采样，然后用 ROCK 或 DBSCAN 这样的强大算法对样本进行聚类。接下来，将 k 设为已经识别出来的聚类的数量，然后用样本聚类的中心点作为初始中心点运行 k-means 算法。比较这两种方法：哪一个的聚类结果更好（通过观察数据，根据经验做判断）？哪种方法效率更高？能否通过分析证明你的结论？

Download at http://www.pin5i.com/

4.10 参考资料

Aggarwal, C.C. "Towards Meaningful High-Dimensional Nearest Neighbor Search by Human-Computer Interaction." *ICDE*, 2002. http://citeseer.ist.psu.edu/aggarwal02towards.html.

Arthur, D. and S. Vassilvitskii. "k-Means++: The advantages of careful seeding." *Symposium on Discrete Algorithms* (SODA), 2007. http://www.stanford.edu/~darthur/kMeansPlus-Plus.pdf.

Beyer, K., R. Ramakrishnan, U. Shaft, J. Goldstein. "When is nearest neighbor meaningful?" ICDT Conference 1999.

Bradley, P.S., U. Fayyad, and C. Reina. "Scaling clustering algorithms to large databases." Proc. 4th International Conference on *Knowledge Discovery and Data Mining* (KDD-98). AAAI Press, pp. 9 – 15.

Dhillon, I.S., J. Fan, and Y. Guan. "Efficient clustering of very large document collections." *Data Mining for Scientific and Engineering Applications*, 2001. Kluwer Academic Publishers.

Dhillon, I.S., S. Mallela, and R. Kumar. "A Divisive Information-Theoretic Feature Clustering Algorithm for Text Classification." *Journal of Machine Learning Research* 3 (March 2003).

Eisner, J. "State-of-the-art algorithms for minimum spanning trees – A tutorial discussion." 1997. http://citeseer.ist.psu.edu/eisner97stateart.html.

Kruskal, J.B. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem." Proceedings of the American Mathematical Society, Vol 7 (1), pp. 48–50. 1956. 163

Ordonez, C. and Cereghini, P. "SQLEM: Fast clustering in SQL using the EM algorithm." *ACM SIGMOD* Rec. 29 (2), pp.559 – 570. 2000.

Prim, R.C. "Shortest connection networks and some generalizations." *Bell System Tech. J.* 1957.

Sheikholeslami, G., S. Chatterjee, and A. Zhang. "WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases." *Proceedings of the 24th VLDB Conference.* 1998.

Tung, A.K.H., J. Hou, and J. Han. "Spatial clustering in the presence of obstacles." In *Proceedings of the 17th ICDE*, 359-367, Heidelberg, Germany. 2001.

Download at http://www.pin5i.com/

5 分类：把事物放到它该在的地方

本章内容提要：

- 了解基于概率与规则的分类技术
- 自动分类电子邮件
- 用神经网络检测金融交易欺诈

“这是什么？”这也许是小孩问得最多的一个问题。好奇是孩童有趣而固执的天性，因而他们热衷于这个问题的普遍性也就不足为奇了。为了理解我们生活的这个世界，我们需要把自己的认知组织成群组与类别。在第 4 章里，我们提出了很多聚类算法来帮助我们把数据点进行分组。在本章，我们还会提出很多分类算法来帮助我们把每个数据点分配到合适的类别中，即所谓的“分类”。分类所承担的角色就如同回答小孩的问题：“这是一只船。”“这是一棵树。”“这是一间房子。”等。分类需要依赖先验的参考结构，它们把所有可能的数据点组成的空间划分为一些类别的集合，这些类别通常（但并非一定）是不重叠的。作为对比，第 4 章所述的聚类，其结构本质上可以是任意的。

可以说，作为心理过程的一部分，聚类是先于分类的[1]，因为我们在分类中使用

1 即分类是一项更高级的认知活动。——译者注

Download at http://www.pin5i.com/

的参考结构是对知识结构的一种更为丰富的表示，而不单是像“X 与 Y 是同一个组别”这样的模糊表述。参考结构中通常用“本体”这个术语来表征构成这个世界（或者这个世界中我们感兴趣的那部分）的知识。本体维护在分类中通常略于讨论，但在实际中却很重要，有很多书专门讨论“本体引擎”与“本体管理”（参看 Staab 与 Studer Gómez-Pérez，Fernández-López 与 Corcho 的论文）。

5.1 节会提供一些现实的例子来说明分类的重要性，同时会给出一个抽象本体的定义，以及本体结构与 Java 代码结构的类比。以此来激发你的积极性之后，我们进入 5.2 节，该节会给出分类器的概述。当然，我们不可能在本书中覆盖所有已知的分类器，所以该概述的作用就是帮助你找到相关的文献。

在 5.3 节，你会学习到朴素贝叶斯分类算法，这是有史以来最著名的分类算法之一。我们会讨论电子邮件过滤中的具体例子：垃圾邮件过滤，以及更一般的情况，即把邮件分置到几个合适的文件夹中。要讲述怎么通过统计分类算法对自由文本进行分类，这是一个很好的例子。

但邮件系统里最常见的分类算法却是基于规则的。5.3.2 节从规则引擎的角度来叙述邮件分类，我们会介绍所有相关的概念，并使用 Drools（JBoss）规则引擎来阐述规则的使用。在 5.4 节，我们会把欺诈检测作为一个分类问题来处理，在那部分内容里，我们会介绍另一个广泛使用的分类方法——神经网络分类器。

该如何判断分配给数据点的类别是否合适？如何确定分类器 A 确实比分类器 B 更好？如果你曾经阅读过商务智能工具的手册，你也许会对类似于“我们的分类器精确度为 75%”这样的描述感到熟悉。这种描述的意义是什么？它对我们有帮助吗？这些问题会在 5.5 节得到解答。我们还会讨论大规模数据的分类、超大型的本体结构的分类，以及怎么实现高效的在线分类。这三个相互交叉的主题需要给予特别的重视，因为它们在实际应用中非常常见。

作为开启本章的旅程，让我们先讨论分类潜在的应用，同时介绍那些在旅途中会反复遇到的技术术语。那么，分类擅长做什么？它可以为我们解决什么实际问题？

5.1 对分类的需求

不管是否意识到，我们在每天的基本生活中都会遭遇分类。依据日常经验，食物可以根据类别——沙拉、开胃菜、特色菜、面条、海鲜等列在饭店的菜单上；报

Download at http://www.pin5i.com/

166

纸上的文章或互联网上的新闻组也可以根据它们的主题——政治、体育、商业、国际、娱乐等分类排列。

图书馆里的书有一个索书号（call number），它由两个数字组成：杜威分类号与 Cutter 号。这套系统的顶层分类如综合、宗教、自然科学、数学等。美国国会图书馆也有一套用于组织和规划其藏书的分类系统，它最早是在 19 世纪晚期到 20 世纪初期开发的。

在整个 20 世纪，国会图书馆的这套系统也同时被其他图书馆所采用，特别是美国的大型学术图书馆。我们之所以提到这两种书籍分类系统，是因为国会图书馆的分类系统并不像杜威分类系统那样具有严格的层次结构，对于后者，主题间的层次关系是反映在分类号中的。我们很快就会看到，区分层次与非层次型的参考结构是很重要的。

在医学上，有大量用于诊断损伤或疾病的分类系统。比如，Schatzker 分类系统被放射科医生与骨科医生用来分类胫骨平台骨折（一种复杂的膝盖损伤）。类似的还有脊髓损伤、昏迷、脑震荡及外脑损伤等分类系统。

按照美国政府的要求，《职业性损伤与疾病分类手册》（Occupational Injury and Illness Classification，OIIC）提供了一个分类系统，用于为职业性损伤与疾病调查（Survey of Occupational Injuries and Illnesses，SOII）及致命职业性损伤普查（Census of Fatal Occupational Injuries，CFOI）中的损伤、疾病与死亡案例的特征进行编码。世界卫生组织（World Health Organization，WHO）的 ICD-10 编码系统在 1990 年 5 月第 43 届世界卫生会议获得通过，并在 1994 年为成员国所使用。这个系统用于对疾病与健康问题进行分类，这些疾病记录来自死亡证明或医院病历中各种类型的健康与致死的记录。在你看完医生后，这个分类结果就会被保险公司用做保险金额比例的参考。顶层的分类包括某些传染与寄生病、肿瘤、内分泌疾病、营养疾病与代谢性疾病等。在生物学中，林奈（Linnaean）分类系统使用两个属性对所有的生命体进行分类——genus 与 species。你也许听说过“Homo sapiens”这个术语，其中 Homo 就是 genus，sapiens 就是 species。这个系统还可以扩展以包含其他的属性，如 family、order、class、phylum 等。

我们稍作离题以提醒你对维数的注意。一般来说，采用的属性越多，分类的效果会越好。“大量”的属性通常是一件好事，但这个一般性的原理也有警戒线。处理大量属性时会面临一个臭名昭著的问题叫“维度灾难”（curse of dimensionality），这

Download at http://www.pin5i.com/

点我们会在 4.7.2 节再讨论。通常，“大量”的意思是说我们处理的属性大于 16 个。

也许你会记得，维度灾难涉及这样的事实：随着属性数量的增加，属性空间会变得越来越同质化。换句话说，无论选用什么距离度量，任意两个数据点间的距离都会变得大致相等。一旦遇到这种情况，要分辨出哪个类别与某个数据点更“接近”，会变得越来越困难，因为不管你“站”在空间的哪个位置，所有的数据点离你的距离都是差不多的。不过，不断地往你的本体里添加属性是可以的，这样就可以在一个空间里包含所有领域的知识，但是在随后的分类时需要选择适量的属性，而不一定是全部。 167

从这些例子里可以明显地看到：平坦参考结构比层次参考结构要“贫瘠”，而层次参考结构又不如浓缩了层次与语义的参考结构那么“丰富”。这些观察结果又要再次归结到对本体的讨论中。我们不会为“本体”这个术语提供一个清晰的定义，因为对它的定义也并没有一个一致的定论。

在本书，本体由三个方面构成：概念（concepts）、实例（instances）、属性（attributes）。

图 5.1 中，我们描绘了一个基本的常见本体的片段，描绘的对象是“vehicle”这个概念。概念用椭圆表示，实例用长方形表示，属性用圆角方形表示。要注意属性分配的遗传性，如果属性被分配到概念树的根节点，那么它会扩散所有的叶子节点。所以，属性 1 的值会被分配到 boat 与 automobile 中，但属性 2 的值只归 automobile 所有。属性 1 可以是“名字”这个属性，依你的实际情况而定，而属性 2 可以是“车轮数量”。属性是定义在概念这一层上的，但只有实例才有具体和唯一的值，因为只有实例才是表示真实的“事物”。

可以把概念与 Java 的类相类比，实例相当于 Java 类的实例，属性则是 Java 类的变量。无疑，拥有如下特性的源代码显得更为出众：通过包来组织不同功能与组件的类；使用继承来抽象共同的结构与行为；适当地使用封装。与本体中定义的属性类似，当你定义一个类的同时，你会定义变量的数据类型，但不会为变量赋值（除非它是一个常数）。这种定义方式可以为你节省 80%～90%的时间，如果你对此存疑，不妨借助这种类比方式来更深入地审视你的类结构。

很显然，我们还会面对更多随处可见的分类系统，分类数据的关键在于如何把数据结构化或者组织起来。分类系统通过降低因信息模糊而导致的误差来达到改善交流的效果，它们还可以帮助我们组织自己的想法和计划我们的行动。用于组织我

Download at http://www.pin5i.com/

168

们数据的参考结构，可以是一些简单的标签集合，也可以是高级的语义本体。你听说过语义网吗？语义网的核心（参看 Antoniou 与 van Harmelen 的文章）就由大量如何创建、使用与维护语义本体的技术与标准定义所组成。本体在以模型驱动的架构（参看 Gasevic、Djuric 和 Devedzic 的文章）中也有使用，这也是对象管理组织（Object Management Group，OMG，http://www.omg.org/）创立的初衷。

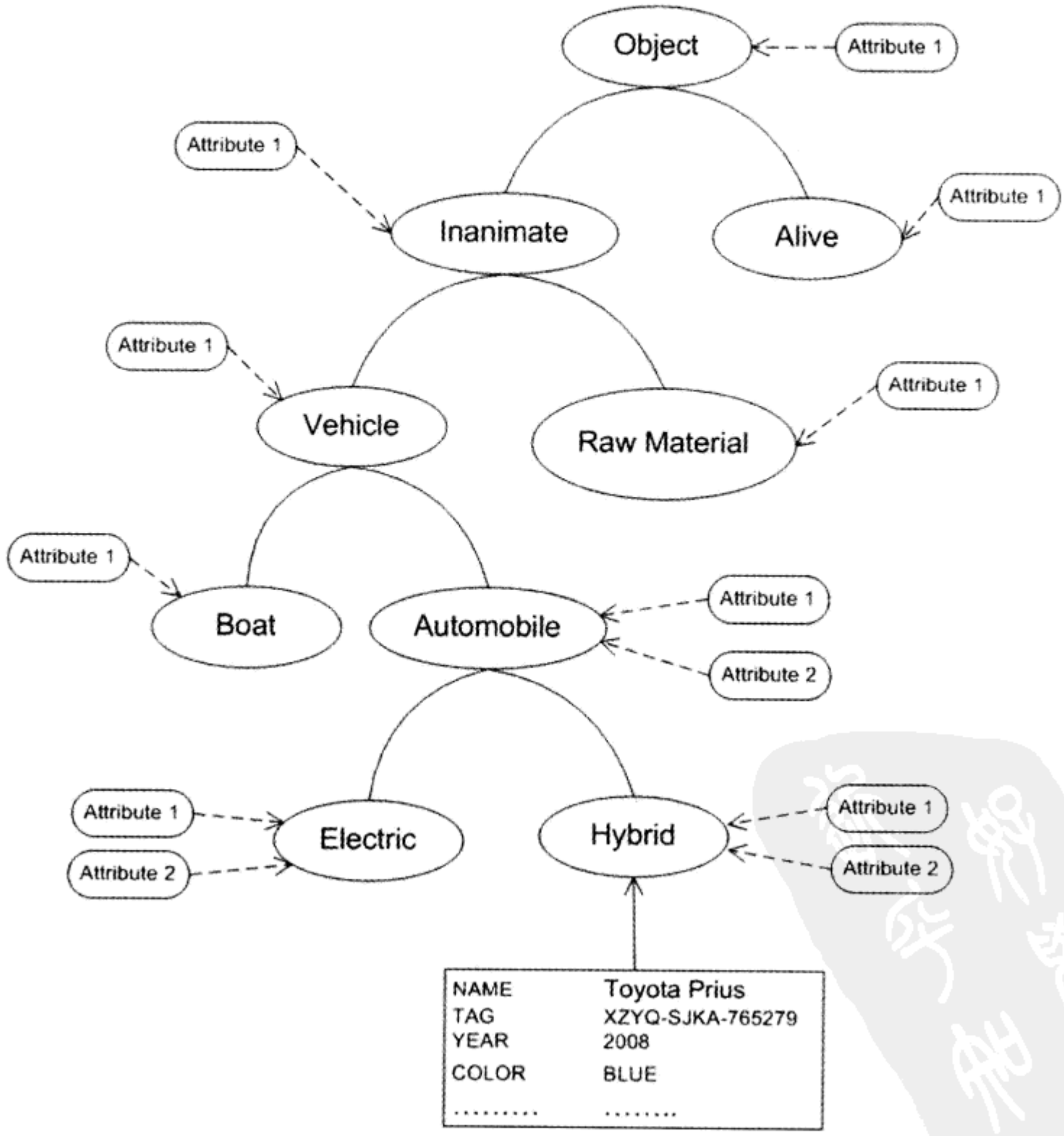


图 5.1 一个描述参考结构原理的例子（一个基本的本体）

Download at http://www.pin5i.com/

看看你的数据库，你的应用或许是一个在线商店、一个内部网文档管理系统、一个互联网的 mashup，又或是其他类型的 Web 应用。当你考虑数据及其组织方式时，你就能意识到分类系统对于应用的价值。从 5.3 节一直到第 6 章的大部分内容，我们会通过一些假想的 Web 应用来介绍分类的机制，阐明分类算法的应用，以及会面临的问题。但首先我们会对分类系统做一个概述，如果你想迅速跳到实践环节，可以忽略 5.2 节的内容。 169

5.2 分类器的概述

纵览所有分类系统的一个方法是关注它们所使用的参考结构。在这种视角的最顶层，我们可以把所有的分类系统划分为两大类——二分类与多分类。二分类系统，顾名思义，对问题做出是或否的回答，例如，这个数据点属于类别 X 吗？医疗诊断系统会回答某个病人是否患有癌症这样的问题，而移民分类系统会回答某个人是否是恐怖分子这样的问题。多分类系统把数据点分配到多个类别中的一个，比如把新闻文章归类为不同类别的新闻。

在多分类系统中，我们还可以基于以下两个标准进一步细分：多个类别是离散的还是连续的；多个类别是“平坦”的（只是标签的列表）还是有层次结构的。5.1 节提及的杜威分类体系与 ICD-10 就属于离散有限类别的多分类系统。分类的结果也可能是连续变量，比如把分类用于预测的场合。如果你把一只股票在周一、周二、周三与周四的价格作为输入，希望得到这只股票在周五的价格，你可以把这个问题转换成离散或连续的多分类问题。离散的版本会预测这只股票在周五的涨、跌，还是保持不变，连续的版本会提供该股票在周五价格的一个预测值。

要根据所使用的技术对分类系统进行归类，就不像上面的归类那么清晰与广为接受了。但还是有两种宽泛的类别在工业界得到有效的采用，第一种为统计算法，第二种为结构算法，如图 5.2 所示。

统计算法有三个主要分支。回归算法特别擅长做预测——预测连续变量的值。回归算法基于这样的假设：用一个特定的模型就足以拟合我们的数据，而这个模型通常是待处理变量的线性函数。另一类统计分类算法源于贝叶斯理论，我们在第 2 章曾简要地介绍过这个理论。第三类，贝叶斯网络是一种相当成功与时髦的统计方法，它把贝叶斯理论与概率网络结构组合起来，用于描述分类问题中多个变量间的

Download at http://www.pin5i.com/

依赖关系。

结构算法也有三个主要分支：基于规则的算法，包括 if-then 规则与决策树；基于距离的算法，可分为函数式方法与最近邻方法；神经网络（neural networks，NN）。神经网络其实是自成体系的，但这里我们依据确定性神经网络与那些已经研究得很透彻的高级统计算法（如高斯过程）本质上是等同的这一点，把它归入这一类中。在下面各子节中，我们会对各分类器的类别做简要说明。

170

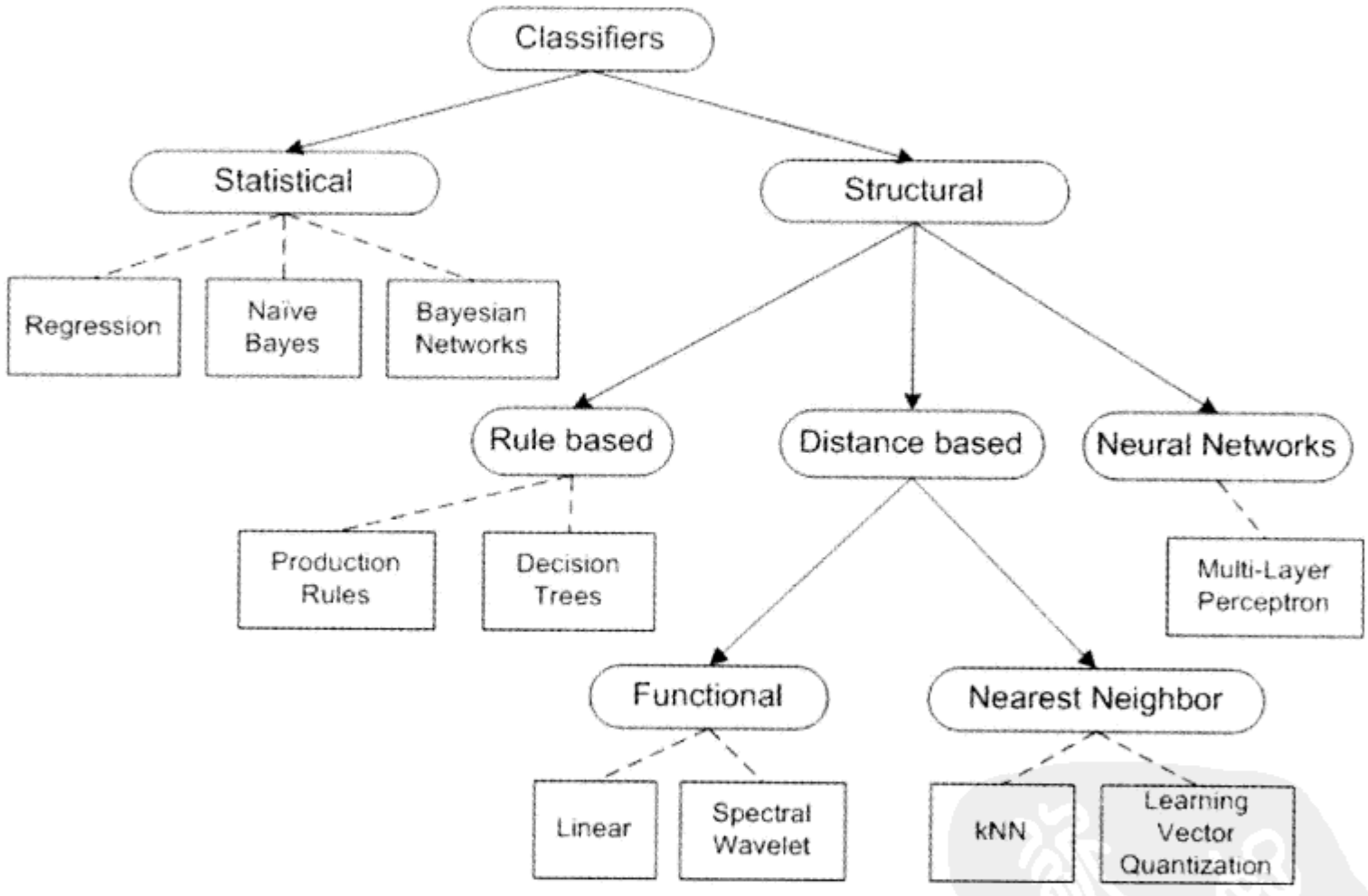


图 5.2 按设计组织的分类算法概览

5.2.1 结构分类算法

如图 5.2 所示，基于规则的结构算法由生成规则（if-then 子句）与决策树（DT）算法构成。生成规则可通过人类专家手工生成，或通过决策树推断得到。基于规则的算法通常被实现为前向链式生成系统（forward-chaining production systems）——一个很吓人的术语。这个类别里最好的算法是 Rete（参看 Russell 与 Norvig 的文章），rete 的拉丁语原义是“网络”，它是很多著名库如 CLIPS、Jess、Soar 的基础。

在本书中，我们会使用由 JBoss 项目提供的面向对象的 Rete 实现。这是 JBoss

Download at http://www.pin5i.com/

Rules 库，也被称为 Drools（这个规则引擎的项目原名）。这个项目已经很稳定，也有丰富的文档，它的基础代码非常适合于学习与开发。顺便说一下，我们很喜欢用 Drools 的 API 来做事情，就像我们很喜欢用 Lucene（第 2 章已涉及）的 API 做事情一样。这两个是实实在在的生产环境可用的项目。

决策树算法基于一个简单但强大的想法。你看过查尔斯·狄更斯的《圣诞颂》吗？在该书中，狄更斯描写了一个是与否的游戏，Scrooge 的侄子先想一个东西，其他人要指出那是什么，他们可以问他问题，但他只回答是或否。在许多民族都有这个游戏的不同版本——在西班牙语国家中，它被称为 veo veo，在孩子们中是相当流行的。跟这种家庭游戏类似，大多数 DT 算法的思想就是询问问题，并通过对问题回答得到的信息来消除尽可能多的候选。决策树算法有几个优点，比如简单易用与计算高效。它们的缺点是不能很好地处理连续变量，所以，我们不得不对数据做一个离散化——树的构造需要把连续值划分到有限个桶中。一般来说，决策树算法不具备很好的泛化性，所以，对于未经训练的数据，它的表现并不好。这个类别中常用的算法是 C5.0（UNIX 系统）或者 See5（微软 Windows 系统），在很多商业产品中都能看到它的身影，如 Clementine(http://www.spss.com/clementine/)和 RuleQuest (http://www.rulequest.com/)。

171

结构算法的第二个分支是基于距离的算法。在前面的章节里，我们介绍过相似性度量与广义上距离的概念，并已经广泛地使用。这些算法都很直观，但很容易被误用，其结果就是得到一个糟糕的分类结果，因为数据点的属性之间可能根本没有什么直接的关系。不同属性的度量有其独特之处，单一的一个相似性度量并不能一劳永逸地捕获它们之间的差异，谨慎地对数据做标准化，并对属性空间进行分析是成功应用距离算法的决定性因素。然而，在很多低维低复杂度的情况中，基于距离的算法会表现得很好，实现起来也相当简单。我们可以进一步把基于距离的算法划分为函数式算法与最近邻算法。

函数式分类器，如其名字所暗示的，用函数来对数据进行近似。这有点像回归，但我们还是通过它们在使用函数时基本原理上的不同对它们进行区分。对于回归，我们把函数作为概率分布的模型（参看 Dunham 的文章）；而对于函数式分类器，我们只对数据的数值近似感兴趣。实际中，其实很难（或许也是没有意义的）去区分线性回归与通过最小化平方误差得到的线性近似。

最近邻算法尝试着为每一个数据点寻找离它最近的类别。通过使用我们之前见

Download at http://www.pin5i.com/

过的广义距离公式，可以计算得到每个数据点与每个有效类别的距离，距离对象最近的那个类别被指定给该对象。这种算法最常用的可能就是 K 最近邻（K nearest neighbors，kNN）了。另外，对学习矢量量化（learning vector quantization, LVQ），人们也有过充分的研究与广泛的使用。

神经网络算法整体作为结构算法的一个子类别。这类算法需要有相当的数学背景才能把它们恰当地叙述出来，我们会尽量避免使用数学工具，纯粹从计算的角度去描述它们。这个家族的分类算法的主要思想在于通过有计算能力的节点构造人工神经网络，以此来模仿人脑中由神经元与连接神经元的突触构成的生理结构。

172

神经网络算法已经被证明在众多的问题中都表现良好，但它有两个主要的缺点：我们没有一个可以应用于普遍问题的网络设计的方法论；神经网络分类器得到的结果很难解释，它也许能得到极小的误差，但我们不知道为什么。这就是为什么我们把神经网络称为“黑盒”技术，以与决策树或基于规则的算法对特定数据点的分类结果极好解释做对比。

5.2.2 统计分类算法

回归算法的思想在于要找到公式与数据的最佳拟合，最常用的公式是输入值的线性方程（参看 Hastie、Tibshirani 与 Friedman 的文章）。回归算法经常被用于数据点本质上是数值变量的场合（例如对象的维数、人的重量或者空气的温度），但跟贝叶斯算法不一样的是，它们不擅长处理类别型数据（例如雇佣状态或信用等级描述等）。此外，如果使用线性模型拟合数据，就很难表现出此类算法的“统计性”，因为线性回归本质上与中学里求一条能拟合一组 x-y 点的直线的经典练习题没有什么区别。

当进入 logistic 回归时，事情变得越来越有趣，因为终于闻到了统计方法的味道。在这种情况中，模型（logistic 函数）的输出在 0 与 1 之间取值，这可以解释为类别的隶属程度，这在二值分类中表现良好（参看 Dunham 文章）。

大多数统计算法都使用了名为贝叶斯规则或贝叶斯定理的概率理论（参看 Papoulis 与 Pillai 的文章），我们在第 2 章讨论从用户点击中学习时提到过贝叶斯规则。这类统计分类算法最基本的共同点在于显式地假设了待解问题中各属性的独立性。贝叶斯算法最吸引人的地方在于尽管当独立性假设明显不满足的时候，它居然还能表现得很好。在 5.3 节中，我们会学习这类方法中最著名的算法——朴素贝叶斯

Download at http://www.pin5i.com/

分类算法。

贝叶斯网络是机器学习领域中相对较时髦的一种方法，它试图把贝叶斯理论的强大与结构方法（如决策树）的优势结合起来。朴素贝叶斯分类器及其家族成员可以表示简单的概率分布，但无法捕获数据的概率结构（如果有的话）。通过利用有向无环图（directed acyclic graphs，DAG）强大的表示能力，属性间的概率关系就可以生动地描画出来。我们不会在本书中更多地涉及贝叶斯网络，如果你对于这方面的主题有更大的兴趣，可以参阅 Richard E. Neapolitan 的 *Learning Bayesian Networks*。 173

5.2.3 分类器的生命周期

不管你为自己的应用选择了什么类型的分类器，它的生命周期都符合图 5.3 所示的一般过程，即一个分类器的生命周期有三个阶段：训练、测试以及生产。

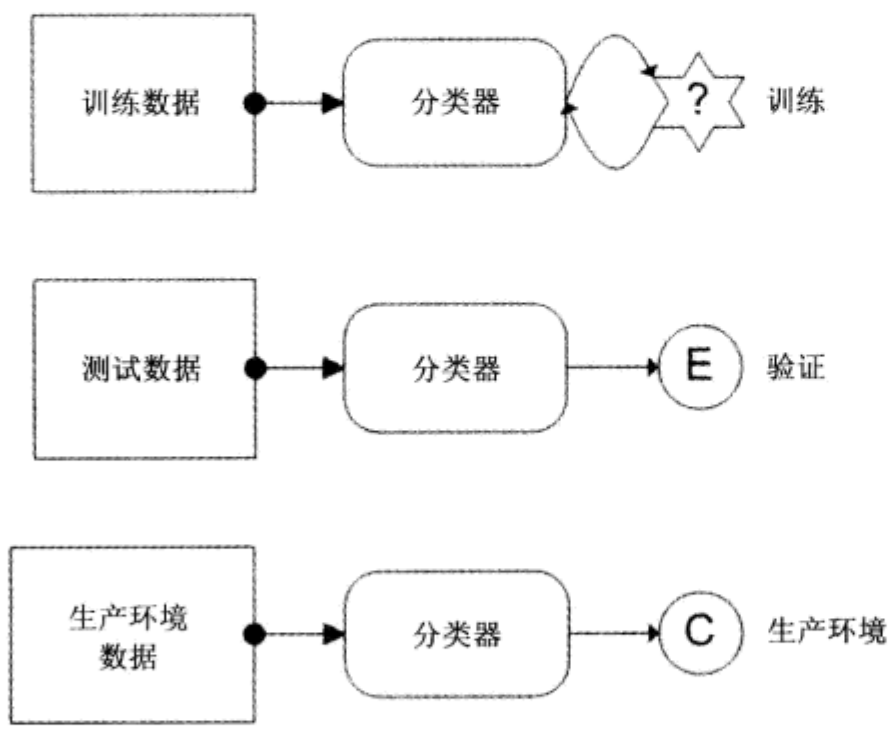


图 5.3 分类器的生命周期：训练、测试、在生产环境中使用

在训练阶段，我们给分类器提供已经分配了合适类别的数据样本，每一个分类器都包括若干参数，在正式使用分类器前必须先确定它们的取值。训练阶段的目的就在于确定这些参数的取值，图 5.3 中，我们在星形图中标记了一个问号就是为了指示这一步主要目的是确定参数。在验证阶段，我们希望对分类器进行验证，并且在把它架设于生产环境之前，要先确认我们得到的结果具有一定程度的可靠性。我们在圆圈里标记字母 E 指示这一步的主要目的是确定分类误差，但质量标准可能并

Download at http://www.pin5i.com/

且也应该通过更多的指标来度量（参看 5.6 节关于分类可靠性与成本的讨论）。验证阶段的数据（测试数据）必须区别于训练阶段的数据（训练数据）。

在把分类器转移到生产阶段之前，训练阶段与验证阶段可能会反复迭代很多次，因为可能有某些在分类器设计阶段引入的参数并不能通过一次的训练过程便调节完成。这个要点意味着我们可以编写一个软件，把分类器及其配置参数都隐藏在内部，便于对大量分类器设计的自动化测试，甚至本质上不同的分类器，比如朴素贝叶斯、神经网络以及决策树，都可以参与到测试中。我们可以通过验证阶段的质量指标选取最好的分类器，也可以把所有的分类器组合为元分类器。这种方法在工业界正获得长足的发展，在大范围的应用中都一致地得到比较好的结果。我们会在第 6 章讨论分类器的组合。

174

在生产环境阶段，我们会把分类器架设在在线系统中，并产生实时的分类结果。一般来说，在生产阶段，分类器的参数不作改变，但为了增强分类器的结果，也可以在生产环境中嵌入一个短期的训练阶段，根据用户的反馈循环更新。随着我们从生产环境中获得更多的数据，或者随着我们对分类器设计上的改良，训练、验证、生产这三个阶段可以反复迭代。

我们已经纵览了分类算法的宏大图景，在文献里，你能找到很多不同的关于分类算法的综述（比如 Holmström 等人的文章）。在 5.3 节，我们会介绍最著名的统计分类算法之一：朴素贝叶斯分类器，具体来说是分类算法在正常邮件归类与垃圾邮件过滤中的应用。

5.3 邮件的自动归类与垃圾邮件过滤

本节我们希望能达到两个目的：其一是粗略范畴的工作——区分正常邮件与垃圾邮件，这是一个二分类的例子；其二是精细范畴的工作——分类邮件，我们希望精练分类结果，使其能把正常邮件归类为商业、国际、美国与体育中的一种，这是一个多分类的例子。

电子邮件无须过多地介绍，它是首批伴随着互联网诞生的应用之一，或许也是目前最常用的应用。对大部分用户来说，所有的邮件都直接进入收件箱。如果你可以定义一些文件夹，然后你的邮件会（自动地）被分配到合适的文件夹中，这可不就是一种更为友好的方式？于是你就可以建立一个标识为“地狱”的文件夹，然后

Download at http://www.pin5i.com/

你的垃圾邮件（也被称为电子广告邮件）都会被直接踢进去！

也许你的邮件客户端已经做了这样的事情，如今大多数的邮件客户端至少都实现了某种形式的规则分类。也就是这种情况，导致大多数的这些客户端并不擅长学习，因为它们并不懂得从之前“看过”的邮件中进行归纳学习，对邮件的分类仍需手工进行。Web 邮件客户端在这方面提供了空前的机遇，因为一个具有归纳泛化能力的算法可以快速地覆盖极大范围的邮件信息，比如新型的垃圾邮件。

我们在这里讨论的算法可以应用于任意的文档集，你可以把它们用在可由用户上传 Word 或 PDF 文档的应用中，然后把上传的文档归类到一组由用户提供的类别中。

在本节的例子里，邮件集的生成与第 2 章网页的生成方法一样。对于第 2 章中的每一个网页，我们都创建一个相对应的邮件。如果你读过那一章，应该对这些内 175
容很熟悉了，如果你还没读过，让我们告诉你邮件包括以下内容（内容的选择是随机的，时间是 2006 年 11 月）：

- 七封与商业新闻有关的邮件，其中三封是关于 Google 扩张到报纸广告业的内容，另有三封主要是讨论 NVidia 的股票，最后一封宽泛地谈及了股票价格与指数变化。
- 三封关于 Lance Armstrong 尝试参加纽约马拉松比赛的邮件。
- 四封关于美国政治（具体地说是国会选举）的邮件。
- 五封关于国际新闻的邮件，其中四封为 Ortega 赢得尼加拉瓜大选的邮件，一封讨论全球变暖的邮件。
- 四封垃圾邮件。

邮件分类在很多方面都很有趣，其中一个很奇怪的地方在于总会有“某个人”在试着攻击你的分类器。实际上，垃圾邮件制造者可能也同时在使用这里描述的技术来攻击你的分类体系，所以时刻警惕着竞争！这里我们有意使用“体系”这个词，提醒你最好不要单一地使用一个分类器来做邮件过滤或邮件组织。下面的这个列表有助于你对分类器做补充：

- 邮件头信息测试。
- 自动邮件地址白/黑名单。
- 手动邮件地址白/黑名单。
- 垃圾邮件协同鉴定数据库。

Download at http://www.pin5i.com/

- 实时黑名单（Real-time blackhole lists，RBLs）。
- 字符集与区域识别。

我们的焦点在于邮件分类系统的智能特性，所以不会覆盖我们提到的其他方面的技术。先给出一个很有用的模块，参看 Apache 项目 Spam Assassin (http://spamassassin.apache.org/)。我们会给出两个有助于你创建这类系统的分类方法，第一个分类方法是基于朴素贝叶斯分类器的，我们在第 2 章有过介绍，第二个分类方法是基于规则的，我们会使用 Drools 规则引擎。

5.3.1 朴素贝叶斯分类

在本节，我们会使用一个封装在 EmailClassifier 类中的统计分类器。如我们之前所提及的，它通过扩展 NaiveBayes 类（朴素贝叶斯算法的一般化实现）而引入了朴素贝叶斯算法。通常，分类器对分类对象是无知的，它们只关注概念、实例与属性。一个分类器的职责就是把概念分配给一个实例，这就是它的全部工作。为了清楚应该给某个特定的实例分配什么概念，分类器需要理解一个训练集——已经分配了概念标签的实例集。根据载入的实例，分类器开始“训练”自己。换句话说，它依据训练集中概念分配的信息来“学习”概念与实例间的映射关系。不同的分类器有其自身的训练方式。本章我们会交叉地使用概念、分类、类别这些可互换的术语。

176

邮件抽样与分类器训练

让我们以邮件载入和分类器训练作为开始，代码清单 5.1 展示的 BeanShell 脚本就可用于这两个目的。

代码清单 5.1 载入邮件训练数据集并训练朴素贝叶斯分类器

```
EmailDataset trainEmailDS = EmailData.createTrainingDataset();❶
trainEmailDS.printEmail("biz-04.html");
trainEmailDS.printEmail("spam-biz-03.html");
EmailClassifier emailFilter = new EmailClassifier(trainEmailDS, 10);❸
emailFilter.train();❹
```

❶ EmailData 类负责载入曾经在第 2 章使用过的 HTML 文件，并把它们转换成 Email 类的实例，Email 类只是一个简单的封装了 from、to、subject 与 textBody 等属性的类。createTrainingDataset()方法载入有待用于分类器训练的文档列表，这个列表通过一个

Download at http://www.pin5i.com/

二维的 String 数组 TRAINING_DATA 提供，类似的测试数据集由另一个二维 String 数组 TEST_DATA 提供。你可以改变这些列表的内容，并观察其对分类结果的影响。一个可靠的评估应该对训练和测试使用不同的文件集。

❷ 这一步打印出两封邮件的内容——一封正常邮件和一封垃圾邮件，只是在处理之前要确认数据已经被正确地载入了，同时也考察将要进行操作的内容的类别。

❸ EmailClassifier 的实例化通过传递 EmailDataset 的引用与词数这两个参数完成，这里词数指在分析邮件时所考虑的有效单词的数量。对于每封邮件，我们分析其内容并保留前 10 个（在这个例子里）最频繁的单词。

❹ 训练分类器。这是一个健全性的检查，以确保我们拥有一个可以用于训练的实例：它设置将用于训练分类器的属性集，调用父类 NaiveBayes 的 train()方法，并为未知输入的属性值设置一个临时的概率阈值。

邮件分类器实战

一旦训练完分类器，我们就可以测试它了。代码清单 5.2 是代码清单 5.1 的延续，所以你需要在同一个 Shell 中执行这些脚本，你会看到在这个层面上使用分类器是多么的简单，只需要两行代码。

代码清单 5.2 使用朴素贝叶斯分类器检测垃圾邮件 177

```
EmailDataset testEmailDS = EmailData.createTestDataset();←从测试数据集中载入电子邮件
email = testEmailDS.findEmailById("biz-01.html");←通过文件名获取电子邮件
emailFilter.classify(email);←分类合法邮件
email = testEmailDS.findEmailById("sport-01.html");
emailFilter.classify(email);
email = testEmailDS.findEmailById("usa-01.html");
emailFilter.classify(email);
email = testEmailDS.findEmailById("world-01.html");
emailFilter.classify(email);
email = testEmailDS.findEmailById("spam-biz-01.html");
emailFilter.classify(email);
```

结果显示如图 5.4 所示，注意看所有的邮件都被正确分类了，这给你试验不同的设置时提供了一个基准线，同时，当你扩大或减少训练集时，也可以把分类器精确度的改变与之进行比较。

Download at http://www.pin5i.com/


```
*** Classifying instance: biz-01.html
P(NOT SPAM|biz-01.html) = 0.944444444444445
P(SPAM|biz-01.html) = 0.055555555555556

Classified biz-01.html as NOT SPAM

*** Classifying instance: sport-01.html
P(NOT SPAM|sport-01.html) = 0.894736842105263
P(SPAM|sport-01.html) = 0.105263157894737

Classified sport-01.html as NOT SPAM

*** Classifying instance: usa-01.html
P(NOT SPAM|usa-01.html) = 0.882352941176471
P(SPAM|usa-01.html) = 0.117647058823529

Classified usa-01.html as NOT SPAM

*** Classifying instance: world-01.html
P(NOT SPAM|world-01.html) = 0.962264150943396
P(SPAM|world-01.html) = 0.037735849056604

Classified world-01.html as NOT SPAM

*** Classifying instance: spam-biz-01.html
P(NOT SPAM|spam-biz-01.html) = 0.468750000000000
P(SPAM|spam-biz-01.html) = 0.531250000000000

Classified spam-biz-01.html as SPAM
```

图 5.4 朴素贝叶斯分类器对垃圾邮件过滤的结果（二分类）

要注意到这样一个事实：如果所有的未知邮件都与训练集里的类似，那么邮件分类要获得成功并不是多难的事情。一般来说，如果训练集与测试集非常相似，要获得高精确度是很好办的，这是因为过拟合的存在，欲知更多的在特化与泛化之间权衡的细节，可参看 To Do 事项的第 1 点。

178

这时候，通过修改一些参数设置来重复实验也许有助于我们对问题的认识，如改变频率词的数量（代码清单 5.1 第 3 步），重新训练分类器，观察这个修改对分类结果的影响。你可以马上通过执行 EmailClassifier 的 sample()方法来得到分类结果。

我们曾提到过，分类器的工作就是把一个概念分配给一个实例。对于 Email Classifier 来说，在邮件过滤（二分类）情景中，概念就是垃圾与非垃圾；在邮件归类（多分类）情景中，概念就是类别名。邮件的实例被封装在 EmailInstance 类中，它继承自 BaseInstance 类。这个例子展示的是一般类的特殊情况，以适应于邮件这个特定的应用领域。

Download at http://www.pin5i.com/

EmailClassifier 通过 EmailDataset 实例的 getTrainingSet 方法来获得它的训练集，载入这些邮件实例之后，分类器对其自身进行训练，以学习怎么基于训练集中的分配信息把概念映射到实例。EmailClassifier 在训练中并不会使用所有的邮件信息，它只使用一个属性，该属性的取值是在 EmailInstance 构造的时候生成的，如代码清单 5.3 所示。

代码清单 5.3 创建 EmailInstance

```
public EmailInstance(String emailCategory, Email email, int topNTerms) {
  super();
  this.id = email.getId();
  this.setConcept(new BaseConcept(emailCategory));
  String text = email.getSubject()+" "+email.getTextBody();
  Content content = new Content(email.getId(), text, topNTerms);
  Map<String, Integer> tfMap = content.getTFMap();
  attributes = new StringAttribute[1];
  String attrName = "Email_Text_Attribute";
  String attrValue = "";
  for(Map.Entry<String, Integer> tfEntry : tfMap.entrySet()) {
    attrValue = attrValue + " " + tfEntry.getKey();
  }
  attributes[0] = new StringAttribute(attrName, attrValue);
}
```

首先，把邮件的标题与正文拼接起来，然后使用 Content 类（我们在第 3 章见过这个类）来分析刚拼接的文本，并得到前 *N* 个最频繁词的列表。文本分析是利用继承自 Lucene StandardAnalyzer 类的自定义分析器实现的，并使用 PorterStemFilter 类来分词。所有 Lucene 的类都可以在包 org.apache.lucene.analysis 中找到。

如你所见，实例唯一的属性（Email_Text_Attribute）即是拼接文本的前 *N* 个频繁词，这当然只是一个简化的模型假设。尽管如此，这种方法在很多案例里都提供了很好的结果。请记住一点，当你为实际的应用设计（或选择）一个智能算法时，永远都应该从最简单可行的设计开始。如果你愿意去类比，这就等同于代码优化中的格言：“过早的优化是魔鬼”。

179

简单的解决方案也许不会是你工作的终点，但是它让你从一开始就避免了复杂的情况，更好地理解你所拥有数据的性质以及所面临问题的难点。当然还有很多其他的选择，比如选择两个属性，一个针对邮件标题，另一个针对邮件正文，你还可

Download at http://www.pin5i.com/

以把 from（来信地址）这个属性包含进来。如果你的邮件有时间戳，你还可以包含这封邮件是正常上班时间发出还是深夜发出这个因素。在本章的 To Do 节，我们会邀请你探索这些及其他更多的可能性（随意发挥），比较它们的结果，以及探讨当为分类器训练考虑更多的邮件信息时，它的复杂度又会如何。

朴素贝叶斯分类器的细节

现在，是时候细究一下朴素贝叶斯分类算法的实现了。你在代码清单 5.4 中看到的是 NaiveBayes 类的实现，这里已经刨除了直观的构造函数、文档注释、部分记录输出与一对无关大局的 getter 函数的实现。除此之外，所有的代码都已经在这里，有史以来最鲁棒、最成功、使用最广泛的分类算法之一，仅仅两页纸而已！

回忆一下，分类器会从训练实例中学习到实例与类别间的关联，然后给一个指定的实例提供与之相应的类别。很自然地，Classifier 的接口需要每一个分类器实现 boolean train()方法与 Concept classify(Instance instance)方法。当然，每个分类器都有自己的方式来实现这些方法，下面让我们看看 NaiveBayes 是怎么做的。

代码清单 5.4 NaiveBayes：通用的贝叶斯分类器

```
public class NaiveBayes implements Classifier {
  private String name;
  protected TrainingSet tSet;
  protected Map<Concept,Double> conceptPriors;   ❶
  protected Map<Concept, Map<Attribute, AttributeValue>> p;   ❷
  protected ArrayList<String> attributeList;   ❸
  public boolean train() {   ❹
    boolean hasTrained = false;
    if ( attributeList == null || attributeList.size() == 0) {
      System.out.print("Can't train the classifier
➥ without attributes for training!");
      System.out.print("Use the method -->
➥ trainOnAttribute(Attribute a)");
    } else {
      calculateConceptPriors();
      calculateConditionalProbabilities();
      hasTrained = true;
    }
    return hasTrained;
```

180

Download at http://www.pin5i.com/


```
}
public void trainOnAttribute(String aName) {
  if (attributeList ==null) {
    attributeList = new ArrayList<String>();
  }
  attributeList.add(aName);
}
private void calculateConceptPriors() {❺
  for (Concept c : tSet.getConceptSet()) {
    int totalConceptCount=0;
    for (Instance i : tSet.getInstances().values()) {
      if (i.getConcept().equals(c)) {
        totalConceptCount++;
      }
    }
    conceptPriors.put(c, new Double(totalConceptCount));
  }
}
protected void calculateConditionalProbabilities() {❻
  p = new HashMap<Concept, Map<Attribute, AttributeValue>>();
  for (Instance i : tSet.getInstances().values()) {
    for (Attribute a: i.getAtrributes()) {
      if (a != null && attributeList.contains(a.getName())) {
        if ( p.get(i.getConcept())== null ) {
          p.put(i.getConcept(), new HashMap<Attribute,
➥ AttributeValue>());
        }
        Map<Attribute, AttributeValue> aMap = p.get(i.getConcept());
        AttributeValue aV = aMap.get(a);
        if ( aV == null ) {
          aV = new AttributeValue(a.getValue());
          aMap.put(a, aV);
        } else {
          aV.count();
        }
      }
    }
  }
```

Download at http://www.pin5i.com/


```
}
public double getProbability(Instance i, Concept c) {❼
  double cP=1;
  for (Attribute a : i.getAtrributes()) {
    if ( a != null && attributeList.contains(a.getName()) ) {
      Map<Attribute, AttributeValue> aMap = p.get(c);
      AttributeValue aV = aMap.get(a);
      if ( aV == null) {
        cP *= ((double) 1 / (tSet.getSize()+1));
      } else {
        cP *= (double)(aV.getCount()/conceptPriors.get(c));
      }
    }
  }
  return (cP == 1) ? (double)1/tSet.getNumberOfConcepts() : cP;
}
public double getProbability(Concept c, Instance i) {❽
    double cP=0;
    if (tSet.getConceptSet().contains(c)) {
      cP = (getProbability(i,c)*getProbability(c))/getProbability(i);
    } else {
      cP = 1/(tSet.getNumberOfConcepts()+1.0);
    }
    return cP;
  }
  public double getProbability(Instance i) {
    double cP=0;
    for (Concept c : getTset().getConceptSet()) {
      cP += getProbability(i,c)*getProbability(c);
    }
    return (cP == 0) ? (double)1/tSet.getSize() : cP;
  }
  public double getProbability(Concept c) {❾
      Double trInstanceCount = conceptPriors.get(c);
      if( trInstanceCount == null ) {
        trInstanceCount = 0.0;
      }
      return trInstanceCount/tSet.getSize();
```

Download at http://www.pin5i.com/


```
  }
  public Concept classify(Instance instance) {⑩
    Concept bestConcept = null;
    double bestP = 0.0;
    for (Concept c : tSet.getConceptSet()) {
      double p = getProbability(c, instance);
      if( p >= bestP ) {
        bestConcept = c;
        bestP = p;
      }
    }
    return bestConcept;
  }
}
```

182

这是一个很长的代码清单，所以，在进入它的细节之前，让我们简单回顾一下在第 2 章所接触过的内容。朴素贝叶斯算法需要计算在给定 Y 后 X 的条件概率，也就是说，这个概率告诉我们当观察到实例 Y 时，有多大的可能性会观察到概率 X。具体来说，这类分类器有以下的输入：

- 通常情况下，观察到概念 X 的概率也被称为先验概率，表示为 $p(X)$。
- 当随机从概念 X 中抽出一个实例来，观察到实例 Y 的概率，这被称为似然性，表示为 $p(Y|X)$。
- 通常情况下观察到实例 Y 的概率，这被称为证据，表示为 $p(Y)$。

分类器的输出即观察到的实例 Y 属于概念 X 的估算概率值，这也被称为后验概率，表示为 $p(X|Y)$。计算是按照如下公式进行的（贝叶斯理论）：

$$p(X|Y)=p(X|Y)p(X)/p(Y)$$

在此之前，我们都在有意避免直接使用数学公式，但这个公式是如此的强大，以致我们不可能绕过它。虽然它的形式很简单，但它是大量分类器的基础，从基于贝叶斯算法的实现，到基于高斯过程和贝叶斯信念网络的实现（参看 McKay 的文章）。如果你只想记一个公式，那就是它了！

就我们所关心的分类本身而言，证据 $p(Y)$的计算其实是没有必要的，因为它的值对于所有的类别都是一样的。分类器对所有的类别计算后验概率 $p(X|Y)$，并选择给出最大后验概率的类别，至于是否除以 $p(Y)$，对整个结果的排序并不会有影响。

Download at http://www.pin5i.com/

因此，为了节约计算成本，在实现中可以忽略除以 $p(Y)$这一步。

现在，让我们逐一检查代码清单 5.4 的重点。首先会给 NaiveBayes 分类器设定一个名字，如果只使用一个分类器，这一步是多余的。但你会在第 6 章看到，我们经常希望创建一组分类器，并把它们组合在一起以改进结果。所以，让每一个分类器保存一个标识对后续的工作是有用的。当然，每个分类器都需要一个训练集，分类器的名字及训练集都有意地在其构造阶段就设置好。NaiveBayes 分类器的实例一旦生成，就不可以再改变它的训练集，但总可以获得它的引用并给它增加新的实例。

183

❶ conceptPriors 映射用于存储训练集中每个概念的数量。它本可以直接存储先验概率值，而不单是数量。但因为我们还需要重用这些数量，所以从计算效率上考虑，我们存储的是数量，只要对这些数量进行一次简单的除法，即可得到先验概率。

❷ 变量 p 存储后验概率——当我们观察到实例 Y 时再观察到概念 X 的概率。具体到用户点击的例子里，即当用户 A 提交查询 Q 时，他希望看到 X 链接的概率。

❸ 这个列表指示分类器应该使用什么属性进行训练。训练集实例也许会有很多属性，但可能只有一部分是有用的，所以我们会记录应该选用哪个属性。trainOnAttribute(String)方法可用于构造这个属性列表。

❹ train()方法负责分类器的训练。在通过简单检查确认有至少一个属性可用于训练之后，该方法开始计算概念的先验概率与贝叶斯公式中的条件概率。如果一切运行良好，它会返回 true，否则返回 false。

❺ 这是训练的第一步，计算先验概率 $p(X)$。通过遍历训练集中的所有实例，可以得到每个概念出现的频次。在这个实现中，我们实际记录的是频次值，概念的先验概率还需要用这个频次除以训练集中实例的总数而求得。

❻ 这是训练的第二步，计算某特定的属性值在一个概念中出现的频次。这个量会用于条件概率 $p(Y|X)$的计算，在 getProbability(Instance I, Concept c)方法中能看到这部分计算。对于训练属性列表中的每一个属性，我们通过训练集中所有的实例数据来计算它的各个取值在不同概念中出现的频次。

❼ 这一步计算条件概率 $p(Y|X)$，在这个方法中，你能找到使用“朴素”这个术语的原因。我们要求的是对于一个给定的概念，求某一特定实例出现的概率，而每个实例是由它的属性的取值来唯一确定的。所以，实例的条件概率本质上等于所有的属性值条件概率的联合概率。每个属性值的条件概率通过(aV.getCount()/concept Priors. get(c))求得。在此前的实现中，我们假设所有的属性都是统计独立的，所以它

Download at http://www.pin5i.com/

们的联合概率就简单地等于每个属性值的个体概率的乘积，这就是“朴素”的由来。一般来说，如果属性间并非统计独立，它们的联合概率是不等于这个乘积的。

我们之所以给朴素这个词加上引号，是因为即使在这个独立性假设明显不成立的情况下，朴素贝叶斯算法仍然非常鲁棒并具有广泛的适用性。实际上，朴素贝叶斯算法表现出最佳能力时恰恰是在与其假设相反的情形下——当属性间具有完全确定性依赖的时候（参看 Rish 的文章）。

对于属性值未出现过的情况，我们给它分配一个任意的条件概率值：取实例个数加 1 的倒数。这是一个随意的近似，你可以取它的两倍或三倍值，又或者用其他完全不同的方式来计算这些缺失值的概率。不要低估这些估计对分类结果的影响，特别是训练集并不大的时候要尤为注意。你认为当训练集比较小时，这些值应该怎么取？ 184

❽ 这个方法计算某个类别的后验概率，这也是贝叶斯公式的输出。对于一个给定的实例，它的分类结果就基于对不同的类别反复调用这个方法的结果。这里一个可以进行优化的地方是避免对 getProbability(i)方法的调用及后续的除法操作，因为正如我们之前所提到的，证据（贝叶斯公式中的 $p(Y)$项）的计算对于分类并不是必需的。getProbability(Instance)方法本身是可以忽略的，这里加上它只是为了叙述上的完整。

在 getProbability(Concept, Instance)方法中，我们会判断作为参数传入的概念是否存在。如果是在一个确定的概念集上使用 NaiveBayes 类作为分类，这一步是不必要的。但你是否还记得，我们在第 2 章讨论从用户点击进行学习时，使用过同样的这个类，而那个时候是有可能会传进一个并不包含在训练集中的概念的。

❾ 这个方法计算类别 X 的先验概率 $p(X)$，取值为该类的实例数量与训练集总实例数量之比。

❿ classify(Instance)方法返回发生概率最高的类别，作为对给定实例的分类结果。在多分类情况中，你可以用一个数组存储每个类别的发生概率值，对数组进行排序并返回最优的三个（或五个）类别。在实际的系统中，这可能会是一个更好的方案，因为概率值相互之间可能是很接近的，而且应用也可以给用户显示一些候选，而不是直接分配一个了事。

一个通用的邮件分类器

现在，让我们近距离看看 EmailClassifier 类本身。代码清单 5.5 显示了这个类的

Download at http://www.pin5i.com/

代码，剔除了实例变量定义、构造方法与分类方法这些无关大局的部分。因为我们刚刚才剖析过 NaiveBayes 分类器的细节，所以我们会主要关注其中被重载的方法。

代码清单 5.5 基于通用 NaiveBayes 类的邮件分类器

```
public class EmailClassifier extends NaiveBayes {
  public boolean train() {❶
      if( emailDataset.getSize() == 0) {
          System.out.println("Can't train classifier -
➥ training dataset is empty.");
          return false;
      }
      for(String attrName : getTset().getAttributeNameSet()) {
          trainOnAttribute(attrName);
      }
      super.train();
      return true;
  }
  protected void calculateConditionalProbabilities() {
      p = new HashMap<Concept, Map<Attribute, AttributeValue>>();
      for (Instance i : tSet.getInstances().values()) {
          Attribute a = i.getAtrributes()[0];❷
          Map<Attribute, AttributeValue> aMap = p.get(i.getConcept());
          if ( aMap == null ) {
             aMap = new HashMap<Attribute, AttributeValue>();
             p.put(i.getConcept(), aMap);
          }
          AttributeValue bestAttributeValue =
➥ findBestAttributeValue(aMap, a)❸
          if (bestAttributeValue != null ) {
             bestAttributeValue.count();
          } else {
             AttributeValue aV = new AttributeValue(a.getValue());
             aMap.put(a, aV);
          }
      }
  }
public double getProbability(Instance i, Concept c) {
    double cP=1;
```

185

Download at http://www.pin5i.com/


```
    for (Attribute a : i.getAtrributes()) {
      if ( a != null && attributeList.contains(a.getName()) ) {
        Map<Attribute, AttributeValue> aMap = p.get(c);
        Attribute bestAttributeValue = findBestAttributeValue (aMap, a);
        if (bestAttributeValue == null) {
          cP *= ((double) 1 / (tSet.getSize()+1));❹
        } else {
          cP *= (double)(bestAttributeValue.getCount()/conceptPriors.get(c));
        }
      }
    }
    return (cP == 1) ? (double)1/tSet.getNumberOfConcepts() : cP;
  }
  private Attribute findBestAttributeValue(Map<Attribute,
➥ AttributeValue> aMap, Attribute a) {
    JaccardCoefficient jaccardCoeff = new JaccardCoefficient();❺
    String aValue = (String)a.getValue();
    String[] aTerms = aValue.split(" ");
    Attribute bestMatch = null;
    double bestSim = 0.0;
    for(Attribute attr : aMap.keySet()) {
      String attrValue = (String)attr.getValue();
      String[] attrTerms = attrValue.split(" ");
      double sim = jaccardCoeff.similarity(aTerms, attrTerms);
      if( sim > jaccardThreshold && sim > bestSim) {❻
        bestSim = sim;
        bestMatch = attr;
      }
    }
    return bestMatch;
  }
}
```

186

❶ 这个方法的目的在于确认训练数据已经被装载进来，设置合适的训练属性，然后调用 NaiveBayes 类的 train()方法开始实际的分类器训练。

❷ 这个语句只在这个特定的实现中才有意义，一般来说，需要训练的属性并不止一个。章末的 To Do 事项中有一项就是要求你引入更多的属性来继续探讨邮件分

Download at http://www.pin5i.com/

类的案例。如果面临的是那样的情况，你必须重新审视一下这部分的实现。

❸ 这一步的必要性源于我们对邮件中纯文本的表示方式，当你处理文本时，可以采用这种通用的技术手段，在这个实现中，我们唯一使用的属性来源于邮件标题与正文的连接字符串，你应该还记得这是通过我们自定义的分析器来处理的，这样可以降低噪声并提取尽可能多的信息。如果我们考虑的是属性值间字符串的严格匹配，那么样本数据集里的每个邮件的属性值都是不一样的。为避免这种毫无意义的情况，我们依据 findBestAttributeValue 方法来对两个属性值进行匹配，如果它们满足一定的条件，则认为它们是等价的。

❹ 当遇到之前没有遇到过的属性值时，对它的估计跟 NaiveBayes 类是一样的，给它分配的条件概率值等于训练集中实例数量加一的倒数。不要低估这个近似对分类结果的影响，特别是当训练集不算大的时候。要记得这是某个特殊属性值的条件概率，在某些情况中，这个值可以由领域专家通过他们的经验来给出一个估计值，这个值可以比基于训练集规模的估计更大或更小。你也可以把这个值的估计替换为一个小常数（比如 10^{-4} 或 10^{-5} 这样典型的小常数就能帮到你），然后观察它对分类结果的影响。

❺ 你也许还能记起第 3 章（To Do 节）或第 4 章（用于 ROCK 的实现）关于雅可比系数的内容，它是基于两个集合交集大小与并集大小之比的相似性度量。在这个案例里，集合是把文本属性值切割后单独的单词集。你也可以使用迄今为止我们碰到过的任意一种相似性度量方式，比如你可以用 CosineSimilarity 类代替 Jaccard Coefficient，并比较它们在分类结果上的差异。

187

❻ jaccardThreshold 是一个实例变量，有其对应的 get 与 set 方法。它的默认值被设置为 0.25，但你可以在运行时把它改成任何你想要的值。比如，要在 BeanShell 环境中把它设为 0.3，只需执行命令：emailFilter.setJaccardCoefficient(0.3)。这个变量规定了如果我们要让两个属性值等价，则它们的相似度必须大于这个阈值。

上述内容已经涵盖了整个概率邮件分类器的实现，不过它仅仅展示了这个分类器在过滤垃圾邮件中的能力，所以，下面还要继续看看它在一般（多分类）情况下的应用。代码清单 5.6 显示了所需的步骤，唯一的差别在于对 EmailDataset 类的 setBinary(false)方法的调用。因为不同的类别（或邮件类别）是在数据集构造时分配的，而分类器不会关心二分类与多分类问题之间的差异。

Download at http://www.pin5i.com/

代码清单 5.6 载入邮件训练集并分类邮件

```
EmailDataset trainEmailDS = EmailData.createTrainingDataset();
trainEmailDS.setBinary(false);←使用所有的电子邮件类别
EmailClassifier emailFilter = new EmailClassifier(trainEmailDS, 10);
emailFilter.train();
emailFilter.sample();←通过分类一部分电子邮件来做测试
```

结果显示如图 5.5 所示，你可以看到只有一个邮件（usa-01）被错分了。你可以把概率值解释为邮件属于某个特定类别的置信性度量，如果置信度低于某一水平线（比如 0.7），系统会选出头三个或五个类别，并把它们作为候选展现给用户。这种用户反馈迭代（human-in-the-loop）的工作流设计方式在智能应用中是很常见的，实际中对于持续性地改进分类器性能也是需要的。

正如我们所提及的，一个实际的邮件过滤器所需的不仅限于概率分类器，下一节会带来一个规则引擎的实现，它是概率分类器一个很好的补充。一个好的垃圾邮件过滤器的很多特性都依赖规则，比如白名单、黑名单、协同垃圾邮件鉴定数据库等。

总之，NaiveBayes 分类器可以帮助你把垃圾邮件过滤出来，还能帮助你把邮件分成几个你选定的类别。当然，我们说到的关于电子邮件的一切，同样可应用于所有你可以获得文本形式表示的文档——Microsoft Word 文档、XML 文档、网站的 HTML 文档、PDF 文档等。

5.3.2 基于规则的分类

188

本节我们会体验另一种不同的分类方法：基于规则的分类。那么什么是规则呢？它们跟贝叶斯分类器有什么不同？要回答这两个问题，且让我们纵览编程范式以作说明。目前有很多正在使用的编程范式，一个典型的 Java/J2EE 应用具有一部分指令式编程、面向对象编程，以及面向方面编程的特性。特别地，指令式编程意味着你需要告诉计算机做什么以及怎么做。这是我们平常编写软件时占有统治地位的范式，但还有另一个名为声明式编程的编程范式，它着眼于要做什么，而把“怎么做”丢给运行时引擎来处理。

基于规则的推理就是声明式编程的一个例子，一个基于规则的系统由事实、规则引擎和（当然还有）规则所构成。事实仅仅只是关于世界的数据，规则是一组条件语句，它告诉我们当数据满足特定条件时应该做什么。换言之，它等同于 if-then 的编程子句；规则引擎负责根据事实来执行规则。相对于概率分类器来说，

Download at http://www.pin5i.com/

一个规则引擎在它获取与表示知识的方式上有着显著的不同。对于概率分类器，比如我们上一节了解过的朴素贝叶斯分类器，知识被表示为概念的先验概率及事件的条件概率，它们都可以从训练集中获得。对这些知识（概率）的积累并没有任何人工干预，给定一个任意的、结构良好的训练集，分类器会自动从中提取分类任务所需的信息。而对于一个基于规则的分类器，它以规则的形式来获取知识，所以规则就是系统的知识。这就会产生一个问题，我们是怎么得到这些规则的？答案是，这些规则是通过手工或半手工（提供方便的工作流程，由专家来总结规则）的方式录入系统中的。

```
*** Classifying instance: biz-01.html
P(WORLD|biz-01.html) = 0.085106382978723
P(BIZ|biz-01.html) = 0.765957446808511
P(USA|biz-01.html) = 0.063829787234043
P(SPAM|biz-01.html) = 0.042553191489362
P(SPORT|biz-01.html) = 0.042553191489362
Classified biz-01.html as BIZ

*** Classifying instance: sport-01.html
P(WORLD|sport-01.html) = 0.121212121212121
P(BIZ|sport-01.html) = 0.181818181818182
P(USA|sport-01.html) = 0.090909090909091
P(SPAM|sport-01.html) = 0.060606060606061
P(SPORT|sport-01.html) = 0.545454545454546
Classified sport-01.html as SPORT

*** Classifying instance: usa-01.html
P(WORLD|usa-01.html) = 0.235294117647059
P(BIZ|usa-01.html) = 0.352941176470588
P(USA|usa-01.html) = 0.176470588235294
P(SPAM|usa-01.html) = 0.117647058823529
P(SPORT|usa-01.html) = 0.117647058823529
Classified usa-01.html as BIZ

*** Classifying instance: world-01.html
P(WORLD|world-01.html) = 0.805970149253731
P(BIZ|world-01.html) = 0.089552238805970
P(USA|world-01.html) = 0.044776119402985
P(SPAM|world-01.html) = 0.029850746268657
P(SPORT|world-01.html) = 0.029850746268657
Classified world-01.html as WORLD

*** Classifying instance: spam-biz-01.html
P(WORLD|spam-biz-01.html) = 0.121212121212121
P(BIZ|spam-biz-01.html) = 0.181818181818182
P(USA|spam-biz-01.html) = 0.090909090909091
```

图 5.5　把 EmailClassifier 用于邮件的多分类处理

Download at http://www.pin5i.com/

基于规则的系统有两种基本的运作方式，第一种是前向、数据驱动的，因为是先给出数据，然后寻找可用于该数据集的规则。第二种是反向、目标驱动的，因为是先定出目标，然后尝试使引擎去满足它（如果可以的话）。像 Prolog 与 ECLiPSe 之类的编程语言就支持了这样的特性，不过本书不会涉及反向推理的内容（欲知细节，可参看 Russell 与 Norvig 的文章）。 189

DROOLS 规则引擎

有两个基于 Java 的规则引擎实现已经可以应用于生产环境。第一个是圣地亚国家实验室（Sandia National Laboratories）出品的 Jess，在本书写作（2008 年春天）之时，它已经发布第七个发行版（7.1）了，所以是一款相当稳定的软件。它对于学术上的使用是免费的，但对于商业性的使用则不是（Jess 规则引擎的细节可参看 Friedman-Hill 的文章）。第二个规则引擎实现叫 Drools（http://www.jboss.org/drools/），也有人叫它 JBoss Rules。JBoss 是一个著名的中间件开源项目，现在由 RedHat 主持维护。Drools 是一个鲁棒的规则引擎，拥有丰富的文档与相当宽松的开源许可（Apache 2.0），这意味着你可以在你的应用中免费使用它。在过去四年里，我们曾利用 Drools 获得过巨大的成功，就我们的观点而言，Drools 是 Java 世界里规则引擎的首选。

Drools 规则引擎由两个主要的模块构成：模式匹配模块与议程模块（agenda module）。模式匹配模块负责识别哪个规则可被事实所触发，一旦这些规则被识别出来，它们会被安放到议程模块中。图 5.6 显示了 Drools 引擎的基本元素。为了进行模式匹配，Drools 实现并扩展了 Rete 算法——单词 rete 拉丁语原义为“网络”，在欧洲发“re-tay”音，在美国发“ree-tee”音。Rete 算法由 Charles Forgy 于 1974 年所设计，它至今仍是其众多变形中最高效最成功的模式匹配算法之一。Rete 在 Drools 中的实现被称为 ReteOO，以标示 Drools 对 Rete 算法进行了改进，使得它能与面向对象的软件适配。Rete 算法采取了用内存空间换取处理速度的策略，它的理论性能独立于系统中规则的数量。但在实际中，随着规则数量的增加，一定会遭遇 AI 系统中那个著名的问题：实用性问题。我们会在第 5.7 节讨论更多这方面

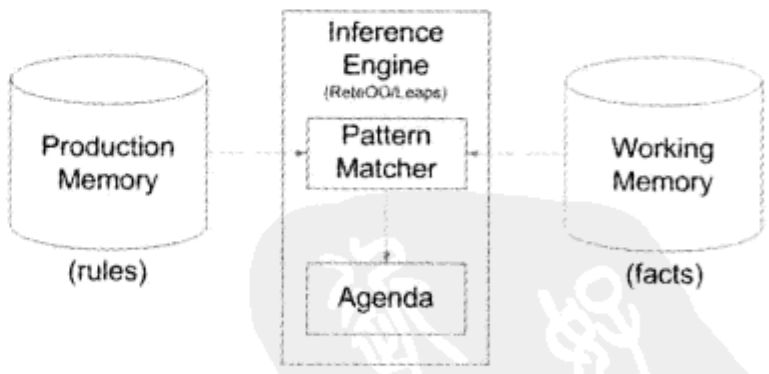


图 5.6 Drools 规则引擎系统的基本元素
（来源：Drools online）

190

Download at http://www.pin5i.com/

的内容。不过我们不打算深入了解 Rete 算法本身的细节，如果你对它的实现感兴趣，Drools 的源代码、文档以及本章后面的参考资料应该足够让你对它的内部工作机制有一个彻底的理解。此外，由 Manning 出版社出版，Ernest J. Friedman-Hill 博士撰写的 *Jess in Action* 也为 Rete 算法的工作原理提供了一个详尽的解释。

在一番简短的对规则引擎的介绍之后，我们已经准备好描述 Drools 库在过滤垃圾邮件这个任务中的应用了，具体来说，就是看看该如何写规则。Drools 引擎自带了一个简单易学的脚本（非 XML）语言，它是如此的简单，你甚至可以直接把它开放给应用的用户进行编辑，让我们看看它的内容。

代码清单 5.7 一个用于垃圾邮件过滤的简单规则集

```
package demo;
import iweb2.ch5.classification.data.Email;
import iweb2.ch5.classification.rules.ClassificationResult;
global ClassificationResult classificationResult;
rule "Tests for viagra in subject"
when
  Email( $s : subject )
  eval( classificationResult.isSimilar($s, "viagra" ) )
then
  classificationResult.setSpamEmail(true);
end
rule "Tests for 'drugs' in subject"
when
  Email( $s : subject )
  eval( classificationResult.isSimilar($s, "drugs" ) )
then
  classificationResult.setSpamEmail(true);
end
```

在电子邮件标题中识别“Viagra”的规则

在电子邮件标题中识别“drugs”的规则

191

代码清单 5.7 显示了 spamRules.drl 文件的内容，它包含两个简单的规则。这些内容几乎就是不言而喻的，你可以在子目录 C:\iWeb2\data\ch05 中找到这个文件。很快我们会看到，规则是以包的形式提交给 Drools 的，所以第一件事情就是给这些规则所属的包命一个名——demo。Import 语句通知规则执行引擎在规则中会用到的对象的类定义——Email 类与 ClassificationResult 类。global 语句允许我们通过 classificationResult 变量获取对象的定义，这等同于我们把 classificationResult 声明为

Download at http://www.pin5i.com/

规则集中的全局变量。那么剩下的那些规则是什么意思呢？

第一条规则称为“Tests for viagra in subject.”，顾名思义，它检查 Email 对象中的变量 subject 是否包含 viagra 这个单词。如果条件满足，则把 ClassificationResult 的 isSpamEmail 变量设置为 true。类似地，名为“Tests for drugs in subject”的第二条规则，检查 Email 对象里的变量 subject 是否包含单词 drugs，如果条件满足，同样把 ClassificationResult 的 isSpamEmail 变量设置为 true。

我们并非意指这样的条件就百分之百地能判定某封邮件为垃圾邮件，使用它们仅仅只是为了说明 Drools 文件的结构。如你所见，在 Drools 中定义规则的一般性结构是相当直观的：

```
rule "Put the name of your rule here"
when
  <Put here your conditions>
then
  <Put here the actions that must be taken
➥ when the above conditions are satisfied>
end
```

这里面，你可以包含多个条件，也可以包含多个操作。我们不认为还有比这个更简单的了！尽管 Drools 规则语言是如此的朴实无华，但可实现的功能却是惊为天人的，我们相信这个引擎的真正强悍之处在于它对对象的支持。注意，每个规则的条件判断都发生在 ClassificationResult 类的 isSimilar 方法的内部，这种时髦的面向对象的处理方式让我们可以采用相当复杂的评估过程。

现在，让我们看看这些规则的具体使用。代码清单 5.8 的第一行从测试数据集中载入邮件，我们曾经在代码清单 5.2 中使用过它。所以，让我们接着看代码清单里其他的步骤。

代码清单 5.8 对一个数据集使用垃圾邮件规则

```
EmailDataset ds = EmailData.createTestDataset();
EmailRuleClassifier classifier =
     new EmailRuleClassifier("c:/iWeb2/data/ch05/spamRules.drl");❶
classifier.train();❷
classifier.run(ds,"Expecting one spam email. :-(");❸
```

❶ 创建了一个基于规则的邮件分类器，它被适当地封装在 EmailRuleClassifier 192

Download at http://www.pin5i.com/

类中。注意，这个类的构造函数只有一个参数，即代码清单 5.7 中描述的 Drools 文件。

❷ 让分类器对其自身进行“训练”。与代码清单 5.1 中训练概率分类器不同，这里我们并没有创建一个训练集。我们从未把数据集的引用传递给分类器的构造函数，而只是把 Drools 文件名传递给它，为什么？在概率方法中，我们试图推断出蕴含在训练集中的知识。而对于基于规则的系统，规则就是知识，所以这种系统中“训练”一步所需做的仅仅只是把规则从文件中载入进来。

❸ 把数据集与一个描述性信息传递给分类器，即可把规则“应用”于测试数据集。尽管使用规则系统的商务人员与普通用户中都流行使用“应用规则”这样的字眼，但实际上，Rete 算法的执行更像是通过一个漏斗对数据（事实）进行过滤，这个漏斗由节点的网络所组成（rete 之名的由来）。当每个事实从上往下流过这个漏斗时，它会通过若干的测试（规则的条件），当它到达底部时，我们就能精确地知道这个事实触发了哪条规则。Rete 结构的详细描述可参看 Doorenbos 的文章。

代码清单 5.8 的执行结果如图 5.7 所示。spam-biz-01.html 文件所代表的垃圾邮件触发了分类器，因为其标题包含的单词“drugs”匹配上了代码清单 5.7 的垃圾规则，一旦条件被满足，该规则即可发动。

```
bsh % classifier.run(ds,"Expecting one spam email.  :-(");

Expecting one spam email.  :-(
------------------------------------------------------------

Classifying email: world-01.html ...
Rules classified email: world-01.html as: NOT-SPAM

Classifying email: spam-biz-01.html ...
Invoked ClassificationResult.setSpamEmail(true)
Rules classified email: spam-biz-01.html as: SPAM

Classifying email: sport-01.html ...
Rules classified email: sport-01.html as: NOT-SPAM

Classifying email: usa-01.html ...
Rules classified email: usa-01.html as: NOT-SPAM

Classifying email: biz-01.html ...
Rules classified email: biz-01.html as: NOT-SPAM
------------------------------------------------------------
```

图 5.7 基于代码清单 5.7 中的规则把垃圾邮件 spam-biz-01 识别出来

Download at http://www.pin5i.com/

代码细节

193

现在你对于 Drools 引擎的使用已经很熟悉了，让我们仔细看看用于代码清单 5.7（规则定义文件）与代码清单 5.8 中的包装类的实现。想象一下，我们之前居然可以把大量的功能打包成仅仅只有几行代码的调用，什么样的代码能把规则引擎的使用抽象为仅仅三个简单的步骤呢？让我们先从这个实现的核心 RuleEngine 类开始，见代码清单 5.9。

代码清单 5.9 RuleEngine：基于 Drools 库创建一个规则引擎

```
public class RuleEngine {
  private RuleBase rules;
  public RuleEngine(String rulesFile) throws RuleEngineException {
    try {
        Reader source = new InputStreamReader(
            new BufferedInputStream(new FileInputStream(rulesFile)));
        Properties properties = new Properties();
        properties.setProperty("drools.dialect.java.compiler", "JANINO");
        PackageBuilderConfiguration cfg =
➡       new PackageBuilderConfiguration( properties );
        PackageBuilder builder = new PackageBuilder(cfg);
        builder.addPackageFromDrl(source);
        Package pkg = builder.getPackage();
        rules = RuleBaseFactory.newRuleBase();
        rules.addPackage(pkg);
    } catch (Exception e) {
        throw new RuleEngineException(e);
    }
  }
  public void executeRules(ClassificationResult classificationResult,
➡ Email email ) {
    WorkingMemory workingMemory = rules.newStatefulSession();
    workingMemory.setGlobal("classificationResult",
➡ classificationResult);
    workingMemory.insert(email);
    workingMemory.fireAllRules();
  }
}
```

❶决定运行时编译器

❷包含我们的规则

❸运行时的规则容器

❹有状态的工作内存

❺把事实插入工作内存

❻执行所有的规则

Download at http://www.pin5i.com/

Drools 规则引擎的创建包含两部分：编集（authoring）与运行时。编集部分从解析 Drools 文件开始——带.drl 扩展名的文件，解析器检查 Drools 文件的语法一致性，并创建一个中间抽象语法树（AST）。在这一步需要一个词法分析器，Drools 使用的是开源项目 ANTLR——Another Tool for Language Recognition（http://www.antlr.org/）——作为词法分析器。有效规则会以 Package 类的序列化对象形式被载入，一个 Package 实例是一个独立可部署的单元，可包含一条或多条规则。Drools 引擎的运行时部分是基于 RuleBase 类的，RuleBase 的实例可以随时添加或删除 Package 实例。

194

让我们检查用于创建 Drools 规则引擎的每一步，如代码清单 5.9 所示：

❶ 首先创建一个包含规则文件的引用，然后创建 Properties 的实例，并给该实例的属性 drools.dialect.java.compiler 赋值。这个属性做什么用？所赋的值 JANINO 又是什么意思？它可以让你把 Java 代码直接合并到 Drools 规则文件中，这个属性确定运行时编译器，由 Drools 用于编译 Java 代码。Janino 是一个嵌入式 Java 编译器的名字，它包含在 Drools 的发行版里，遵循 BSD 许可（http://www.janino.net/）。

接着，为完成编辑部分，我们需要创建 PackageBuilder 类的实例，再由它生成类 Package 的实例。PackageBuilderConfiguration 是辅助 PackageBuilder 进行配置的类，PackageBuilder 的默认值可以通过调用 set 方法进行修改，或者可以采用这里的方式，通过属性配置的方式来定义。在这个例子里，只传送了一个属性，但其实还可以提供更多的信息。设置的核心是 ChainedProperties 类，它会搜索多个目录以寻找 drools.packagebuilder.conf 文件。按照优先顺序，这些目录依次是系统属性、系统属性中用户定义的文件、用户的家目录、工作目录以及多个 META-INF 目录。PackageBuilderConfiguration 处理 AccumulateFunctions 的注册、Dialects 的注册，以及主要的 ClassLoader。欲知更多的细节，可参考 Drools 的在线文档：http://downloads.jboss.com/drools/docs/4.0.7.19894.GA/html_single/index.html - d0e766。

❷ PackageBuilder 的实例生成后，就可以生成包含规则的包了。把文件的引用传递给 addPackageFromDrl 方法，就可以立即调用 getPackage 方法。规则就绪！

❸ 这是为引擎建立运行时部分的第一步，一个 RuleBase 对象可以拥有一个或多个 Packages，它可以随时实例化一个或多个 WorkingMemory 对象，除非特别配置，否则默认会对它们保持一个弱引用。WorkingMemory 类还有很多子部件，可以从 Drools 在线文档中获得更多的细节。

Download at http://www.pin5i.com/

❹ StatefulSession 类继承自 WorkingMemory 类，它为规则的插入、更新与触发增添了异步的方法，还有 dispose()方法。RuleBase 类为它创建的每个 StatefulSession 实例保持一个引用，当新规则添加进来时，就可以对这些实例进行更新。dispose()方法让 RuleBase 可以释放 StatefulSession 的引用，以防内存泄漏的发生。

在代码清单 5.7 的 Drools 文件中，我们使用 global 语句来声明对 classification Result 对象的访问，这等效于在规则内把 classificationResult 声明为全局变量。但这个效果只有当 WorkingMemory 实例调用 setGlobal 方法后才能最终实现，这个方法的参数必须跟 Drools 规则文件中的声明完全保持一致。 195

❺ 在这里我们使用 insert 方法把事实添加到 WorkingMemory 实例。当插入一个事实时，Drools 引擎会把它跟所有的规则进行匹配，这意味着所有的工作都会在插入阶段完成，但直到你调用 fireAllRules()方法时，规则才会起作用，这是下一步的事情。

❻ 这一步调用规则的执行方法。你应该在完全插入所有的事实之后，才调用 fireAllRules()。之前已经说过，关键的匹配阶段已经在事实插入时完成，所以可以在所有的事实匹配上规则之后，再执行规则。

要利用 Drools 创建一个规则引擎，这几乎就是你所需做的全部事情了。现在来看看 EmailRuleClassifier 类是怎样把它的操作托付给 RuleEngine 来完成邮件的分类工作的。这里 RuleEngine 的实现是针对邮件的，我们建议你可以参照本章的 To Do 事项之一利用 Instance 接口创建一个通用的实现。代码清单 5.10 展示了 EmailRule Classifier 类的代码，剔除了 main 方法，因为它跟代码清单 5.8 的代码差不多。

代码清单 5.10 使用基于 Drools 的规则引擎检测垃圾邮件

```
public class EmailRuleClassifier {
  private String ruleFilename;
  private RuleEngine re;
  private Concept spam;
  private Concept notSpam;
  public EmailRuleClassifier(String ruleFilename) {
    this.ruleFilename = ruleFilename;
  }
  public void train() {
    re = new RuleEngine(ruleFilename);❶
    spam = new BaseConcept("SPAM");❷
    notSpam = new BaseConcept("NOT-SPAM");
```

Download at http://www.pin5i.com/


```
}
public Concept classify(Email email) {
  ClassificationResult result = new ClassificationResult();❸
  re.executeRules(result, email);❹
  if( result.isSpamEmail() ) {❺
    return spam;
  } else {
    return notSpam;
  }
}
public void run(EmailDataset ds, String msg) {
  System.out.println("\n");
  System.out.println(msg);
  System.out.println("____________________________________________");
    for(Email email : ds.getEmails() ) {❻
      Concept c = classify(email);
      System.out.println("Email: "+
➥ email.getId()+" classified as: "+c.getName());
    }
  System.out.println("____________________________________________");
  }
}
```

196

❶ 首先我们需要创建一个 RuleEngine 实例，这样就可以把对规则的应用交付给它。我们把包含规则的文件名称传递给 RuleEngine，让它来进行繁重的匹配工作。

❷ 这是两个由 classify 方法使用的辅助变量。因为在这个例子里它们是常数，所以不管是什么规则或邮件，我们都把它们看做是实例变量。你还可以定制 ClassificationResult 类的实现，使得它可以从更复杂的概念数据结构（例如，实体）中辨识出正确的概念来。

❸ 这个类封装了两个重要的动作。它包含对规则条件的测试（通过 isSimilar 方法），以及规则的操作（通过 setSpamEmail 方法）。你也可以创建不同的对象来封装条件与操作。如果你的条件或操作涉及复杂的算法实现，最好把这两部分的实现清晰地隔离开。

❹ 这是把规则的应用托付给 RuleEngine 的地方，我们会在代码清单 5.9 中再查看这个方法。

❺ 我们再次使用 ClassificationResult 实例来获取（被触发）规则生成的结果。

Download at http://www.pin5i.com/

这个结果也可以记录在持久化媒体中（如数据库或文件），但在这个简单的例子里，我们把 ClassificationResult 类作为所有相关动态的记录者。

❻ 这个方法帮助我们把数据集里所有的邮件一次性地进行分类，也可以把数据集本身传递给分类方法，并改写 RuleEngine 类的 executeRule 方法，这样就可以把所有的邮件一次性载入到工作内存里。但要注意，在基于规则的系统中，对其他邮件的判断不会影响到某一封邮件是否是垃圾邮件的判定。

冲突解决

最后一点事关（但又有区别）另一个有趣的主题。让我们看看如下情况：如果规则 A 的操作修改了事实 X，这个改动激活了规则 B，规则 B 会修改事实 Y，这个改动再度触发规则 A，怎么办？如果没有办法停止这个循环，系统就会陷入死循环。再如，如果一个规则把一封邮件判定为垃圾，但另一个规则却把同样的这封邮件判定为正常，又如何是好？换言之，当两个或多个规则发生冲突时该如何处理？在 executeRule 方法结束之前，我们必须要有一个答案，那答案应该是什么？

比较幸运的是，Drools 已经通过规则属性为这些问题（还有更多的其他问题）提供了现成的答案。第一个问题通过规则属性 no-loop 来解决；第二个问题通过规则属性 salience 来解决。规则属性让你可以通过 Drools 规则文件中的声明来影响规则的行为。有一些很简单的属性，如 salience，另有一些会比较复杂，比如 ruleflow-group。查阅 Drools 的官方文档与源代码，你可以得到对每个属性详尽的了解。 197

下面看看怎样通过引入 salience 属性来解决冲突。代码清单 5.11 显示了一个包含三条规则的 Drools 规则文件，这一次某些邮件规则间会存在冲突，因为多个规则的条件可能会被同时满足。这个文件看起来跟代码清单 5.7 第一个 Drools 规则文件一样，但这里给每条规则引入了 salience 属性，取值为整数。

代码清单 5.11 一个简单的垃圾邮件过滤规则（带冲突）

```
package demo;
import iweb2.ch5.classification.data.Email;
import iweb2.ch5.classification.rules.ClassificationResult;
global ClassificationResult classificationResult;
rule "Rule 1: Tests for viagra in subject"←在电子邮件标题中识别“Viagra”的规则
salience 100
when
  email: Email( $s : subject )
```

Download at http://www.pin5i.com/


```
  eval( classificationResult.isSimilar($s, "viagra" ) )
then
  email.setRuleFired(1);
  classificationResult.setSpamEmail(true);
end
rule "Rule 2: Tests for 'drugs' in subject"←在电子邮件标题中识别“drugs”的规则
salience 100
when
  email: Email( $s : subject )
  eval( classificationResult.isSimilar($s, "drugs" ) )
then
  email.setRuleFired(2);
  classificationResult.setSpamEmail(true);
end
rule "Rule 3: Tests for known sender address"←可能与规则 1 或 2 存在冲突的规则
salience 10
when
  email: Email( $sender : from )
  eval( classificationResult.isSimilar($sender, "friend@senderhost" ) )
then
  email.setRuleFired(3);
  classificationResult.setSpamEmail(false);
end
```

198

我们从文档 spam-biz-01.html 中创建一封邮件，它的标题包含单词“drugs”，所以会触发规则 2。同时，因为它的发信者为 friend@senderhost，所以规则 3 也会被触发。按照规则 2，这是一个垃圾邮件；而根据规则 3，这是一个正常邮件。也即是说，规则 2 与规则 3 在这个特殊的邮件里发生了冲突，所以我们需要一个冲突解决机制，而 salience 属性这时就成了救星。

salience 这个单词在规则系统语境里或许是源于 semiotics，即符号学的研究——希腊单词 σημειωση 意为记号、符号或者基于上下文的注解。在符号学中，salience 指在任何指定时刻一个人所接收到的大量符号中比较重要的一个。类似的，在我们的规则的上下文中，当所有的规则应用于一个特定的事实或一个事实集时，salience 属性指示一个规则相对于其他规则的重要程度。salience 值越低，规则的重要性就越高。从本质上说，规则的重要程度反映在执行次序上。如果规则 X 的 salience 值比规则 Y 小，那么规则 Y 会先执行，而 X 会后执行。不管 X 与 Y 的操作在何时何处发生了冲突，X 的操作总能覆盖掉 Y 的操作。

Download at http://www.pin5i.com/

让我们实际看看这些情况，代码清单 5.12 显示的脚本跟代码清单 5.8 的几乎一样，只是现在我们会使用带冲突的规则。

代码清单 5.12 利用 salience 属性解决邮件规则的冲突

```
EmailDataset ds = EmailData.createTestDataset();
EmailRuleClassifier classifier = new EmailRuleClassifier(
➥ "c:/iWeb2/data/ch05/spamRulesWithConflict.drl");
classifier.train();
classifier.run(ds," Hurray! No spam emails here.");
```

图 5.8 展示了执行的结果。你可以看到，规则 2 和规则 3 都被邮件 spam-biz-01.html 触发了，但规则 2（salience=100）先被触发，规则 3 随后才被触发，并把邮件的标志重置为 NOT-SPAM。

```
bsh % EmailDataset ds = EmailData.createTestDataset();
bsh % EmailRuleClassifier classifier = new EmailRuleClassifier(
?   "c:/iWeb2/data/ch05/spamRulesWithConflict.drl");

bsh % classifier.train();
bsh % classifier.run(ds," Hurray! No spam emails here.");

Hurray! No spam emails here.
____________________________________________________

Classifying email: world-01.html ...
Rules classified email: world-01.html as: NOT-SPAM

Classifying email: spam-biz-01.html ...
Invoked Email.setRuleFired(2), current value ruleFired=0,
emailId: spam-biz-01.html
Invoked ClassificationResult.setSpamEmail(true)
Invoked Email.setRuleFired(3), current value ruleFired=2,
emailId: spam-biz-01.html
Invoked ClassificationResult.setSpamEmail(false)
Rules classified email: spam-biz-01.html as: NOT-SPAM

Classifying email: sport-01.html ...
Rules classified email: sport-01.html as: NOT-SPAM

Classifying email: usa-01.html ...
Rules classified email: usa-01.html as: NOT-SPAM

Classifying email: biz-01.html ...
Rules classified email: biz-01.html as: NOT-SPAM
____________________________________________________
```

图 5.8 使用规则的 salience 属性解决规则冲突

Download at http://www.pin5i.com/

这是一个简单的例子，你可以看到每一步，并理解在引入 salience 规则属性后所产生的效果。规则系统的实际价值在于它们可以高效地处理复杂的规则以及数百万的事实，同时允许你通过声明的方式用不同的条件来做试验，而不需要修改一行代码。想一想：如果要在代码中书写成千上万行嵌套的 if-then 语句，仅仅是想一下就已经让我颤抖不已！

现在我们已经结束了邮件分类之旅。到目前为止，你已经学到了怎样利用概率朴素贝叶斯算法与 Drools 规则引擎对一般的文档进行分类，下一节会提出另一个分类算法。为了使学习更加有趣，我们会通过一个新的例子（欺诈检测）来讲解。

199

5.4 用神经网络做欺诈检测

欺诈在电子世界里是非常普遍的，从保险欺诈到互联网拍卖欺诈，从虚假的救济金申请表（fraudulent benefit application forms）到电信欺诈。如果你在互联网上从事一些吸引了众多用户与法人实体的业务，当有人不遵守游戏规则时，你能把他们识别出来，这样的能力是很有价值的。在这一节中，我们会以欺诈性购买交易作为例子，通过引入分类算法把欺诈交易从正常购买中找出来。

5.4.1 交易数据中关于欺诈检测的一个用例

我们的样例数据是人造的——你也不希望看到你的交易数据被印到一本技术书籍上吧？尽管如此，我们会尽量使数据看起来像是真实的，也会努力让你觉得你看到的就是真实的数据。以下场景对每个人来说应该都很熟悉：你为一家大银行从事信用卡方面的工作，你需要确定你们的系统可以快速地（如果不能做到实时）检测出欺诈行为，以便采取合适的机制来保护你的客户。我们考虑如下与交易紧密相关的典型属性：

200

- 交易的描述。
- 交易总额。
- 交易地点。

我们已经建立了一个正常交易描述的集合，都包含在名为 descriptions.txt 的文件中，还有一个欺诈交易描述的集合，包含在名为 fraud-descriptions.txt 的文件中。你可以在目录 data\ch05\fraud 中找到这两个文件。对用户，我们需要五个不同的描述

Download at http://www.pin5i.com/

特征，因为消费习惯取决于很多方面，一次 3000 美元的交易对于某一个账户来说可能是有欺诈嫌疑的，但对于另一个账户却可能是完全正常的。五个描述特征足够做出判断了，当然，现实中会有更多的消费描述。消费总额是从高斯分布中提取的，它取决于交易总额的均值与标准差。如果你不知道高斯分布或标准差，可查看附录 A。

现在是时候让你知道大交易数据集中一个有趣的性质了。如果你从不同的源聚合了交易数据，观察这些数字里第一个有效数字等于 1 的频率，你会发现它出现的频率比你预料的要高得多。每一个正常人（我是说不一定是数学家）都会告诉你，既然我有 9 个数字，那么看到数字 1 的可能性当然就是 11.1%，看到数字 2 的可能性也是 11.1%，以此类推。真的吗？错！贝德福德定律（Benford's law）告诉我们这个概率应该服从对数分布，而非均匀分布。以此得到结果，第一个有效数字为 1 的概率约为 30%。在这个强大的统计事实后面有一个有趣的故事：1995 年，这个定律被布鲁克林地方检察办公室成功地用于对纽约的 7 家公司的欺诈检测（参看 Hill 的文章）。

回到对我们对交易数据的描述，我们把交易地点简化为欧几里德坐标（x，y）。一个实际的系统可以通过 GPS 精确地描述交易的地点，但在这里，简化的（x，y）坐标同样能很好地满足了我们的需要，同时避免了不必要的问题复杂化。交易的（x，y）坐标从一个设定好最大最小值的均匀分布中得到，换句话说，对于每一个用户档案，我们为 x 与 y 设置一个最大与最小值，对于一个给定的交易，分配给它一个随机的地点，坐标取值（x，y）可从最大最小值之间等概率地选取。

你可以试验一下代码，然后生成自己的数据，你也可以增添更多的描述或者更多的用户，以及给每个用户更多的交易。TenUsersSample 类作为学习的启动正合适，你可以在包 iweb2.ch5.usecase.fraud.util 中找到这类及其他的辅助类。这个类的 main 方法的执行会生成两个文件，一个名为 generatedtraining-txns.txt，另一个名为 generated-test-txns.txt。你也许从名字就能猜到，这两个文件分别包含训练数据与测试数据。在文件夹 data\ch05\fraud 中可找到本节使用的数据，我们称为 training-txns.txt 与 test-txns.txt。训练集大概有 10 000 条交易记录，测试集有 1 000 条，每条交易由如下的属性值所确定（按序罗列）：

- 用户 ID。
- 交易 ID。
- 交易的描述。

201

Download at http://www.pin5i.com/

- 交易总额。
- 交易的 x 坐标。
- 交易的 y 坐标。
- 一个用于确定交易是（true）否（false）属于欺诈的二值变量。

我们的目的很直接，即创建一个分类器，它能基于训练集的交易数据学习如何辨识一个欺诈交易。一旦训练好分类器，我们希望在来自同一个统计分布的测试集中检验它。接下来的章节会利用两种不同的分类系统来达到这个目的。第一个基于神经网络算法，第二个基于决策树。我们曾在 5.3 节的概述中讨论过这些分类方法，现在是时候与它们亲密接触了。

5.4.2 神经网络概览

本节我们会简要地提出神经网络背后的核心思想。神经网络这个主题很宽泛，我们会聚焦于被称为计算神经网络（computational neural networks）的这一领域——我们会避免使用“人工”（artificial）这个词，因为已经有基于硬件的神经网络实现了（如 Maier 等人的文章所述），这里会专注于神经网络的软件实现。

一般而言，一个神经网络由神经元节点（或简称神经元）与节点之间的连接（突触）所构成。一些节点负责简单地把数据传入或传出网络，另一些节点则负责处理数据。前者让网络具备 I/O 能力，所以按照它们是插入数据到网络或从网络导出数据的不同职责，分别被形象地称为输入层和输出层。其他所有的节点都称为隐层节点，它们不直接与“外面的世界”进行交互。

图 5.9 展示了一个典型的神经网络。对于一个给定的输入向量$\{x1, x2, x3\}$，神经网络生成一个由输入与网络参数构成的函数。图中网络的输出标识为 y，一般来说，输出不一定是一个单一的值，也可以是一个向量。你可以想象初始值$\{x1, x2, x3\}$从左边传播到右边，每个节点都收集它的输入值并计算输出值，最后结果（y）依赖于初始值$\{x1, x2, x3\}$与这些值在网络中传播的方式。

202

突触用于连接节点，信息可以在突触连接的节点之间进行传递交换。信息的交换由突触的权值参数所控制，这个权值大致上反映了两个节点连接的重要程度。在神经网络的训练阶段，突触的权值会根据训练集的值不断地进行评估与修正。

神经网络的图形化表示是很常见的，仅仅通过观察这种图形化的表示，你就可以得到大量关于这个神经网络的信息。

Download at http://www.pin5i.com/

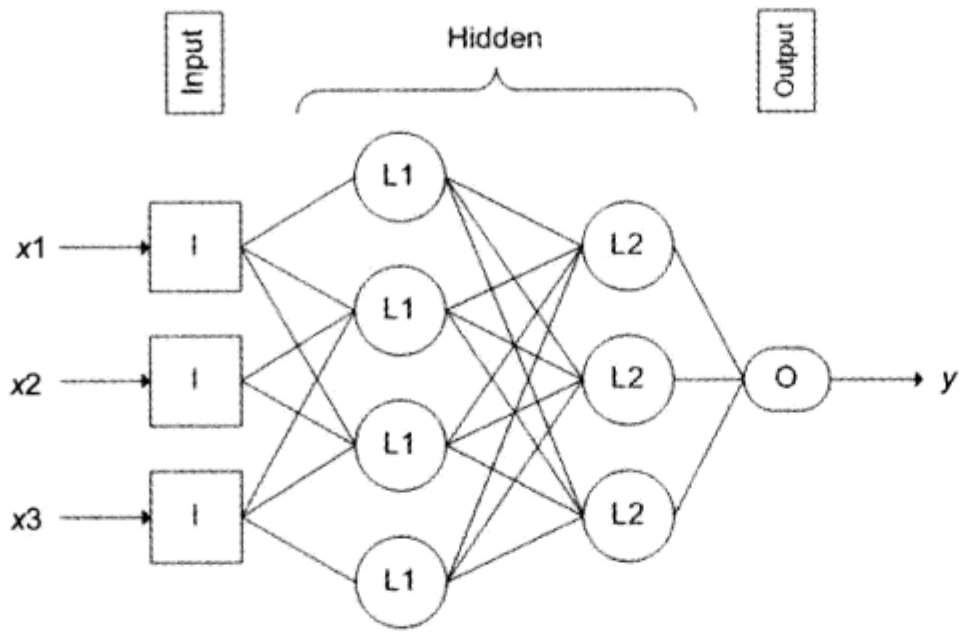


图 5.9 一个典型的神经网络，具有三个输入节点、两层隐层节点和一个输出节点

第一，注意到只有两种连接加了箭头：输入变量的节点以及输出答案 y 值的节点。如果节点与突触构成了有向无环图（directed acyclic graph，DAG）——检验这种情况的一个经验法则是看是否所有的箭头都从左指向右，若是，说明我们得到的是一个前向神经网络，否则，我们得到的就是一个反向神经网络。

第二，我们把节点按照垂直带的方式从左往右排列，这并非死规定，但习惯上通常是这么表示的。我们把每个垂直带称为“层”，按照惯例，把第一个隐层表示为 L1 节点，第二个隐层表示为 L2 节点，输入层表示为 I，输出层表示为 O。

第三，可以看到输入节点不一定要连接所有的第一隐层的节点，但第一隐层的节点会跟第二隐层所有的节点都相连。当某一层的所有节点与下一层的所有节点都有连接时，我们称它们是完全连接的。

以上观察结果是如下判语的心理学准备：我们可以通过如下三个基本要素来完全定义一个神经网络（参看 McKay 的文章）。

- 神经网络结构：描述网络中节点的数量，输入/输出节点的数量，以及突触连接和连接的方向。
- 激活规则：描述节点间直接交互的规则。
- 学习规则：描述节点间间接交互的规则，以及网络中传播的信息。

所有这些灵活因素都让我们对要定义的神经网络拥有无穷的可能性，同时也让我们难以确定现实中什么样的神经网络才是最理想的。我们并不打算用过多的篇幅来全面介绍神经网络，对于我们来说，这也未免过于不自量力。在附录 E 中，你能找到更多的关于神经网络的文献。

203

Download at http://www.pin5i.com/

5.4.3 一个可用的神经网络欺诈检测器

我们现在要做的第一步是使用神经网络来帮助辨识欺诈交易。代码清单 5.13 显示了如下流程：

- 载入一个交易数据集，并为它计算用户统计信息。
- 构建 NNFraudClassifier 分类器，训练它，并把它存入磁盘。
- 从磁盘中载入 NNFraudClassifier 的一个实例，并把它用于交易的分类。
- 载入一个新的交易数据集，通过 FraudErrorEstimator 类的一个实例来测试你的分类器。

代码清单 5.13 NNFraudClassifier：一个用于欺诈检测的神经网络分类器

```
TransactionDataset ds = TransactionLoader.loadTrainingDataset();❶
ds.calculateUserStats();❷
NNFraudClassifier nnFraudClassifier = new NNFraudClassifier(ds);❸
nnFraudClassifier.setName("MyNeuralClassifier");
nnFraudClassifier.useDefaultAttributes();❹
nnFraudClassifier.setNTrainingIterations(10);❺
nnFraudClassifier.train();❻
nnFraudClassifier.save();❼
NNFraudClassifier nnClone = NNFraudClassifier
➥ .load(nnFraudClassifier.getName());❽
nnClone.classify("1");❾
nnClone.classify("305");
TransactionDataset testDS = TransactionLoader.loadTestDataset();❿
FraudErrorEstimator auditor = new FraudErrorEstimator(testDS, nnClone);⓫
auditor.run();
```

204

如你所见，使用我们的代码来创建并使用基于神经网络的分类器相当简单，所有的事情都可以通过简单的几个步骤写就。下面我们就一步一步地分析。

❶ 文件 training-txns.txt 中的交易数据集通过 TransactionDataset 类来封装，也可以用 iweb2.ch5.usecase.fraud.*包中的代码来创建你自己的数据集，还可以只给出一个交易文件名，然后剩下的就不用管了。但这是一本实践性的书籍，通过创建你自己的数据集（也可以扩展我们所提供的数据），有助于你对自己应用中的数据进行建模。使用正确的数据与合适地使用这些数据在智能应用中是极其重要的。

Download at http://www.pin5i.com/

❷ 获得原始交易数据后，我们就可以收集每一个用户消费习惯的统计信息。要牢记：在现实生活中，这些数据本该是从后台或数据仓库系统中获得的。我们需要通过挖掘这些数据得到一些信息，为每个用户计算得到一个基准线。如果用户 A 的消费范围为 20 美元到 200 美元，用户 B 的消费范围为 100 美元到 5000 美元，那么一个 2000 美元的交易对于这两个用户而言意味着完全不同的事情。这个过程属于宽泛意义上的数据预处理，名为数据规范化（即数据标准化）。

查看 UserStatistics 类，它封装了每个用户的消费基准线。有三件事情是需要注意的，第一是我们曾经提到过的消费范围，它是训练集中合法交易的最大最小金额。第二，留意合法交易描述中的单词集。第三，除了最大最小位置坐标值，我们还会计算位置的中心点，理由是大多数的交易都会发生在居住地的附近，所以，如果一个新的交易地点离基准点（该用户的位置中心点）很远，我们应该对其进行慎重考察，但这个因素贡献并非是主导性的，因为居民会有偶发性的旅游。

❸ NNFraudClassifier 是交易分类的主类，这个类本身不是一个神经网络，它把自己函数中关于学习方面的要求交付给 TransactionNN 类来实现，我们会在本节稍后再来查看这些类。我们为分类器取了名字，便于之后引用。这个名字会用于构造它的文件名（序列化时），所以应该给它取一个能表义的名字。

❹ 如果你还记得之前的实验（特别是第 2 章用户点击的实验），我们需要确定应把交易的什么属性用于分类。在现实中，这是很重要的，因为你的交易里即便没有数百，也会有数十个属性。你需要慎重地决定哪些属性应当被用于分类训练，不相关的属性会“玩死”你的分类器，并极大地干扰它对于欺诈交易的分类能力。这个方法自动选择三个我们讨论过的交易属性：总额、位置与描述。

❺ 这一步决定数据会在网络中传播多少次。这个值的选择取决于数据性质与网络结构。如你所见，在我们的例子中，10 次就可以工作得很棒。 205

❻ 一旦为我们的神经网络配置好所有的参数，就可以开始训练过程了。完成对这个方法的调用后，分类器已经从训练集中学到了所有它应该学习的，可以听命待用了。

❼ 我们把训练好的分类器实例存放在磁盘里，这一步的重要性在于你可能要把分类器发布到多个系统中，又或者使用分类器的系统跟创建分类器的系统并不是同一个。除此之外，这个措施还能有效地避免因系统故障而造成的损失。想象一下，你花了两个小时的时间为数百万个拥有 10 个属性的交易训练出来一个分类器，却因

Download at http://www.pin5i.com/

为某些意外的原因，系统宕机了！除了欲哭无泪，你还能怎么办？训练好的分类器应当享受像其他内容可变的电子文档同等的待遇——当在使用期内的分类器做出额外的训练之后，它的副本应当被持久化存储，例如，可以存放在磁盘中。

❽ 这里说明如何载入一个已训练好的分类器，你所需知道的只是分类器的文件名。在我们的实现中，所有序列化后的分类器都存放在磁盘的同一个路径，这个路径由 NNFraudClassifier 类的常量 SERIALIZATION_PATH 决定。如果你觉得这个不方便，可以修改这个变量，或者修改相关的代码，以使你的分类器存储更灵活。

❾ 开动吧！我们已经准备好要对两个交易进行分类，第一个 ID（1）是合法交易，第二个 ID（305）属于欺诈交易。这只是一个检查性的测试，并非对分类器全面的评估，因为这里所选的交易都是分类器在训练阶段已经遇到过的。

❿ 创建了一个新的数据集，分类器在此前还从未遇见过那里的交易。我们把它称为训练数据集，并标识为 testDS。

⓫ FraudErrorEstimator 是一个辅助类，它帮助我们评估分类器的精确度。这个过程通过调用 run()方法来启动，最后它概述了正确分类的交易的数量以及错分交易的数量。尤其是，它还单独提供了被分类器认为欺诈但实际是有效交易的数量，以及被分类器判为合法但实际是欺诈交易的数量。这两种误分类具有不一样的意义，我们会在下一节讨论。

在深入神经网络内部之前，让我们先看看结果。图 5.10 是代码清单 5.13 的执行输出。哇！看到了没？这个分类器看起来简直太完美了！这是真的吗？其实未必如此！这对于使用黑盒库但并不掌握其内部分类算法工作原理的使用者来说，是一个很常见的陷阱。为了理解这一点，用任意的文本编辑器打开 test-txns.txt 文件，把其中的每一个“BLACK DIAMOND COFFEE”条目换成“SOME DUDE”。

206

如果你再次执行代码清单 5.13 中的最后三行，你会看到如图 5.11 所示的输出结果，其中会包含标准化后交易属性的值，为了具有可读性，这里我们把它省略掉了。那几个置换过描述的交易原本都是合法的——它们最后一个属性为 false。输出结果显示有四个合法的交易（VALID_TXN）被错分为欺诈交易。我们完美无瑕的记录这次染上了污点，因为之前的置换给我们的数据引入了噪声。换句话说，我们现在处理的数据是之前从未遇到过的。

Download at http://www.pin5i.com/


```
bsh % nnClone.classify("1");
Transaction:
  >> 1:1:EXPEDIA TRAVEL:63.29:856.0:717.0:false

Assessment:
  >> This is a VALID_TXN
bsh % nnClone.classify("305");
Transaction:
  >> 1:305:CANADIAN PHARMACY:3978.57:52.0:70.0:true

Assessment:
  >> This is a FRAUD_TXN
bsh % TransactionDataset testDS =
å   TransactionLoader.loadTestDataset();

bsh % FraudErrorEstimator auditor =
å   new FraudErrorEstimator(testDS, nnClone);

bsh % auditor.run();
Total test dataset txns: 1100, Number of fraud txns:100
Classified correctly: 1100,
Misclassified valid txns: 0,
Misclassified fraud txns: 0
```

图 5.10 神经网络分类器 NNFraudClassifier 的分类结果（代码清单 5.13）

```
bsh % TransactionDataset testDS =
å  TransactionLoader.loadTestDataset();

bsh % FraudErrorEstimator auditor =
å  new FraudErrorEstimator(testDS, nnClone);

bsh % auditor.run();

userid = 25.0 - txnid = 500523 - txnamt = 63.79 -
å  location_x = 533.0 - location_y = 503.0 -
å  description = SOME DUDE --> VALID_TXN

userid = 26.0 - txnid = 500574 - txnamt = 127.97 -
å  location_x = 734.0 - location_y = 507.0 -
å  description = SOME DUDE --> VALID_TXN

userid = 23.0 - txnid = 500273 - txnamt = 47.76 -
location_x = 966.0 - location_y = 991.0 -
description = SOME DUDE --> VALID_TXN

userid = 21.0 - txnid = 500025 - txnamt = 50.47 -
location_x = 980.0 - location_y = 996.0 -
description = SOME DUDE --> VALID_TXN

Total test dataset txns: 1100, Number of fraud txns:100
Classified correctly: 1096,
Misclassified valid txns: 4,
Misclassified fraud txns: 0
```

图 5.11 通过置换有效交易中的“描述”属性来给数据引入噪声

第一个测试集（图 5.10 显示了对它的分类结果）的交易没有任何一个来自训练

Download at http://www.pin5i.com/

集，但它们却来自跟训练集一样的统计分布，更何况，交易描述都来自一个确定的描述集，没有任何变化。所以，尽管每一个测试交易都从未被遇到过[2]，但它们却恰好都属于同一个数据空间。几乎所有的分类算法面对这样的情况时都能做得很好，但当测试数据相对于训练集发生显著性变化时，它们又都会产生误差。一个分类器是否可以优雅地处理自己从未遇到过的数据的能力，是其泛化能力的重要度量。

207

这个事例的寓意在于说明对数据的充分了解是何等重要。尤其重要的是，需要清楚你的数据在多大程度上代表了你的应用实际会面临的所有可能的数据。一般来说，收集大量的数据有助于我们获得更多的相关数据。

实际中，这并不像说起来那么简单，因为同样的数据在不同的环境中可能有不同的意义。此外，过少的属性会导致数据意义的不明确，但过多的属性又会因为鱼龙混杂的信息而让分类本身产生混淆。我们必须找到这样一个平衡，使分类器对已知的信息能做到精确的反映，同时又具备泛化的能力。

很重要的一点是，当引入噪声时，你需要知道分类器会表现出多大的敏感性。在之前的例子中，我们改变了 1100 个测试交易中 39 个交易的描述信息，而这 39 个“被污染过”的交易中有 4 个被神经网络分类器错分了。如果你把置换后的字符串换成其他，又会有什么后果呢？随着受污染的交易数量增加，又会有多少个污染交易会被错分呢？你可以把自己的名字作为置换后的字符串来测试，研究一下是什么结果。

208

5.4.4 神经网络欺诈检测器剖析

是时候近距离看看 NNFraudClassifier 类了，具体代码显示在代码清单 5.14 中。它的核心依赖于 TransactionNN 类，这是一个神经网络，专门为我们欺诈检测案例的需求而建立。然后，TransactionNN 继承自一个名为 BaseNN 的通用神经网络基类，你可以以这个基类为基础来编写自己的神经网络，我们会在代码清单 5.16 中查看到 BaseNN 类。

代码清单 5.14 基于一个特殊神经网络的欺诈检测分类器

```
public class NNFraudClassifier
  implements Classifier, java.io.Serializable {
private String name;
```

2 这里“从未遇到过”是这么定义的：如果我们把任意一个测试交易与训练集中每一个交易进行属性值间的比较，我们会发现训练集中没有一个交易与测试交易是相同的。

Download at http://www.pin5i.com/


```
private TransactionNN nn;
private TransactionDataset ds;
private transient TrainingSet ts;
private TransactionInstanceBuilder instanceBuilder;
private List<String> availableAttributeNames;
public NNFraudClassifier(TransactionDataset ds) {❶
  this.ds = ds;
  this.ts = ds.createTrainingDataset();
  this.instanceBuilder = ds.getInstanceBuilder();
  this.availableAttributeNames = new ArrayList<String>();
  nn = createNeuralNetwork();
}
public Concept classify(String transactionId) {❷
  setVerbose(true);
  Transaction t = ds.findTransactionById(transactionId);
  return classify(t);
}
public Concept classify(Transaction t) {
  return classify(instanceBuilder.createInstance(t));
}
public Concept classify(Instance instance) {❸
  double[] x = createNNInputs(instance);
  double[] y = nn.classify(x);
  Concept c = createConceptFromNNOutput(y);
  return c;
}
public boolean train() {❹
  if( ts == null ) {
    throw new RuntimeException("Can't train classifier -
➥ training dataset is NULL.");
  }
  if( nn == null ) {
    throw new RuntimeException("No Neural Network found.");
  }
  if( nn.getInputNodeCount() != availableAttributeNames.size()) {
    throw new RuntimeException("Number of attributes doesn't match");
  }
  if( nn.getOutputNodeCount() != 1) {
```

Download at http://www.pin5i.com/


```
    throw new RuntimeException("Classifier expects network
➥ with only one output node.");
  }
  trainNeuralNetwork(nTrainingIterations);
  return true;
}
private void trainNeuralNetwork(int nIterations) {❺
  for(int i = 1; i <= nIterations; i++) {
    for(Instance instance : ts.getInstances().values()) {
      double[] nnInput = createNNInputs(instance);
      double[] nnExpectedOutput = createNNOutputs(instance);
      nn.train(nnInput, nnExpectedOutput);
    }
  }
}
public double[] createNNInputs(Instance instance) {❻
  int nInputNodes = nn.getInputNodeCount();
  double[] x = new double[nInputNodes];
  for(int i = 0; i < nInputNodes; i++) {
    String attrName = this.availableAttributeNames.get(i);
    Attribute a = instance.getAttributeByName(attrName);
    if( a instanceof DoubleAttribute ) {
      x[i] = (Double)a.getValue();
    } else {
      if( a == null ) {
        throw new RuntimeException("Failed to find attribute with name:
➥ '"+attrName);
        } else {
      throw new RuntimeException("Invalid attribute type.");
     }
    }
  }
  return x;
}
public double[] createNNOutputs(Instance i) {❼
  int nOutputNodes = nn.getOutputNodeCount();
  double[] y = new double[nOutputNodes];
 if( TransactionConcept.CONCEPT_LABEL_FRAUD.equals(i.getConcept().getName()) ) {
```

Download at http://www.pin5i.com/

210

```
    y[0] = 1;
  } else {
    y[0] = 0;
  }
  return y;
}
private Concept createConceptFromNNOutput(double[] y) {❽
  double threshold = 0.5;
  Concept c = null;
  if( y[0] >= threshold ) {
    c = new TransactionConcept(TransactionConcept.CONCEPT_LABEL_FRAUD);
  } else {
    c = new TransactionConcept(TransactionConcept.CONCEPT_LABEL_VALID);
  }
  return c;
}
public void useDefaultAttributes() {❾
    trainOnAttribute(TransactionInstance.ATTR_NAME_N_TXN_AMT);
    trainOnAttribute(TransactionInstance.ATTR_NAME_N_LOCATION);
    trainOnAttribute(TransactionInstance.ATTR_NAME_N_DESCRIPTION);
  }
}
```

你已经在代码清单 5.13 中见过 NNFraudClassifier 类的基本方法，为了简洁起见，这里剔除了文档、getters 方法、setters 方法等。如你所见，这个分类器是对几个更基本的类的包装，以使得我们可以把交易欺诈这个案例映射到基准的“实例到概念”这个框架中。以下按其出现顺序对这些方法进行注释：

❶ 构造函数获取交易数据集的引用，并生成一个对象作为后续分类所用。要记得我们的数据是交易，所以需要为它们创建实例，才能用于分类算法，这就是 TransactionInstanceBuilder 类所做的事。createNeuralNetwork()方法的调用创建了 TransactionNN 类的一个实例，我们会在代码清单 5.15 中对它进行介绍。

❷ 基于这个分类器的特定用途，classify 方法对原函数进行了重载。按照 iweb2.ch5.classification.core.intf.Classifier 接口的定义，一个分类器有义务提供一个实现，它以一个 Instance 对象作为其唯一的输入参数，并返回一个 Concept 对象。为方便分类器的使用，我们提供了额外的 classify 方法，但最终它会调用主要的 classify 方法。

Download at http://www.pin5i.com/

❸ 这是分类器必要的方法，神经网络会接受一个 double 型的数组作为输入，然后产生一个 double 型的数组作为输出。它的实现分三步，第一步是把一个交易实例转换为一个 double 型的数组；第二步是把这个数组输入神经网络并获取神经网络的分类结果（单一的一个 double 值）；第三步，因为我们并不关心神经网络返回的精确的 double 值，而只希望分类器告诉我们这个实例是否属于欺诈，所以需要把 double 值转换为两个 Concepts 中的一个——CONCEPT_LABEL_FRAUD 或是 CONCEPT_LABEL_VALID，这就是 createConceptFromNNOutput 方法所做的事情。

211

❹ 这是 NNFraudClassifier 的训练方法，它返回一个训练好的神经网络。但在把计算交给主训练方法之前，这个方法会进行一系列的检查。具体地说，它会检查如下条件：

- 训练集的存在。
- TransactionNN 实例的存在。
- 输入与 TransactionNN 实例规范的一致性。
- 输出与 TransactionNN 实例规范的一致性。

❺ 这个方法是训练的主体部分，它需要一个唯一的参数来确定训练集里的实例应该在神经网络中传播的次数。每一个实例的输入都会导致神经网络突触的权值被修正，以使神经网络对于所有已知的实例得到最优的分类结果。换句话说，你不断地告诉神经网络对应于给定输入的应答是什么，它就会尝试着调整自身以使其能“记得”这个应答，同时也不会忘记之前见过的所有其他输入的应答。

❻ 这个辅助方法以一个 Instance 作为参数，并为神经网络生成输入值。

❼ 这个辅助方法以一个 Instance 作为参数，并为神经网络生成输出值。它只被用于训练阶段。

❽ 这个辅助方法以神经网络的输出值作为参数，并把它转换为两个概念之一——CONCEPT_LABEL_FRAUD 或 CONCEPT_LABEL_VALID。

❾ 这个辅助方法定义用于分类的交易实例的属性。在这个例子里，我们并不想要太多的属性，而这个包装简化了我们的脚本。一般来说，把训练属性列表定义在代码的同一个地方是更方便和明智的。还可以为 availableAttributeNames 变量编写一个 getter 方法。

现在，你可能已经对我们这个基于神经网络的欺诈分类器的顶层定义有了一个很好的理解。但是，这个神经网络是怎么定义的呢？如果你想基于不同的神经网络

Download at http://www.pin5i.com/

来编写自己的欺诈检测分类器，你又该遵循什么步骤？代码清单 5.15 显示了来自 TransactionNN 类的代码，这是我们在欺诈分类器中所使用的神经网络。但就如你所看到的，在这个类的定义中并没有什么特别与欺诈或交易相关的东西，它只传递了一个信号，那就是该如何把欺诈检测转换为神经网络框架可解的问题。 212

代码清单 5.15 一个特别的用于欺诈检测案例的神经网络

```
public class TransactionNN extends BaseNN {❶
  public TransactionNN(String name) {❷
      super(name);
      createNN351();
  }
  private void createNN351() {
      Layer inputLayer = createInputLayer(❸
            0, // 层 Id
            3 // 节点数量
            );
      Layer hiddenLayer = createHiddenLayer(❹
            1, // 层 id
            5, // 节点数量
            new double[] {1, 1.5, 1, 0.5, 1} // 节点偏移值
            );
      Layer outputLayer = createOutputLayer(❺
            2, // 层 id
            1, // 节点数量
            new double[] {1.5} // 节点偏移值
            );
      setInputLayer(inputLayer);
      setOutputLayer(outputLayer);❻
      addHiddenLayer(hiddenLayer);
      setLink("0:0", "1:0", 0.25);❼
      setLink("0:0", "1:1", -0.5);
      setLink("0:0", "1:2", 0.25);
      setLink("0:0", "1:3", 0.25);
      setLink("0:0", "1:4", -0.5);
      setLink("0:1", "1:0", 0.25);
      setLink("0:1", "1:1", -0.5);
      setLink("0:1", "1:2", 0.25);
```

Download at http://www.pin5i.com/


```
        setLink("0:1", "1:3", 0.25);
        setLink("0:1", "1:4", -0.5);
        setLink("0:2", "1:0", 0.25);
        setLink("0:2", "1:1", -0.5);
        setLink("0:2", "1:2", 0.25);
        setLink("0:2", "1:3", 0.25);
        setLink("0:2", "1:4", -0.5);
        setLink("1:0", "2:0", -0.5);
        setLink("1:1", "2:0", 0.5);
        setLink("1:2", "2:0", -0.5);
        setLink("1:3", "2:0", -0.5);
        setLink("1:4", "2:0", 0.5);
        System.out.println("NN created");
    }
}
```

213

基于这个代码清单，我们的欺诈检测神经网络中定义的各步骤如下：

❶ TransactionNN 类是 BaseNN 类的扩展，我们随后会深入描述这个基类。你可以通过扩展 BaseNN 来创建一个通用的神经网络。在本书所带的代码中，你还能找到一个实现了异或门（XOR gate）的神经网络，即它以两个 double 值作为输入，并输出一个 double 值。如果所有的输入值都大约等于 1 或都大约等于 0，输出为 0；如果一个输入值大约等于 1，而另一个大约等于 0，输出为 1。这个类被称为 XORNetwork，它比 TransactionNN 类还要简单。阅读并执行它的代码可以加强你对如何建立一个神经网络的理解。

❷ 构造函数把所有基本的初始化步骤交给 BaseNN 的构造函数来处理，并创建一个特定的网络拓扑：三个输入节点、五个隐层节点与一个输出层节点。等一下！三个输入节点？我们之前描述的交易数据可是拥有大量属性值的，包括用户 ID、交易金额、用两个坐标表示的交易地点和交易描述字符串。描述字符串不是数值，但可以通过多种方法把它转换为数值，它在输入数据中至少占据一个节点是没有问题的，这样，至少就有五个输入数据了，那为什么我们只使用三个输入节点呢？

我们看看这中间都发生了什么转换。首先，传进神经网络的输入数据都是经过标准化的，为了确信这一点，你可以查看 TransactionInstanceBuilder 类的 createInstance（Transaction t）方法。交易金额通过合法交易的最大最小值来进行标准化，这样，它得到的值总会在 0～1 之间。对于交易描述，我们用雅可比系数（JaccardCoefficient）

Download at http://www.pin5i.com/

也得到了同样标准化的结果。对于交易地点，要做得更复杂一点，我们先对用户中心点的位置与交易地点做标准化（基于 x 与 y 坐标的最大最小值），然后计算这两个标准化地点的距离，这个距离就作为 TransactionNN 这个神经网络三个输入值中的一个。这就说明了为什么我们只需要三个输入节点。很明显，这只是一个设计上的选择，在神经网络设计中也是常有的事。尽管看起来或多或少地有点随意性，但这并非是一个糟糕的选择，你可以验证这一点，实际上，本章中的一个 To Do 事项里就让你去做这个工作。

总体的网络拓扑（3/5/1 节点，只有一个隐层，全连接）同样也不是一成不变的一个设计决策，但它可以通过实验来优化。你尝试不同的拓扑结构，就可以得到不同的分类器。

在与数据标准化同一个 To Do 事项里，我们还强烈要求你去实现自己的网络拓扑，并比较它们生成的分类器的结果。最优的网络拓扑依赖于输入数据的性质以及待解决问题的本质，欺诈检测这个案例以及基于 TransactionNN 的实现提供了一个很好的基准，你可以从中研究它们之间的依赖性。 214

❸ 以 ID 号为参数，我们生成了一个输入层，并且设定输入层应该有三个节点。

❹ 为网络创建隐层。注意，这次我们引入了一个数组作为 biases 参数（数组中的每个值对应于隐层中的一个节点），这是一个新的参数。之前我们已经谈到突触的权值，现在，可以把这些值看做是额外的权值常量，它们表示节点输出值应该加上的偏移量。

同样要注意下面添加隐层对象时使用的前缀是 add 而非 set。这样的意图是为了说明你可以拥有不止一个隐层，作为神经网络设计的一个方面，我们建议你研究一下不同隐层数量所产生的不同效果。

❺ 这是三层里的最后一层。我们同样为输出层定义一个偏移值，你也可以选择不为输出节点设置偏移值。在后一种情况下，只要简单地把偏移设置为 0 即可。

❻ 我们把三个层的引用赋值给网络。到这个时候，所有的节点已经就绪，剩下的事情就是为网络创建连接（边）。

❼ 我们逐个为节点间建立连接（突触）。第一个参数决定连接的起始点，形式为 LayerID:NodeID；第二个参数决定连接的目标点，形式同样为 LayerID:NodeID；第三个参数为连接初始化权值。前文我们已经叙述过，权值会在训练过程中连续不断地发生改变。

Download at http://www.pin5i.com/

5.4.5 创建通用神经网络的基类

前一节叙述的内容仅限于欺诈检测这么个特殊的案例中。为了创建神经网络，以及在每次需要的时候都能访问到神经网络的内部功能，我们需要把调用交付给神经网络的通用实现——BaseNN 类。因为它的重要性以及广泛的适用性，本节会对这个类进行详细讲解。

为了更好地叙述，我们会把这个类分成两个代码清单来展示。代码清单 5.16 是结构层面的主题（配置生成神经网络），代码清单 5.17 展示了操作层面的主题（训练、分类相关的代码）。

代码清单 5.16 BaseNN（结构层面）：通用神经网络基类代码摘录

```
public Layer createInputLayer(int layerId, int nNodes) {❶
  BaseLayer baseLayer = new BaseLayer(layerId);
    for(int i = 0; i < nNodes; i++) {
      Node node = createInputNode(layerId + ":" + i);
      Link inlink = new BaseLink();
      inlink.setFromNode(node);
      inlink.setWeight(1.0);
      node.addInlink(inlink);
      baseLayer.addNode(node);
    }
    return baseLayer;
  }
  public Layer createHiddenLayer(int layerId,
  int nNodes, double[] bias) {❷
    if( bias.length != nNodes ) {
      throw new RuntimeException("Each node should have bias.");
    }
    BaseLayer baseLayer = new BaseLayer(layerId);
    for(int i = 0; i < nNodes; i++) {
      Node node = createHiddenNode(layerId + ":" + i);
      node.setBias(bias[i]);
      baseLayer.addNode(node);
    }
    return baseLayer;
  }
public Layer createOutputLayer(int layerId,
```

215

Download at http://www.pin5i.com/


```
➡ int nNodes, double[] bias) {❸
    if( bias.length != nNodes ) {
      throw new RuntimeException("Each node should have bias.");
    }
    BaseLayer baseLayer = new BaseLayer(layerId);
    for(int i = 0; i < nNodes; i++) {
      Node node = createOutputNode(layerId + ":" + i);
      node.setBias(bias[i]);
      baseLayer.addNode(node);
    }
    return baseLayer;
  }
  public void setLink(String fromNodeId, String toNodeId, double w) {❹
    Link link = new BaseLink();
    Node fromNode = allNodes.get(fromNodeId);
    if( fromNode == null ) {
      throw new RuntimeException("Unknown node id: " + fromNodeId);
    }
    Node toNode = allNodes.get(toNodeId);
    if( toNode == null ) {
      throw new RuntimeException("Unknown node id: " + toNodeId);
    }
    link.setFromNode(fromNode);
    link.setToNode(toNode);
    link.setWeight(w);
    fromNode.addOutlink(link);
    toNode.addInlink(link);
  }
  protected Node createInputNode(String nodeId) {❺
    Node node = new LinearNode(nodeId);
    node.setLearningRate(learningRate);
    return node;
  }
  protected Node createHiddenNode(String nodeId) {
    Node node = new SigmoidNode(nodeId);
    node.setLearningRate(learningRate);
    return node;
  }
  protected Node createOutputNode(String nodeId) {
```

216

Download at http://www.pin5i.com/


```
        Node node = new LinearNode(nodeId);
        node.setLearningRate(learningRate);
        return node;
    }
    public abstract double fireNeuron();
    public abstract double fireNeuronDerivative();
}
```

我们先从代码清单 5.16 的结构层面开始。这并非是其全部的实现，我们保留了需要描述神经网络结构最少的方法。

❶ 这个方法为网络创建输入层，它以层的 ID 和节点数量作为参数，实例化一个 BaseLayer 对象，这个对象是我们框架里神经网络层的基础实现，它由一个层 ID 和一个节点列表组成。接下来，循环会迭代 nNodes 次以创建输入层的所有节点。输入层的每个节点都被分配一个连接（突触），我们称为入链（inlink），以说明它负责把数据传输进网络。连接的权重设置为 1，并且在训练过程中也不发生改变，因为我们不希望扭曲数据的原值。因为这个原因，很多作者都不会把输入层视为神经网络自身的一部分。

❷ 这个方法为网络创建隐层，它以层的 ID、节点数量以及这些节点的偏移值作为参数。在确认偏移值的量跟节点数量一致之后，它实例化了一个 BaseLayer 对象，接下来循环会迭代 nNodes 次，为隐层创建所有的节点。隐层的每个节点按照循环迭代的顺序都被分配一个偏移值，因为这是构造阶段，我们假设这就是有意的顺序。与输入层节点不一样的是，这一步不会生成连接（突触），所以也不用提供权值。权值的设定是通过 setLink 方法单独进行的，我们很快就会看到。

❸ 通过构造输出层来为神经网络各层的构建做个收尾。这一步类似于隐层的构造，潜在的区别在于输入层的节点是 LinearNode 类的实例，而隐层节点是 SigmoidNode 类的实例。

217

❹ 之前的方法负责建立神经网络节点，而这个方法则负责生成神经网络的连接（突触）。之前拥有连接的只有输入层节点，剩下的节点都通过这个方法来连接。它的参数是需要连接的两个节点的 ID，以及连接的权值。你也可以定义其他的方法，比如 connectFully(Layer x, Layer y)用于为两层间所有可能组合的节点生成连接。试着根据自己的需要去实验和探索各种可能性。

❺ 代码清单 5.16 剩下的方法负责生成神经网络节点的具体实现，以 BaseNode 为抽象类，我们为它写了两个具体的实现。第一个实现通过 LinearNode 类给出，并

Download at http://www.pin5i.com/

由输入与输出层使用。第二个实现通过 SigmoidNode 类给出，并由隐层节点来使用。一旦节点创建完毕，我们会设置学习率（learning rate）。几乎每一个节点所有的功能都由基类来提供，LinearNode 与 SigmoidNode 只提供了两个方法的实现——fireNeuron()与 fireNeuronDerivative()。你是否还记得在 5.4.2 节提出的设计准则：我们可以通过定义网络结构、激活规则与学习规则来充分决定一个神经网络。为神经网络创建层、节点、节点间的连接就建立了网络的结构，但还没确定节点是如何对一个给定的输入做出响应的（激活规则），或者网络是怎么学习的（学习规则）。fireNeuron()方法就定义了神经元节点是如何对一个给定输入做出响应的，这正是激活规则的关键所在；而 fireNeuronDerivative()方法（它必须提供 fireNeuron()方法的数值版本。）则直接与学习规则相关，学习率 learningRate 通常取 0～1 之间的值，这个参数并不依赖于节点的具体实现。

上述方法从结构层面对神经网络做了充分的定义。接下来，让我们把视线转移到代码清单 5.17，看看从操作层面上对网络的描述。

代码清单 5.17 BaseNN（操作层面）：通用神经网络基类代码摘录

```
public void train(double[] tX, double[] tY) {
  double lastError = 0.0;
  int i = 0;
  while( true ) {❶
    i++;
    double[] y = classify(tX);
    double err = error(tY, y);
    if( Double.isInfinite(err) || Double.isNaN(err) ) {❷
      throw new RuntimeException("Training failed.");
    }
    double convergence = Math.abs(err - lastError);
    if(err <= ERROR_THRESHOLD ) {❸
       lastError = err;
       break;
    }
    if( convergence <= CONVERGENCE_THRESHOLD ) {❹
       break;
    }
    lastError = err;
    outputLayer.setExpectedOutputValues(tY);❺
```

Download at http://www.pin5i.com/


```
      outputLayer.calculateWeightAdjustments();
      for(Layer hLayer : hiddenLayers) {
         hLayer.calculateWeightAdjustments();
      }
      outputLayer.updateWeights();❻
      for(Layer hLayer : hiddenLayers) {
         hLayer.updateWeights();
      }
    }
  }
  public double[] classify(double[] x) {❼
    inputLayer.setInputValues(x);
    inputLayer.calculate();
    inputLayer.propagate();
    for(Layer hLayer : hiddenLayers) {
      hLayer.calculate();
      hLayer.propagate();
    }
    outputLayer.calculate();
    double[] y = outputLayer.getValues();
    return y;
  }
```

每个神经网络都有两个主要的操作特性：它必须能训练自身，它也应该可以对输入进行分类——生成期望的输出。在我们的实现里所采取的算法被称为后向传播（back propagation）算法，这是一个在线[3]的梯度下降学习算法。实际操作中，这个算法检查每一个训练实例，并依此调整自身连接的权值（突触），以使输出值与期望值之间的误差最小化，最小化过程依赖于对误差斜率的计算。对于每个实例，我们会进入一个无限循环中，有三种条件可以让我们跳出循环。

❶ 进入一个开环回路中，在这里我们需要提升分类器的精确度，终止条件在 2～4 点中描述。

❷ 第一个终止条件通过计算误差实现，如果无法计算误差，则再进行迭代就毫无意义。这只是一个简单的检查性测试。如果这个条件恰好为真，那么通常意味着是一个错误的神经网络设计，或是你把问题转换为一个神经网络分类问题的尝试是

3 即一个输入一次训练。——译者注

Download at http://www.pin5i.com/

错误的。

❸ 第二个终止条件与误差的大小相关，如果这个实例的分类误差已经小于一个预设的阈值，我们就可以停止了。 219

❹ 第三个终止条件检查随着迭代次数的增加，得到的误差减少是否显著。我们希望减少误差，所以一直修改着权值希望能得到一个更优的输出值。但很有可能无法到达那个预设的误差阈值。换句话说，也许为分类器设置的门槛过高了。看看我们处理这个情况的实现方法，你能想出一个更好的收敛判定准则来吗？

❺ 到现在，我们已经见过所有的终止条件了。所以把输出节点的值设置为期望值，并开始对网络权值进行重新评估。从输出层开始，为各层调用 calculateWeightAdjustments()方法。

❻ 在前一步我们计算了权值的调整量，但还没具体应用这些调整量。这一步，在为所有的节点都完成权值调整计算之后，通过调用 updateWeights()方法来实现权值的更新。就是这样，一次迭代结束，并已经准备好进行下一次的迭代，直到碰到三个终止条件中的一个为止。

❼ 这个方法是对分类过程的顶层进行包装。如我们早前所述，当神经网络开始运作时，可以把信息看做从输入节点进入，流经整个网络，然后从输出节点出来，这个过程被简洁地捕获在这个方法里。我们先从输入层节点开始，转移到隐层，然后计算出网络输出值，结束这个过程。每个节点会基于权值与偏移做其自身的计算，这由 calculate()方法来负责。节点会通过 propagate()方法把它的输出传送到与之相连的节点。一旦神经网络训练完毕，这些操作都是非常直观的。

还有大量依赖于数学预备知识的内容在这里没有提及，因为作为一本面向普通读者的书籍来说，这是不需要的。我们聚焦于神经网络分类器的工作机制，而非它们的基础。如果你有兴趣学习更多的神经网络的内部工作机制，那么有大量的文献著作可作为你的参考。附录 E 列出了很多神经网络方面好的著作，可以帮你扩充在这个领域的知识面。

前一节聚焦于分类算法的定义与描述，以创建一个分类器。从一个产品的角度来说，还有一些很重要的议题，如分类的可信度、分类结果在大数据集中的一致性，还有分类时的计算量也要加以考虑。我们会在下面两节讨论其中的一些议题。

Download at http://www.pin5i.com/

5.5 你的结果可信吗?

假定你基于贝叶斯理论、神经网络或其他技术建立了自己的分类器。你如何得知自己是否干了一项漂亮的工作呢？你如何得知是否可以把自己的智能模块应用于生产环境中，并获得同行的景仰以及老板的赞赏呢？评估分类器和创建它一样重要，如同在销售会议上，你会听到大量的夸大之词及纯粹的废话，但没有评估就没有结论。本节的目的在于帮助你评估自己的分类器，如果你是一个开发者或产品经理，这有助于你理解第三方产品的合理与否。

220

“没有人知道所有的事情”、“人都会犯错”。这些箴言在计算机领域也有其对应的版本：没有一个单分类器可以解决所有的问题，也没有一个单分类器在所有的数据集中都能良好地工作。在分类范畴中的学习技术属于有监督学习（supervised learning）（无监督学习的例子可参看相关的 To Do 事项），学习是“有监督”的，是因为分类器会利用已知的分类结果历经一个训练的过程，通过这种监督，它会尝试着学习蕴含在训练数据集中的信息。你可以想象得到，训练数据集与你部署环境中实际数据的相关性会是分类是否成功的关键。

为了表述清晰，这里引入一些术语。为让事情简化一些，我们考虑一个标准的二分类问题：辨识垃圾邮件，例如，让我们假定要辨别一个特定的邮件是否应该标定为垃圾，一个评估分类器可信度的基本工具是混淆矩阵（confusion matrix），也以它作为这个调查的起始点。这是一个简单的矩阵，行对应于分类器给一个特定实例所分配的类别，列对应于实例实际所属的类别。在二分类中，矩阵只有四个单元格，在通常情况下（多分类），本质上跟二分类问题没有什么区别，只是需要更复杂的分析而已。

表 5.1 展示了一个针对于垃圾邮件过滤或欺诈检测的混淆矩阵。这个表捕获了所有二分类中可能出现的情况。如果分类器把一个特定的邮件标识为垃圾类别，那我们称这个分类是正（positive）的；否则我们说这个分类是负（negative）的。当然，分类的本身有可能是正确（True）或是错误（False）的。所以矩阵包含四种可能的结果——positive/negative 与 true/false 的组合结果。这同样让我们意识到有两种错误情况的存在，一种错误由 false positive 的分类结果组成，这类错误称为第一类错误（type I error）；另一种错误由 false negative 的分类结果组成，这类错误称为第二类错

Download at http://www.pin5i.com/

误（type II error）。用简单易懂的话来说，当你犯了第一类错误时，你把无辜者判定为有罪了；当你犯了第二类错误时，你把罪犯给放跑了！这个类比特别适合用于指出分类成本（classification cost）的重要性。伏尔泰说他宁可释放 100 个有罪的人，也不要冤枉一个好人，这种敏感性同样存在于欧洲法庭中。这种道德轶事的存在在于决策都是有后果的，而后果的严重程度是不一样的。这在多分类问题中尤其是真理，我们后面会再次讨论这个问题。 221

表 5.1 简单二分类问题中典型的混淆矩阵

	Positive	Negative
True	True Positive (TP)	True Negative (TN)
False	False Positive (FP)	False Negative (FN)

基于表 5.1 中的值，我们引入如下的定义：

- FP rate = FP / N，其中，$N = TN + FP$
- Specificity = 1 – FP rate = TN / N
- Recall = TP / P，其中，$P = TP + FN$
- Precision = $TP / (TP + FP)$
- Accuracy = $(TP + TN) / (P + N)$
- F-score = Precision * Recall

按照这个定义，假设我们得到一个分类器的精确度（accuracy）为 75%，那么这个估计距离分类器真正的精确度有多远呢？换言之，如果你用不同的数据重复分类任务，你有多大的可能性得到的精确度还是 75%？为了回答这个问题，我们需要求助于统计学中的伯努利过程（Bernoulli process），它被描述为一系列的独立事件，这些事件的结果可能是成功或者失败的。对于我们的垃圾邮件过滤案例与欺诈检测案例来说，这正是一个极好的例子，当然对于一般性的二分类问题都是适用的。如果你把真实的精确度表示为 A*，测量精确度表示为 A，那么你希望知道 A 是否是 A* 的一个良好估计。

你也许还记得在统计课中学到的置信区间（confidence interval）这个概念，这是一个对于指定断言的确定性的度量。假设我们的精确度为 75%，如果这是在 100 封邮件集中得到的结果，这个置信度不算高，但如果是在 100 000 封邮件集中得到的结果，这个置信度会高得多。直观上，我们能理解这一点，当数据集的规模增加时，置信区间会变小，我们对结果会更确定。具体地说，对于 100 个采样的伯努利

Download at http://www.pin5i.com/

过程，真实的精确度会以 80%的置信度落在 69.1%～80.1%之间(参看 Witten & Frank 的文章)。如果我们把用于度量分类器精确度的集合扩大 10 倍，那么在同样 80%的置信度下，得到新的置信区间是 73.2%～76.7%。任何一本合格的统计教科书都会给出计算这个区间的公式。理论上说，当你的样本量大于 30 个实例时，这些结果都是有效的。在实际中，你应该使用尽可能多的实例。

不幸的是，实际中未必能有你想要的那么多的实例。为了解决这个挑战，机器学习领域的人们设计了大量的技术来帮助我们评估数据量不足时分类结果的可信度。标准的评估方法被称为 10 步交叉验证（10-fold cross-validation）。这个简单的过程可以通过一个例子来说明，假设我们有 1 000 封已经手工分类好的邮件，为了评估我们的分类器，我们需要用其中一部分作为训练集，另一部分作为测试集。10 步交叉验证告诉我们可以把 1 000 封邮件分为 10 组，每组 100 封。每一组中的 100 封邮件包含合法邮件与垃圾邮件的比例应该与 1 000 封邮件集中相应的比例大致相同。随后，我们用其中的 9 组邮件对分类器进行训练。一旦训练完成，就用不存在于训练集中的那一组邮件来测试分类器。我们可以计算不同的质量度量，有一些在之前已经提到过，最典型的就是分类器的精确度了。把这个过程重复 10 次，每一次都留下不同的邮件组作为测试集。在这些试验的最后，我们就拥有了 10 个精确度的值，对它们取平均，就得到一个精确度的平均值。

222

你也许会问，如果把原数据集分割为 8 份或者 12 份，精确度是否会改变。是的，当然会，你不太可能会获得一样的结果。然而，新的精确度的平均值应该与你之前获得的平均值很接近。通过大量的实验用不同的分类器在多个数据集上的验证，结论是 10 步交叉验证可以得到对分类器相当具有代表性的测量值。

考虑 10 步交叉验证的极端情况，把除一个实例外的所有邮件作为训练集，然后把剔除的实例作为测试。很显然，这种技术就被称为留一法（leave-one-out）。这种方法有一些理论上的优势，但在实际数据集中（数十万甚至上百万的实例），这种方法的计算成本通常是无法接受的。你可以选择剔除一个实例来做实验，但不要以此应用到数据集中所有的实例。这个技术就叫 bootstrap，bootstrap 的基本思想是我们可以通过采样原数据集的一些实例，对它们做置换，得到一个新的训练集。换句话说，我们可以对原数据集中的实例使用多次，例如，创建一个有 1 000 封邮件的训练集，其中某封邮件可能出现过不止一次。如果你照这样做了，那么最终你也许会得到一个有 368 封未用于训练的邮件测试集，而你的训练集仍然保持有 1 000 封邮

Download at http://www.pin5i.com/

件，因为剩下的 632 封邮件中的某一些在训练集中有重复[4]。对于这些数字，更多的数学上的解释可参看 Witten & Frank 的文章。

据了解，把 *TP* 率（TP rate，TPR）相对于 *FP* 率（FP rate，FPR）画出来有助于分析分类器的可信度。这类图被称为 ROC 曲线（ROC curves），源自于 20 世纪 70 年代的信号检测理论。ROC 曲线的基本思想在于它应该尽量远离 TPR/FPR 这条对角线。关于 ROC 图的分析，我们会推荐来自 Tom Fawcett 优秀的技术报告，它包含了伪算法说明及实际使用中的很多小技巧。

在真实的世界里，分类系统通常被用做决策支持系统，错误的分类会导致错误的决策。在一些案例里，错误的决策虽然不是我们想要的，但相对来说还是没有太大祸患的；但在另一些案例中，这可能就是生与死的差别了，想想如果一个内科医生漏过了一个癌症的诊断，又或者一个太空宇航员遭遇紧急情况急需等待你的分类器的结果。分类系统的评估应该同时检查可信度及做出分类时的对应成本。对于二分类问题，方法是指派一个由 FPR 与 FNR 计算得到的成本函数（cost function）。对于多分类问题的成本设定，可参看相应的 To Do 事项。 223

总的来说，一个分类器最重要的方面在于它结果的可信度。这一节我们描述了若干度量以帮助我们计算分类器的可信度，如准确度（precision）、精确度（accuracy）、召回率（recall）以及特异性（specificity）。这些度量的组合可以得到新的度量，如 F 值。我们还讨论了通过不同的方法分割训练集对结果进行交叉验证的方法，可以从中看到随着数据集的变化对这些分类度量指标的影响。我们讨论了 ROC 曲线的概念，它即是 TPR 与 FPR 的简单绘图。在 5.6 节里，我们会讨论若干与大数据集相关的主题。

5.6 大数据集的分类

相对于现实世界中的情况，很多用于学术与研究目的的数据集都很小。大公司的交易数据集的规模至少都会在千万条到亿条记录之间，保险索赔、电信日志文件、股票价格记录、点击流日志、审计日志等（这个列表很长）都在同样的数量级里。所以，在生产环境的应用中处理大数据集是常态而非意外，不管它们是否是基于 Web 的。对极大数据集的分类如此值得我们给予特别关注的（至少）有三个原因：（1）

4 也即放回式抽样得到的结果。——译者注

Download at http://www.pin5i.com/

训练集中数据的代表性；（2）训练阶段的计算复杂性；（3）分类器运行在极大数据集时的表现。

不管应用所在的具体领域与分类器所支持的功能如何，你都必须保证训练数据对在生产环境里遇到的数据具有代表性。你也不应期望一个分类器会表现得同验证阶段的指标所指示的那样好，除非训练数据对生产数据具有极强的代表性。我们要不断地强调这一点！在很多情况下，早期的兴奋很快地就转变为失望，仅仅只是因为遇到从未遇到过的情况。所以，你当然想知道，到底我应该怎样才能确信训练数据具有代表性呢？

二分类的情况很好处理，因为它们只有两个类别——一封邮件是否是垃圾的，一个交易是否是欺诈的，等等。既然如此，我们就假定你从两个类别中都收集到足够数量的训练实例，我们的重点应该是训练集中实例属性值的覆盖范围。你的评估可以纯粹是经验主义的（“对，这已经足够好了，因为我们有了足够的值，把它放到生产环境去吧！”）、完全科学主义的（持续地对数据进行采样，并测试采样数据跟训练数据是否来自相同的统计分布），又或是介于这两种极端的中间地带。实际中，最后一种方案是更可能的，我们称这为监督学习的半经验之路。从经验方面说，在评价训练集的完整性的过程中，你做出若干合理的假设，用以反映你对应用所使用数据的理解与期望。从科学方面说，你应该为你的数据计算一些基本的统计信息，比如最大最小值、均值、中值、有效奇异值、属性值中缺失数据的百分比等，你可以利用这些信息对应用中那些从未见过的数据进行采样，并把它们包含到训练集中。

224

多分类的情况与二分类在原理上是类似的，但除了之前提到的指导方针外，我们现在要面临一个额外的挑战，这个新的挑战就是我们挑选的训练实例要使所有的类别都同样具有代表性。相对于二选一的情况，要识别 1 000 个不同的类别是一件要难得多的任务。对于多维（多属性）多类的分类问题，还有因维度灾难带来的额外的问题（见第 4 章）。

如果你的数据库包含 1 亿条记录，你当然希望发挥所有数据的优势，利用好蕴含于其中的信息。所以，在分类器的设计阶段，你就应该为分类器在训练与验证阶段的可扩展性做好规划，如果把训练数据翻倍，那么问问自己：

- 训练分类器需要多长时间？
- 分类器在新（更大）的数据集中的精确度是多少？

Download at http://www.pin5i.com/

你可能还想包含更多的质量指标，而不单是精确度，你也许还想把数据规模多增加几倍（原数据的 4 倍、8 倍等）。但你也明白我们上述问题的含义，有可能你的分类器在一个小规模的采样数据集工作良好（它训练迅速并得到了很好的精确度），但当它作用于一个充分大的数据集时，其表现却急剧下降。留意这一点很关键，因为进入市场的时机总是很重要的，应用的“智能”模块应该与软件的其他部分一样遵循生产规则。

同样的道理也适用于分类器在运行时的表现，这属于它生命周期的第三个阶段——生产阶段。很可能你的分类器训练很迅速并得到很好的精确度，但如果它不能推广到生产环境，就只是徒劳而已！在分类器的验证阶段，你应该度量它的性能及其对数据规模的依赖性。我们假设你使用的分类器与数据规模的依赖是二次方的——如果数据翻倍，用于处理这些数据的时间是原来的四倍。让我们进一步假设你的智能模块会利用后台的分类器来检测欺诈交易，如果你使用 10 000 条记录作为验证，所有记录的分类可以在 10 分钟内完成，那么当你处理 1 千万条记录时，你就需要 1 千万分钟！我想你应该不会有那么多的时间，所以，要么选择别的分类器，要么提高现有分类器的性能。在生产系统中，人们通常不得不以精确度来换取速度，如果一个分类器极其精确但极其缓慢，它其实与废物差不多！ 225

要重视分类系统的个性特点，如果你使用的是基于规则的系统，也许会遇到可用性问题（utility problem），学习的过程——规则的收集——会导致生产环境中的系统整体变得缓慢起来。有很多办法去避免或至少缓解可用性问题（参看 Doorenbos 的文章），但你要充分了解它们，并确保你的实现跟这些技术是兼容的。当然，在这里性能的退化并不是唯一的问题，你还需要想办法去管理与组织这些规则，这是一个工程问题，它的解决严重依赖于应用的特定领域。一般来说，分类器的实现越复杂，你就越应该谨慎于分类器的性能特性（包括速度与质量）。

5.7 总结

分类是智能应用的主要部件之一。本章一开始展示了一系列不同形式的分类案例，并讨论与多个应用领域相关的解决方案作为参考，从图书馆分类到医疗保险手册，然后得到分类是无处不在并极具价值这样的结论。随后我们介绍了构成分类的三大模块——概念、实例与属性，这三大模块定义了本体——对某特定领域知识的

Download at http://www.pin5i.com/

完全描述，如果还有语义信息，它就可以被称为语义本体了。分类总可以看做是把“最优”的概念分派给指定实例这么一个问题。分类器间的区别在于它们表示及度量最优化分派形式的不同，然而它们都有类似的生命周期三部曲：训练、验证与生产阶段。

你也已经学习到，从广义上说，所有的分类器可以分为两类——二分类与多分类，区别在于分类器需要做出的是两个选择还是多个选择。你还学习到，按照底层技术来划分，分类器可分为统计的或结构的。我们概述了文献中最大的共性，随后从较高的层面叙述回归算法、贝叶斯算法、基于规则的算法、函数算法、最近邻算法以及神经网络。

你也学到了两个用于文本分类的强大的算法。第一个是应用于单字符串属性的朴素贝叶斯算法，第二个是 Rete 算法的面向对象的实现——Drools 规则引擎，它让我们可以声明规则并把规则用于分类的目的。很可能你的邮件客户端已经包含某种形式的规则引擎，当你声明发自*.foo.com 这个域的邮件，如果它的标题包含某个特定的单词，则可视为垃圾邮件，实质上，你就是在定义一个规则。现在，你应该已经准备好把我们的算法应用于很多其他自由的或半结构化的文本分类任务中。

226

此外，我们还介绍了计算神经网络的构建，并提出了一个基本但很鲁棒的实现，可以把它用于建立一般化的神经网络。我们提供了设计的准则，以及对数据结构的观察，还提出了反映生产数据特征的训练集与测试集的重要性。

尽管分类带来的效益是巨大的，但我们仍然需要指出，在把分类器引入实际的应用前，对可信度与计算成本相关主题的调查是很重要的。

总的来说，我们有如下结论：

- 分类算法对于构建智能应用是很重要的，因为它们能帮助我们（自动地）利用并（系统地）增强我们对这个世界的知识。
- 我们总是在针对一个参考结构进行分类，这个结构可以如二元集一样简单（真或假），也可以是一个更大的本体。
- 从最高的层面来看，分类器可以被视为统计的或是结构的。
- 分类器的选择严重依赖于你的数据以及待分类问题的本质。
- 需要给予可信度与分类成本特别的关注。
- 极大的数据集、极大的本体、在线需求，或其他关于这三点的组合都会给我们带来麻烦。
- 我们所描述的每一个分类算法都可以做好本职工作，但没有一个分类器具备

Download at http://www.pin5i.com/

提供永无差错决策的能力。实际上，如果你以一个完美主义者的眼光看待分类器，那你就要失望了！

在第 6 章，我们会着眼于几种把分类器组合起来的技术，以期能改进目前为止我们所描述过的任一单分类器的结果。

5.8 To Do

1．特化与泛化的权衡。每一个你能想到的分类算法都是以若干个变量作为输入，并得到若干个变量作为输出。输入由两种变量组成，第一种变量与实例的属性值有关，第二种变量则是若干个针对分类器的模型变量。在训练阶段，我们依据训练集的输入与输出变量来估计模型的变量。换言之，对于任何模型，我们都遵循如下原则来校正模型的参数：对于训练集里的一个输入，模型输出值要重现所期望出现的输出值。

很明显，你可以作弊，引入与数据点一样多的模型参数，这样就可以对训练集得到非常高的分类精确度，这被称为过拟合（overfitting），它可以使你的分类器专门用于这个训练集，但很有可能在与此不同的数据集里只能得到很糟糕的结果。一般来说，过拟合（我们也可以称为特化）并不是什么好事，应该尽量避免。反过来说，你也许只有很少的几个模型参数，以致无法捕获训练集中的信息量，这被称为低度拟合（underfitting）。使用更少的参数，但仍然足以表达训练集中的信息量，这样也许可以提升分类器对于未知数据（未包含在训练集中的数据点）的精确度。这样的能力通常被称为泛化能力（generalization）。

所以，一个好的分类器应当致力于在特化与泛化之间寻求一个精巧的平衡。用本章提供的数据集做实验，并引入新的实例作为测试数据。把分类器误差相对于训练集实例数量的变化绘画出来。画两条曲线，第一条描画训练集中实例的误差，第二条描画不包含在训练集中的实例的误差。你能在一个特定的分类器里观察到特化与泛化间的平衡吗？你还可以引入第三个维度用于捕获模型的复杂度，在规则引擎与决策树中表示模型的复杂度也许会比较直观一些（例如规则的数量），但你该如何表示基于贝叶斯理论的分类器（比如 `NaiveBayes` 类）的复杂度呢？对于神经网络的情况又该如何？

2．奥坎剃刀（Occam's razor）与训练属性的量。与本节事项 1 同样的道理，我们可以证明，模型中若包含的训练属性越多，得到的结果就越好。但这种方法有两

Download at http://www.pin5i.com/

个问题。第一，从实际实现的角度而言，我们的资源是有限的，可用于分类的时间不能太长。所以，分类方案应该易于维护、易于测试，并且还要能够迅速地产生结果，相信你不会希望为了一个垃圾邮件的分类判决而等待五分钟。当然，有时候也是需要长时间的计算的，比如要发现一个富含石油的地点，又或是生成一个能帮助你的用户做出关键的商务决策的报告。

使用尽量多的训练属性会带来的第二个问题是基于这样一个事实：更多的数据并不一定意味着更多的信息量。我们有很多指标来度量信息量，但且让我们绕开数学术语而用以下方式来考虑。我们一般会对某些与我们应用相关的变量的值比较感兴趣，它们可能是纳斯达克一只或多只股票的价值、一封邮件合适的类别，以及描述我们是否应该购买 eBay 上某个商品的一个布尔变量，等等。我们通常会假设有一个潜在的模型可用于描述我们面临的问题，而这个模型的解就是由我们希望计算的变量组成的。提供一个数据集和一个分类器，我们试图尽己所能去近似与估计原来的物理模型。如果我们使用的数据与物理模型无关，又或者分类器被多余的信息所淹没，那么结果是我们只能得到对于物理模型的一个歪曲的反映。当提到信息量时，我们的意思是说那些可以帮助我们改善表现底层物理模型的数据，如果你把那些对分类器精确度提升毫无效果的训练属性加进来，我们可以肯定地说新的训练属性自身并没有携带什么明显的信息量。

228

当然，在挑选训练属性时需要小心，因为不同的分类器也许能从某个特定的训练属性里挖掘出或多或少的信息。一般来说，我们可以使用奥坎剃刀准则：如果两种方法能得到一样的结果，那么最简单的那个就是最好的那个。所以，考虑 5.3 节里邮件过滤的例子，给你的分类器加入更多的训练属性。一开始你可以把我们在 5.3.1 节中使用的单一属性分割为两个，一个对应于邮件的标题，另一个对应于邮件的正文。你得到的结果是不是有了明显的不同？在信息量的范畴里，你该如何解释分类规则在邮件分类中的适用性？

3. 一个一般用途的规则引擎类。我们实现的 RuleEngine 类在 executeRules 方法里以 Email 类作为参数，这对于单一的邮件分类没有什么问题，但对于一个一般性的实现是不够的。在一般的情况里，你会在激活规则之前就把所有的事实插入到工作内存里。修改 RuleEngine 类，让它可以应对更一般的情况。

此外，创建一个能处理更复杂的规则、条目与操作的规则引擎使用案例。ClassificationResult 类可以引导你定制自己的条件与操作。要注意的是，因为 Drools 引擎面向对象的特性，所以你可以建立包含了条件与详细操作的复杂的规则。这正是一

Download at http://www.pin5i.com/

个合适的实践，你可以使用像 ClassificationResult 这样的辅助类，把代码都放在这个类里，而避免了把代码写在 Drools 规则文件中。

接下来你该如何去创建一个基于规则的一般用途的分类系统呢？想象这样一个系统，它可以把任意的文本分为多个类别，这些文本可以是邮件、Word 文档、PDF 文档等。你需要什么规则？你需要什么条件？你又该如何用代码表示它们？

4. 数据标准化的重要性及神经网络拓扑的影响。注意，我们传入神经网络的数值都是经过标准化的。交易金额基于合法交易的最大最小值进行标准化，所以它的取值总在 0～1 的区间里。我们还使用雅可比系数来使交易的描述也能得到同样的结果。至于交易地点，我们需要多做些工作。首先把用户的中心点及交易地点均做标准化，随后计算这两个地点间的距离，这个距离就作为神经网络 TransactionNN 的三个输入值中的一个。为什么我们要对数据做标准化？这一步骤重要吗？不妨用同样的代码来做实验，但输入数据不要做标准化，并且使用一样的训练数据与测试数据，这样你就可以只对算法本身进行比较。当你想了解你的改变所产生的效果时，你最好一次只改变一个因素。

229

另一个相当具有启发性的实验是网络拓扑对分类器结果的影响。最佳的网络拓扑依赖于输入数据的性质以及待解决问题的本质。欺诈检测这个用例及基于 TransactionNN 的实现提供了一个基准，可以帮助我们研究这种依赖性。实现你自己的网络拓扑，并比较由它们构成的分类器的结果。例如，你可以尝试着显式地提供 x 和 y 坐标作为网络的输入，而非每个地点到用户地点中心的距离。你还可以把交易的描述信息作为多个节点的输入，例如，把描述分词，以及每个词语与最高频的五个描述词语的相似度作为输入，这样就能得到跟描述对应的五个输入节点。或者更一般化的是取每个词语跟最高频的前 N 个描述词语的相似度作为输入，这样就得到 N 个对应于描述的输入节点。当你增加 N 时，结果会怎么变化？相比于基本的 TransactionNN 实现，这些神经网络的结果又是怎样的？

5. 非监督学习：Hebbian 学习及自组织特征映射（SOM）。在这一章，我们只覆盖了监督学习技术，因为它们既常见，又有用。但非监督学习技术同样有用，也值得你去关注。在监督学习中，你总会感觉到自己是从“后门”把答案输入进去——数学上这叫插值（interpolation），这并不是一个光明正大或值得炫耀的名词。无论如何，事实是我们告诉分类器应该懂得什么，然后它就尝试着通过修改自身的参数来同化这些知识，不管是计算先验概率与条件概率（贝叶斯方法），还是调整大量的权值（神经网络），又或是把所有的东西都“写下来”（基于规则的系统）。相反，非监督学习

Download at http://www.pin5i.com/

的惊艳之处在于它可以“记得”自己所看到的，而无须从人类那里得到反馈。

1949 年，Donald Hebb 在他的 *The Organization of Behavior* 书中介绍了一个简单的模型，很好地揭示了无须监督的学习能力。如果你没有告诉分类器什么是正确的，那它该如何工作？考虑一个神经网络，它的节点间通过对称双向连接（突触）构成全连接（这里的对称是指节点 i 到节点 j 的连接权值等于节点 j 到节点 i 的连接权值）。什么样的激活规则与学习规则可以帮助我们打造一个非监督学习的神经网络？[5]

230

6. 计算误分类的成本。实际生活中，分类系统多用于决策支持系统，所以分类出错会导致决策错误。在某些情况中，做出错误的决策尽管是不愿意看到的，但相对而言，其害处也不大。在另一些情况下，这可能就是生与死的差别了，想想如果一个内科医生漏掉了一个癌症的诊断，又或是一个在太空遭遇紧急情况并等着你的分类器结果来救命的宇航员。所以，对分类系统的评估应该把可信度及做出分类时对应的成本都考虑在内。对于二分类的情况，方法是指派一个成本函数（cost function），它是 *FP* 率与 *FN* 率的函数。对于多分类问题，你又该如何推广这一方法？

在多分类问题中，如果你有 N 个类别，而你的分类出了错，那么有 $N-1$ 种错误的可能性。所以，我们需要有方法来为 $N\times(N-1)$ 种情况指派成本。很显然，矩阵在这里最合适了。在多分类问题中，混淆矩阵是 $N\times N$ 的规模，那么你可以定义一个 $N\times N$ 的成本矩阵（cost matrix），它的对角线元素取 0 值（你不应当因分类器给出正确的结果而对它进行惩罚）。把细节设计完成并对它进行评估，例如，可以对 NaiveBayes 分类器取不同的成本矩阵进行实验与评估。

5.9 参考资料

分类法

The dewey Decimal classification (DDC) system. http://www.oclc.org/dewey/

International Classification of Diseases (ICD). World Health Organization (WHO). http://www.who.int/classifications/icd/en/.

The Library of Congress: Cataloging Distribution Service. http://www.loc.gov/cds/

Myhre, A.P., and M. L. Richardson, “A Web-based Tutorial for Teaching the Schatzker Classification for Tibial Plateau Fractures.” http://uwmsk.org/schatzker/.

Occupational Injury and Illness Classification Manual. Bureau of Labor Statistics, U.S. Department of Labor. http://www.bls.gov/iif/oshoiics.htm.

5 自组织特征映射网络 SOM 就是一个类似于这种构型的神经网络。——译者注

Download at http://www.pin5i.com/

书籍与文章

Antoniou, G., and F. van Harmelen. *A Semantic Web Primer*. The MIT Press, 2004.

Doorenbos, R.B. *Production Matching for Large Learning Systems*. Ph.D. Thesis, Carnegie Mellon University, 1995.

Dunham, M.H. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, Pearson Education Inc. 2003.

Fawcett, T. "ROC Graphs: Notes and practical considerations for researchers." 2004. http://home.comcast.net/~tom.fawcett/public_html/papers/ROC101.pdf.

Friedman-Hill, E. *Jess in Action*. Manning Publications, 2003.

Gasevic, D., D. Djuric, V. Devedzic. *Model Driven Architecture and Ontology Development*. Springer, 2006. 231

Gómez-Pérez, A., M. Fernández-López, and O. Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer, 2004.

Hastie, T., R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

Hill, T. (1996). "A note on distributions of true versus fabricated data." *Perceptual and Motor Skills*. Vol 83, pp.776-778. http://www.math.gatech.edu/~hill/publications/cv.dir/truvsfab.pdf.

Holmström, L., P. Koistinen, J. Laaksonen, and E. Oja. "Neural and statistical classifiers—taxonomy and two case studies." *IEEE Transactions on Neural Networks*, Vol 8 (1), pp. 5-17, 1997.

Maier, K.D., C. Beckstein, R. Blickhan, W. Erhard, and D. Fey. "A multi-layer-perceptron neural network hardware based on 3D massively parallel optoelectronic circuits." Proceedings of the 6th International Conference on Parallel Interconnects, pp. 73-80, 1999.

MacKay, D.J.C.. *Information Theory, Inference, & Learning Algorithms*. Cambridge University Press, 2003.

Neapolitan, R.E. *Learning Bayesian Networks*. Prentice Hall, 2003.

Papoulis, A., and S.U. Pillai. *Probability, Random Variables, and Stochastic Processes*, Fourth Edition. McGraw-Hill, 2002.

Rish, I. "An empirical study of the naïve Bayes classifier." *IBM Research Report*, RC22230 (W0111-014). http://www.cc.gatech.edu/~isbell/classes/reading/papers/Rish.pdf.

Russell, S., and P. Norvig. *Artificial Intelligence: A Modern Approach* (Second Edition). Prentice Hall, 2002.

Staab, S., and R. Studer. *Handbook on Ontologies*. Springer, 2004.

Download at http://www.pin5i.com/

6 分类器组合

本章内容提要：

- 分类器评估的基准
- 分类器比较，理解复杂的数据集
- Bootstrap 集成的具体细节
- Boosting 基础

古希腊哲学家 Epictetus 曾经说过，“人不应当把生活只寄托在一个希望上，就如同船不能只绑在一个锚上”。同样的道理，我们不应该只依赖于一个分类器。没有一个分类器有能力提供绝对可靠的决策。实际上，有大量的例子能证明分类器组合所蕴含的极大潜力，本章就将引领你进入这个激动人心的主题。在推荐系统领域（参看第 3 章），Bell、Koren 与 Volinsky 最近也引入了类似的思想，并获得了巨大的成功。

分类器组合的主要思想在于以计算复杂性的增加与计算成本的提升（如更长的计算时间或额外的计算资源）来换取更好的分类结果。分类器组合大体上可分为两类——分类器融合及分类器选择。对于分类器融合，所有的分类器对给定的分类都有贡献，所以，每个分类器都必须覆盖所有可能的数据点所分布的所有区域。对于分类器的选择，每个分类器只负责特定区域内的数据点，它们也只需要在这个自己能发生影响的区域内表现良好即可。

Download at http://www.pin5i.com/

分类器组合类型的区分有助于我们在这个领域中找到方向，但这也并非是绝对的。我们会看到（见本章 To Do 事项第 5 条）融合与选择的概念也并非总是那么易于区分的。

如果你想阅读分类器组合的理论分析，可以参看 Dietterich 的报告，该报告大致列出了使用分类器组合的三类原因。

第一类是从统计的角度考虑的，主要的论点在于分类器的组合未必能超过分类器集群里表现最好的那个单分类器，但它可以显著性地降低把单分类器应用于未知数据时的风险。

第二类是从计算的角度考虑的，它指出分类器对于训练通常很敏感，所以 Dietterich 论述说，分类器组合通过平滑化每个分类器在训练集中的敏感性，应能得到更好的结果。

第三类是表达性的。为了理解它，假设我们需要对一批简短的法语文本描述（例如 porte、fenêtre、guerre）进行分类。问题在于我们并没有一个能理解法语的分类器，而只有能理解英语和西班牙语的分类器。表达性动机的主要思想在于英语与西班牙语的组合可以很好地表达出法语来，尽管它们中单独的每一个都无法做得很好。Guerre 的英语翻译是“War”，而西班牙语是 guerra。反过来，你同样可以找到跟英语对应词比较相似而跟西班牙语对应词不那么相似的法语词汇。在我们接下来的讨论中，你也能找到这几类动机的影子。

作为本章的开始，我们会介绍贷款申请人的信贷价值评估研究案例。这是一个热门主题，因为它跟我们今天（2008 年前后）所经历的经济衰退紧密地联系在一起。具体地说，我们会探讨如何对贷款申请人的信用进行评估，这从另一个角度来看就是对信贷风险的评估。这个用例的数据是非常逼真的人造数据——因为信贷风险评估数据与模型都是私密的。很自然，首先要解决的是怎么把单分类器应用于这个用例中。所以，我们创建了三个分类器把申请人分为五个信用类别。具体来说，这些分类器包括基于贝叶斯算法的分类器、神经网络分类器和一个新引入的决策树分类器。

然后会检查分类器之间是否存在着优劣，为此，我们会介绍四种用来比较两个或多个分类器优劣的统计技术。对于两个分类器的比较，引入 McNemar 检验与差额比例（difference of proportions）检验。对于多于两个分类器的比较，引入 Cochran 的 Q 检验与 F 检验。不要担心这些颇为吓人的名字，我们会把它们的实际应用简化为几个方法的调用。

Download at http://www.pin5i.com/

234

接下来我们会展示两类分类器组合的技术。第一个被实现用于分类器组合的类别称为 bagging，它的原本形式是 bootstrap 聚合（bootstrap aggregating），我们会使用它来提升决策树分类器的精确度，并研究当分类器组合数量增加时分类结果的变化。第二种用于分类器组合的方法叫 boosting，我们会介绍一个拥有神秘名字（arc-x4）的算法，并且会讨论它的前身 AdaBoost。很自然地，我们会把这些组合技术应用于信贷价值的鉴定问题中。

6.1 信贷价值：分类器组合案例研究

本节首先介绍这个案例：为贷款申请而评估一个人的信贷价值。我们会用三个针对本问题而设计的分类器来审查申请。具体来说，是一个基于朴素贝叶斯算法的分类器、一个神经网络分类器和一个新引入的决策树分类器。但是，信贷价值是什么，为什么评估它很重要？这里有一些背景知识用以介绍案例的来龙去脉，以及强调优秀的智能应用对于全球经济的影响（也许是意想不到的影响）。

2007 年，一个历史性的信贷混乱及其对全球影响的事件在美国被披露出来。人们被允许（也许还有人鼓励他们这样做）享受他们不应得的信用额度，去购买他们根本承担不起的房子。当房价下跌、市场资本流动紧缩时，那些其实没有购买力的人们开始拖欠贷款。我们不知道有多少取消抵押品赎回权及抵押房产被迫出售的事件被催生，在本书撰写的时候危机仍在持续。

为了避免更深层次的经济困境与社会动荡，美国政府正设法让大量的购房者能保住自己的房子。据估计，有两百万房主的贷款是浮动利率的，其中很多人的初始固定利率都很低。除非这些人获得直接的金钱援助，否则他们将面临高额的赔款，而不管联邦储备金利率[1]是多少。不可避免地，取消抵押品赎回权及抵押房产被迫出售都会打击总体的房价，并减少大多数家庭的财富与消费。

这个问题会产生全球性影响是基于两方面的原因。专家相信同样的金融动乱有可能会在欧洲抵押信贷市场和大的出口国爆发，特别是像中国这样的发展中国家，原本是美国商店的主要供货商，现在却迫切需要寻求新的客户，因为资金吃紧的美国人已经不再购买他们的商品了。

你也许会问，这都是些什么呀，它们与智能软件应用有一点关系吗？事实上，

1 联邦储备金利率是存款机构在联邦储备银行将盈余借给其他存款机构的过夜利率。

Download at http://www.pin5i.com/

关系大着呢！这整个金融问题其实是很复杂的，并还在持续地产生。然而，目前为止，有一件事情是确定的：不仅信贷金融中的欺诈和造假会影响输入数据，对各个交易的信用价值的评估能力也在此次危机对整个社会的影响不断扩大的过程中扮演了至关重要的角色。 235

在本章，我们会学习如何基于一系列的标准来对特定的个人进行信贷价值的评估。换句话说，我们会关注这个人的若干项可以反映他财务状况、社会地位、信用历史的属性，并依此把他置为五个信贷价值类别中的一个——优秀、很好、好、差与危险。

我们会把信贷价值看做 11 个属性的函数，属性的选择是随意的。我们会提供足够数量的属性使问题更有趣和真实，但你可不要认为这就是信贷价值计算中最复杂的模型。观察这些属性中哪一些应该包含，哪一些应该剔除，也是一件很有启发性的事情，本章末 To Do 节的第一个事项会有属性选择重要性的动机与指引。

6.1.1 数据的简要说明

下面我们按字母表顺序来一一介绍这 11 个属性。第一个属性是个人的实足年龄（chronological age），这与没有正式记录的心理年龄相反。很显然，年龄是一个连续变量，因为时间是一个连续变量。但是我们的第一个假设是我们要以年来计量年龄，所以这个属性的取值从 0 到 100。如果你查看实现代码，你会发现最终年龄范围的精确值并不那么重要，实际上，我们把从 18 到 100 的年龄范围分割为 10 个区间，如表 6.1 所示。

表 6.1 年龄属性的 10 个区间段。这个划分不算细致，也不算粗糙，目的在于充分表示在一个人的生命中财务发展的各个阶段

Attribute ID	From	To	Attribute ID	From	To
1	18	25	6	51	60
2	26	30	7	61	70
3	31	35	8	71	75
4	36	40	9	76	80
5	41	50	10	81	100

第二个我们需要考虑的属性是这个人是否已经宣布破产。破产的重要性在不同的国家是不同的，在美国宣布破产与在德国完全不是一回事，前者会被认为是企业 236

Download at http://www.pin5i.com/

家珍藏的一枚值得自豪的徽章；而后者也许就是你在金融与社交领域的污点，这也几乎意味着你生命的终结！此外，破产也分多种类型，为了评估信贷价值，了解破产是因为医疗原因还是因为奢靡的个人生活是非常重要的。但在我们的例子里，破产只被作为一个二值属性处理。

第三个属性是对一辆汽车的所有权。虽然拥有一辆兰博基尼（Lamborghini）跟拥有一辆起亚（KIA）汽车可谓是天壤之别。然而，这里我们仍然只把它考虑为一个二值属性。因为我们关注的是信贷价值，是否拥有一辆汽车是最强烈也是最简单的区分因素。那什么时候考虑用一个正整数值（比如汽车价格的特定范围）来描述汽车所有权会比较有意义呢？当你需要为信贷价值做一个更精细的分类时，比如把原来的 5 类再细分到 20 类。此外，你还得考虑异常情况——有人即使能负担得起一辆汽车，但因为他无法驾驶（有身体上的残疾）或他并不需要也不想要一辆车，他也就不会买车。

第四个属性是个人的信用得分。有三家主要的公司会收集信用卡消费、抵押、房屋净值信贷等信息，并依此给每个人计算一个信用得分。得分取值范围因公司不同而异，但通常是一个 0～800 的值。我们会把这个信用得分分为 8 个不同的区域，如表 6.2 所示。

表 6.2　信用分值属性的取值范围

Attribute ID	From	To	Attribute ID	From	To
1	0	500	5	651	700
2	501	550	6	701	750
3	551	600	7	751	800
4	601	650	8	801	—

第五个属性标识个人是否有过犯罪记录。虽然一个犯过小罪或受过轻微诉讼的人跟一个持械抢劫犯从根本上就是不可相提并论的，但我们并没有那么严格地进行区分，仍用一个二值属性来表示。为是否“犯罪”设定某个合适的阈值也是一件简单的事情，重罪或另一些犯罪列表的罪名可把二值变量设置为真，而轻度犯罪则保持这个犯罪记录变量的默认值（假）不变。

237

第六个属性是结算时申请人愿意支付的首付比例。表 6.3 显示了首付的四个区间，如果申请人可以支付比贷款总额 25%更多的金额，就无所谓她能支付多少了，值得关注的可区分区域是 0%～25%之间。

Download at http://www.pin5i.com/

表 6.3 首付属性的区间范围（相对于贷款总额的比例）

Attribute ID	From	To	Attribute ID	From	To
1	0	5	3	11	20
2	6	10	4	21	—

第七个属性是收入。与首付属性类似，我们并不关注某人收入是 5 百万美金还是 1 亿美金，而更加关心的是收入值在较低范围中的不同层次。而且，因为收入会是个从零到几十亿的数字，所以我们会使用更大的数值范围来对申请人进行区分。表 6.4 展示了这个属性的 10 个值及其所对应的范围，取值的单位是千美金。

表 6.4 收入属性的取值范围

Attribute ID	From	To	Attribute ID	From	To
1	0	25	6	81	100
2	26	35	7	101	125
3	36	45	8	126	150
4	46	60	9	151	200
5	61	80	10	201	—

第八个属性由个人所从事的工作性质决定。仅仅针对这个案例，我们提出了五种工作类别，但进一步细分也是可以的。表 6.5 显示了工作类别属性中五个可能的取值。

表 6.5 工作类别属性的取值

Attribute ID	Job Class Name	Attribute ID	Job Class Name
1	C-level executive	4	Business owner
2	Professional	5	Contractor
3	Employee	—	—

第九个属性与机动车所有权相关，这是一个二值属性。在某些国家，拥有机动车是一种奢侈，而对于另一些国家，这不过是一般的交通工具而已（也许还有多个）。这个属性不应该对结果造成太大的影响，因为在产生数据时，我们就以等概率把机动车所有权分配给每一个类别（除了危险类）。稍后我们会依据这些属性的取值来讨论每个类别的定义，到时会再详细地描述这个主题。 238

第十个属性与财产所有权有关——土地产权——这也是一个二值属性。另一个很自然的扩展是土地价值的货币表示，但在这个例子里，我们仍把这个属性看做是

Download at http://www.pin5i.com/

二值变量。

第十一个属性与退休账户这个金融资产总额有关。当然，这又将是一个被我们离散化的连续变量。表 6.6 展示了这个属性的八个取值及其对应的范围，取值的单位是千美金。

表 6.6 退休账户属性的范围

Attribute ID	From	To	Attribute ID	From	To
1	0	25	5	301	500
2	26	75	6	501	1000
3	76	150	7	1001	2000
4	151	300	8	2001	—

这时候，需要一再强调 11 这个数字或者这些特定的属性并没有什么神秘的。实际中，你会有很多属性可供选择，而属性选择也会是公式化并解决分类问题的一部分，这个可参看本章 To Do 一节的第一个事项，其关键点在于理解数据本身的重要性。你可以学习到目前人们所知道的所有算法，但如果你不理解数据的性质，就很难让你的问题得到一个满意的答案。

239

6.1.2 为真实问题生成人工数据

让我们审查一下那些为了定义数据所必须做的决策，并以此来说明数据性质是怎么回事。如我们所述，每个用户都会用 11 个属性值来表征，为了给用户定义一个类别，我们需要为每个类别里的用户分配一个适当的属性值。换句话说，我们在 11 维的空间里“画”出五个区域来，并给每个区域一个标签。如果我们想为某个类生成一个用户，那么只需要从合适的数组中随机选取一个——即对于每个属性，我们在该类的区域中挑选一个值。

你可以在每个类别对应的 Java 类 ExcellentUserType、VeryGoodUserType、GoodUserType、BadUserType、Dangerous UserType 中找到合适的属性值，这些类都继承自抽象类 UserType，这个抽象类包含了常用的功能。代码清单 6.1 显示了 GoodUserType 的内容，这是信贷价值中最主要的类别。如你所见，按照这个定义，拥有“好”信用的用户可以是任何年龄，但他们不应该宣布过破产，他们应该有一辆汽车，如此，等等。

Download at http://www.pin5i.com/

代码清单 6.1 GoodUserType：在属性值中定义“好”类别

```
public class GoodUserType extends UserType {
  {
    setAge(new int[] { 2, 3, 4, 5, 6, 7, 8 });
    setBancruptcy(new int[] { 0 });
    setCarOwnership(new int[] { 1 });
    setCreditScore(new int[] { 3, 4, 5, 6 });
    setCriminalRecord(new int[] { 0 });
    setDownPayment(new int[] { 2, 3 });
    setIncome(new int[] { 5, 6, 7, 8 });
    setJobClass(new int[] {2, 3, 4, 5});
    setMotorcycleOwnership(new int[] { 0, 1});
    setPropertyOwnership(new int[] { 0, 1 });
    setRetirementAccounts(new int[] { 1, 2, 3, 4 });
  }
  @Override
  public String getUserType() {
    return UserType.GOOD;
  }
}
```

查看其他的 UserType 类，你能看到什么？你也能预想到，在相邻的类别之间会有一些小小的重叠。这些模糊的边界使数据更具有现实性。你也许想知道，有没有可能一个属于“差”类别的用户的属性值却完全落入“好”类别的定义域里？在现实的数据中，这是很有可能的，这也正是我们尝试着去解决的问题！给定一个属性值的集合，我们希望有一个办法能自动分辨出一个用户的信用价值。如果现实的数据能完美地符合一些预想模型，那么“学习”就毫无意义了，只要把所有的认识都注入一个先验模型中，然后拿来用即可。所以，为了仿真实际数据的特性，我们引入了噪声级。 240

噪声级定义了一个属性值落入 X 类别定义域的用户实际上却是属于 Y 类别的可能性。噪声级允许我们把类别的定义混淆到一个任意的程度。代码清单 6.2 显示了 DataGenerator 类中的几个重要步骤。

代码清单 6.2 DataGenerator：为所有的类别创建一个用户集

```
public List<User> generateUsers(List<UserType> userTypes) {
  List<User> allUsers = new ArrayList<User>();
```

Download at http://www.pin5i.com/


```
    for(UserType userType : userTypes) {
        allUsers.addAll(
        generateUsers(userType, userType.getNUsers() ));
    }
    return allUsers;
}
public List<User> generateUsers(UserType userType, int n) {
    List<User> users = new ArrayList<User>();
    userTypeDistributions.put(userType, n);
    for(int i = 0; i < n; i++) {❶
        User u = generateUser(userType);
        users.add(u);
    }
    return users;
}
public User generateUser(UserType userType) {
    User user = new User();
    long userId = generateNextUniqueUserId();
    String username;
    if (isNoiseOn) {❷
        username = userType.getNoisyType();
    } else {
        username = userType.getUserType();
    }
    username = username + String.valueOf(userId);
    user.setUsername(username);
    user.setAge(userType.pickAge());
    user.setCarOwnership(userType.pickCarOwnership());
    user.setCreditScore(userType.pickCreditScore());
    user.setIncome(userType.pickIncome());
    user.setJobClass(userType.pickJobClass());
    user.setDownPayment(userType.pickDownPayment());
    user.setBicycleOwnership(userType.pickMotorcycleOwnership());
    user.setPropertyOwnership(userType.pickPropertyOwnership());
    user.setCriminalRecord(userType.pickCriminalRecord());
241 user.setBankruptcy(userType.pickBancruptcy());
    user.setRetirementAccount(userType.pickRetirementAccounts());
    return user;
}
```

Download at http://www.pin5i.com/

❶ 对于每个用户类别，依据对这类用户预定义的分布生成一批用户。

❷ 如果存在噪声，则依据 getNoisyType 方法的定义来混淆用户的类别。否则只使用每个用户类别预定义的分布。

因为这个例子里的数据是人造的，不同信贷价值类别的用户比例是完全可以控制的，每个类别用户的默认分布见表 6.7。

表 6.7 用户在各个信贷类别中的比例分布

类　别	用户比例(%)	类　别	用户比例(%)
优秀	5	差	25
非常好	15	危险	5
好	50	—	—

很显然，我们可以修改每个类别中的用户比例，对此，你只需要修改 UseCaseData 类中 createUserTypes 方法的实现即可。一个需要注意的重要事项是，分布几乎不可能是均匀的，大多数用户属于“好”类别，但对于信用等级低于“好”的用户，他们的比例会比“好”小 2 到 3 倍，而两个信用等级“更好”的类别在它们的分布上也是同样的情况。

你可以通过执行代码清单 6.3 的命令来生成自己的人造数据。

代码清单 6.3 为信贷价值案例生成人造数据

```
UseCaseData useCaseData = new UseCaseData(40000,20000);←定义新的数据集
UserType.addNoiseLevel("EX",
➥ new Double[] {1.0d, 5.0d, 8.0d, 10.0d});
UserType.addNoiseLevel("VG",
➥ new Double[] {1.0d, 2.5d, 6.0d, 10.0d});
UserType.addNoiseLevel("GD",
➥ new Double[] {1.0d, 3.0d, 4.0d, 8.0d});
UserType.addNoiseLevel("BD",
➥ new Double[] {1.0d, 3.0d, 7.5d, 10.0d});
UserType.addNoiseLevel("DN",
➥ new Double[] {1.0d, 6.0d, 10.0d, 14.0d});
useCaseData.create(false);←用 True 值初始化会覆盖已有的数据文件
```

为每个类定义噪声级

如果你希望使用默认的噪声级，可以跳过对 UserType.addNoiseLevel 方法的调用。要注意最后一个方法的调用，如果你为这个二值参数设置了真值，就会覆盖已经存在的数据集，根据代码清单执行得到的结果就会与你在本书中看到的不一样。 242

Download at http://www.pin5i.com/

但你可以随时通过把数据从 clean_40k_20k.7z 文件中解压而恢复原始的文件，这个数据文件可以在随代码发布的 data/ch06/samples/clean 目录中找到。

在你进入下一步之前，让我们仔细看一下噪声级的定义。在代码清单 6.2 中，一个重要的步骤就是对 getNoisyType 方法的调用，代码清单 6.4 展示了“优秀”用户类别与方法相关的代码，应该有助于解释 UserType.addNoiseLevel 调用的含义。

代码清单 6.4 UserType.getNoisyType：给“优秀”用户类别加入噪声

```
public String getNoisyType() {
  double gaussian = rnd.nextGaussian();
  String noisyType=null;
  String userType= getUserType();
  Double[] nLevels = noiseLevels.get(userType);
  if (getUserType().equals(EXCELLENT)) {
     if (gaussian <= nLevels[0]) {
        noisyType = EXCELLENT;
     } else if (gaussian > nLevels[0] &&
        gaussian <= nLevels[1]) {
          noisyType = VERY_GOOD;
     } else if (gaussian > nLevels[1] &&
        gaussian <= nLevels[2]) {
          noisyType = GOOD;
     } else if (gaussian > nLevels[2] &&
        gaussian <= nLevels[3]) {
          noisyType = BAD;
     } else {
          noisyType = DANGEROUS;
     }
  }
```

java.util 包的 Random 类用于根据标准的正态分布生成一个 double 型的数字，即是说，这个数字是以 0 为中心，并且有 68.2%的可能性在−1～1 之间，有 95%的可能性在−2～2 之间，99.7%的可能性在−3～3 之间。当我们生成一个带噪声级的数据集时（如代码清单 6.3 所示），代码清单 6.4 的代码告诉我们一个属于“优秀”类别的用户最终有 16%的可能性会落入“很好”类别的定义域。同时我们也知道，这个用户属于“差”或“危险”用户类别的可能性几乎为零。数据集的噪声级越低，分类器能得到的精确度就越高。

Download at http://www.pin5i.com/

6.2 用单分类器做信用评估

现在让我们把每一个分类器都应用于这个案例的数据，并评估它们的精确度。在 6.2.1 节，使用朴素贝叶斯分类器；在 6.2.2 节，会引入一个基于决策树的新分类器；在 6.2.3 节，会使用基于神经网络实现的分类器。本节的目的是估计出一个分类基准线，这样就可以评价通过组合分类器得到的结果是否获得了精确度上的提升。

6.2.1 朴素贝叶斯的基准线

你可以遵循代码清单 6.5 中的步骤快速地载入、训练与评估贝叶斯分类器。默认情况下，这些命令会从 c:/iWeb2/data/ch06/training-users.txt 文件中载入训练数据集，如果你想从另一个文件载入数据，则可以把这个文件名作为参数传进去。对于测试集，数据会从 c:/iWeb2/data/ch06/test-users.txt 文件中载入。

代码清单 6.5 用朴素贝叶斯分类器评估信贷价值

```
UserDataset ds = UserLoader.loadTrainingDataset();
NBCreditClassifier naiveBayes = new NBCreditClassifier(ds);←创建朴素贝叶斯分类器
naiveBayes.useDefaultAttributes();
naiveBayes.train();←训练分类器
UserDataset testDS = UserLoader.loadTestDataset();
CreditErrorEstimator nb_err =
➥ new CreditErrorEstimator(testDS, naiveBayes);
nb_err.run();←误差估计
```

图 6.1 显示了代码清单 6.5 的命令在 20 000 条交易记录测试集上计算的结果。如你所见，朴素贝叶斯分类器对这个数据集产生了相当好的精确度：0.826。在对 20 000 个信贷价值评估中，有 16 520 个记录被正确分类，仅有 3 480 个被错分。混淆矩阵有助于我们形象地观察到这些错分的严重程度。这个矩阵的主对角线是正确分类的数字，而远离主对角线的数字则是错误的分类。如果分类落在对角线的左边（在矩阵中可见），则信贷价值因错分而被高估了；而对于落在对角线右边的条目，分类器低估了一个用户的信贷价值。

Download at http://www.pin5i.com/


```
Classification completed in 1.575 seconds.

 Total test dataset txns: 20000
    Classified correctly: 16520, Misclassified: 3480
                Accuracy: 0.826
_____________________________________________________________

                CONFUSION MATRIX
_____________________________________________________________

         EX     VG     GD     BD     DN
  EX    828     24     18      0      0
  VG    161   2149   1900      4      0
  GD      1    418   8482    800      0
  BD      0      0      0   4208    147
  DN      0      0      0      7    853
```

图 6.1　朴素贝叶斯分类器对信贷价值的分类结果

对分类结果的混淆矩阵进行可视化是很重要的，因为如我们在第 5 章所述，实际问题中错分的成本是非均匀的。我们还可以为两个不同的分类器的混淆矩阵计算度量指标，以此来对它们进行比较，这方面的主题会在 6.3 节进行更多的讨论。CreditErrorEstimator 记录分类结果并负责产生如图 6.1 所示的输出。

244

代码清单 6.6 展示了来自 NBCreditClassifier 类的代码，一些无关紧要的辅助方法并没有包含在内，而这个类的基本要素都已经由代码清单中的代码表现出来。我们再一次看到以 NaiveBayes 类为基类定义一个分类器是非常简单的，你基本不需要写什么代码。下面让我们仔细看看。

代码清单 6.6　用于评估信贷价值的 NaiveBayes 分类器

```
public class NBCreditClassifier extends NaiveBayes {
  private UserInstanceBuilder instanceBuilder;  ❶
  public NBCreditClassifier(String name, TrainingSet ts,
                            UserInstanceBuilder instanceBuilder) {  ❷
      super(name, ts);
      this.instanceBuilder = instanceBuilder;
  }
  public Concept classify(Instance instance) {❸
      return super.classify(instance);
  }
  public Concept classify(User user) {❹
      return classify(instanceBuilder.createInstance(user));
```

Download at http://www.pin5i.com/


```
    }
    public void useDefaultAttributes() {❺
        trainOnAttribute(CreditInstance.ATTR_NAME_JOB_CLASS);
        trainOnAttribute(CreditInstance.ATTR_NAME_INCOME_TYPE);
        trainOnAttribute(CreditInstance.ATTR_NAME_AGE);
        trainOnAttribute(CreditInstance.ATTR_NAME_CAR_OWNERSHIP);
        trainOnAttribute(CreditInstance.ATTR_NAME_CREDIT_SCORE);
        trainOnAttribute(
➥ CreditInstance.ATTR_NAME_MORTGAGE_DOWN_PAYMENT);
        trainOnAttribute(
➥ CreditInstance.ATTR_NAME_MOTOR_BICYCLE_OWNERSHIP);
        trainOnAttribute(
➥ CreditInstance.ATTR_NAME_OTHER_PROPERTY_OWNERSHIP);
        trainOnAttribute(CreditInstance.ATTR_NAME_CRIMINAL_RECORD);
        trainOnAttribute(CreditInstance.ATTR_NAME_BANKRUPTCY);
        trainOnAttribute(
➥ CreditInstance.ATTR_NAME_RETIREMENT_ACCOUNT);
    }
}
```

245

❶ UserInstanceBuilder 类负责把一个 User 对象转换为 Instance 对象，如果你处理的是另一个问题，比如评估汽车，那么你的分类器就会有一个相似的辅助类把 Car 对象转换为 Instance 对象。这是因为我们继承了 NaiveBayes 分类器，它会无视一般的对象，而只处理 Concepts 与 Instances。判断把各种属性看做是数值变量还是类别变量，这一关键的步骤也是在 UserInstanceBuilder 类中实现的。

❷ 构造函数需要一个名字、一个训练集，以及一个 UserInstanceBuilder 实例作为参数。

❸ 我们必须指出最终所有的实现都会把分类过程交给 NaiveBayes 分类器的基本方法去做。

❹ 重载的 classify 方法，利用 UserInstanceBuilder 类生成一个 Instance 对象，然后把它传递给基本的分类方法。

❺ 你可以通过注释某些行来定制这个方法。默认情况下，我们为分类使用了所有的 11 个属性，如果把其中某些行注释，相应的属性就不会被用于分类。

在大多数情况下，你都可以信赖朴素贝叶斯分类器，它能给你合理的结果。它同时也很稳定，这是说如果训练集中的某些实例不够典型，或是某些实例错误地进

Download at http://www.pin5i.com/

入你的数据中，结果并不会受到显著的影响。但我们很快会看到，当我们组合分类器时，并不需要这样的性质。相反，对于组合的方法，我们希望基分类器是非稳定的（参看 6.4 节）。我们可以使用决策树来生成一个非稳定的分类器，所以，接下来看看如何为这里的问题创建并使用一个决策树分类器。

6.2.2 决策树基准线

代码清单 6.7 展示了创建、训练与评估一个基于决策树分类器的各个步骤。我们没有在第 5 章讨论决策树，但在本章第二个 To Do 事项中有一个简短的描述。从本章的重点来说，特定算法的实现并不真的那么重要，我们会有一个基于多个分类算法的分类器，重要的是这些分类算法是不一样的。

246

代码清单 6.7 的步骤与代码清单 6.5 类似，如果你执行代码清单 6.7 时的 Shell 与代码清单 6.5 是一样的，就无须再载入训练集与测试集了。

代码清单 6.7 用决策树分类器评估信贷价值

```
UserDataset ds = UserLoader.loadTrainingDataset();
DTCreditClassifier decisionTree = new DTCreditClassifier(ds);←创建决策树分类器
decisionTree.useDefaultAttributes();
decisionTree.train();←训练分类器
UserDataset testDS = UserLoader.loadTestDataset();
CreditErrorEstimator dt_err =
➥ new CreditErrorEstimator(testDS, decisionTree);
dt_err.run();←误差估计
```

图 6.2 显示了执行代码清单 6.7 中命令的结果。如你所见，决策树分类器是很快的——比朴素贝叶斯分类器快一个数量级。它得到的分类精确度是 0.8262，从数字而言，在同一个数据集上，它比贝叶斯分类器的精确度更好。具体来说，在 20 000 个信贷价值的评估中，有 16 524 个被正确分类，而有 3 476 个被错分。在后面的章节我们会看到，朴素贝叶斯分类器与决策树分类器在结果上的差异是不具有统计显著性的。

247

如果在 Shell 中执行 decisionTree.printTree()命令，你会看到决策树是什么样的。图 6.3 显示了 printTree()的头几行输出。要确认你的 Shell 窗口缓冲的高度是可以容纳几千行的。同时要注意，由于本章在格式上的限制，图中的树形结构是不完全的。

Download at http://www.pin5i.com/

从图中可见，分类器优先关注的是犯罪记录属性，按照决策树的判断，如果你被标识为有过犯罪记录的人，你就妄想能得到贷款了！如果用户没有任何犯罪记录，那么决策树会继续查看这个用户过去是否宣告过破产，如果曾经有过，该用户的信贷价值就会被分类为“差”，否则，分类会按屏幕中所显示的输出信息那样持续下去。

```
Classification completed in 0.132 seconds.

Total test dataset txns: 20000
    Classified correctly: 16524, Misclassified: 3476
                 Accuracy: 0.8262
_____________________________________________________

                 CONFUSION MATRIX
_____________________________________________________

          EX      VG      GD      BD      DN
  EX     831      24      15       0       0
  VG     164    2321    1725       4       0
  GD       0     585    8319     797       0
  BD       0       0       8    4200     147
  DN       0       0       0       7     853
```

图 6.2 决策树分类器对信贷价值分类的结果

```
Node:attrName=priorCriminalRecord,isLeaf=false,concept=null
-> Branch: [priorCriminalRecord=1]
     Node:attrName=null,isLeaf=true,concept=DN
-> Branch: [priorCriminalRecord=0]
Node:attrName=priorDeclaredBankruptcy,isLeaf=false,concept=null
 -> Branch: [priorDeclaredBankruptcy=1]
      Node:attrName=null,isLeaf=true,concept=BD
-> Branch: [priorDeclaredBankruptcy=0]
   Node:attrName=carOwnership,isLeaf=false,concept=null
-> Branch: [carOwnership=1]
   Node:attrName=mortgageDownPayment,isLeaf=false,concept=null
-> Branch: [mortgageDownPayment=3]
   Node:attrName=otherPropertyOwnership,isLeaf=false,concept=null
-> Branch: [otherPropertyOwnership=1]
   Node:attrName=retirementAccount,isLeaf=false,concept=null
-> Branch: [retirementAccount=3]
   Node:attrName=creditScore,isLeaf=false,concept=null
-> Branch: [creditScore=3]
   Node:attrName=null,isLeaf=true,concept=GD
```

图 6.3 决策树对于信贷价值数据的顶层节点

Download at http://www.pin5i.com/

分类结果的可解释性是决策树流行的主要原因之一，但当我们把 10 或 100 个分类器组合起来时，要直接解释其输出就变得困难了。在那样的情况下，我们主要是对决策树固有的非稳定特性感兴趣。当然，决策树并非唯一的非稳定的分类算法，还有其他典型的非稳定分类器，比如神经网络分类器。

6.2.3 神经网络的基准线

下面我们再添加一个基于神经网络实现的分类器到集群中。代码清单 6.8 显示了如何利用神经网络信用分类器来评价用户的信贷价值。如果你在与代码清单 6.5 和代码清单 6.7 一样的 Shell 环境下执行代码清单 6.8 的命令，则可以忽略对训练集与测试集数据的载入。

248

代码清单 6.8 用神经网络分类器来评价信贷价值

```
UserDataset ds = UserLoader.loadTrainingDataset();
NNCreditClassifier neuralNet = new NNCreditClassifier(ds);←创建神经网络分类器
neuralNet.setLearningRate(0.025);←设置反向传播的学习率
neuralNet.useDefaultAttributes();
neuralNet.train();←训练分类器
UserDataset testDS = UserLoader.loadTestDataset();
CreditErrorEstimator nn_err =
↪ new CreditErrorEstimator(testDS, neuralNet);
nn_err.run();←误差估计
```

如你所见，这些代码清单的不同之处仅仅在于分类器的配置，但这些重复是有意义的，也是重要的。在分类器比较的情况（或者别的比较）中，你应该尽可能地做到有条理、有系统，因为随着分类器的数量增加，要保证事物间的可比性会变得越来越困难。图 6.4 显示了代码清单 6.8 的执行结果。

iweb2.ch6.usecase.credit.NNCreditClassifier 类封装了专门用于这个信贷价值案例的神经网络分类器，代码清单 6.9 则显示了这个神经网络在 iweb2.ch6.usecase.credit.UserCreditNN 类中的定义。让我们看这个类，这里并没有显示所有连接的定义与不重要的方法。

Download at http://www.pin5i.com/


```
Classification completed in 0.266 seconds.

 Total test dataset txns: 20000
    Classified correctly: 14330, Misclassified: 5670
                Accuracy: 0.7165
_____________________________________________________________

                CONFUSION MATRIX
_____________________________________________________________

          EX      VG      GD      BD      DN
  EX     498       0     372       0       0
  VG      91       0    4100      23       0
  GD       0       0    8804     897       0
  BD       0       0      33    4175     147
  DN       0       0       0       7     853
```

图 6.4 神经网络分类器对信贷价值分类的结果

代码清单 6.9 用于贷款信用风险评估的一个自定制神经网络

```
public class UserCreditNN extends BaseNN {
  public UserCreditNN(String name) {
      super(name);
      create();
  }
  public void create() {
      createNN_11_7_5();
  }
  private void createNN_11_7_5() {
      Layer inputLayer = createInputLayer(0, 11);←定义有 11 个节点的输入层
      Layer hiddenLayer = createHiddenLayer(1, 7,
➥ new double[] { 0.5, -1, 1.5, 0.5, 1, -0.2, 0.1 });←定义有 7 个节点的隐层
      Layer outputLayer = createOutputLayer(2, 5,←定义有 5 个节点的输出层
          new double[] {-1.5, 0.5, -1, 0.5, 1});
      setInputLayer(inputLayer);
      setOutputLayer(outputLayer);
      addHiddenLayer(hiddenLayer);
      setLink("0:0", "1:0", 0.25);
      setLink("0:0", "1:1", -0.7);
      setLink("0:0", "1:2", 0.25);   定义节点间的连接
      setLink("0:0", "1:3", 0.25);
      setLink("0:0", "1:4", -0.3);
```

249

Download at http://www.pin5i.com/


```
        setLink("0:0", "1:5", 0.25);
        setLink("0:0", "1:6", -0.5);
        setLink("0:1", "1:0", 0.25);
        setLink("0:1", "1:1", -0.5);
        setLink("0:1", "1:2", 0.25);
        setLink("0:1", "1:3", 0.25);
        setLink("0:1", "1:4", 0.50);
        setLink("0:1", "1:5", 0.25);
        setLink("0:1", "1:6", 0.50);
    [... Snip ...]
  }
}
```

定义节点间的连接

我们使用 11 个属性的选择决定了输入层有 11 个输入节点，但隐层的 7 个节点与输出层的 5 个节点的选择则是任意的。实际上，我们可以为网络设定两个或多个隐层，别忘了我们在第 5 章关于神经网络复杂性过高的讨论，巨大的灵活性是以复杂度的提升为代价的。虽然有经验原则的存在，但遵循神经网络设计原则时应当谨慎，设计出来的神经网络也应该经过实际应用环境的检验。在其中一个 To Do 事项中，请你在 UserCreditNN 类中定义自己的神经网络架构，这样你就可以通过本章的方法来比较两个或更多的神经网络，看看你的设计决策在实际中会产生什么效果。

这一节展示了会在本章随后部分继续使用的三个分类器。我们想给出的最后一个观察报告是神经网络的训练时间。除了分类器的质量指标特性，你还应该关注两个基本的性能特性——训练时间与运行时。训练时间在每个分类器执行完 train()方法后会打印在 Shell 中（见代码清单 6.5 至代码清单 6.9）。对于朴素贝叶斯、决策树和神经网络分类器，它们的训练时间分别是 0.5 s、5.6 s 与 265.6 s[2]。训练时间通常不是一个热点问题，分类器在运行时的表现更为重要。然而两个分类器在训练时间上存在着几个量级的差别足以让我们把较慢的分类器排除掉，尤其是生产环境的数据量预期要比训练数据更大的时候。

250

前面提到决策树分类器的分类精确度是 0.8262，在同一个数据集中，这是一个比朴素贝叶斯分类器精确度更好的结果。而决策树与朴素贝叶斯分类器看起来都比神经网络分类器要好，但这是真的吗？这些精确度上的差异具有统计显著性吗？在

2 这几个值并没有显示在图 6.1 至图 6.4 中，图中显示的是分类时间，不是训练时间。——译者注

Download at http://www.pin5i.com/

进入分类器的组合版本之前，这是一个急需回答的重大问题。下一节，我们会开发出相应的工具以帮助回答这些问题。

6.3 在同一个数据集中比较多个分类器

本节会展示四种假设检验方法来对分类器进行比较。McNemar 检验与差额比例检验（difference of proportions test）可用于比较两个分类器；Cochran Q 检验与 F 检验可用于比较三个或更多的分类器。这些都是统计检验方法，需要对统计学有一定的了解，但这并非是本书所要求的。幸运的是，它们的大致思想还是很好理解的。所有的统计检验都有一个初始假设（术语为零假设，null hypothesis），这个假设可通过统计量与一个阈值（这个阈值可通过查表或某个已知公式计算得到）间的数值比较而被证明或被证伪。所以，在我们提供的所有的检验方法中，你会看到它们都遵循两个步骤：第一步，计算相关的统计量；第二步，把统计量与一个硬编码的值（即阈值）进行比较。如果你对其中的数据细节感兴趣，Ludmila Ilieva Kuncheva 的 *Combining Pattern Classifiers: Methods and Algorithms* 是一本极好的启蒙读物，在本节，我们也是遵循她的表述框架来介绍。

不管我们是希望从多个分类器中挑选一个，还是基于多个分类器设计分类器，都可以只使用成对的比较。但是讨论应用于三个或多个分类器的检验也是有现实原因的：分类器对的数量会随着分类器数量的增加而快速增长。实际上，当分类器数量为 N 时，分类器对的数量为 $N*(N-1)/2$。所以，我们需要有一个方法来快速分辨出一系列的分类器是否具有统计意义上的显著性差异。最终，我们仍需做成对的比较，以找到到底哪个分类器是“罪魁祸首”。

这些检验的论证已经超出了本书的范围，感兴趣的读者可以从本章后的文献列表和附录 C 中找到大量的参考文献。从实用的角度而言，你所需要的最重要的知识有如下三项：

- 要理解分类器精确度上的差异并不总是显著的，它们存在成对比较的假设检验，也存在同时对多个分类器进行比较的假设检验。
- 要了解如何为每个检验计算合适的统计量，我们的代码提供了本章使用的四种假设检验的实现，以及一个一般的抽象类，你可以通过继承它来编写自己的检验。

251

Download at http://www.pin5i.com/

- 要知道如何为每一个检验查找合适的阈值，它们可以在公用的表中根据给定的显著性水平与自由度（如果要用到的话）查到。你也可以编写自己的类，依据数学定义来计算这些值。

6.3.1 McNemar 检验

我们假定你已经在同一个 Shell 中执行过代码清单 6.5、6.7 与 6.8 的命令，那么就可以接着运行代码清单 6.10 的脚本，以两两地对这三个分类器进行比较。如你所预期的那样，McNemar 检验被封装在 McNemarTest 类中，它的构造函数需要两个参数——来自两个分类器的 ClassifierResults 实例。

代码清单 6.10 用于比较两个分类器的 McNemar 检验

```
McNemarTest mnTest1 = new McNemarTest(nb_err.getResults(),
➥ nn_err.getResults());
McNemarTest mnTest2 = new McNemarTest(nb_err.getResults(),
➥ dt_err.getResults());
McNemarTest mnTest3 = new McNemarTest(dt_err.getResults(),
➥ nn_err.getResults());
mnTest1.evaluate();
mnTest2.evaluate();
mnTest3.evaluate();
```

为每个分类器对做检验

评估每个检验

结果显示在图 6.5 中，这应该也是在你看到每个分类器的分类测试报告时本能地想到的结果。神经网络得到较低分类精确度的结果，从检验结果看，这个结论也是具有统计显著性的。你在输出中看到的变量 Chi2 被称为 χ^2（chi 方统计量），它的概率分布可用自由度 n 的取值来定义，在本例中，我们取 n=1。对于 χ^2 统计量的细节，可参看本章 To Do 事项的第四条，以及附录 C 提到的参考文献。

χ^2 统计量有很多不同的用法，在这个实现中，我们遵循 Dietterich 与 Kuncheva 的方法，像代码清单 6.11 那样计算 χ^2。变量 n01 表示第一个分类器错分而第二个分类器正确分类的次数；变量 n10 则表示刚好与之相反的情况。

注意，取绝对值保证了 n01 与 n10 相减是可交换的，此外，对分子取平方可以保证最终的结果是对称的。换句话说，传入 McNemar 构造函数时分类器结果的顺序是无关紧要的。一旦得到了统计量的值，我们把它与 3.841 阈值进行比较，这是 χ^2 概率分布在显著性水平为 0.05 的取值。如果你不知道其中的意思，可参看附录 C 里

Download at http://www.pin5i.com/

关于假设检验显著性水平的参考文献。也即是说，如果 χ^2 统计量的值大于 3.841，从统计显著性意义上说，这两个分类器肯定是有差异的。

252

```
bsh % mnTest1.evaluate();
      Evaluating classifiers
      NBCreditClassifier and NNCreditClassifier:

      ____________________________________________________
      NBCreditClassifier accuracy: 0.826
      NNCreditClassifier accuracy: 0.7165
      N = 20000, n00=3050, n10=2620, n01=430, n11=13900

      ____________________________________________________
      Confidence Interval              : 0.05
      Degrees of Freedom               : 1
      Statistic threshold (Chi-square): 3.841

      ____________________________________________________
      Chi2 = 1571.0560655737704 > 3.841
      The two classifiers are different: TRUE

bsh % mnTest2.evaluate();
      Evaluating classifiers
      NBCreditClassifier and DTCreditClassifier:

      ____________________________________________________
      NBCreditClassifier accuracy: 0.826
      DTCreditClassifier accuracy: 0.8262
      N = 20000, n00=3252, n10=224, n01=228, n11=16296

      ____________________________________________________
      Confidence Interval              : 0.05
      Degrees of Freedom               : 1
      Statistic threshold (Chi-square): 3.841

      ____________________________________________________
      Chi2 = 0.01991150442477876 <= 3.841
      The two classifiers are different: FALSE

bsh % mnTest3.evaluate();
      Evaluating classifiers
      DTCreditClassifier and NNCreditClassifier:

      ____________________________________________________
      DTCreditClassifier accuracy: 0.8262
      NNCreditClassifier accuracy: 0.7165
      N = 20000, n00=2872, n10=2798, n01=604, n11=13726

      ____________________________________________________
      Confidence Interval              : 0.05
      Degrees of Freedom               : 1
      Statistic threshold (Chi-square): 3.841

      ____________________________________________________
      Chi2 = 1413.6534391534392 > 3.841
      The two classifiers are different: TRUE
```

图 6.5 把 McNemar 检验应用于三个分类器对的结果

Download at http://www.pin5i.com/

代码清单 6.11 McNemar 检验中对 χ^2 统计量的计算

```
protected void calculate() {
  int n = c1.getN();
  for(int i = 0; i < n; i++) {
   if( c1.getResult(i) && c2.getResult(i) ) {
      n11++;←两个分类器都判断正确
   } else if( c1.getResult(i) && !c2.getResult(i) ) {
          n10++;←第一个正确，第二个错误
   } else if( !c1.getResult(i) && c2.getResult(i) ) {
          n01++;←第一个错误，第二个正确
   } else {
          n00++;←两个分类器都判断错误
   }
  }
  double a = Math.abs(n01 - n10) - 1;
  chi2 = a * a / (n01 + n10);←χ²统计量的值
}
```

253

在本章末的一个 To Do 事项中，请你以不同的方式对 χ^2 统计量加以应用。基本思想是判断来自两个不同的分类器的分类结果是否来自同一个概率分布。在与本案例——分布在五个级别中的申请人的信贷价值——相似的问题中，我们提出的方法比起 McNemar 检验也许是一个更好的比较度量方式，详见本章 To Do 事项。

6.3.2 差额比例检验

再次强调，假定你已经在同一个 Shell 中执行了代码清单 6.5、6.7 与 6.8 的代码，这样就可以运行代码清单 6.12 的脚本来比较三个分类器对的差异，这次使用差额比例检验。差额比例检验被封装在 Diff2PropTest 类中，跟 McNemarTest 一样，它的构造函数需要两个参数。

代码清单 6.12 用差额比例检验来比较两个分类器

```
Diff2PropTest d2pTest1 = new Diff2PropTest(nb_err.getResults(),
➥ nn_err.getResults());
Diff2PropTest d2pTest2 = new Diff2PropTest(nb_err.getResults(),
➥ dt_err.getResults());
Diff2PropTest d2pTest3 = new Diff2PropTest(dt_err.getResults(),
➥ nn_err.getResults());
```

为每个分类器对做检验

Download at http://www.pin5i.com/


```
d2pTest1.evaluate();
d2pTest2.evaluate();   评估每个检验
d2pTest3.evaluate();
```

结果显示在图 6.6 中，它进一步证实了我们从 McNemar 检验得到的结论。根据这个检验，同样可得到结论：神经网络精确度更低的结果是具有统计显著性的。输出结果里的变量 z 是一个来自标准正态分布的统计量，所以，它的均值是 0，方差是 1。关于正态分布的细节，可参看附录 C 所提及的参考文献。 254

```
bsh % d2pTest1.evaluate();
      Evaluating classifiers
      NBCreditClassifier and NNCreditClassifier:

      ____________________________________________
      NBCreditClassifier accuracy: 0.826
      NNCreditClassifier accuracy: 0.7165

      ____________________________________________
      Confidence Interval              : 0.05
      Statistic threshold (Std Normal): 1.96

      ____________________________________________
      |z| = 26.069696745398772 > 1.96
      The classifiers are different: TRUE

bsh % d2pTest2.evaluate();
      Evaluating classifiers
      NBCreditClassifier and DTCreditClassifier:

      ____________________________________________
      NBCreditClassifier accuracy: 0.826
      DTCreditClassifier accuracy: 0.8262

      ____________________________________________
      Confidence Interval              : 0.05
      Statistic threshold (Std Normal): 1.96

      ____________________________________________
      |z| = -0.05276718103090302 <= 1.96
      The classifiers are different: FALSE

bsh % d2pTest3.evaluate();
      Evaluating classifiers
      DTCreditClassifier and NNCreditClassifier:

      ____________________________________________
      DTCreditClassifier accuracy: 0.8262
      NNCreditClassifier accuracy: 0.7165

      ____________________________________________
      Confidence Interval              : 0.05
      Statistic threshold (Std Normal): 1.96

      ____________________________________________
      |z| = 26.12132981820125 > 1.96
      The classifiers are different: TRUE
```

图 6.6 把差额比例检验应用于三对分类器比较的结果

Download at http://www.pin5i.com/

代码清单 6.13 展示了 z 统计量的计算，其分布假定是服从标准正态分布的。这个统计量在机器学习文献中很常用，但独立性假设是存疑的，你能指出是为什么吗？（提示：参看 Dietterich 的文章）

255

代码清单 6.13 差额比例检验中对 z 统计量的计算

```
double diff = c1.getAccuracy() - c2.getAccuracy();←计算精确度的差
double mean = 0.5 * (c1.getAccuracy() + c2.getAccuracy());←计算精确度的平均值
double b = ( 2.0 * mean * ( 1 - mean ) ) / n;
z = diff / Math.sqrt(b);←定义 z 统计量
```

在代码清单 6.13 中，变量 n 指测试集中实例的数量，这个值对于两个分类器都是一样的。一旦得到 z 统计量的值，就可以用它的绝对值与 1.96 相比较，这是一个显著水平为 0.05 的双边检验（对于这样的检验，分类器的输入顺序就无关紧要了）。

6.3.3 Cochran Q 检验与 F 检验

现在开始进入三个或更多个分类器比较的情况。如你所预期的，这些检验相对于比较两个分类器的检验要更复杂，但基本思想是一致的——计算合适的统计量，并把它与一个合适的阈值进行比较，这个阈值是来自统计量自身所属概率分布的值。既然如此，我们的统计量当然会涉及所有分类器的数据。代码清单 6.14 展示了只需要一行代码，即可使用 Cochran Q 检验或 F 检验（F 检验意指 Fisher，因为是 Ronald A. Fisher 在 1920 年引入这个检验的）对三个分类器进行比较。我们再次假定你已经在同一个 Shell 中执行过代码清单 6.5、6.7 和 6.8 的代码。

代码清单 6.14 用 Cochran Q 检验与 F 检验比较三个或更多的分类器

```
CochransQTest cqTest = new CochransQTest(
➥ nb_err.getResults(),dt_err.getResults(), nn_err.getResults());
cqTest.evaluate();
FTest fTest = new FTest(nb_err.getResults(),
➥ dt_err.getResults(), nn_err.getResults());
fTest.evaluate();
```

Cochran 检验基于这样的假设：如果两个分类器之间没有差异，那么 q 统计量（计算方法见代码清单 6.15）应该服从自由度为 2 的 χ^2 分布。一般来说，对于 N 个分类器，其自由度为 $N-1$。

Download at http://www.pin5i.com/

代码清单 6.15 Cochran Q 统计量的计算

```
protected void calculate() {
  int n = c1.getN();
  double T = calculateT();←正确分类的总数
  double T2 = 0.0;
  for(int i = 0; i < n; i++) {
   double x = 0.0;←正确分类给定条目的分类器的数量
   if( c1.getResult(i) ) {
       x++;
   }
   if( c2.getResult(i) ) {
       x++;
   }
   if( c3.getResult(i) ) {
       x++;
   }
   T2 += (x * x);
  }
  double sum = 0.0;
  sum = (double)c1.getNCorrect() * c1.getNCorrect() +
       (double)c2.getNCorrect() * c2.getNCorrect() +
       (double)c3.getNCorrect() * c3.getNCorrect() ;
  double a = L * sum;
  q = (L - 1) * (a - T * T) / (L * T - T2);
}
```

256

在分类器的比较范畴内，F 检验是由 Looney 提出的。F 统计量的计算很有讲究，你可以研读一下 FTest 类 calculate()方法的实现，我们会遵循 Looney 对它的数学解释。从实用角度而言，最重要的是理解 F 统计量是与 Fisher-Snedecor 分布的值进行比较的。就我们的使用来说，后者可以被看做是两个 χ^2 分布的比，其中每一个 χ^2 分布都先除以它的自由度。一个 χ^2 分布有 $N-1$ 个自由度，另一个 χ^2 分布有$(N-1)\times(M-1)$个，N 是分类器的数量，M 是测试集中实例的数量。Fisher-Snedecor 分布（或简写为 F 分布）的阈值可以通过 *Engineering Statistics Handbook* 一书的在线网站（http://www.itl.nist.gov/div898/handbook/）查找，你可以查到与不同自由度相对应的不同显著性水平的值。

Cochran Q 检验与 F 检验的结果显示在图 6.7 中，检验失败了，说明分类器是不

Download at http://www.pin5i.com/

同的，因为每个统计量的值都比相应的阈值高。这是我们所期望的结果，因为我们之前已经发现神经网络分类器与贝叶斯分类器、决策树分类器在分类精确度的差别是具有统计显著性的。

你已经学会了如何对多个分类器进行比较，现在是组合它们的时候了。下一节会覆盖分类器融合的内容，即汇集所有分类器的贡献，以得到一个统一的分类结果。而分类器的选择（即每个分类器只负责数据空间中的特定领域，同时也假定每个分类器只在其所属领域才有好的表现）则不会被包含在内。但可以参看 To Do 事项中与专家混合相关的条目——这其实是一种分类器选择技术。

257

```
bsh % cqTest.evaluate();
      Evaluating classifiers NBCreditClassifier,DTCreditClassifier,NNCreditC
      ____________________________________________________________
      NBCreditClassifier accuracy: 0.826
      DTCreditClassifier accuracy: 0.8262
      NNCreditClassifier accuracy: 0.7165
      ____________________________________________________________
      Confidence Interval                : 0.05
      Degrees of Freedom                 : 2
      Statistic threshold (chi-square): 5.991
      ____________________________________________________________
      Q = 2783.821552723059 > 5.991
      The classifiers are different: TRUE

bsh % fTest.evaluate();
      Evaluating classifiers NBCreditClassifier,DTCreditClassifier,NNCreditC
      ____________________________________________________________
      NBCreditClassifier accuracy: 0.826
      DTCreditClassifier accuracy: 0.8262
      NNCreditClassifier accuracy: 0.7165
      ____________________________________________________________
      Confidence Interval      : 0.05
      Degrees of Freedom (1st): 2
      Degrees of Freedom (2nd): 39998
      Statistic threshold      : 3.08
      ____________________________________________________________
      F = 264.49439710228575 > 3.08
      The classifiers are different: TRUE
```

图 6.7　Cochran Q 检验与 F 检验同时应用于三个分类器得到的结果

6.4　bagging: bootstrap 聚合（bootstrap aggregating）

前面我们说过，bagging 这个术语是 bootstrap aggregating 的缩写，由 Breiman

Download at http://www.pin5i.com/

发明。在统计学里，bootstrap 这个术语意指对一个统计量的采样分布的非参数估计方法，那么在这里它又是什么意思呢？假定有一个由数值组成的采样数据集，我们希望计算采样生成这个数据集的原分布的均值。注意，如果你把所有的数字相加后除以数据集的大小，你会得到对原分布真实均值的一个估计。bootstrap 的意图在于通过对原数据集的部分复制（放回采样）以生成几个数据集，并使用这多个数据集来提高对原分布均值的估计。

在分类的范畴，采样数据集是训练集或测试集的标签数据。所以，bootstrap 聚合的主要思想是很直接的，首先，从标签数据集中随机采集一部分样本，对每一份采样数据训练一个分类器，然后让每一个分类器对给定的实例进行分类，并得到占主导性的分类结果（多数票决）。再次声明，随机采样并不是从系统里获得新的训练集和测试集，而是从现有数据集的放回抽样得到。

尽管有着理论上的动机，但仍需注意其中的陷阱。使用 bagging 的前提是你从重采样同一份数据中得到的分类器是有差异的，它们会产生不同的分类结果，否则所谓的重采样也就毫无意义了！一个分类器如果输入受到轻微的扰动时会产生不一样的输出，就称为是非稳定的。所以，我们可以预期 bagging 在非稳定分类器上能得到好的表现，比如神经网络和决策树。这跟我们较早前对理解与探索数据本质的重要性的观察是相关的，同时也跟我们在第 5 章讨论的过拟合的风险相关。 258

bagging 试图去探索存在于数据中的多样性，这个通过采样生成不同的数据集以训练分类器来实现，我们创建的分类器强调了数据的不同方面。所以，如果我们的数据是富含信息的，bagging 就可以确保数据中多方面的信息都能被分类器所捕获。bagging 特别适合于只有小训练集但却想对大数据集进行分类的情况。为阐明这个特性，我们会使用 100 个实例的训练集和 10 000 个实例的测试来做实验。代码清单 6.16 显示了我们用于生成数据的脚本，这与代码清单 6.3 几乎相同，但要注意训练集的大小以及噪声级。

代码清单 6.16 以一个小训练集与大量的噪声创建一个人工数据集

```
UseCaseData useCaseData = new UseCaseData(100,10000);❶
UserType.addNoiseLevel("EX",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});
UserType.addNoiseLevel("VG",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});
UserType.addNoiseLevel("GD",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});
UserType.addNoiseLevel("BD",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});
UserType.addNoiseLevel("DN",new Double[] {0.5d, 1.5d, 3.0d, 4.0d});
useCaseData.create(false);❷
```

定义每个类别的噪声级

Download at http://www.pin5i.com/

❶ UseCaseData 的第一个参数是训练集的大小，第二个参数是测试集的大小。

❷ 这个方法启动了数据的创建过程。参数取 true 会覆盖所有已有的文件，参数取 false 则只会在没有数据文件时才生成数据。随书已经附带了代码清单 6.16 所生成的数据集，你也可以创建不同的数据集，如果能把算法应用于自己的数据集就更好了。计算结果会随着生成数据的多样性而发生变化，一般来说，随着训练集的增长，分类精确度的提升就会失去统计上的显著性。bagging 对分类精确度的显著性提升源于在放回采样时对训练集的智能选择，实现了数据集最大程度的多样性，Friedman 等人在 boosting 相关的工作中，也得到了与 bagging 类似的结论。

6.4.1 bagging 实例

现在是时候应用 bagging 算法并考察它在我们的数据集上能得到的结果了。代码清单 6.17 演示了如何创建并执行 BaggingCreditClassifier 类中实现的 bagging 算法。

259

代码清单 6.17 使用 BaggingCreditClassifier 类

```
UserDataset ds = UserLoader.loadTrainingDataset();
BaggingCreditClassifier bagClassifier =
➥ new BaggingCreditClassifier(ds);❶
bagClassifier.setVerbose(false);←设置为 true 可以看到输出结果
TrainingSet ts1 = bagClassifier.getBootstrapSet();❷
DTCreditClassifier dt1 = new DTCreditClassifier(ts1);
dt1.useDefaultAttributes();
dt1.setPruneAfterTraining(true);
bagClassifier.addMember(dt1);
bagClassifier.train();
UserDataset testDS = UserLoader.loadTestDataset();
CreditErrorEstimator bagee1 =
➥ new CreditErrorEstimator(testDS, bagClassifier);
bagee1.run();←创建 CreditErrorEstimator 以进行评估
TrainingSet ts2 = bagClassifier.getBootstrapSet();←对所有的成员重复以上流程
DTCreditClassifier dt2 = new DTCreditClassifier(ts2);
dt2.useDefaultAttributes();
dt2.setPruneAfterTraining(true);
bagClassifier.addMember(dt2);
bagClassifier.train();
CreditErrorEstimator bagee2 =
```

Download at http://www.pin5i.com/


```
new CreditErrorEstimator(testDS, bagClassifier);
bagee2.run();
```

❶ 创建 bagging 分类器实例，并让它指向原始数据集。

❷ 创建第一棵决策树，并把它添加为分类器集群中的一员，然后训练分类器。可以使用我们在 6.2 节提供的三个分类器中的任意一个，这里使用决策树分类器 DTCreditClassifier 是基于它的非稳定特性。你也可以使用神经网络分类器 NNCredit Classifier，它同样是非稳定的。使用贝叶斯分类器 NBCreditClassifier 也是无妨的，甚至还可以创建一个混合了这三个分类器的 BaggingCreditClassifier 分类器。在随书发布的代码里，我们还提供了更多的脚本，所以你可以仔细研读它们，主要的步骤都相同。

输出结果很冗长，因为其中包含了混淆矩阵。表 6.8 以精确度与计算时间（毫秒）对输出结果进行了整理。每一个决策树分类器的训练都需要 5～10 ms 的时间，从时间的测定来看，重要的是训练阶段与运行时都非常快，100 000 个具有 11 种属性的实例只需几秒即可分类完毕。

表 6.8 对于代码清单 6.17 的BaggingCreditClassifier分类器，精确度与执行时间相对于分类器数量的函数

分类器数量	精 确 度	执行时间（ms）	分类器数量	精确度	执行时间（ms）
1	0.60517	752	7	0.64258	1945
2	0.62158	968	8	0.63536	2194
3	0.63714	1173	9	0.64129	2435
4	0.62955	1327	10	0.63625	2662
5	0.646	1540	11	0.64305	2870
6	0.63719	1741	—	—	—

这个实验中生成数据的噪声是相当严重的，所以只能得到比较低的精确度，但要注意到精确度随着集群中分类器数量的增加而提升，同时看到这样的提升并不是单调的。换句话说，虽然精确度的总体趋势是上升的，但集群的精确度会随着集群成员数量的增加而发生波动。这说明当你采用 bagging 作为提升分类质量的策略时，你应当监督它的结果，如果它们不能提升总体精确度，可以减少一部分分类器。本章的一个 To Do 事项就是与此相关的练习，以及更多的 bagging 使用技巧。 260

你得到的结果不会与表 6.8 的完全一样，因为每次运行代码清单 6.17 的脚本时，得到的采样结果都是不同的。然而，运行时间的数量级以及精确度提升的趋势应该

Download at http://www.pin5i.com/

与这里的结果是一致的。不妨多运行几次脚本，都会得到一致的结果。

6.4.2 bagging 分类器底层细节

核心的 BaggingCreditClassifier 类继承自抽象类 ClassifierEnsemble，基于跟 bagging 一样的组合分类器的思想，你可以通过继承这个抽象类来创建不同的集成分类器。代码清单 6.18 显示了 BaggingCreditClassifier 类的代码。

代码清单 6.18 扩展自 ClassifierEnsemble 的 bagging

```
public class BaggingCreditClassifier extends ClassifierEnsemble {
  private UserInstanceBuilder instanceBuilder;←基于用户数据生成实例
  private BootstrapTrainingSetBuilder bootstrapTSetBuilder;←通过放回采样生成训练数据集
  public BaggingCreditClassifier(UserDataset ds) {
      super(BaggingCreditClassifier.class.getSimpleName());
      instanceBuilder = new UserInstanceBuilder(false);
      TrainingSet originalTSet =
  instanceBuilder.createTrainingSet(ds);
      bootstrapTSetBuilder =
  new BootstrapTrainingSetBuilder(originalTSet);
  }
  public TrainingSet getBootstrapSet() {
      return bootstrapTSetBuilder.buildBootstrapSet();
  }
  public UserInstanceBuilder getInstanceBuilder() {
      return instanceBuilder;
  }
  public Concept classify(User user) {
      return classify(instanceBuilder.createInstance(user));
  }
}
```

261

就我们所关心的算法而言，这个类并没什么特别的。分类器应该可以处理信贷价值数据，所以它被赋予了 UserInstanceBuilder 类，用于把数据解析为合适的形式，以符合一般的 Classifier 接口的需要。此外，我们还提供了 BootstrapTrainingSetBuilder 用于创建之前所描述的放回采样[3]训练集。从本质上说，BaggingCreditClassifier 是一

3 根据上下文的意思，本章会把部分的 bootstrapp 翻译成放回采样。——译者注

Download at http://www.pin5i.com/

个为方便使用而实现的包装类。实际有价值的代码蕴含在 BootstrapTrainingSetBuilder 类与抽象类 ClassifierEnsemble 中，代码清单 6.19 显示了来自 BootstrapTrainingSetBuilder 的源代码，剔除了 javadoc 注释与 import 语句。

代码清单 6.19 辅助类：对原训练集进行放回采样

```
public class BootstrapTrainingSetBuilder {
  private TrainingSet originalTrainingSet;
  public BootstrapTrainingSetBuilder(
➥ TrainingSet originalTrainingSet) {
    this.originalTrainingSet = originalTrainingSet;
  }
  public TrainingSet buildBootstrapSet() {
    int N = originalTrainingSet.getSize();❶
    Map<Integer, Instance> instances =
➥ originalTrainingSet.getInstances();
    Instance[] selectedInstances = new Instance[N];
    Random rnd = new Random();
    int center = rnd.nextInt(N);❷
    int countN =0;
    while (countN < N) {
      if (countN % (N/5) == 0) {❸
        center = rnd.nextInt(N);
      }
      int selectedInstanceId = pickInstanceId(N,center);❹
      Instance selectedInstance = instances.get(selectedInstanceId);
      selectedInstances[countN] = selectedInstance;
      countN++;
    }
    TrainingSet tS = new TrainingSet(selectedInstances);
    return tS;
  }
  private int pickInstanceId(int N) {❹
    Random rnd = new Random();
    boolean loop = true;
    int selectedInstanceId=-1;
    double scale = (double) (N/2) / 4.0d;
    while (loop) {
      selectedInstanceId = new Double(center +
```

262

Download at http://www.pin5i.com/


```
rnd.nextGaussian()*scale).intValue();
    if (selectedInstanceId >=0 && selectedInstanceId < N) {
      loop=false;
    }
  }
  return selectedInstanceId;
 }
}
```

❶ 每个采样数据集中实例的数量都跟原数据集一样。

❷ 为了获得更好的多样性，需要选定一些中心点，每个训练集都围绕这些中心点对实例进行采样。

❸ 这个条件决定围绕着每个中心点采样的实例数量。

❹ 第 3 步与 pickInstanceId 方法是关键的，因为它们一同定义了选择放回采样训练集的策略。具体来说，首先从一系列可取的实例中定义出一个中心点，然后用高斯分布对该中心点的邻居采样 *N*/5 次。变量 scale 是 *N* 的函数，它控制从中心点到某个特定距离获得一个实例的概率。

你可以重写这些方法，或者自行写一个方法，但要记得训练集的选择是 bagging 获得好的表现的关键。实际上，提升 bagging 结果的最好方法之一就是使用高级的训练集选择技术。那你认为多高级才算是高级呢？可以随便去试验多种选择机制，并研究其结果，你已经拥有了所有的基本设施了，剩下这样探索的事情应该会很有趣！

263

Bagging 的基本思想是分类器的集成，当然 bagging 只是分类器集成的一种方法，你也许有理由相信另一种集成分类器的方法也能给你带来分类精确度的显著性提升。所以，我们提供给你的并不是一个特定于 bagging 的实现，而是一个抽象类，用于说明分类器集成的一般思想，以便你日后对它进行扩充与探索。实际上，无论是 bagging 还是 boosting，我们使用的都是同一个抽象类。

6.4.3 分类器集成

代码清单 6.20 展示了抽象类 ClassifierEnsemble 的代码，它把分类器组合（就像 bagging 那样）的基本要素与方法都已经封装了起来。它实际上并不关注集群中的分类器是同一个分类器的不同版本还是不同分类器类型的不同版本，所有这些需求都

Download at http://www.pin5i.com/

是遵循 Classifier 接口的结果。

代码清单 6.20 集成分类器的抽象类

```
public abstract class ClassifierEnsemble implements Classifier {
  private String name;
  private List<Classifier> baseClassifiers =
  new ArrayList<Classifier>();←集群中的分类器
  public ClassifierEnsemble(String name) {
      this.name = name;
  }
  public Concept classify(Instance instance) {
      ConceptMajorityVoter voter =
new ConceptMajorityVoter(instance);←多数票决的分类决策
      for( Classifier baseClassifier : baseClassifiers ) {
         Concept c = baseClassifier.classify(instance);
         voter.addVote(c);←每次分类代表一个投票
      }
      return voter.getWinner();←由投票结果得出胜利者
  }
  public boolean train() {
      for( Classifier c : baseClassifiers ) {
         c.train();←集群的训练即各个成员分类器的训练
      }
      return true;
  }
  public void addMember(Classifier baseClassifier) {
      baseClassifiers.add(baseClassifier);
  }
  public void removeMember(Classifier c) {
      baseClassifiers.remove(c);
  }
}
```

264

ConceptMajorityVoter 类的意义很容易理解，就是封装了每个分类器对给定实例的投票集，然后根据简单的多数规则来决定优胜者。但你也可以设计更复杂的遴选策略来决定集成分类器的分类结果。另一种可行的策略是考虑分类器的特殊性，对不同分类器的投票赋予不同的权重。还可以按照每个分类判断的（概率）置信度来

Download at http://www.pin5i.com/

分配权重。为完整起见，我们在代码清单 6.21 中列出了 ConceptMajorityVoter 的源代码。

代码清单 6.21 集成分类器的一个简单选择策略

```
public class ConceptMajorityVoter {
  private Map<Concept, Integer> votes =
  new HashMap<Concept, Integer>();←通过 HashMap 保存投票结果
  private Instance i;
  public ConceptMajorityVoter(Instance i) {
     this.i = i;
  }
  public void addVote(Concept c) {←投票量的叠加
     Integer conceptVoteCount = votes.get(c);
     if( conceptVoteCount == null ) {
        conceptVoteCount = new Integer(1);
     } else {
        conceptVoteCount = conceptVoteCount + 1;
     }
     votes.put( c, conceptVoteCount );
  }
  public Concept getWinner() {←多数票决规则的实现
     int winnerVoteCount = 0;
     Concept winnerConcept = null;
     for(Map.Entry<Concept, Integer> e : votes.entrySet()) {
        if( e.getValue() > winnerVoteCount ) {
           winnerConcept = e.getKey();
           winnerVoteCount = e.getValue();
        }
     }
     return winnerConcept;
  }
  public int getWinnerVoteCount() {←胜利者的票数
     Concept winner = getWinner();
     return votes.get(winner);
  }
}
```

265

最后需要补充的是，要获得最有效的多样化（通过输入数据的变异来保证）效

Download at http://www.pin5i.com/

应，训练数据集的大小应该有一个最有效点，这个最有效点依赖于候选分类器的类型及训练的方法。如果数据集很小，通过 bagging 集成获得的收益并不能补偿单个模型精确度下降的损失，因为这时候每个分类器只能得到一个更小的训练集。另一方面，如果数据集非常庞大，导致计算时间已经是一个问题，这时即使是一个灵活的单分类器，也已经相当够用了。

总之，bagging 是通过利用训练数据的多样性来获得收益的，如果我们的训练集本身已经足够多样化，那么单分类器也能够对它的方方面面有着最大（即便不是完全）的“理解”。如果你已经阅读并试验过本章相关的 To Do 事项，那么相信你已经有足够的心理准备进入 boosting 了！

6.5 boosting：一种迭代提高的方法

boosting 的主要思想在于其分类器集群的增长方式：对于同一个数据集，新的分类器应该提高之前的分类器分类失败的实例的分类效果。对于 bagging，我们创建的分类器是相互独立的，寄希望于新成员的加入能提升整体分类的精确度。所以，bagging 的成功依赖于对训练集的精挑细选，但我们无法事先得知这是否可行。

对于 boosting，我们增加分类器进行迭代的每一步都希望分类结果有所提升。这次不是依赖于随机选择合适的训练数据集，而是通过仔细提取有偏的训练集（来自之前的集群错分的那部分实例）以提升分类的结果。

我们可以在 boosting 与实际生活中的咨询体验之间做类比。假定公司的金融问题需要一个委员会来做咨询，问题可能会很复杂，它的解决方案也取决于多个领域的知识。对于其中的一些领域，比如 X，委员会中的成员并不能给出确定性的建议。在这种情况下，引入一个对领域 X 很擅长的专家是有意义的。一旦我们按这种方式去做，委员会就拥有了所有必备的知识，以对研究中的金融问题做出有指导意义的决策和有价值的建议。这就是 boosting 的实质，可以归纳为：找到你所不擅长的，引入对此熟悉的人去解决它！

这里我们会介绍 Leo Breiman 引入的名为 arc-x4 的 boosting 算法。Breiman 为自适应重采样组合（adaptive resampling and combining）这一类分类器选择了 arcing 这个名字。这类分类器与另一种名为 AdaBoost（adaptive 与 boosting 的拼接）的 boosting 方法类似，AdaBoost 这种最初出现在 Freund 和 Schapire 的文章中的算法是一种威名

Download at http://www.pin5i.com/

266 在外并已经被广泛使用的 boosting 算法。而你只要对 arc-x4 做适量的修改，就可以得到 AdaBoost 的实现。AdaBoost 有着坚实的理论基础，并且经过严格的推导得到，而 arc-x4 则是一种特定性（ad hoc）的算法。从算法的角度来看，这两种方法有两个关键的区别，我们稍候会对它们进行讨论。

6.5.1 boosting 分类器实例

现在让我们用 arc-x4 算法来提升决策树分类器的表现，代码清单 6.22 的脚本正是用于实现该目的的，整个流程与代码清单 6.17 中 bagging 的情况类似。

代码清单 6.22 评估 BoostingCreditClassifier

```
UserDataset ds = UserLoader.loadTrainingDataset();
BoostingCreditClassifier arcx4 = new BoostingCreditClassifier(ds);❶
arcx4.setClassifierType("decision tree");❷
arcx4.setClassifierPopulation(1);❸
arcx4.setVerbose(false);
arcx4.train();
UserDataset testDS = UserLoader.loadTestDataset();
CreditErrorEstimator arcx4ee =
➥ new CreditErrorEstimator(testDS, arcx4);❹
arcx4ee.run();
arcx4.setClassifierPopulation(3);
arcx4.train();
CreditErrorEstimator arcx4ee =
➥ new CreditErrorEstimator(testDS, arcx4);
arcx4ee.run();
```

❺

❶ 生成一个 boosting 分类器的实例，并把训练集的实例传递给它。

❷ 把决策树算法作为集群的基分类器。

❸ 定义集群的数量，一开始集群只有一个成员。

❹ 生成 CreditErrorEstimator 的实例以评估分类器的结果。

❺ 给集群添加一个新成员并重新评估分类结果。

你可以使用 6.2 节列出的任一个分类器，代码清单 6.22 中使用的是 DTCredit Classifier，通过调用 setClassifierType 方法并把参数设置为 decision tree 即可实现。这里使用跟代码清单 6.17 一样的基分类器的意义在于我们希望比较 boosting 和

Download at http://www.pin5i.com/

bagging 的结果，这个选择的基本原则跟之前一样——决策树是非稳定的。此外，决策树有着最快速的执行时间，这点对于在交互式 Shell 前实验的你来说当然更方便。但你也可以使用 NNCreditClassifier，只要把字符串参数设置为 neural network（大小写不敏感）即可，同样可以使用 NBCreditClassifier，这需要把字符串参数设置为 naive bayes（同样大小写不敏感）。

这里先岔开一下话题。我们也可以创建一个“混合”的 BoostCreditClassifier，其中包含所有这些基分类器的混合，而不是仅使用一种基分类器。这并不是 AdaBoost 或 arc-x4 的标准变体，但这是有道理的，因为对于某部分实例，一类分类器会表现得比别的更好，如果你可以利用这点信息，就没有理由失去这个提升分类质量的机会。boosting 是自适应的，从原则上说，这个自适应不单指训练集选择的自适应，还应当包含分类器类型选择的自适应。在本章的一个 To Do 事项中，我们建议你去实现这一种一般化的分类器。 267

现在检查来自代码清单 6.22 的输出。我们已经把结果总结在了表 6.9 中，其中包括代码清单 6.22 脚本执行的精确度与计算时间，这个表与表 6.8 类似。

表 6.9 代码清单 6.22 中BoostCreditClassifier的精确度与计算时间相对于分类器数量的函数

分类器数量	精 确 度	执行时间（ms）
1	0.57314	1108
3	0.63862	1638
5	0.64203	2153
7	0.65317	2792
11	0.64668	3947
31	0.65572	9828
41	0.65676	12870
61	0.66044	19032

如你所见，boosting 给出了比 bagging（见表 6.8）稍好的结果。从计算性能上看，应该要提的是，如果我们使用贝叶斯分类器作为基分类器，训练时间仍然很小，与决策树的训练时间相比几乎没有什么差别。但基于贝叶斯分类器的集成精确度却比决策树集成的精确度好很多，而且集群只需要 11 个成员就够了。但计算时间相差就很明显了，对于只有 11 个分类器成员的集群，基于决策树的 boosting 比基于朴素贝叶斯的 boosting 在计算时间上快 63 倍。

看到这里，你也许想知道这些精确度上的差异是否具有统计上的显著性。你可

Download at http://www.pin5i.com/

以使用 McNemar 检验（见 6.3.1 节）对 bagging 和 boosting 做一个直接的比较，不管你选择了什么基分类器或其他什么参数，你总可以用这样的方式来对两个分类器进行比较。从工程的角度讲，希望能得到代码复杂度、分类精确度、训练与执行时间的一个折中，所以，你应该把你所面临的问题的成本视为这些参数的函数。一般来说，仅精确度这个因素并不足以构成选择某个算法的工程决策。

268

6.5.2 boosting 分类器底层细节

我们的主体类是 BoostingCreditClassifier，这是对 BoostingARCX4Classifier 类的继承。后者又是对 ClassifierEnsemble 类的继承，跟 BaggingCreditClassifier 一样的继承关系。这是因为 boosting 与 bagging 类似，都是致力于通过组合不同的分类器以求得到更好的结果。BoostingCreditClassifier 的代码显示在代码清单 6.23 中。

代码清单 6.23 arc-x4 算法的一个实现

```
public class BoostingCreditClassifier extends BoostingARCX4Classifier {
  private UserInstanceBuilder instanceBuilder;  ←基于用户数据生成实例
  private ClassifierMemberType classifierType;  ←通过放回采样生成训练数据集
  public BoostingCreditClassifier(UserDataset ds) {
     this(BoostingCreditClassifier.class.getSimpleName(), ds,
➥         new UserInstanceBuilder(false));
  }
  public BoostingCreditClassifier(String name, UserDataset ds,
➥        UserInstanceBuilder instanceBuilder) {
     this(name, instanceBuilder,
➥    instanceBuilder.createTrainingSet(ds));
  }
  public BoostingCreditClassifier(String name,
➥ UserInstanceBuilder instanceBuilder, TrainingSet tSet) {
     super(name, tSet);
     this.instanceBuilder = instanceBuilder;
  }
  public UserInstanceBuilder getInstanceBuilder() {
     return instanceBuilder;
  }
  public Concept classify(User user) {
     return classify(instanceBuilder.createInstance(user));
```

Download at http://www.pin5i.com/


```
}
@Override
publicClassifiergetClassifierForTraining(TrainingSet set) {←根据 classifierType
                                                          生成新的分类器
    Classifier baseClassifier = null;
    switch(classifierType) {
      case NEURAL_NETWORK:
        NNCreditClassifier nnClassifier =
➥ new NNCreditClassifier(set);
        nnClassifier.setLearningRate(0.01);
        nnClassifier.useDefaultAttributes();
        baseClassifier = nnClassifier;
        break;
      case DECISION_TREE:
        DTCreditClassifier dtClassifier =
➥ new DTCreditClassifier(set);
        dtClassifier.useDefaultAttributes();
        dtClassifier.setPruneAfterTraining(true);
        baseClassifier = dtClassifier;
        break;
      case NAIVE_BAYES:
        NBCreditClassifier nbClassifier =
➥ new NBCreditClassifier(set);
        nbClassifier.useDefaultAttributes();
        baseClassifier = nbClassifier;
        break;
      default:
        throw new RuntimeException("Invalid type!");
    }
    return baseClassifier;
  }
  public ClassifierMemberType getClassifierType() {
   return classifierType;
  }
  public void setClassifierType(String type) {
   if (type.equalsIgnoreCase("decision tree")) {
    this.classifierType = ClassifierMemberType.DECISION_TREE;
   } else if (type.equalsIgnoreCase("neural network")) {
```

269

Download at http://www.pin5i.com/


```
        this.classifierType = ClassifierMemberType.NEURAL_NETWORK;
      } else if (type.equalsIgnoreCase("naive bayes")) {
        this.classifierType = ClassifierMemberType.NAIVE_BAYES;
      }
    }
}
```

再次强调，这个类的角色就是把问题的各个方面封装起来。注意，我们提供了根据字义设置基分类器的方法，代码可以自动生成新的分类器实例。arc-x4 算法本身的实现可以在抽象类 BoostingARCX4Classifier 中找到，它的代码显示在代码清单 6.24 中，剔除了辅助方法、javadoc 注释与 import 语句。

代码清单 6.24 Breiman 的 arc-x4 算法的一个实现

```
public abstract class BoostingARCX4Classifier
➥ extends ClassifierEnsemble {←集成分类器
  private TrainingSet originalTSet;
  private int classifierPopulation = 2;
  public BoostingARCX4Classifier(String name, TrainingSet tSet) {
      super(name);
      this.originalTSet = tSet;
  }
270 public Concept classify(Instance instance) {
      ConceptMajorityVoter voter =
➥ new ConceptMajorityVoter(instance); ←多数票决的分类决策
      for( Classifier baseClassifier : baseClassifiers ) {
         Concept c = baseClassifier.classify(instance);
         voter.addVote(c);
      }
      return voter.getWinner();
  }
  public abstract Classifier
➥ getClassifierForTraining(TrainingSet set);←生成不同类型的分类器
  public boolean train() {
      baseClassifiers = new ArrayList<Classifier>();
      int size = originalTSet.getSize();
      double[] w = new double[size];←样本的选择权重
      int[] m = new int[size];←实例被错分的次数
      double w0 = 1.0 / size;
```

Download at http://www.pin5i.com/


```
        Arrays.fill(w, w0);
        Arrays.fill(m, 0);
        for(int i = 0; i < classifierPopulation; i++) {
            TrainingSet tSet = buildTSet(originalTSet, w);
            Classifier baseClassifier =
➥ getClassifierForTraining(tSet);
            baseClassifier.train();
            updateWeights(originalTSet, w, m, baseClassifier);
            baseClassifiers.add(baseClassifier);
        }
        return true;
    }
    public TrainingSet buildTSet(TrainingSet tSet, double[] w) {←指定如何创建新的训练集
        WeightBasedRandom wRnd = new WeightBasedRandom(w);
        int n = w.length;
        Instance[] sample = new Instance[n];
        Map<Integer, Instance> instances = tSet.getInstances();
        for(int i = 0; i < n; i++) {
            int instanceIndex = wRnd.nextInt();
            sample[i] = instances.get(instanceIndex);
        }
        return new TrainingSet(sample);
    }
    public void updateWeights(TrainingSet tSet, double[] w,
➥ int[] m, Classifier baseClassifier) {←指定如何更新权重
        int n = w.length;
        for(int i = 0; i < n; i++) {
            Instance instance = tSet.getInstance(i);
            Concept actualConcept =
➥ baseClassifier.classify(instance);
            Concept expectedConcept = instance.getConcept();
            if( actualConcept == null ||
              !(actualConcept.getName()
➥ .equals(expectedConcept.getName()) ) ) {
              m[i]++;
            }
        }
        double sum = 0.0;
```

271

Download at http://www.pin5i.com/


```
        for( int i = 0; i < n; i++) {
            sum += ( 1.0 + Math.pow(m[i], 4) );
        }
        for( int i = 0; i < n; i++) {
            w[i] = ( 1.0 + Math.pow(m[i], 4) ) / sum;
        }
    }
}
```

四次幂是任意的一个选择

我们的实现是 ClassifierEnsemble 的另一个实例，该抽象类的分类结果是基于各分类器成员的多数票决。arc-x4 算法的关键部分可以在 train()方法中找到，我们为训练集中的每个实例引入了一个权值，这些权值在采样生成新训练集时会发挥作用。一开始，所有的权值都置为 1/*N*，*N* 是训练集中的实例数量。一旦有一个分类器加入了集群并进行了训练，我们就会通过 updateWeights 方法对权值进行更新。

要注意的是，每次我们为集群添加新的分类器后，新训练集的选择并不是在所有的实例中平均选取的，而是由新的权值所决定的。实现这个功能的类被称为 WeightBasedRandom，它的目的在于优先选择权值比较大的实例，而避免选择权值较小的实例。

每个分类器都会把自己的错分记录累加在数组 *m* 中，而这个数组又会以特定的方式来影响权值 *w*。算法名字中的数字 4 源于在 updateWeights 方法中标准化并得到实例权值时对错分次数的 4 次幂这个运算。你也可以试验不同的幂值，比如 5，这样就可以称为 arc-x5 算法了，这样也许能得到较好或较差的结果，实际情况如何，则更多地取决于你的数据。对幂值参数选择的任意性也是我们称为 ad hoc 算法的原因。

272

当把它具体地应用于我们的信用分类问题时，你认为会产生什么效果？有没有可能通过调整这个参数而得到更好的结果？为了改良采样过程以生成新的训练集，你还能想到用其他函数可以把错分次数与权值联系起来的吗？

6.6 总结

本章以贷款申请中评估信贷价值应用为背景，介绍了分类器组合这个主题。我们的案例涉及的数据有 11 个属性，同时也提出了我们到底真正需要多少个属性这一

Download at http://www.pin5i.com/

问题。在一一说明这些属性的过程中，我们有机会去强调理解数据的重要性，以及你的选择对于数据正确表征现实世界信息的影响。分类器的组合在某些场合中能得到显著性提升的结果，但在另一些场合，单分类器可能会更好。现在你拥有一个任你使用的数据生成工具后，就可以进一步探索数据的特性及其对单个或是分类器组合的偏好性。

当然，为了评价单个或组合分类器是否比别的更好，需要有办法去分辨两个或多个分类器在精确度上的差别是否具有统计上的显著性。我们提供了四种假设检验方法让你可以对分类器进行比较。对分类器进行比较的能力是很重要的，因为集群应该以尽可能少的分类器数量来获得对可用信息尽可能大的挖掘，所以只把最好的分类器保留在集群中就是举足轻重的一步了。为此，我们展示了用于比较两个分类器的 McNemar 检验与差额比例检验，以及用于比较三个或更多分类器的 Cochran Q 检验与 F 检验。

如你所见，我们的比较是依靠统计假设完成的，它有一个初始的可被证明或证伪的零假设，验证过程是根据一个统计量的取值与一个现成的阈值在数字上的比较完成的。我们提供的所有的检验方法都遵循同一套模式，它包含两步：第一，为相应的统计量计算一个值；第二，把统计量的取值与阈值进行比较。

我们的第一个集成分类器是基于 bagging 的，这种技术集成的成员来自对原数据集不同子集的训练得到的分类器，希望通过集成来提升分类精确度。这样的做法，我们是希望每一次从原数据集进行放回式采样时，都能着重于数据的不同方面，以达到增加集群的知识总量这个目的。bagging（或者说 bootstrap 聚合）只是这个领域的冰山一角而已，正因为如此，我们引入了一个名为 ClassifierEnsemble 的抽象类，你可以使用它来检验不同的想法，同时希望对你特定的项目也有帮助。

另一个稍有不同的分类集成方法是 boosting，它不会随机生成新的训练集，也不会把对数据的先验知识加入新训练集的选择过程中，而是试图给予错分的实例更多的训练机会，以达到迭代趋于更优结果的目的。我们给出了 Breiman 的 arc-x4 boosting 算法的实现，并在测试集中检查它的精确度与执行时间。我们把算法的主要部分分为三类——BoostingCreditClassifier、BoostingARCX4Classifier 和 WeightBasedRandom。你现在已经懂得有哪些途径可以提升人造数据的分类效果，但别忘了，更重要的是，把这些思想应用到现实数据中。

273

本章总结了一系列本书中会用到的智能应用工具，实际应用中，这些工具只是

Download at http://www.pin5i.com/

一幅大蓝图中的简单组件。第 7 章我们会讨论如何把这些智能零件组装到一个新闻门户应用中。

6.7 To Do

1．属性选择。为了描绘每个申请人的信贷价值，我们选择了 11 个属性。为什么是 11 个，而不是 17 个？直觉上，你也许会认为使用更多的属性就能得到更好的结果，但事实并非如此，当可选的属性变多时，对属性的正确选择就变得尤为重要。有些算法（比如决策树）通过精心设计可以识别出富含在属性中的信息，而其他算法则不像决策树那么明显，所以可能需要对不同的属性集进行直接的实验。

那你该怎样选择属性呢？通过修改 UserType 类别中的属性定义，有些属性会变得与类别不相干，而另一些则会变得举足轻重。哪一个分类器受这种改变的影响最大？为什么？你也可以定义自己的属性，以对属性空间进一步扩展。实际问题中，理解数据的性质，明白算法在不同配置参数下的工作方式，无论是从精确度，还是从内存效率及计算时间上看，都是很重要的。

2．决策树。决策树因其对分类结果解释的简单而被广泛应用于分类问题中。想象一个树形结构，其每个节点都与一个属性相关，而节点下的每个连接则相当于该属性的一个特定的值。树的每一个叶子节点都对应于分类问题中一个特定的类别。这就是决策树。

为了理解它是怎么工作的，需要考虑树的根节点，它位于树的顶层，所有其他的节点都在它下面。对于每一个待分类的实例，我们从根节点开始，并检查与根属性相应的值。这个值会引导我们进入下一个节点，沿着适当的边进入一层节点后，我们再次检查与该层节点属性相应的值，如此反复，直到到达叶子节点。因为每个叶子节点都对应着一个类别，所以我们的分类过程也就结束了，这是不是很直观明了呢？

274

当然，为把决策树应用于分类问题中，首先你必须建立一棵树，这可不像使用树那么简单了。我们邀请你研究我们的实现代码，并从 Witten、Frank 与 Dunham 的书中获得更多关于决策树的知识。

3．比较两个或更多的神经网络。神经网络架构的设计是一件相当复杂的事情，需要对神经网络有很好的理解，以及对有待解决问题的充分认识。6.2.3 节介绍了 UserCreditNN 类，这是一个专门为信贷价值用例而设计的神经网络。

Download at http://www.pin5i.com/

如果尝试用其他的神经网络架构做实验，并利用本章所介绍的方法来比较相关分类器的差别，则是非常有启发意义的事情。例如，你可以创建输入层有 11 个节点、隐层有 5 个节点、输出层有 1 个节点的一个神经网络。你还可以尝试为已有的神经网络多增加一个隐层。即使是在单节点神经网络架构中，不同的突触激发规则与不同的偏移量都会导致完全不同的结果。怀着大无畏的精神，勇敢地去探索尽可能多的可能性吧。

复杂性的增加相比于它带来的麻烦到底值不值得？当有大量可供你支配的参数，你的优化设计又该从何做起？你又如何判断它是最优的还是仍有缺陷的呢？

顺便说一下，本章中的神经网络分类器是作为“差”分类器而挑选进来的。因为我们的工作是比较分类器，所以需要一个结果与另两个分类器不同（相对较差）的分类器。最简单的方法是选择一个并非最优的神经网络——在大量的候选神经网络设计中，随机挑选一个就几乎可以肯定能得到一个非最优的选择。

4．两个分布的 χ^2（chi 方）检验。6.3.1 节中引入了 χ^2 统计量来进行 McNemar 检验，我们还可以用不同的方式来引入这个概率分布。为此，我们需要把比较问题转变成与 McNemar 检验不一样的事情。比较的意义是这样的——我们有两个分类器，一个公用的训练集，一个公用的测试集，如果我们训练分类器并测试它，可以分辨出结果的差异具有统计上的显著性吗？

考虑把问题转变为如下方式。回忆我们在第 5 章介绍过的混淆矩阵——对于每个分类器，当运行 CreditErrorEstimator 时，这个矩阵就会打印在 Shell 的输出中。混淆矩阵可以表示为一个二维的数组，比如 int[][] confusion，如果两个分类器在统计上是等同的，那么我们有理由预期混淆矩阵的值也是相似的。所以，我们来计算如下的统计量：

```
double chi2=0;
  for(int i = 0; i < n; i++) {
      for(int j = 0; j < n; j++) {
      int diff = confusion1[i][j]-confusion2[i][j];
      int sum = confusion1[i][j]+confusion2[i][j];
      chi2 = chi2 + (diff*diff)/sum;
      }
  }
```


其中，n 是类别的数量（在我们的案例中，n=5），confusion1 是第一个分类器的

Download at http://www.pin5i.com/

混淆矩阵，confusion2 是第二个分类器的混淆矩阵。编写一个新的继承自 Test 的检验类，就像 McNemarTest 类一样，但这次是计算上述 χ^2 统计量。那你怎样为这个统计量计算阈值呢？应该取多大的自由度呢？如果统计量只包含矩阵的对角线元素，是否会更有价值？为什么？

[提示：参考附录 C 的文献中通过不完全伽玛函数计算 chi 方值的介绍。自由度的答案是 n-1。]

5．专家混合——一种分类器选择技术。就在本章的简介中我们说过，分类器组合这个主题可以分为两个主要的方向，一个是分类器选择；另一个是分类器融合，如 bagging（见 6.4 节）和 boosting（见 6.5 节）。作为一名软件工程师，你一定会为蕴含在分类器选择方法中的分治法原理的优雅与高效而赞赏不已。

一个特别有趣并被深入研究过的分类器选择方法是（局部）专家混合（mixture of (local) experts，参看 Jacobs 等人与 Jordan、Jacobs 的文章），这种方法最初源于多层（监督）神经网络模块化实现，这项技术要求我们可以通过使用若干个局部专家与一个门网络（另一个分类器）来改善学习效率与泛化能力（能正确分类未知实例的能力）。门网络决定对于不同的训练实例应该使用哪一个专家，所以经过训练之后，每个分类器都是输入（训练集的实例）在某个特定域的专家。这种方法的细节可参看 Jacobs 与其合作者的文章。

创建三个适用于信贷价值用例的神经网络分类器，并把它们与作为门网络的第四个网络组合起来，这样就得到了一个专家混合分类器。其中也凸现出多个问题，应该用什么训练方法？专家混合分类器的结果比单个神经网络分类器好吗？这样做是否值得？相对于 bagging 与 boosting 那样的融合方法，专家混合的优点与缺点又在哪里？

276

6．bagging 的技巧。在 6.4 节，我们曾学习把 bagging 用于提升分类的效果。尽管其原理很简单，但你要想从 bagging 中获得最大的提升还是需要一些额外工作的。对 bagging 算法最具决定性影响的因素是集群中成员分类器所使用的训练集的选择。

另一个关键性的决策是集群中分类器的数量。换言之，我们应该使用多少个分类器？10 个？20 个？50 个？100 个？先从最直观的方式开始，编写一个类，其目的在于调查通过在集群中增加新分类器所得到的提升。你需要做的第一项检验是利用 6.3 节介绍的技术中的一种来区分两个或多个分类器的分类结果是否具有统计上的

Download at http://www.pin5i.com/

显著性。如果是，你可以再进行第二项检验，这次会检验精确度上的提升是否高于某一个阈值，这个阈值可以由你来设定。如果两项检验都通过了，就可以把新分类器保留在集群中，否则就抛弃它，并选择一个新的再做检验。

尝试用不同的分类器并检查其结果，这些结果的异同是什么？如果你混合不同类型的分类器又会怎样？这样 bagging 还能正常工作吗？你会得到较好还是较差的结果？你可以在刚刚用于优化精确度的类中实验这些情况。

为 BootstrapTrainingSetBuilder 类的 pickInstanceId 方法编写你自己的实现。提升 bagging 结果的最佳途径之一就是使用更复杂的训练集选择技术，你觉得需要哪一种复杂的技术？可以随意用多种选择机制来进行试验并研究其结果。你已经拥有所有必备的基础设施，剩下的工作应该会很有趣。

7. 把 boosting 推广为对分类器类型的自适应。在 6.5 节介绍了 boosting 并解释了 arc-x4 算法的工作原理。Breiman 提到 arc-x4 算法“之所以可行并非因为它是 arc-fs 算法的特殊形式，而是因为它的自适应采样”（Breiman 喜欢用 arc-fs 这个术语指代大家熟知的 AdaBoost）。

如果 Breiman 的观点是对的，我们就可以创建一个包含混合基分类器（而不仅是一种基分类器）的 BoostCreditClassifier 类。这并不是 AdaBoost 或者 arc-x4 的标准变体，但这是有意义的。此外，如我们曾提到的 arc-x4 本身是一种特定性的算法，很有可能对于某特定的实例集，一类分类器比另一类分类器好，所以，如果你可以利用这个事实，则没有理由失去这个提升分类质量的机会。Boosting 是自适应的，原则上，你可以让它自适应的不仅是训练集的选择，还有分类器类型的选择。

这样的推广有多种变体可以考虑，比如你可以选择在每次需要给集群添加成员时，引入每个类型分类器的一个代表。你也可以基于预设的分类器类型的比例来生成集群，例如，50%的决策树，30%的朴素贝叶斯，20%的神经网络。 277

要把权衡的思想放在心上，设计一个策略，在获得最大的精确度收益的同时，又不至于使训练时或运行时的计算时间过于离谱。精确度上的提升是否就足以证明在复杂性与计算成本上的付出是合理的？我们又怎样进一步提升算法的精巧度？在 6.3 节，我们学习到如何去比较两个或更多的分类器，由于我们只对分辨分类器是否不同感兴趣，所以这种比较的方法实际上可以直接嵌入分类算法中。

Download at http://www.pin5i.com/

6.8 参考资料

Bell, R., Y. Koren, and C. Volinsky. "Chasing $1,000,000: How we won the Netflix progress prize." *ASA Statistical and Computing Graphics Newsletter*, Vol 18 (2), pp. 4-12, 2007. http://stat-computing.org/newsletter/v182.pdf.

Breiman, L. "Bagging predictors." *Machine Learning*. Vol 24 (2), pp. 123-140, 1996.

Breiman, L. "Arcing classifiers. "*The Annals of Statistics*. Vol 26 (3), pp. 801-849, 1998.

Dietterich, T.G. "Ensemble methods in machine learning." *Multiple Classifier Systems*, (Editors: J. Kittler and F.Roli) volume 1857 of *Lecture Notes in Computer Science*, pp.1-15. Cagliari, Italy. Springer, 2000. http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=743935.

Freund, Y., and R. E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting." *Journal of Computer and System Sciences*. Vol 55 (1), pp. 119-139, 1997.

Friedman, J., T. Hastie, and R. Tibshirani. "Additive logistic regression: A statistical view of boosting." *Annals of Statistics*, Vol 28 (2), pp. 337-407, 2000.

Jacobs, R.A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton. "Adaptive mixtures of local experts." *Neural Computation*. Vol 3, pp. 79-87, 1991.

Jordan, M.I., and R. A. Jacobs. "Hierarchical mixtures of experts and the EM algorithm." *Neural Computation*, Vol 6, pp. 181-214, 1994.

Kuncheva, L.I. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

Looney, S.W. (1988). "A statistical technique for comparing the accuracies of several classifiers." *Pattern Recognition Letters*. Vol 8, pp.5-9, 1988.

Download at http://www.pin5i.com/

7 智能技术大汇集：一个智能新闻门户

本章内容提要：

- 从互联网获取并清洗内容
- 新闻搜索
- 分配新闻类别
- 创建新闻组

前面各章都介绍了某一特定类型的算法，我们从中学习到了搜索算法、推荐算法、聚类算法与分类算法，并在各算法对应的领域与特定的用例中对它们进行了检验。同一个算法可以用于多个不同的案例，多个算法也可以汇集在一起以增强同一个应用的功能。

本章的目的在于展示在一个应用中怎样利用各种不同的算法。应该在哪里使用聚类算法？分类结果又会受到聚类带来的哪些影响？PageRank 与 DocRank 可被用做搜索之外的用途吗？我又该如何利用本书所学到的知识，来为自己的应用编写一个推荐引擎？这些问题的答案都在本章一一为你道来。

我们不会编写一个完整的 Web 应用，而只是简单地把我们置于这个应用环境中，以讨论智能算法在其中的使用。换句话说，我们会讨论这些算法在一个假想的 Web

Download at http://www.pin5i.com/

应用环境中的使用。具体地说，我们面临的例子是一个新闻门户，一个类似于 Google 新闻的网站。

需要清楚的是，当我们把智能算法引入应用中时，会遇到一些实际的问题。例如，对于很多智能算法来说，对数据操作的顺序会是很重要的，所以元算法（meta-algorithm）——一个用于控制或组合[1]其他算法的算法——是能否把所有算法成功地融合在一起的关键。

本章也让我们有机会把之前各章讨论过的所有内容做一次回顾。这里会以从互联网抓取并聚合新闻这件事情作为启动用例。我们曾在第 2 章介绍过这个任务，不过现在可以利用分类技术对这个过程进行多方面的改进。为了使结果是可重现的，以及为讨论建立一个数据参考系，我们会用到第 2 章介绍的新闻数据集的一个扩充集。

很自然地过渡到下一节，我们会针对多个查询来搜索这些新闻数据集，这会引出大量有趣的问题。在这个例子里（新闻门户 Web 应用），我们并不是要对全网的内容进行搜索，而是聚焦于从具有特定属性的文档（新闻）的正文中获取结果。所以，我们可以充分利用这个知识来提升检索结果的质量。这是一个广泛适用的经验，毕竟没有人比你更了解自己的数据。

进入新闻分组与新闻类别标定的范畴（7.4 节），会涉及一系列不同的主题，这是因为那里使用了多种智能算法来实现我们的目标。如果先把一个聚类算法应用于新闻数据集，然后对这些聚簇进行分类，得到的结果跟先对新闻数据进行分类而后对每个新闻类别进行聚类的结果是不同的。这个不同会引发很多问题，我们会在 7.4 节和 7.5 节详细讨论这些问题。

在本章的最后一节，我们把推荐带进了这幅蓝图里。具体地说，我们对用户会给新闻打分这个案例进行了探讨。第 3 章介绍的纯粹的机器推荐算法是适用于这个案例的。我们会应用基于条目的推荐方法，并且提供了大量可扩展的建议。

我们还编写了大量 To Do 事项，它们应能帮助你探讨智能算法在新闻门户中应用的方方面面，这个研究方向对于大量的其他应用应该也同样具有价值。尽量享受这一切吧！

1 我们在第 6 章介绍的分类器组合即是元算法出现的一种特殊形态，并已经在实际中使用了。

Download at http://www.pin5i.com/

7.1 功能概览

280

如果你还没用过 Google 新闻网站，用你的浏览器访问http://news.google.com/，以获得对我们即将讨论内容的一个现实的参考。进入这个网站，你似乎会认为不过是在访问一个普通的门户网站而已，但这种感觉其实是一种假象。我们会看到，在 Google 新闻门户与其他新闻门户之间具有很多重要的差别。必须一开始就申明一点，我们并不是要沿着 Google 的做法来实现，并且我们也不知道智能技术在何处及如何被用在 Google 新闻网站中，我们只知道终端用户能用到的那些功能。我们的目标是从一个更大的主题视角来展示我们的内容，而非帮助你复制某一个特定的网站。

Google 新闻网站与其他门户网的差别在哪？它到底有何独特之处？首先，它是基于从全世界成千上万个新闻源收集回来的内容，再由机器组织生成的。这些新闻被分组并分配类别，如国际、美国、商业、科技等。关键在于，分组与类别标定的过程都不需要人工干预，而是自动完成的。在读过第 4、5 章之后，你应该已经明白新闻分组是通过聚类完成的，而类别标定则是通过分类完成的，所以这个网站与我们的主题是相关的。还有很多我们感兴趣的主题。

有一个名为焦点新闻的栏目，很显然，它是通过相关性（排序）度量做出来的，这与我们在第 2 章介绍的内容类似。基于我们在本书中学到的知识，这样的一批新闻可以通过使用 PageRank 排序或基于文本的排序方法(类似于 DocRank)识别出来。用户权重可用于决定或至少影响这些排序结果，并提供更好的用户体验。

个性化同样是直接而明显的。页面的链接是个性化的，它允许你对页面上主题的顺序重新排列，也允许你引入自定义的栏目，这些栏目可以反映出你对特定类别新闻的兴趣。这些个性化的信息给那些有着大量用户的网站提供了极好的数据挖掘机会。我们还可以利用文章相似性以及用户对文章的偏好来做出推荐，换言之，我们可以基于用户评分与新闻内容来构建一个推荐系统，这与我们在第 3 章所看到的一样。

Google 新闻网另一个有趣的特征是“In the News”这个栏目。这是一个列表，标识了今日最“重要”的个人、非盈利组织、公司、政府之类的实体。当然，也可以仅仅依靠基本的统计信息来建立这样的功能，但当你深入思考这个问题时，你会发现某个特定名字的重要性是受多方因素影响的，而不仅仅是它出现在新闻中的频

Download at http://www.pin5i.com/

次。当内容来源于大量异构的数据源时尤其是这样，而这正是像 Google 新闻这样的网站所面临的情况。在这种情况下，我们可以从第 2 章的方法（像 PageRank 和 DocRank 那样的相关性度量方法）中获得极大的帮助。

281

新闻提醒功能让你可以通过邮件来接收内容，这些内容是由你定义的搜索关键词与媒体类型（例如新闻稿、视频、博客、讨论帖）决定的。只有当内容对用户是有“价值”的才应该发出提醒，很显然，不同的用户对于“价值”的评判标准是不一样的，这时智能算法又可起作用了。

这个网站的设计者应该不希望失去本地化的特性，所以很自然地，Google 新闻网上有大量的基于新闻使用语言与用户地理位置的可定制选项。在我们的讨论中，这只是一些从 NLP（自然语言处理）角度来说是重要的特性，所以我们不会关注它们。要得到关于 NLP 的优秀的综述，我们推荐参考 Jurafsky 与 Martin 的书。

作为项目启动，我们首先介绍如何抓取新闻网站的内容，以及把新闻聚合在系统中以备将来之用。

7.2 获取并清洗内容

一个如 Google 新闻网那样的新闻门户的起点，当然是收集新闻。通过爬虫从互联网不同的数据源把文章收集回来。已经有几个现成可用的免费爬虫，但出于方便性与完备性的考虑，我们会使用专门为本书写就的爬虫，它是我们在第 2 章使用的爬虫的变体。

我们还会对搜索预备知识进行一次回顾，因为经过前面 6 章对智能算法的学习之后，现在我们可以从一个新的视角来重新审视每一个步骤了。尽管你可以通过抓取网页来获得自己的新闻数据，但为保证实验的可重复性，我们的例子会使用一个确定的数据集。在本节的末尾，我们会描述将用在本书剩余部分的默认新闻数据集。

7.2.1 各就位、预备、开抓！

如你所预期的，我们的新闻爬虫名为 NewsCrawler，它的构造函数有三个参数：

- 存储抓回数据的根目录。
- 链接结构遍历的深度。

Download at http://www.pin5i.com/

- 抓取文档数量的最大值。

让我们在 BeanShell 中使用它来抓取 Manning 网站[2]。代码清单 7.1 显示了仅有的几个配置爬虫的步骤。

代码清单 7.1 抓取 Manning 网站

```
String rootDir = Ch7Constants.CRAWL_DATA_ROOT_DIR;←设定根目录
NewsCrawler crawler = new NewsCrawler(rootDir, 2, 100);
crawler.addSeedUrl("http://www.manning.com");
crawler.run();
```

282

第一件事情就是设定抓取的文档应该存放的位置。默认情况下，我们使用 C:/iWeb2/data/ch07/news-crawls 这个路径，但你可以通过修改 CRAWL_DATA_ROOT_DIR 的值将文档放在任意目录中。当爬虫开始从访问 URL 中获取内容时，它会把数据存放在这个根目录的子目录里，新子目录的名字会以 crawl-字符串打头，然后紧跟一个以微秒（μs）表示的爬虫的时间戳，如 crawl-1200697910111。NewsCrawler 类把它的工作交付给 BasicWebCrawler 类，就像第 2 章中 FetchAndProcessCrawler 类所做的那样。

让我们回忆爬虫工作的成果。BasicWebCrawler 类会获取数据并解析数据，处理的结果会存储在 processed 子目录下。对于每一组被处理过的文档，都有 4 个子目录——fetched、knownurls、pagelinks 与 processed。fetched 目录包含原始的 HTML 页面，knownurls 目录包含在抓取过程中识别出来的所有 URL 地址所组成的一个文件，pagelinks 目录包含所有访问过的 URL 地址相互链接所组成的一个文件，processed 目录的结构看起来跟 fetched 目录类似，但这时网页的内容是被分解过的，特定的页面属性（如页面标题、与 URL）跟核心文本是分开存放的，每个页面的出链也被抽取出来分别存放。如果你把 fetched 子目录的文件与 processed/content 子目录的文件加以比较，你会发现后者的文件不同之处在于已经删除了大部分 HTML 结构信息，如 header、table、div 等。在一个一般配置的笔记本电脑或 PC 上运行爬虫，在正常速度的互联网连接速度下，爬虫用几分钟就可以完成它的工作。

我们不会深入讨论爬虫的工作机制，因为照目前来看，它并没有涉及什么智能方面的东西。但在本书中随带一个基本的爬虫，可以吸引你把智能算法应用于爬虫

2 请你在抓取的时候遵守互联网的基本公德。我们所有的抓取例子都设定了一个很小的深度值与一个（相对）较小的文档数量。

Download at http://www.pin5i.com/

之中。参看本章的第一个 To Do 事项，就是这方面的挑战。

7.2.2 搜索预备知识回顾

在第 2 章，我们曾经审查过搜索的基本步骤：

- 抓取
- 解析
- 分析
- 索引
- 搜索

为了完整起见，让我们回顾每一个步骤的内容。可以把解析与分析这两步统称为清洗。在第 2 章介绍的文档解析器用于把不同的文档（如 XML、HTML、Word、PDF）转换为一般的、纯粹的文本形式。对于 HTML 解析，我们使用了来自 NekoHTML 项目（http://nekohtml.sourceforge.net/）的代码。NekoHTML 包含了一个简单的 HTML 解析器，它可以扫描 HTML 文档并修改大量 HTML 文档常见的错误。它的特点包括补全缺失的父元素，用可选的结束标签自动关闭元素，以及处理不匹配的内联元素标签。NekoHTML 非常鲁棒，使用速度也足够快，但如果你只抓取特定的一些网站，你也许想写一个自己的解析器。

283

如果你计划抓取 PDF 文档，可以使用 PDFBox 项目（http://www.pdfbox.org/）的代码，它是基于 BSD 许可的，有大量的文档。PDFBox 包含 LucenePDFDocument 类，该类可用于获取 Lucene 的 Document 对象，且只需要如下一行代码就可立即实现：

```
Document doc = LucenePDFDocument.convertDocument(File file)
```

参看 Java 文档可获得更多的信息。有 PDF 文档的解析器，当然也有 Word 的解析器，Apache POI 项目就提供了处理基于微软 OLE 2 复合文档格式的文件的 API，该 API 纯粹使用 Java 实现。此外，TextMining（http://www.textmining.org/）也提供了从 Microsoft Word 97、2000、XP 与 2003 文档中提取文本的 Java 库。

分析文档这一步同样很重要，在本书中，我们都是使用 Lucene 的 Standard Analyzer 类提取各个文档中文本的“意义”。很显然，NLP 是成功的文档分析的关键，附录 D 提供了 NLP 的简介和大量的参考文献。NLP 的目的在于帮助我们保留那些最重要且最能表达文档内容的词语，同时滤掉不相关的词语。如果你在分析阶段漏

Download at http://www.pin5i.com/

掉了某些对你来说有意义的词语，你就无法在后续的数据处理中再利用这部分信息了。反过来说，如果你囊括了大量不相关或不重要的词语，你的处理质量就会受制于数据中的大量噪声，结果是当你开始对新闻进行聚类或分类时，就无法得到有意义的聚簇，甚至可能会把商业文章归类到体育类别中。

基本上，如果你拿到的是“肮脏”的数据，也无法指望聚类或分类算法能给出合理的结果。作为类比，想象一辆因为燃料混合物不纯而无法启动的汽车，无论是多么好的汽车，只要燃料混合物没有达到适于燃烧的汽油空气比，它就无法发动起来。数据清洗有助于把算法的杂质去除，并得到合适的燃料混合物。

对于特定的网站来说，清洗的力度取决于网站设计的复杂程度。现代网站有大量的图片或富媒体组件，相对于纯粹的 HTML 页面或 PDF 文档来说，其内容的获取与处理要困难得多。对于互联网上一般的语料数据，我们的代码以及引用的工具在很大程度上都可以处理得不错。 284

一个专业的爬虫应该可以检测某个网站的语言（甚至地区），并对其使用最合适的清洗工具。一旦开始抓取，你就有可能到达任何一个地方，你也许会从来自美国的一组 URL 出发，经过半个地球的漫游之后到达班加罗尔（印度）或中国北京。爬虫应该有能力去鉴定该如何合适地“读取”内容。

虽然识别服务器地理位置已经有标准的技术，但却很可能从一个源获得多种语言混合的内容，又或是在印度抓回英文文章，在美国的服务器抓回中文的内容。这就是可以让我们的分类器再次闪耀的地方了。例如，朴素贝叶斯算法是与语言无关的，所以它可被用于识别特定文档或文档部分内容的语言。相比于我们曾用进行新闻分类的朴素贝叶斯分类器来说，这是一个有着同样的算法但却是不同的分类器（我们可以称为 NBLanguageDetector）。这个分类器的训练集会包含经过简单解析但未分析过的文本，而分类的概念则会是地区或语言。这里我们不会再细究这个例子，但如果你以后处理多语言文本时，可以把它翻出来想一想。另外，考虑世界各地存在的多种编码方式的情况可能会带来什么问题，它对于正确地检测语言又会造成什么影响。

7.2.3 一个抓取并处理好的新闻数据集

在本章的余下部分，我们会使用一组网页集做实验，它包含了我们在第 2 章见过的那些网页。为了使案例更具体且实验具有可重复性，拥有一个默认的数据集是

Download at http://www.pin5i.com/

有必要的。你可以在 data/ch07/目录中找到这些网页，所有的页面都存放在 all 子目录下，该目录包含（内容的选择是随机的）以下内容：

- 20 个关于商业新闻的文档。
- 24 个关于体育的文档。
- 23 个关于健康与医药的文档。
- 26 个关于技术新闻的文档。
- 16 个关于美国政治的文档。
- 20 个关于国际新闻的文档。

这意味着我们的新闻门户在默认情况下会聚焦于这 6 个主题类别，稍后会讨论如何添加自定义的类别。现在，让我们载入默认的数据集并看看这些新闻文档。代码清单 7.2 显示了如何载入一个文档集，与第 2 章的方法类似。我们提供了 News Dataset 接口的两种实现，第一种是 CrawlResultsNewsDataset，它可以用于任意（几乎所有）由代码清单 7.1 中的 NewsCrawler 类从网络爬回来的文档。第二个实现是代码清单 7.2 的 FileListNewsDataset 类。两者的区别为：对于 FileListNewsDataset，新闻的类别隐含在文档的名字中；而对于 CrawResultsNewsDataset，新闻的类别是未知的。

285

代码清单 7.2 载入默认的新闻数据集并启动一个自定义的新闻浏览器

```
NewsDataset dataset = new FileListNewsDataset("DefaultDS");
dataset.setDocumentDir("C:/iWeb2/data/ch07/all");←设置文档目录
dataset.setTopTerms(15);←要记住的单词的数量
dataset.loadTopics();
dataset.loadStories();
NewsUI ui = new NewsUI(dataset);
NewsUI.createAndShowUI(ui);
```

基本的 UI 显示在图 7.1 中，它提供了一个能快捷地了解新闻内容的途径，并随着各种智能特性的逐渐加入，我们也能方便地检查工作成果。在图 7.1 左边的面板，你能看到 6 个新闻类别及其相应的新闻报道，每个主题文章的数量显示在括号内。如果你单击某个新闻报道，就可以在右边面板看到它的内容。

总之，新闻的收集与清洗过程看起来也许很肤浅、很平凡，但它确实是一项有挑战性的工作。要使大规模的数据抓取有效且高效，要对来自各色各样的网站的文档进行清洗，这都是很艰巨的任务。在我们的应用环境里，数据抓取与清洗可以说

286

Download at http://www.pin5i.com/

是培育智能算法成长的富饶土壤。下面介绍对新闻索引与搜索的处理。

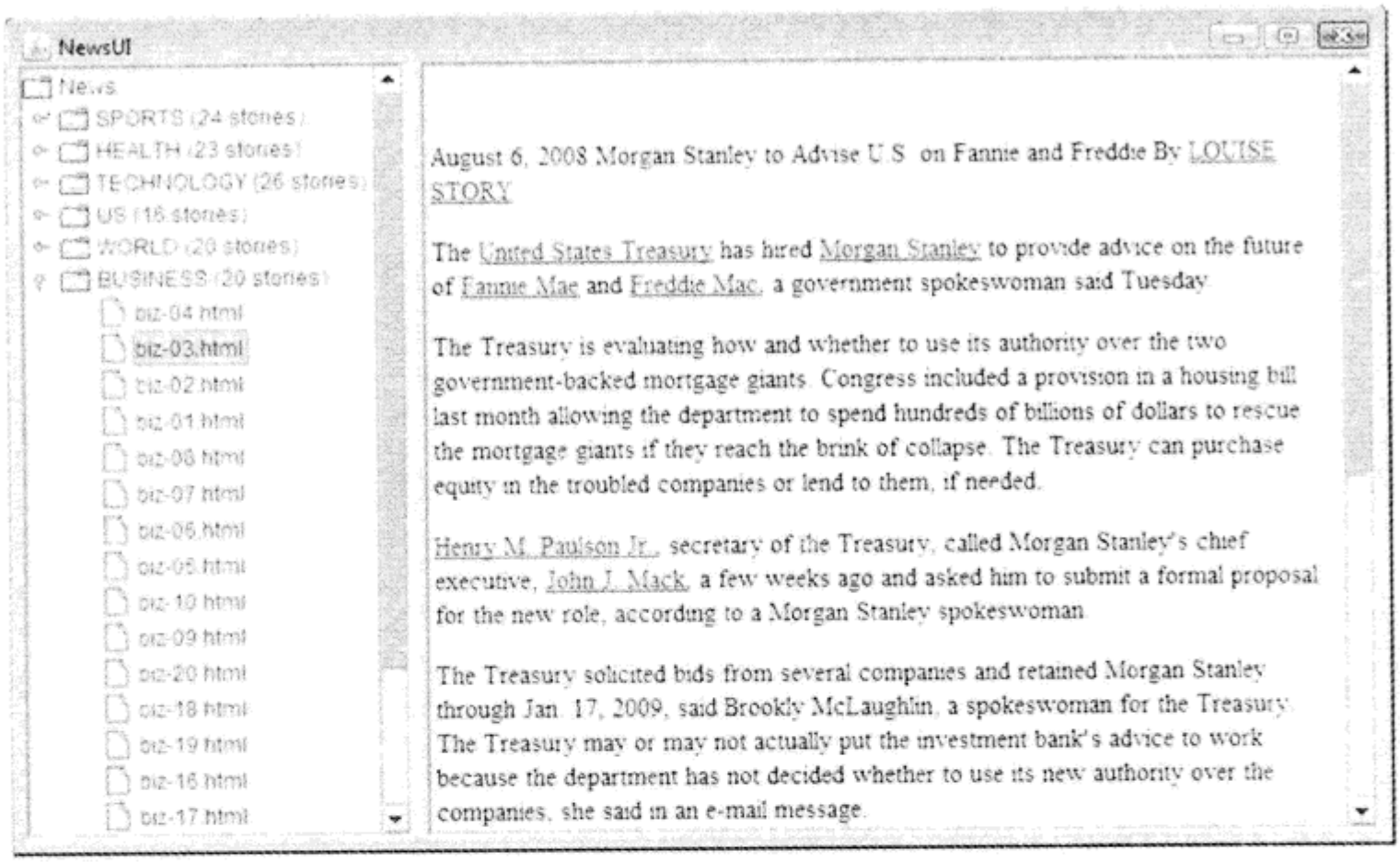


图 7.1 一个自定制的用于浏览默认新闻报道的用户界面

7.3 搜索新闻

通常，我们会使用 Lucene 来索引并搜索这些文档，但我们会把新闻门户的所有处理功能都封装到 NewsProcessor 类中。代码清单 7.3 生成了 NewsProcessor 的一个实例，为默认的数据集创建索引，并做了再次查询。这里默认已经在同一个 Shell 环境里执行过代码清单 7.2 的代码。

代码清单 7.3 索引并搜索默认的新闻数据集

```
NewsProcessor newsProcessor = new NewsProcessor(dataset);
newsProcessor.buildIndexDir();
newsProcessor.runIndexing();
newsProcessor.search("cell",5);
newsProcessor.search("football",5);
```

我们曾在第 2 章讨论过索引与搜索，所以现在只关注上面的结果，并分析各种

Download at http://www.pin5i.com/

有助于提升新闻门户的用户体验的因素。对“cell”的搜索的返回结果如图 7.2 所示。

```
bsh % newsProcessor.search("cell",5);

Search results using Lucene index scores:
Query: cell

Document Title: health-21.html
Document Terms: warn, limit, phone, cell, children, us, risk,
herberman, ronald, becaus, especi, institut, brain, cancer, dr
Document URL: file:/C:/iWeb2/data/ch07/all/health-21.html -->
Relevance Score: 0.610166192054749

---------------------------------------------------------------
Document Title: health-23.html
Document Terms: phone,concern,cell,us,tumor,research,risk
herberman,pittsburgh,dont,publish,studi,brain,institut,cancer
Document URL: file:/C:/iWeb2/data/ch07/all/health-23.html -->
Relevance Score: 0.537154197692871

---------------------------------------------------------------
Document Title: tech-15.html
Document Terms: control, home, phone, can, your, cell, starner,
you, us, technolog, what, could, kientz, gestur, we
Document URL: file:/C:/iWeb2/data/ch07/all/tech-15.html   -->
Relevance Score: 0.341093242168427

---------------------------------------------------------------
Document Title: health-02.html
Document Terms: work, c, tumour, treatment, cell, inject, suggest
research, dose, bbc, human, mice, becaus, vitamin, cancer,
Document URL: file:/C:/iWeb2/data/ch07/all/health-02.html  -->
Relevance Score: 0.337665110826492

---------------------------------------------------------------
Document Title: health-09.html
Document Terms: telomeras, have, chemic, structur, cell, block
most, activ, bbc, help, new,  could,  immort,  cancer,  drug,
Document URL: file:/C:/iWeb2/data/ch07/all/health-09.html     -->
Relevance Score: 0.334201782941818
```

图 7.2　仅仅利用索引信息的单词“cell”的搜索结果

文档标题（document title）是对一篇新闻的标识，即之前新闻浏览器中左边面板的名字；文档单词（document terms）是经过分析后每个文档中前 15 个频繁单词。相关得分（relevance score）是 Luncene 给每个文档的打分。你的新闻浏览器现在应该是开启的，可以单击左边面板里的各个文档标题以查看全文内容。头两则新闻是关于移动电话对人体健康的影响，第三个搜索结果是一则技术新闻——用移动电话来定位错位的物体，第四、五个搜索结果与移动电话完全无关，而是两则大量服用维生素 C 对延缓癌症扩散有潜在作用的文章。

Download at http://www.pin5i.com/

这些结果很典型，你可以在 Google 新闻网上进行同样的搜索，会得到涉及不同主题的大量文章。这些结果会带来一些很重要的问题。首先，当用户通过新闻门户搜索“cell”时，她到底想要什么？是干细胞研究？是某些技术器材？是移动电话的电磁辐射对人的影响？很显然，脱离上下文来回答这些问题是毫无意义的。

我们难道不可以使用在第 2 章学到的 PageRank 或 DocRank 算法吗？答案是肯定的。这些排序算法有助于改善搜索结果，但效果有限。排序算法提升最明显的地方在于识别出一个网站相对于其他网站（检索链接图的一部分）的重要性，但新闻报道中大部分的外链都是自引的，甚至是没有外链的（例如 PDF 格式的新闻报告、新闻通知等），在第 2 章中它们被称为悬孤节点（dangling nodes）。 287

不要觉得沮丧！我们还可以有别的方法来改善用户体验。我们在第 2 章中学到的另一项技术是用户点击，看起来，在新闻门户网站中，用户点击也许与我们改善搜索结果的诉求更加直接相关。显然，这并不是给予新用户的红利，而是照顾老用户的长远利益。我们在较早前提到，相关性是很主观的，这也是为什么相关性的计算很困难的原因之一。如果我和你都在寻求“cell”这个查询结果，你感兴趣的也许是干细胞的研究，而我感兴趣的也许是 Android 或 Nokia 最新的配件。所以，对于一个用户来说，最相关的结果常常并非是另一名用户最相关的结果，就算两者的查询词完全一样！

在我们的新闻门户中，用户点击可以应用在两个层面，第一个层面是用户点击的单篇文章，跟第 2 章的例子类似；第二个层面的信息是每篇文章所属的主题。此外，还有一个改善搜索结果的方法是把查询词与文档词都放在属性列表里做训练，这正是本章末第二个 To Do 事项的主题。 288

这些例子同样提醒我们 NLP 在搜索领域的重要性。具体来说，有三个高层次的自然语言的因素是无法从低层次的语法或句法结构中直接获得的，它们是语义、语用和篇章。语义——关于意义的知识——利用我们对世界的知识；语用涉及词义与用户意图间的关系；篇章包含的词汇结构比单一的观点或句子蕴含了更多的知识。这些高层次的自然语言概念对于搜索结果有着特别的重要性，它的普遍性也已经超越了语言处理，它就是我们称为“大局”或“上下文”的东西（参看附录 D 及其文献）。

智能搜索的价值不可小视，但通过搜索把相关的新闻检索出来并不是把新闻展现给用户的唯一途径。实际中，新闻门户的布局就反映了其潜在的分类计划——把页面组织为多个新闻类别。下面我们会进入给新闻分配类别的阶段。

Download at http://www.pin5i.com/

7.4 分配新闻类别

给新闻分类的工作可以单独针对一篇新闻报道，也可以针对一组新闻，本节的内容会两者兼顾。我们曾在第 5 章讨论过分类问题，这里也会使用这些方法把新闻归类到合适的新闻类别中。但是，当我们在一个功能更强大的包括了搜索、聚类、推荐功能的应用环境中考虑分类这个问题时，会出现很多问题，这些问题都是你孤立地进行分类操作时难以浮现的。

本节的第一个主题是算法使用中的顺序效应。一旦爬虫完成抓取工作，我们就得到了无序、无结构的新闻文档。为了组织这些文档，一种可行的操作流程是把新收集回来的文档都拿出来，并对它们分组；另一种方案是首先把每篇新闻报道归类到六个主题中的一个，然后在各个主题内对相似的文章进行分组。你认为这两个操作是等价的吗？对于一般的新闻文档来说，它们几乎是不可能等价的，从技术术语方面说，这两个操作是不可交换的。

接下来的两节（7.4.2 节与 7.4.3 节）详细介绍了自定制的新闻分类器。你会再次见识到朴素贝叶斯算法作为一个通用型分类器的“多才多艺”。这个自定制的分类器最有趣的部分也许是分类策略（classification strategy）的引入，所以在本节的最后一节，我们解释了什么是分类策略，为什么我们需要它，以及该如何才能有效地利用它。

289

下面开始给新闻报道分配新闻类别。我们会比较两种组合方式：对聚类后的结果进行分类，以及对分好类别的新闻进行聚类。

7.4.1 顺序问题

我们已经暗示过，不同的操作顺序会导致不同的结果，但结果会有多大的不同？哪一种操作顺序是更好的呢？这两个问题的答案取决于数据的性质和算法本身，然而一般的观点都会倾向于第二种方式（先分类，后聚类），因为数据的噪声在分类之后都会得到显著的遏制，剩下聚类算法的工作就容易得多了。

图 7.3 显示了把两个操作（分类与聚类）作用于默认的数据集上的结果，顺序是先聚类，后分类。

Download at http://www.pin5i.com/

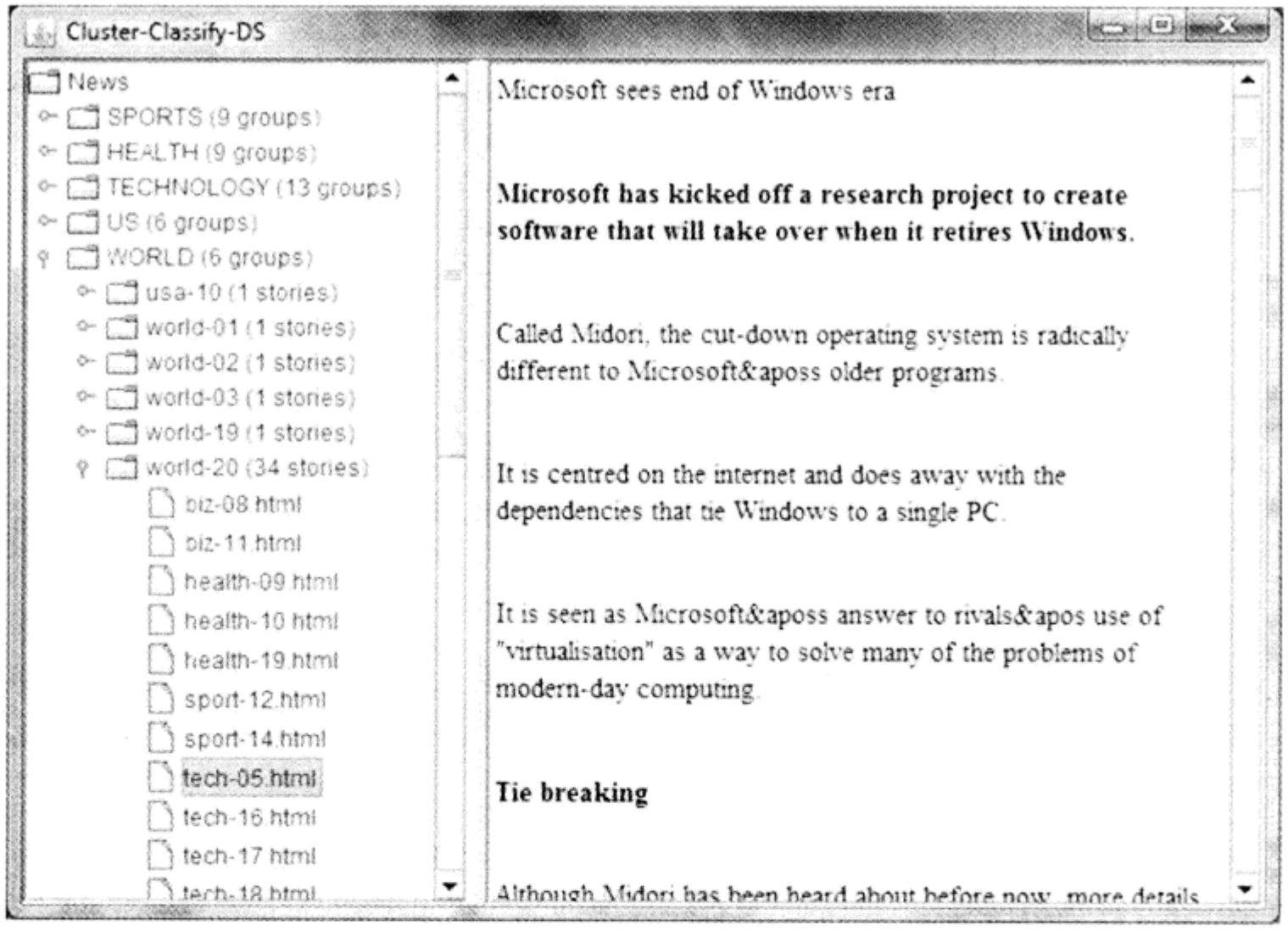


图 7.3 先对文章聚类，然后把它们分类到各个主题

图 7.3 中被选中的那篇文章（tech-05）是一个研究项目，即关于微软开始取代 Windows 操作系统家族之路。很显然，这是一篇技术文章，但却被划分到国际新闻里。具体来说，这是因为它与一篇关于俄罗斯作为政治与经济大国复苏的国际新闻聚到同一个群组里，但是，从概念上说，这两个主题差得要多离谱就有多离谱——除非，你有丰富的联想。 290

就在和俄罗斯复苏那篇文章的同一个群组里，我们还发现了大量如 tech-05 那样和俄罗斯根本毫不相干的文章，甚至连国际新闻都不算。默认数据集有 129 篇文章，仅这个群组就占了 34 篇。默认数据集的文章占各个类别的数量为：20 篇商业类文章、23 篇健康类文章、24 篇体育类文章、26 篇技术类文章、16 篇美国国内新闻、20 篇国际新闻。所以，一个拥有 34 篇文章的群组显然是不平衡的，除非聚类算法经过极大的调整，否则必然会出现类别不均衡的情况。

代码清单 7.4 展示了如何创建这些新闻聚簇，并把它们显示在自定义的浏览器里。你现在就可以执行这些脚本，并考察分组的结构及文章在各个主题中的分布。

Download at http://www.pin5i.com/

代码清单 7.4 通过先聚类后分类来创建新闻聚簇

```
NewsDataset trainingDS = new FileListNewsDataset("TrainingDS");←生成训练/测试数据集
trainingDS.setDocumentDir("C:/iWeb2/data/ch07/training");
trainingDS.init();
NewsDataset ds1 = new FileListNewsDataset("Cluster-Classify-DS");
ds1.setDocumentDir("C:/iWeb2/data/ch07/all");
ds1.init();
NewsProcessor newsProcessor = new NewsProcessor(trainingDS);
newsProcessor.trainClassifier();
newsProcessor.createClusters(ds1);←聚类新闻报道
newsProcessor.classifyClusters(ds1);←分类新闻报道的聚簇
NewsUI ui1 = new NewsUI(ds1);
NewsUI.createAndShowUI(ui1);
```

我们还会回到图 7.3，并对这个由代码清单 7.4 生成的分组结构进行更详细的讨论，现在先看看通过调换操作顺序（先分类，后聚类）得到的分组结构。图 7.4 显示了对同一个数据集分组的结果，只是这次是先分类，再聚类。

通过对比图 7.3 与图 7.4 可以立即得到一些直接的结果。在图 7.4 中，技术文章现在已经被聚到了一起，而且被正确地分配到技术类别中。所以跟 world-20 这篇文章聚到一块的文章都是与俄罗斯复苏相关的，而且商业与健康相关的文章再没有出现在这个聚簇中。与之前相比，这当然是一个更好的新闻分类结果，但仍然有一些瑕疵。技术文章 tech-16、tech-17、tech-18 都是涉及垃圾邮件的新闻，可以被聚到同一个聚簇中，但 tech-05 是与微软相关的新闻，与它们并不直接相关，不应该在同一个组里。

291 代码清单 7.5 展示了如何创建这些新闻聚簇，并把它们显示在这个自定制的浏览器中。它大部分的代码都跟代码清单 7.4 差不多，只有一个重要的差别，如我们所说的，聚类与分类操作的顺序做了交换。很自然，调用方法的名字也有改动，因

292 为这次是对每一个个体（而不是对群组）进行分类，而聚类则是在各个主题内部进行。我们使用了一个名为 ds2 的新的 FileListNewsDataset 类的实例，这样就可以对同一份数据进行操作，而分别显示操作的结果。如果你使用同一个实例（ds1），就无法比较这两种方式了。如果你在执行完代码清单 7.4 后未退出过 Shell，那么这次只需要从创建 NewsDataset ds2 这一行开始输入执行即可。

Download at http://www.pin5i.com/

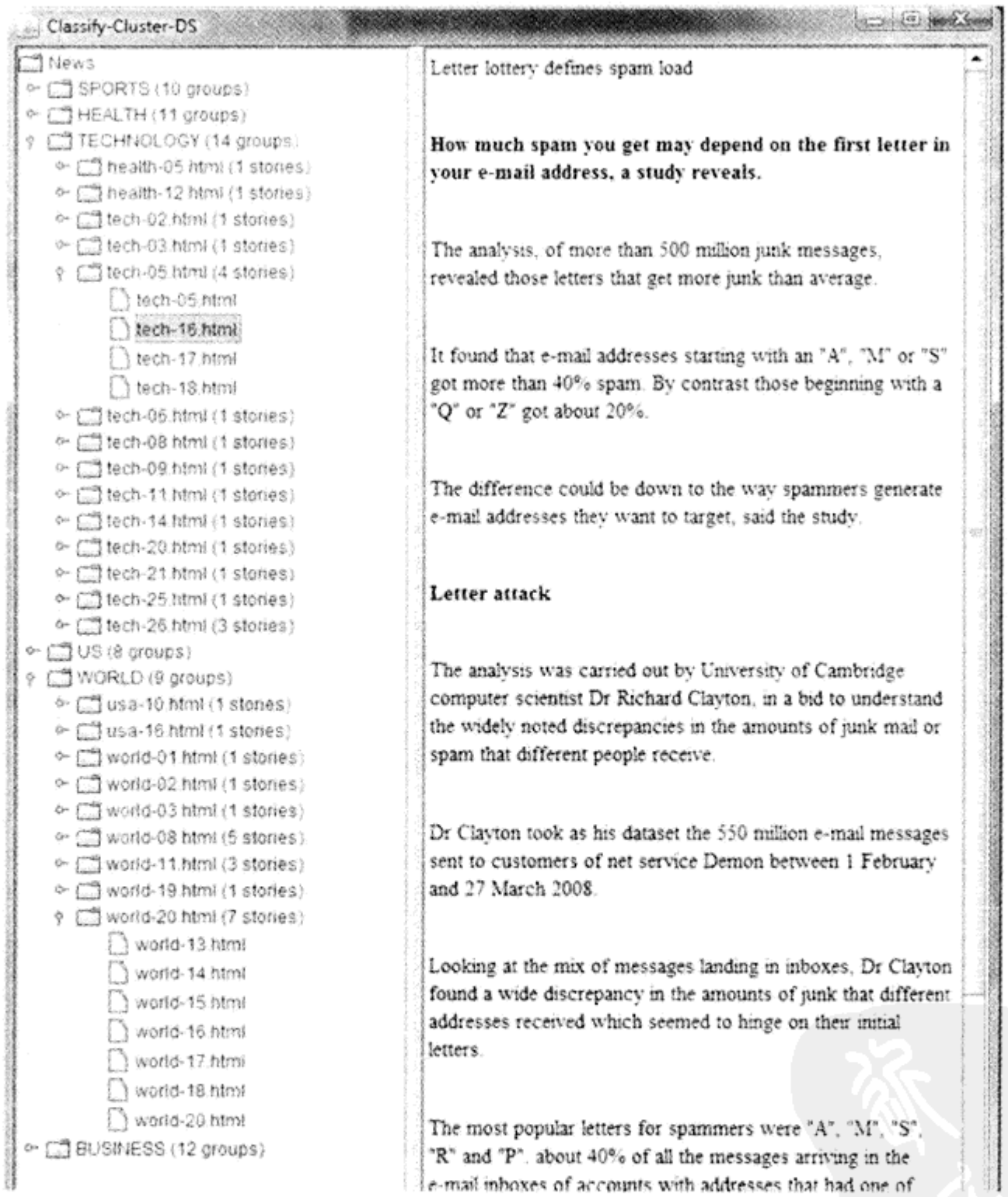


图 7.4　把新闻分类后再进行聚类

代码清单 7.5　通过先分类后聚类创建新闻分组

```
NewsDataset trainingDS = new FileListNewsDataset("TrainingDS");
trainingDS.setDocumentDir("C:/iWeb2/data/ch07/training");
trainingDS.init();
NewsDataset ds1 = new FileListNewsDataset("Cluster-Classify-DS");
ds1.setDocumentDir("C:/iWeb2/data/ch07/all");
ds1.init();
```

Download at http://www.pin5i.com/


```
NewsProcessor newsProcessor = new NewsProcessor(trainingDS);
newsProcessor.trainClassifier();
newsProcessor.createClusters(ds1);
newsProcessor.classifyClusters(ds1);
NewsUI ui1 = new NewsUI(ds1);
NewsUI.createAndShowUI(ui1);
NewsDataset ds2 = new FileListNewsDataset("Classify-Cluster-DS");
ds2.setDocumentDir("C:/iWeb2/data/ch07/all");
ds2.init();
newsProcessor.classifyStories(ds2);←分类所有的新闻报道
newsProcessor.createClustersWithinTopics(ds2);←聚类各个主题中的新闻报道
NewsUI ui2 = new NewsUI(ds2);
NewsUI.createAndShowUI(ui2);
```

现在，两个 swing 客户端 ui1 和 ui2 应该都是可见的。调整图 7.3（Cluster-Classify-DS）与图 7.4（Classify-Cluster-DS）中窗口的尺寸，并把它们放在一起，以检查各聚簇的结构，以及两个方法对新闻类别的分配。图 7.5 中仅显示了节点层次的第一层，窗口的区域也很小，在你的屏幕里可以伸展窗口，一半屏幕容纳左窗口，另一半容纳右窗口。

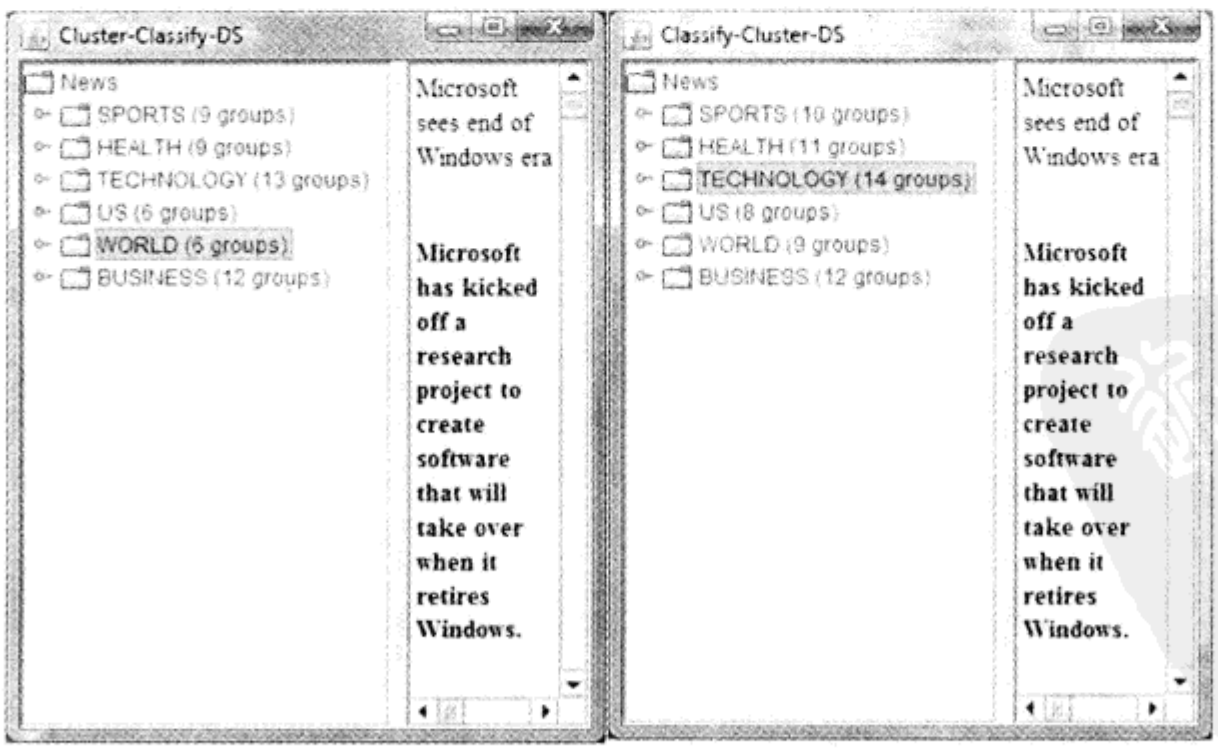


图 7.5　把图 7.3 和图 7.4 中两个不同的树形结构并列在一起

在两个例子里，数据同样都有分类和聚类结构，但聚类与分类操作的不同顺序导致了图 7.5 所显示的不同结构。通过并排对照可以很明显地看到，聚类–分类顺序

Download at http://www.pin5i.com/

得到的结果有更少却更大的聚簇，而分类-聚类顺序得到的结果则是更多的相当紧凑与干净的聚簇。

让我们深入查看聚簇数量差别最大的国际新闻类别，在聚类-分类方式中得到六个聚簇，而分类-聚类方式则得到九个聚簇。文章 usa-10 在两种方式中都被错分，它独立地存在于一个“干净”的聚簇中。 293

要注意的是，这两种方法都是应用在训练集的超集上的，即是说，usa-10 这篇新闻报道的文本信息跟训练集中的每一篇新闻报道都不相近。同样能看出来的是，与其他类别的新闻相比，它跟训练集中涉及国际新闻的报道更相似。这就提醒我们应该把这个例子（新闻 usa-10）纳入我们的训练集！这种分析或者说是质量保证机制（如果你喜欢这么叫的话），能增加系统的知识体系。

usa-16 同样被分类-聚类方法所错分，但这篇文章在聚类-分类方法中甚至都没有被处理的机会。如果你查看一下标记为 world-20 这个组，你会发现 usa-16 与其他 33 篇文章被聚到了一起。发生了什么？在聚类-分类方法中，我们聚类的对象是一批内容多样化的新闻报道，结果是像 world-20 这样的大组看来更加具有鲁棒性，聚类算法也无法再把它们分割为更小的块。当然，通过细调参数，你可以影响聚簇的粒度，但主要的问题仍然没有解决。在分类-聚类方法中，初始的分类对文档进行了划分，这就让后续的聚类工作变得更简单。所以，在国际新闻类别里文档间的噪声大大降低的情况下，聚类算法就可以检测出 usa-16 这篇文章其实并不属于任何其他的聚簇，跟 usa-10 类似，应该单独归为一组。顺便说一下，人们通常把只有一个元素的聚簇称为单例（singletons），如果你是一名软件专家，对这个术语应该很熟悉，它就来自著名的单例设计模式（singleton pattern）。 294

7.4.2 使用 NewsProcessor 类进行分类

现在开始深入介绍 NewsProcessor 类的分类工作的细节，我们会用单独的代码清单对每一个重要的步骤进行描述。首先是 NewsProcessor 的训练阶段，如代码清单 7.6 所示。它由分类器的训练所构成，因为对于聚类来说，并没有训练这一步。但要注意这里并不仅限于定义分类器后调用 train 方法而已。

代码清单 7.6 训练一个朴素贝叶斯分类器并选择一个分类策略

```
public void trainClassifier() {
  if( topicSelector == null ) {
    NBStoryClassifier storyClassifier =
```

Download at http://www.pin5i.com/


```
new NBStoryClassifier("NewsStoryClassifier",trainingDataset);
    storyClassifier.train();
    ClassificationStrategyImpl defaultTopicSelector =
new ClassificationStrategyImpl();←选择分类策略
    defaultTopicSelector.setStoryClassifier(storyClassifier);←为策略分配分类器
    topicSelector = defaultTopicSelector;
  }
}
```

类的名字 NBStoryClassifier 中的含义也许你也猜到了，我们会利用强大的朴素贝叶斯分类器来处理这个问题，在第 2 章和第 5 章都已经见过它。如所有的分类器都必须要做的，NBStoryClassifier 类实现了核心的 Classifier 的接口，并包含了特别的机制，可以将新闻转换为 NaiveBayes 分类器所使用的“语言”——Concept 与 Instance 类。

我们马上就会看到 NBStoryClassifier 类，但首先讨论一下代码清单 7.6 中出现的分类策略的选择。ClassificationStrategyImpl 类做了什么事？这是一个我们未曾见过的类，为什么会需要它？因为所有之前遇到过的分类案例都属于把一系列的实例分类为相应的概念，而现在新闻门户这个例子需要处理的问题是把实例组（groups of instances）分类为相应的概念（新闻类别），有不止一种方法可以做到这件事情，所以需要一个策略以决定该怎么做。

其中一种策略就是选择组内最有“代表性”的文章，然后对它分类，这样，只要该代表性的文章被分类，这个组就被分类了，这有点像美国总统选举团的工作机制。另一个策略是通过多数票决规则来进行分类。你在阅读代码之前，尝试思考其他可行的策略，并考虑策略上的差异会对最终的新闻结构造成的影响。

295

总之，NewsProcessor 类的分类过程比我们之前见过的要复杂得多，它依赖于两个类——NBStoryClassifier 与 ClassificationStrategyImpl。第一个类帮助我们使用朴素贝叶斯分类器，第二个类则封装了组分类的策略决策部分。

7.4.3 分类器

在进一步深入代码之前，让我们仔细查看这两个类。NBStoryClassifier 的代码显示在代码清单 7.7 中，按照惯例，我们已经移除了一些无关紧要的方法以及注释等。除了每个 Classifier 都必须实现的核心的 train 与 classify 方法外，还有几个辅助方法

Download at http://www.pin5i.com/

用于为新闻报道创建 Instance 对象，及为新闻类别创建 Concept 对象。

代码清单 7.7 一个使用了朴素贝叶斯算法的新闻分类器

```
public class NBStoryClassifier implements Classifier {
  private List<ClassificationResult> conceptScores;
  private NaiveBayes nbClassifier;
  private boolean verbose = true;
  public NBStoryClassifier(String name, NewsDataset ds) {
    TrainingSet tSet = createTrainingSet(ds);
    nbClassifier = new NaiveBayes(name, tSet);
  }
  public TrainingSet createTrainingSet(NewsDataset ds) {
    int nStories = ds.getSize();
    List<Instance> allTrainingInstances =
      new ArrayList<Instance>(nStories);
    Iterator<NewsStory> iter = ds.getIteratorOverStories();
    while( iter.hasNext() ) {
      NewsStory newsStory = iter.next();
      Instance instance = toInstance(newsStory);❷
      allTrainingInstances.add(instance);
    }
    Instance[] instances =
      allTrainingInstances.toArray(new Instance[nStories]);
    return new TrainingSet(instances);
  }
  public boolean train() {
    TrainingSet tSet = nbClassifier.getTset();
    for(String attributeName : tSet.getAttributeNameSet() ) {
      nbClassifier.trainOnAttribute(attributeName);
    }
    return nbClassifier.train();
  }
  public Instance toInstance(NewsStory newsStory) {❷
    Concept concept = toConcept(newsStory.getTopic());
    String[] terms = newsStory.getTopNTerms();
    StringAttribute[] attributes = new StringAttribute[terms.length];
    for(int i = 0; i < terms.length; i++) {
      String name = terms[i];
```

❶

❸

296

Download at http://www.pin5i.com/


```
        String value = "Y";
        attributes[i]= new StringAttribute(name, value);
      }
      return new BaseInstance(concept, attributes);
    }
    private Map<String, NewsCategory> allTopics =
    new HashMap<String, NewsCategory>();
    private Map<String, Concept> allConcepts =
    new HashMap<String, Concept>();
    public Concept toConcept(NewsCategory t) {
      if( t == null ) return null;
      String topicName = t.getName();
      Concept c = allConcepts.get(topicName);
      if( c == null ) {
        c = new BaseConcept(t.getName());
        allConcepts.put(topicName, c);
      }
      return c;
    }
    public Concept classify(Instance instance) {❹
      conceptScores = new ArrayList<ClassificationResult>();
      Concept bestConcept = null;
      double bestP = 0.0;
      TrainingSet tSet = nbClassifier.getTset();
      for (Concept c : tSet.getConceptSet()) {
        double p = nbClassifier.getProbability(c, instance);
        ClassificationResult cR = new ClassificationResult(c, p);
        conceptScores.add(cR);
        if( p >= bestP ) {
          bestConcept = c;
          bestP = p;
        }
      }
      return bestConcept;
    }
  }
```

❸

297

❶ createTrainingSet 方法是 NBStoryClassifier 主要做的事情，因为我们打算把分

Download at http://www.pin5i.com/

类的任务交给标准的 NaiveBayes 分类器来实现，但首先我们需要有一个 TrainingSet 对象。

❷ toInstance 方法以 NewsStory 对象作为输入，生成一个 BaseInstance 对象。为了创建这个对象，我们需要具备两个因素：一是 Instance 所属的 Concept 对象，二是一组属性及其取值。属性的数量取决于文章中频繁词的数量，这个量的默认值（设置为 25）在类 FileListNewsDataset 中定义。我们在代码清单 7.5 中生成训练集与两个测试集（ds1 和 ds2）时隐式地使用过这个值。修改频繁词的数量会改变分类的结果，要修改这个值，可以在你调用数据集的 init 方法之前调用 setTopTerms 方法，然后观察其结果与我们之前得到的结果的差异。

另一种方案是可以修改 toInstance 方法的实现，让它不要在所有的频繁词上训练。这项练习在本章第三个 To Do 事项中有所描述。

❸ toConcept 方法是创建数据集的第二个重要的辅助方法，toInstance 也依赖于它。它的实现非常直观，这里也没有太多可以变化的空间。我们得到 NewsCategory 的名字，并生成 BaseConcept 类的一个对象，同时记录下曾遇到过的所有概念。

❹ 如果你把这里的 classify 方法实现与 NaiveBayes 类的 classify 实现相比较，你会发现它们几乎是一样的。不同之处在于这里我们把实例在所有概念中的分类得分存储在 conceptScores 变量中，这个变量在下一次调用 classify 前都会有效。这种方式给我们后续操作带来了更多的选择，当我们使用 ClassificationStrategyImpl 为每个新闻或新闻聚簇分配类别时有更大的自由。

ClassificationStrategyImpl 类的目的在于定义两个分类新闻报道可用的方法，第一个是对新闻聚簇分类；第二个是对单一的新闻报道分类。让我们看看这个类是如何工作的。

7.4.4 分类策略：超越底层的分类

ClassificationStrategyImpl 类实现了 ClassificationStrategy 接口，从两个可选的方法中选择一个用于分类新闻。具体来说，assignTopicToCluster 方法与 assignTopicToStory 方法分别封装了这两个方法——这里的 topic 等同于新闻的类别。闲话少说，下面让我们通过代码清单 7.8 的代码看看这个类到底干了什么，清单中并没有包含该类的全部代码。

Download at http://www.pin5i.com/

298

代码清单 7.8 ClassificationStrategyImpl：定义一个分类策略

```
public class ClassificationStrategyImpl implements ClassificationStrategy {
  private NBStoryClassifier storyClassifier;
  public void assignTopicToCluster(NewsStoryGroup cluster) {❶
     List<NewsStory> newsStories = cluster.getStories();
     NewsStory rpStory = selectRepresentativeStory(newsStories);❷
     NewsCategory bestTopic = selectBestMatchingTopic(rpStory);❸
     cluster.setTopic(bestTopic);❹
     cluster.setRepresentativeStory(rpStory);
  }
  public void assignTopicToStory(NewsStory newsStory) {❺
     Instance instance = storyClassifier.toInstance(newsStory);
     Concept concept = storyClassifier.classify(instance);
     NewsCategory bestTopic = storyClassifier.toTopic(concept);
     newsStory.setTopic(bestTopic);
  }
  private NewsStory selectRepresentativeStory(List<NewsStory> newsStories) {
     return selectLongestStory(newsStories);
  }
  private NewsStory selectLongestStory(List<NewsStory> newsStories) {
     NewsStory representativeStory = null;
     int maxContentLength = 0;
     for(NewsStory newsStory : newsStories) {
        int storyContentLength = newsStory.getContent().getText().length();
        if( storyContentLength > maxContentLength ) {
           maxContentLength = storyContentLength;
           representativeStory = newsStory;
        }
     }
     return representativeStory;
  }
  private NewsCategory selectBestMatchingTopic(NewsStory newsStory) {❸
     Instance instance = storyClassifier.toInstance(newsStory);
     Concept concept = storyClassifier.classify(instance);
     return storyClassifier.toTopic(concept);
  }
}
```

❻

Download at http://www.pin5i.com/

❶ assignTopicToCluster 方法封装了两种可选策略中的一种：应该对给定的新闻聚簇分配什么类别。这里的实现是很简单的，它从新闻聚簇里找出一篇文章来代表该组别，对这篇具有代表性的文章进行分类，然后把该组内的所有文章都归为代表文章所属的类别。本章的第 4 个 To Do 事项邀请你尝试更多的为聚簇分配类别的策略。

299

❷ 代表文章的选择仅仅取决于 selectLongestStory 方法的结果，但你也可以把它实现得复杂一些，同样可以参看本章第 4 个 To Do 事项。

❸ 你也许想知道我们是否可以用 assignTopicToStory 取代 selectBestMatchingTopic 方法。毕竟，后者只是简单地调用了分类器，与前者并没什么区别。但实际中，两个方法并不一定相同，这也是为什么要显式地把它们作为不同的方法来调用的原因。

❹ 这一步记录各聚簇的代表文章与其所分配的类别。

❺ 如前所述，这里的实现跟 selectBestMatchingTopic 方法一样，它利用 NBStoryClassifier 类把新闻报道归类到一个新闻类别。如果仅仅只是把分类的工作交给 NBStoryClassifier 来完成，这个方法并不特别有用，但我们可以以这个分类器作为起点。如果对分类结果做一定的后处理，或者在传送给分类器之前先把文章做预处理，这个方法的价值就会好得多。

❻ selectLongestStory 方法给一个列表文章里代表性的文章的选择提供了一个简单的启发式方法。这种选择方式的初衷在于：文章越长，它与其他文章的重叠就越多。这种方式不算很精妙，基于我们在之前章节（特别是第 4 章）的讨论去改善会对结果有显著的提升。本章第 4 个 To Do 事项有一些可供探索的想法。

总的来说，ClassificationStrategyImpl 类扮演的就是一个元算法的角色。换句话说，它是一个框架，我们可以基于自己的认识（数据）与自己的工具（基础算法），把商业逻辑或复杂的组合技术注入其中，然后它可以帮我们做出最佳的选择。第 6 章曾研究过几种分类器组合的技术，它们同样也是元算法，所以同样可以用于此处，与其他的启发式的或商业的规则一起优化新闻类别分配的结果。

让我们重回 NewsProcessor 类本身，代码清单 7.9 显示了 ClassificationStrategyImpl 类是怎样在它的两个方法（classifyClusters 与 classifyStories）之间做出选择的。这里没有直接包含 NBStoryClassifier 类，ClassificationStrategyImpl 就是所有分类相关工作的外在表现。在 NewsProcessor 中，无论是对新闻聚簇还是新闻报道的分类，都归为一行代码。

Download at http://www.pin5i.com/

代码清单 7.9 给新闻聚簇或新闻报道分配类别

```
public void classifyClusters(NewsDataset ds) {←对给定数据集中的所有群组分类
  List<NewsStoryGroup> clusters = ds.getStoryGroups();
  for(NewsStoryGroup cluster : clusters) {
     topicSelector.assignTopicToCluster(cluster);←委托给 ClassificatioStrategyImpl
                                                  处理
  }
}
public void classifyStories(NewsDataset ds) {←对给定数据集中的所有文章分类
  Iterator<NewsStory> iter = ds.getIteratorOverStories();
  while(iter.hasNext()) {
     NewsStory newsStory = iter.next();
     topicSelector.assignTopicToStory(newsStory)←委托给 ClassificatioStrategyImpl
                                                                    处理
  }
}
```

300

本节总结了对新闻聚簇与新闻报道的分类细节。我们展示了顺序对分类结果的影响，还展示了分类这个范畴——通过分类单一文章或一组文章——是怎样得到了延伸，让我们有了更大的自由来设计更有效的解决方案。具体来说，分类策略这个概念很自然地伴随着对新闻聚簇分类这一更高层面的分类决策而诞生。现在，我们需要更多地描述这些新闻聚簇形成的细节。

7.5 用 NewsProcessor 类创建新闻分组

我们曾在第 4 章讨论过聚类问题，这里再把它用于对新闻报道进行分组也不足为怪。在代码清单 7.5 中有两种可用于分组新闻的方法，第一种是 createClusters 方法，第二种是 createClustersWithinTopics 方法。当你第一次看到代码清单 7.10 的代码时，这两种方法看起来是通过不同的类来进行聚类操作的，但它们实际上都依赖于 4.5.1 节介绍过的 ROCK 算法。

代码清单 7.10 NewsProcessor 的两种聚类方法

```
public void createClusters(NewsDataset ds) {
  NewsClusterBuilder clusterBuilder = new NewsClusterBuilder();←使用 NewsClusterBuilder
  clusterBuilder.setNewsDataset(ds);
```

Download at http://www.pin5i.com/


```
    clusterBuilder.cluster();
  }
  public void createClustersWithinTopics(NewsDataset ds) {
    TopicalNewsClusterBuilder clusterBuilder =
    new TopicalNewsClusterBuilder();←使用 TopicalNewsCluster
    clusterBuilder.setNewsDataset(ds);
    clusterBuilder.cluster();
  }
```

如果代码中的两种方法最终都依赖于 ROCK 算法，那为什么我们还要创建两个独立的类？在目前的实现中，这两种方法最主要的区别在于第一种方法对 NewsDataset 中所有的文章聚类，另一种方法则对 NewsDataset 中的每个 NewsCategory 内的文章进行聚类。与前一节对分类的讨论类似，如果这两种情况被区别对待，就可以对它们分别进行改善以提高聚类质量。随着我们讨论的深入，这两种方法的差异会变得更加清晰，作为开始，接下来先不考虑新闻的类别，而只关注于聚簇的创建。 301

7.5.1 聚类全部文章

createClusters 方法应用于整个默认的新闻数据集的聚类结果显示在图 7.6 中，总共有 55 个聚簇——在这个图里你不能看全，但你可以在 swing 客户端上看到全部的聚簇。具体来说，有 42 个聚簇是单例，2 个包含 2 篇文章的聚簇，5 个包含 3 篇文章的聚簇，1 个包含 4 篇文章的聚簇，2 个包含 5 篇文章的聚簇，1 个包含 6 篇文章的聚簇，1 个包含 14 篇文章的聚簇，以及 1 个包含 34 篇文章的聚簇。

这些数字相对来说是很重要的，在一般的新闻数据集中，你会希望聚簇差不多是等规模的，而大多数的文章应该被归组，而不是作为一个单例存在。如果聚簇的分布是事先知道的，我们就可以评估聚类算法的性能。识别出一个“理想”的聚类结构在某种程度上是主观的，这取决于我们希望聚簇是什么样的。例如，看看默认文章的数据集，你该如何聚类这些文章？为了使讨论更具体，同时也减少文章的数量，可以只关注文件名以 biz-开头的那些文章，你希望把哪些文章聚在一起？为什么？参看本章第 5 个 To Do 事项，它是关于这个要点的。 302

现在，查看一下图 7.6 的结果，你可以通过执行代码清单 7.11 的脚本得到如图 7.6 所示的 swing 客户端及结果。聚类这些新闻的第一个大难题就是那个包含了 34 篇文章的大聚簇，它之所以是一个问题，是因为总共才有 129 篇新闻报道，而要

Download at http://www.pin5i.com/

分成差不多相等的 6 大类主题，这个大聚簇就占了整个数据集 26%的规模，而且它比单纯地用新闻总数除以主题数的这个平均值所能得到的比例都要大。这两个原因已经足以说明这两个聚簇是多么庞大了。这其实无须经过很严密的量化论证，只需要简单地通过 swing 客户端查阅这 34 篇文章就可以得到定性的认识。类似的情况，我们也可以认为拥有 14 篇文章的聚簇 17 也是一个大聚簇。查阅这些新闻报道，并对照本书 4.5.1 节中的内容，以恢复你对 ROCK 算法工作原理的记忆。

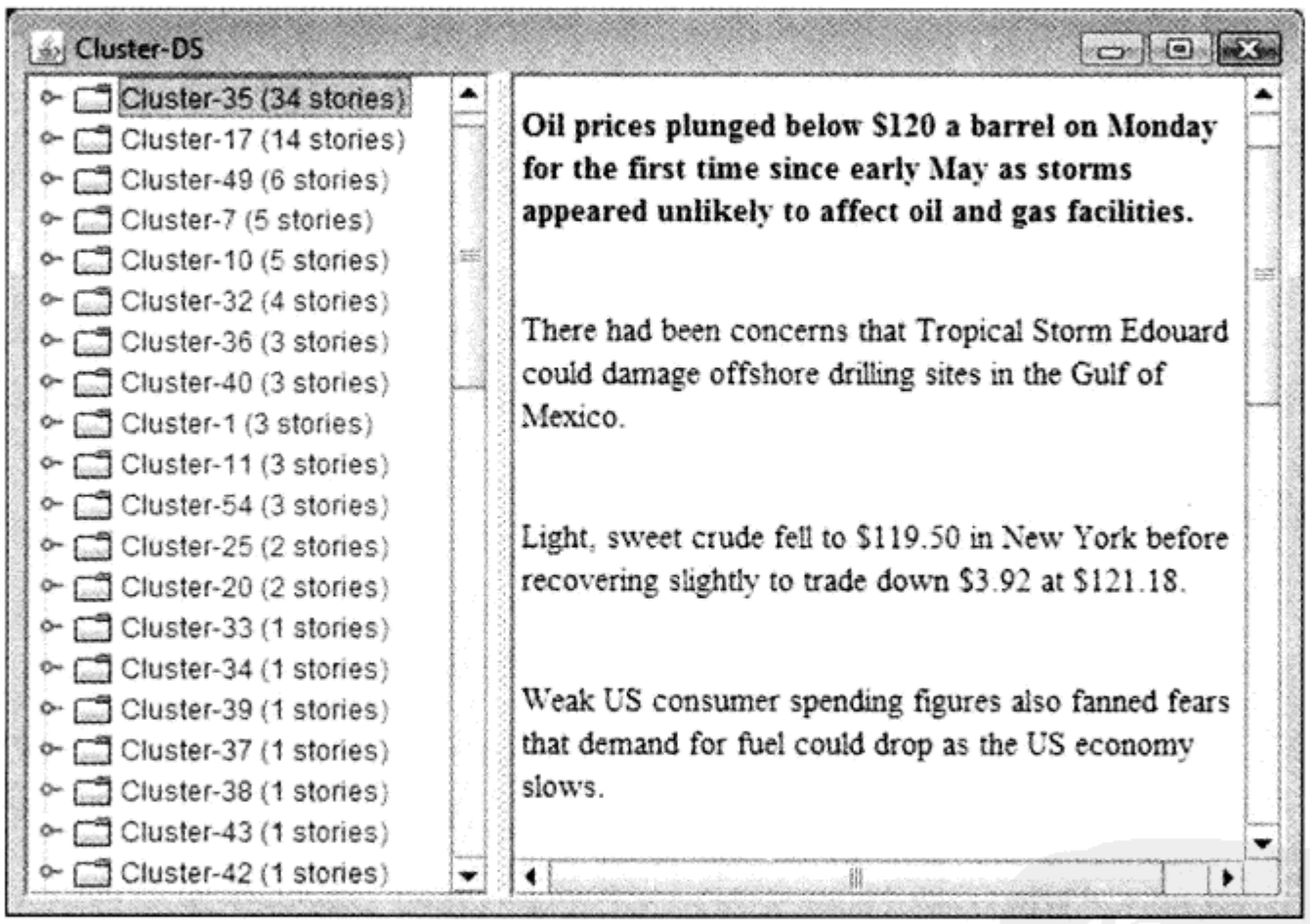


图 7.6　对分类前的默认数据集全部文章的聚类结果

代码清单 7.11　无新闻类别作为参考的情况下对新闻报道进行聚类

```
NewsDataset ds = new FileListNewsDataset("Cluster-DS");←载入默认的新闻数据集
ds.setDocumentDir("C:/iWeb2/data/ch07/all");
ds.init();
NewsProcessor newsProcessor = new NewsProcessor();
newsProcessor.createClusters(ds);←聚类新闻报道
NewsUI ui = new NewsUI(ds);
ui.showClustersOnly(true);←只显示聚簇
NewsUI.createAndShowUI(ui);
```

Download at http://www.pin5i.com/

看起来，聚类数、连接两个数据点（文章）的阈值这两个参数的默认取值促使了超级大类的产生，很自然，我们要做的第一件事情就是设置这两个参数，可以在NewsClusterBuilder类的cluster方法修改这两个参数的取值，如代码清单7.12所示。

代码清单7.12 NewsClusterBuilder类的cluster方法

```
public void cluster() {
  DataPoint[] dataPoints = createDataPoints(ds);
  int k = dataPoints.length / 3;←聚类数量
  double linkThreshold = 0.15;←可形成连接的阈值
  ROCKAlgorithm rock = new ROCKAlgorithm(dataPoints, k, linkThreshold);
  Dendrogram dnd = rock.cluster();
  List<NewsStoryGroup> storyGroups = createStoryGroups(dnd);
  for(NewsStoryGroup cluster : storyGroups) {
      ds.addStoryGroup(cluster);
  }
}
```

303

如你所见，该方法内部把所需的聚类数量设置为数据集规模的三分之一，即我们希望算法能聚出大约43个聚簇。这个选择并无特别之处，对不同的*k*值做试验，并观察它们对最大聚簇的影响。要实现这个目的，你可以把cluster()用cluster(int k, doublelinkThreshold)方法来重载，这个方法你同样可以在NewsClusterBuilder中找到。修改linkThreshold变量同样会影响到你的结果，所以你应该避免同时修改它们，而应该在试验某一个参数的同时保持其他的因素不变。

图7.6展示的聚簇分布的第二个问题是大量单例（只有一个元素的聚簇）的存在。我们知道单例的数量不应该这么大，因为这些新闻都是由我们生成的。那么你觉得ROCK算法的哪部分对单例数量影响最大呢？不要只关注于聚类数量与连接阈值这两个参数，如果你还没复习过4.5.1节，请现在就看一看。如果我们为ROCK算法写一个不一样的实现，你会修改什么？参看本章第6个To Do事项，以得到更多的提示和这个方向的分析。

为了更深入地分析这个结果，让我们把目光关注在文件名前缀为biz-的新闻报道上。图7.7显示了三个包含商业新闻的聚簇，图中显示有一个来自美国新闻类别的文章（文件名带usa-前缀的文章）被包含在第三个聚簇中，该聚簇同时还包含了

304

Download at http://www.pin5i.com/

两个商业类新闻。结果很让人吃惊，这样的聚类结果居然是正确的，你可以查看每一篇文章以验证这些文章其实是相关的。如果新闻报道先被分类到各个类别中，这样的聚簇是很难被发掘出来的，在那样的情况中，usa-01 这篇文章很有可能在分类阶段就已经跟 biz-01 和 biz-05 分离开了。

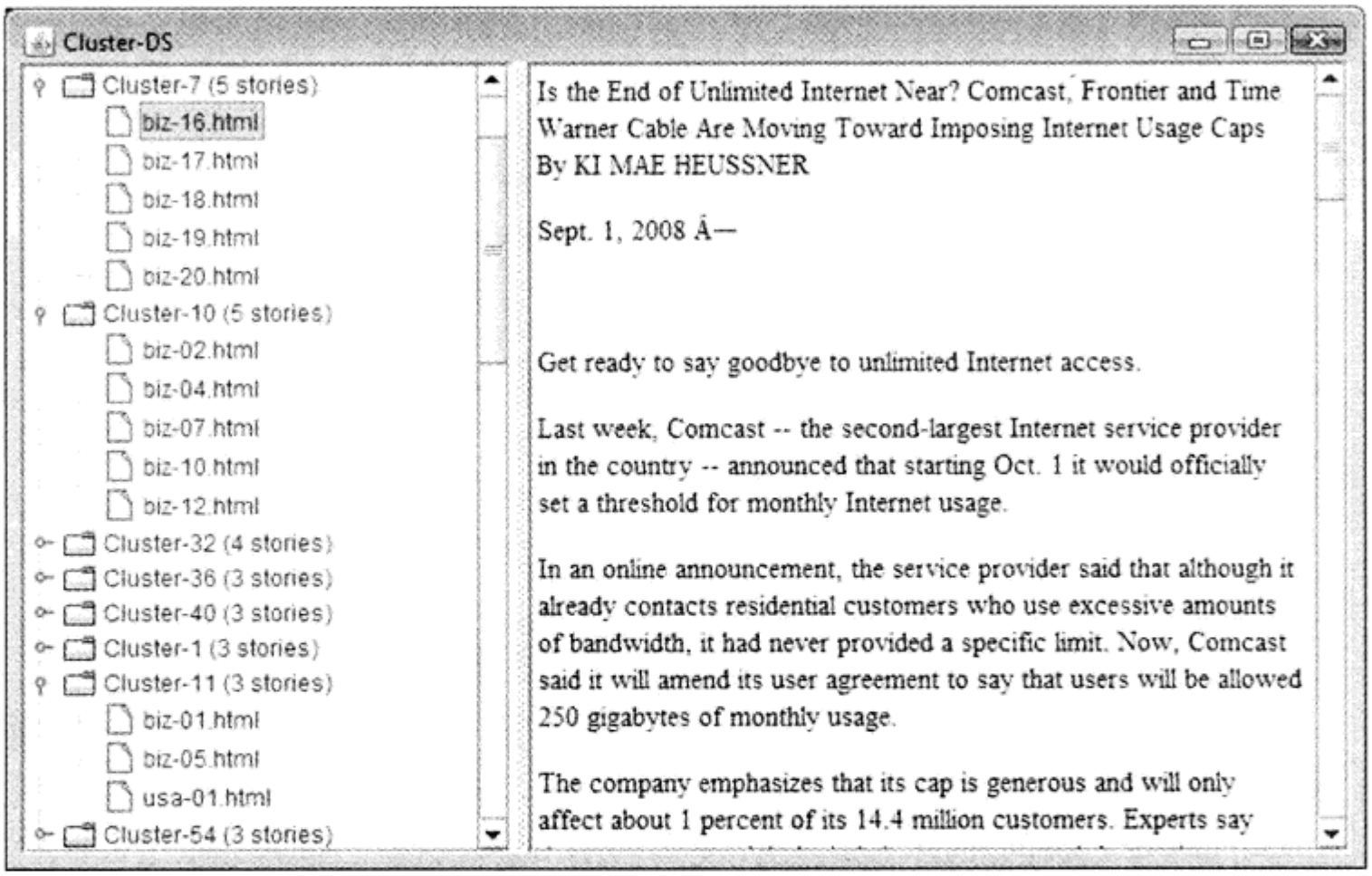


图 7.7　审视三个展开的商业新闻聚簇（左面板）

这个现象带来了一个因智能算法组合而出现的有意思的观点：没有必要认为两种聚类方法是互斥的，而有可能通过对两者的应用而收获它们各自的好处。

如果结果说明对新闻先分类、后聚类能得到质量更高的结果，并不意味着反序的结果一无是处。先聚类、后分类的价值可以获得交叉参考（cross-referencing）的好处，这里说的交叉参考是指我们可以在网站上放置一个独立的模块，来容纳来自其他类别相关的文章。这个功能跟推荐不一样，尽管它可以用类似的方式展现给终端用户，例如，“喜欢这篇文章的人也喜欢相关的文章”。交叉参考涉及为你的数据集建立一个横跨全集的潜在的语义网络。希望这能让你小窥到通过组合智能算法而呈现出的种种可能性。仅仅是通过对算法使用顺序的探讨，就能得到那么有趣的功能。

Download at http://www.pin5i.com/

对聚簇 7 和聚簇 10 的审视又带来了另一个有趣的要点。我们的聚类结构只有一层——聚簇之内再无聚簇。这一部分是因为构造上的需要，因为 Google 新闻门户也是通过这样的结构来展示的。然而，通过创建层次关系，在新闻组内再创建新的聚簇，以此来提升价值也是可行的。

让我们评价一下其中的价值。首先从实用性的角度，在聚簇 7 中，我们有 5 篇确实相关的文章，但这种相关的程度并不一样。biz-17 与 biz-20 是同一个事件（对美国有线电视用户在网络吞吐量上的限制）的不同报道。类似地，biz-18 与 biz-19 是时代华纳有线电视关于互联网使用实验的不同报道。聚簇 7 的最后一篇新闻 biz-16 比较一般，但仍然是与那两个事件都直接相关的。

不同的人对这些新闻报道有不同的看法，但把这些文章聚在一起是有意义的，还可以把 biz-17 与 biz-20、biz-18 与 biz-19 分别聚在一起，而把 biz-16 单列出来，因为它是一个一般性的描述。这样的结构应该对用户更友好并更吸引人。组成这种结构的一个可行的方法是持续地使用聚类算法，而不单应用一次。

实际上，其间你还可以使用不同的聚类算法，一种情况是你先使用 k-means 算法，然后对 k 个聚簇中的每一个使用 ROCK 算法。这种方法很常用，可以提升聚类结果的质量。在本章第 7 个 To Do 事项中，我们邀请你对当前的例子应用这个方法。 305

7.5.2 在一个新闻类别中聚类文章

createClustersWithinTopics 方法应用于默认的数据集得到的分组结果如图 7.8 所示。注意，现在有 64 个聚簇，具体地说，有 45 个单例，4 个含 2 篇文章的聚簇，6 个含 3 篇文章的聚簇，2 个含 4 篇文章的聚簇，3 个含 5 篇文章的聚簇，1 个含 6 篇文章的聚簇，1 个含 7 篇文章的聚簇，1 个含 8 篇文章的聚簇，以及 1 个含 14 篇文章的聚簇。

聚簇的名字现在指明了它所在的新闻类别。用这里的结果与图 7.6 相比，聚簇的分布看起来比之前的要好。超级大类已经消失了，取而代之的是含 2～4 篇文章的聚簇的增加。代码清单 7.13 展示了生成图 7.8 结果的 BeanShell 脚本。现在执行这些脚本，这样，当我们讨论这些结果时，你对聚类结构也有一个直接的参考。该代码清单中的步骤很直观，与代码清单 7.5 的步骤也很类似。

Download at http://www.pin5i.com/

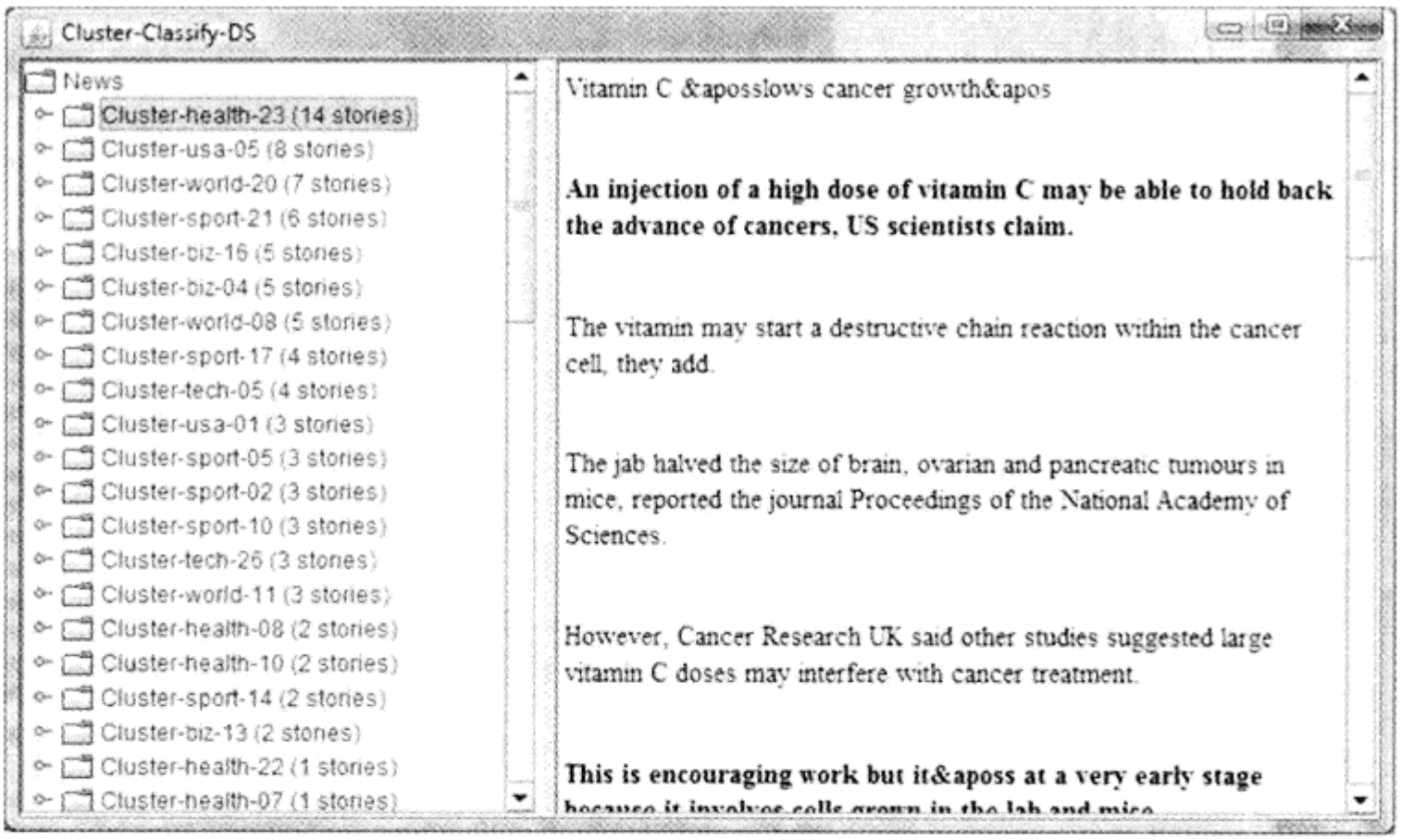


图 7.8　对默认数据集分类后在各类别进行聚类的结果

代码清单 7.13　对各个新闻类别的文章聚类

```
NewsDataset trainingDS = new FileListNewsDataset("TrainingDS");
trainingDS.setDocumentDir("C:/iWeb2/data/ch07/training");
trainingDS.init();
NewsDataset ds = new FileListNewsDataset("Classify-Cluster-DS");
ds.setDocumentDir("C:/iWeb2/data/ch07/all");
ds.init();
NewsProcessor newsProcessor = new NewsProcessor(trainingDS);
newsProcessor.trainClassifier();
newsProcessor.classifyStories(ds);
newsProcessor.createClustersWithinTopics(ds);
NewsUI ui = new NewsUI(ds);
ui.showClustersOnly(true);
NewsUI.createAndShowUI(ui);
```

306

如果你展开最大的聚簇 Cluster-health-23，你会发现它的内容与图 7.6 中的 Cluster-17 是一样的。不仅仅是文章数量一样，实际包含在这两个聚簇中的文章也一样，如图 7.9 所示。就这个聚簇来说，操作的顺序并没有什么影响。

Download at http://www.pin5i.com/

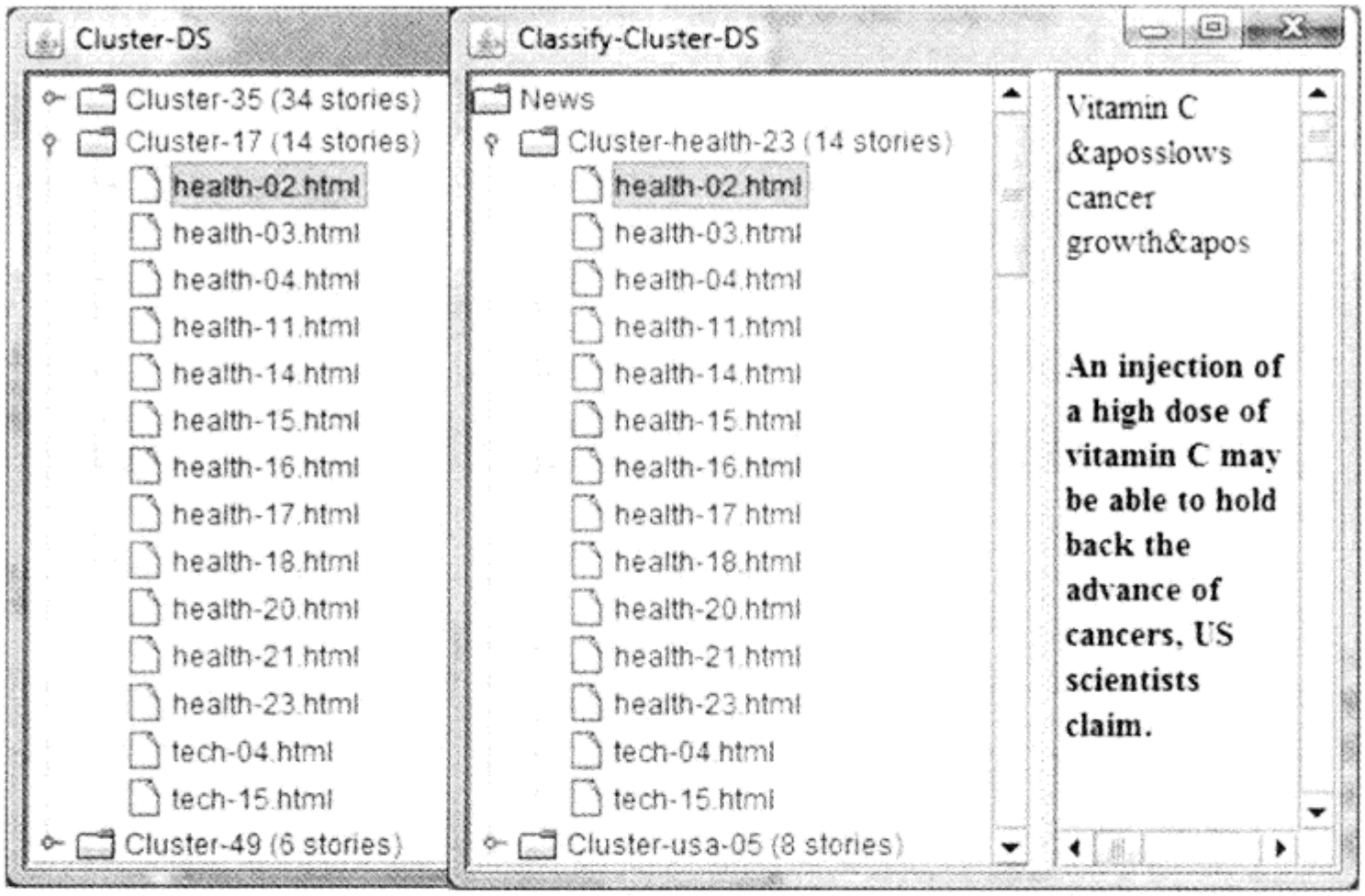


图 7.9 交换聚类与分类顺序后有一个大类别的结果没有发生改变。左边显示的是先做聚类的结果，中间显示的是先做分类的结果

这个聚簇的不变性很重要，它可以被解释为：文章 tech-04 和 tech-15 都不在分类器的训练集里[3]，所以分类器无法把它们分配到合适的新闻类别中。文章 health-16 被包含在训练集中，并且它的内容与这两篇技术文章有重叠，看起来这种重叠比训练集中技术类别的内部重叠更加显著。一旦这两篇技术文章被错分，我们的最佳选择就剩下：把它们各自作为单例或是把它们聚在一起——当然，是在错误的新闻类别中。 307

无论如何，这 14 篇文章的内容毕竟是相似的，无论是在聚类中还是在分类算法中。从实用性角度而言，在先分类、后聚类的顺序里，这种混叠情况可以通过扩充训练集来修补，当分类错分确实存在时，我们希望聚类可以在类别内进行细调。

对结果深入的考察还揭示了一点：在每个类别中的聚簇有很多都跟先聚类后分类得到的聚簇是一样的。这意味着这种顺序的最大好处是把超级大类给打散了，但在把这个发现推广到任何新闻数据集时需要相当谨慎。当训练集对各新闻类别有足

3 训练集与测试集并不一样。——译者注

Download at http://www.pin5i.com/

够的覆盖量时，可以期望这是正确的。

下面看看如何设置聚类数量与数据点（文章）连接阈值的默认值，如代码清单 7.14 所示，这个可以在 TopicalNewsClusterBuilder 类的 cluster 方法中实现。这些参数可以改变在最终结果里获得的单例数量，不管聚类与分类的顺序如何，这一点都是确定的。但是参数的改变对于先分类后聚类这种情况的影响要更加明显，因为输入聚类算法中的数据的噪声级已经被降低了。

代码清单 7.14 TopicalNewsClusterBuilder 的 cluster 方法

```
public void cluster() {
  for(NewsCategory topic : ds.getTopics() ) {
    List<NewsStory> stories = ds.getStories(topic);
    DataPoint[] dataPoints = createDataPoints(stories);
    int k = dataPoints.length / 3;←聚类数量
    double linkThreshold = 0.15;←可形成连接的阈值
    ROCKAlgorithm rock = new ROCKAlgorithm(dataPoints, k, linkThreshold);
    Dendrogram dnd = rock.cluster();
    List<NewsStoryGroup> storyGroups = createStoryGroups(topic, dnd);
    for(NewsStoryGroup cluster : storyGroups) {
      ds.addStoryGroup(cluster);
    }
  }
}
```

308

代码清单 7.14 与代码清单 7.12 类似，主要区别在于多了一层外循环来标识各个新闻类别。就这点差别也值得单独新建一个类来处理吗？你也看到了，这个实现避免了超级大类的出现，但更重要的是，它为后续更复杂的实现做好了铺垫：聚类数量与连接阈值可以定义为新闻类别的函数。你可以通过阅读本章第 8 个 To Do 事项来开启对这种方法的尝试。

现在，我们已经涵盖了两种重要的对新闻报道进行分组的方法——对单一的新闻聚类以及在新闻类别中对新闻聚类。我们已经完成了对本书中除推荐算法以外的所有智能算法的介绍。7.6 节的主题会是一个基于条目的推荐算法在我们这个新闻门户中的应用。具体地说，我们会考虑用户可以给新闻打分的情况，然后可以利用这些打分来针对每个用户动态地对门户的内容进行排序。

Download at http://www.pin5i.com/

7.6 基于用户评分的动态内容展示

如果你曾经对新闻打过分，我们就可以根据这些评分记录向你推荐你没看过的新闻。第 3 章我们介绍过两种应用广泛的用于生成推荐结果的技术——协同过滤与基于内容的方法。我们曾指出，在创建那个在线音乐商店的案例中，我们已经建立了一个可用于你自己应用的一般性推荐引擎系统的架构，现在是时候证实这一点了，让我们基于第 3 章的代码为新闻报道创建一个推荐引擎。如果你需要复习一下第 3 章，又或者没读过该章，现在是时候看看了。

首先，我们会生成一个包含了用户评分数据的文件。你也许还记得，一个推荐引擎的基本概念是条目、用户与评分。在本章，条目即是新闻报道。评分文件 ratings.txt 在 C:\iWeb2\data\ch07\ratings 目录中，它包含的数据如表 7.1 所示。

表 7.1 新闻门户中三个用户的 15 个评分记录

用户	新闻报道	评分
A	Usa-04	4
	Usa-05	3
	Biz-05	5
	Biz-01	4
	World-16	5
B	Usa-01	5
	Sport-04	3
	Sport-03	4
	Biz-05	5
	Tech-06	2
C	World-15	3
	World-16	5
	Usa-04	4
	Biz-01	4
	Usa-01	5

309

代码清单 7.1 显示了三个用户（A、B、C）给出的 15 个评分记录。在 7.5 节，我们曾一次又一次地看到那个包含了文章 biz-01、biz-05 与 usa-01 的聚簇。我们对

Download at http://www.pin5i.com/

评分进行了设计，以使 biz-01 为 A 和 C 共有， biz-05 为 A 和 B 所共有。从直观上看，我们期望 usa-01 会出现在对 A 的推荐列表中。下面执行代码清单 7.15 的脚本，看看会发生什么。

代码清单 7.15 基于评分推荐新闻报道

```
NewsDataset ds = new FileListNewsDataset("NewsDataset");
ds.setDocumentDir("C:/iWeb2/data/ch07/all");
ds.setTopTerms(25);
ds.setUserAndRatingsFilename("c:/iWeb2/data/ch07/ratings/ratings.txt");
ds.init();
StoryRecommender delphi = new StoryRecommender(ds);←创建推荐引擎
delphi.calculateRecommendations();
delphi.recommendStories("1");
```

结果显示在图 7.10 中，你会看到新闻 usa-01 确实出现在用户 A 的推荐列表中。对于这个小规模的一个新闻、用户、评分数据集，这也许不太具有说服力，但你可以把这些代码应用于更大的数据集，同样可以获得满意的结果。需要注意的是，推荐结果已经滤掉用户 A 打过分的文章。

```
bsh % delphi.recommendStories("1");
Skipping item:biz-01.html
Skipping item:biz-05.html
Skipping item:usa-05.html
Skipping item:usa-04.html
Skipping item:world-16.html

Recommendations for user UserA:
Item: usa-01.html                    , predicted rating: 5.000000
Item: sport-03.html                  , predicted rating: 4.000000
Item: world-15.html                  , predicted rating: 3.000000
Item: sport-04.html                  , predicted rating: 3.000000
Item: health-16.html                 , predicted rating: 2.500000
bsh %
```

图 7.10 基于表 7.1 的评分记录向用户 A 做出推荐

310

让我们深入到实现了这个推荐过程的 StoryRecommender 类，代码清单 7.16 显示了该类全部的代码。通过利用 Delphi 推荐引擎，仅仅只用了数行代码便实现了一个自定制的新闻报道推荐引擎。这里的 Delphi 推荐引擎就是在第 3 章曾用到的那个。

Download at http://www.pin5i.com/

代码清单 7.16 StoryRecommender：为用户生成新闻推荐

```
public class StoryRecommender {
  private DatasetAdapter rDs;
  private Recommender delphi;
  public StoryRecommender(NewsDataset ds) {
      this.rDs = new DatasetAdapter(ds);
  }
  public void calculateRecommendations() {
      Delphi d = new Delphi(rDs,
➥ RecommendationType.ITEM_PENALTY_BASED, false);←创建推荐引擎
      d.setVerbose(true);
      this.delphi = d;
  }
  public List<PredictedNewsStoryRating>
➥ recommendStories(String newsUserId) {
      if( delphi == null ) {
        String msg = "Recommender not initialized.";
        throw new RuntimeException(msg);
      }
      User user = rDs.getUserForNewsUserId(newsUserId);
      List<PredictedItemRating> predictedRatings = delphi.recommend(user);
      List<PredictedNewsStoryRating> ratings =
➥ new ArrayList<PredictedNewsStoryRating>();
      for(PredictedItemRating iR : predictedRatings) {
            PredictedNewsStoryRating r = new PredictedNewsStoryRating();
            r.setUserId(newsUserId);
            r.setRating(iR.getRating());
            NewsStory newsStory =
➥ rDs.getNewsStoryForItemId(iR.getItemId());
            r.setStoryId(newsStory.getId());
            ratings.add(r);
        }
        return ratings;
      }
}
```

除了通过 DatasetAdapter 类这个接口把新闻报道转化为第 3 章中一般目的的数据集外，这个类本身没有做什么事情。这里的 Delphi 是带惩罚（类型设置为 Recommendation

Download at http://www.pin5i.com/

Type.ITEM_PENALTY_BASED）的基于条目相似度的推荐引擎，你也可以使用其他可用的类型来进行实验。

StoryRecommender 类及其辅助类（如 DatasetAdapter、PredictedNewsStoryRating）可以作为你自己的应用模板。主要思想很简单：转换你的数据，以便它可以映射为条目、用户与评分，然后把推荐的事情交给 Delphi 推荐引擎来实现，你可试验不同的推荐类型，以找到最符合你的目标的那一个。你的数据结构也许不一样，或者是以不同的方式来存储的，但要把它们转换到一般的形式也不会是太困难的事情。

7.7 总结

本章以一个 Web 应用为背景，即通过一个如 Google 新闻那样的新闻门户的构建，展示了本书所介绍过的所有智能算法的使用。从一开始讲述的智能爬虫，我们就已经可以看到智能算法的价值。很自然，第 2 章的内容在新闻门户的搜索功能上有其广阔的用武之地，不管是基于索引的还是更丰富信息的搜索。

一开始就要发现智能算法在某个应用中的用途及其价值也许是很困难的，然而，当我们开始发问时，你就会发现这些算法在提升用户服务方面发挥着你意想不到的作用。我们看到，一个聚类算法不单可用于分组相似的新闻报道，还可以通过交叉参考来增强相关文章的可达性。

我们也看到了，可以多次使用同一个算法，也可以按一定的顺序使用不同的算法，自然而然地就出现了跟算法的先后顺序相关的问题。我们通过按不同的先后次序将聚类与分类用于新闻数据集来探讨相关的问题。先分类后聚类得到的结果要优于先聚类后分类的结果，这个观察结论是具有普遍性的，在你的数据集中也很有可能就是这样，不管它们是否是新闻数据集。

另一个有趣的现象也因为聚类与分类的使用次序而出现，那就是如何处理聚簇的分类问题。本章之前，我们只涉及如邮件、信用卡交易等简单对象的分类问题，本章我们不但描述了如何对单一的文章分类，还讨论了如何对一组文章分类。有很多种方式可以达到这个目的，所以元算法的构建在这种场合下就成了“救星”。

要回答关于如何使用智能算法才能适用于所有的应用场景这样的问题，实际上都是不可能的。然而，我们希望我们的叙述已经使你对智能算法的使用更近了一步。我们也希望，在读完本章之后，关于如何通过组合多个智能算法以创建满足给定需

Download at http://www.pin5i.com/

求的组件这一点，你已经有清晰的认识。回顾一下文中提到过的问题，然后探索一下我们在 To Do 事项中提到的诸多建议，你将来的工作必定能从中受益。 312

7.8 To Do

1．为 NewsCrawler 添加智能特性。我们的 NewsCrawler 比较适合于用小量的代码来启动整个实验，但如果你只想从一个 URL 列表里获取内容，则有很多其他的爬虫可以做得比我们的更好。编写这个小爬虫的目的在于指出智能算法在很多方面都可以使爬虫更高效与有效地工作。在这个 To Do 事项中，我们会讨论这些改善的可能性，并希望你有动力去实现它们。

考虑那些垃圾网站或有内容不符的网站——你可以按自己的喜好来定义它们，不合适或你不喜欢，都可以。它们属于你没有兴趣去抓取内容的网站，但是尽管你并不中意这部分站点，你的爬虫仍然有可能爬到其中某一个去。所以，如果爬虫可以识别出这些网站，进而避免抓取其内容，当然是最好的，你怎样实现这个功能？

在第 5 章关于邮件过滤的案例中，我们曾使用朴素贝叶斯分类器和基于规则的分类器来过滤垃圾邮件，这里也可以做同样的事情。爬虫可以对这些页面进行采样，并把它们分类为合法或不合法的抓取源。对于朴素贝叶斯分类器，你需要为各个类别提供一些采样页面，以作为训练集；对于基于规则的分类器，你需要一些规则，例如，如果页面包含 xyz，那么忽略它。

同样的算法可用于稍有不同的案例。如果你希望选择（而非过滤）某类特定的网页，例如，你要创建一个关注于 Java 相关文章的爬虫，或者更具体的 Java 开源项目的文章。这种情况下，你可以从一个已知的 URL 列表（或 Google 搜索结果的链接）开始进行遍历抓取，但只下载那些分类器判断结果为与主题相关的网站的页面。

你还可以在这些结果之上再加以更高端的智能。如果网站 A 连接到网站 B，而网站 B 被认为是不合适的，那么计算网站 A 到此类不合适的网站链接数量也是一种合理的方法。我们可以直接使用这些比例作为阈值，以创建一个连接图，可使用类似于 PageRank 的算法对其进行分析。这只是很少的几个设想，如果你实现了它们，而且你喜欢其结果以及其能力，可以尝试把类似的智能算法嵌入爬虫中，你可以获得学习经验上的提升，而且这对于你的工作也大有裨益。

2．利用用户点击以改善搜索结果。第 2 章我们曾见识过如何通过用户与搜索引擎的交互来定义他的兴趣与主观性，并引入朴素贝叶斯分类器以利用用户点击来改 313

Download at http://www.pin5i.com/

善搜索质量。你也许还记得，用户点击的数据记录在 data/ch02 目录的 userclicks.csv 文件中。这个逗号分隔文件包含三个域：用户 ID、搜索词与一个包含 URL 的字符串。

类似地，你可以创建一个文件来捕获用户的点击，以改善我们这个新闻门户的搜索质量。那么这次的训练属性是什么？可以试试用下面的列表作为训练属性：

- 用户 ID
- 搜索词
- 所选文档的单词
- 所选文档的主题

你能证明这些选择是合理的吗？为什么我们把 URL 置换成主题？为什么要引入文档的单词？校正排序分值时该怎么处理？把你的想法写下来，并实现它们。

对于同一个用户及同一个搜索词，用户点击文件中应该存储了不止一个记录。在第 2 章的例子里，一个点击在文件中出现的次数可以使该 URL 在搜索结果排序时更靠前，对于新闻门户中多个主题的案例，也应该遵循同样的规则。比较典型的是，同一个用户对于同一个搜索词会阅读多个不同主题的文章，因为他的兴趣会随着时间而变，又或者因为他倾向于在同一个领域中寻求更多额外的信息。

在新闻门户的案例中，一个有待解决的有趣的问题是：如果在同一个类别下有很多不同的文章，训练好的分类器也可以识别出属于这个主题的文章，那么它又如何帮助一个特定的用户发现他最感兴趣的文章？这能做到吗？如果不行，该如何去解决这个问题？

3. 评估属性选择对结果的影响。我们对一篇新闻报道属性的选择是基于从文章中提取出来的频繁词。从代码清单 7.7 的代码中可以很清楚地看到每一个频繁词都会成为分类器的训练属性。除此之外，显然还有更多的选择。

对代码清单 7.7 中 toInstance 方法的一个明显的修改是创建一个包含了所有频繁词的单一字符串，可以通过空格连接这些频繁词，使得它是易读的。你认为这种方法会导致更好还是更差的分类结果？不管你的观点是什么，尝试从概念上证明它，并实现这个修改，看看会发生什么。

注意 toInstance 方法中我们构建 StringAttribute 实例的方式，实际上，字符串属性被转换为布尔属性。所以另一种可改动的地方就是使用一个固定的频繁词集合，并把频繁词本身作为属性值。你可以通过如下步骤来实现：

Download at http://www.pin5i.com/

314

- 创建一个参数来确定训练属性的数量。
- 用一个简单的命名规则来为每个属性命名。例如，用一个确定的词并对它进行枚举——attr-1、attr-2 等。
- 根据每篇文档中频繁词的顺序依次给各属性赋值。即是说，属性 attr-1 取值为第一频繁词，属性 attr-2 取值为第二频繁词，等等。

通过这样的构造方式，重运行我们的样例程序并观察其结果。你认为会得到什么输出？你认为这样的方法能提供更好的结果吗？如果回答是肯定的，理由是什么？

如果这种方式会产生问题，你能预料到是什么吗？假设你要使用 32 个属性，但因为某些原因，某篇文章并没有 32 个频繁词，这就意味着后面的属性（按 1 到 32 的顺序）无法得到对应的词作为属性值，那么这些属性该如何取值？这些取值对结果有影响吗？

4. 创建 ClassificationStrategyImpl 类的高级实现。代码清单 7.8 中曾描述过新闻处理类 ClassificationStrategy 的基本实现，在该实现中，assignTopicToStory 方法是很简陋的，所以，看看是否还可以有别的实现，以得到有益的结果。

考虑使用两个分类器以得到分类结果的情况，跟第 6 章组合分类器的方法不一样，这里我们希望依次使用这两个分类器。你也许会觉得奇怪，但考虑如下场景：我们基于每个网页的内容训练了朴素贝叶斯分类器，但它并未包含任何上下文知识或内容的元信息。

如果你的网页是从 ESPN 抓回来的，那么它跟体育类的关系肯定是相当高的。我们可以利用网页所拥有的这些元信息来创建规则，并在朴素贝叶斯分类器完成分类工作后执行这些规则。换句话说，我们先让概率分类器对新闻报道进行第一次分类，然后对结果再执行一系列假定是正确的规则，这些规则都是内容无关的。试着在 assignTopicToStory 方法中实现这样的分类策略，并把它与原来的实现得到的结果进行比较。

这种方法在分类精度上应该能得到显著性的提升。当然，你还可以设计其他更好的策略，我们鼓励你做更多的尝试并比较它们的结果。需要注意的是，当你面临的是聚簇的分类时，分类器的依次使用会变得没那么直接，因为一个给定的新闻聚簇内的文章基本上是来自不同源的。对于一个新闻聚簇，这种方法会变得复杂。所以，让我们把注意力放在 ClassificationStrategyImpl 类的第二个重要的方法上。

Download at http://www.pin5i.com/

315

assignTopicToCluster 的特殊实现是相当简单的，它从一个聚簇中挑选出一篇文章作为该聚簇的代表，然后记录下该聚簇与代表文章的类别。代表文章的挑选仅仅取决于 selectLongestStory 方法，它以简单的启发式的方法从一个新闻列表中挑选出一篇代表文章来。这个选择的主要思想在于认为：文章越长，它与其他文章重叠就越多。

这个实现并不很精妙，我们也可以显著地改善它。第 4 章描述 k-means 算法时，还介绍了重心的概念。可以依此实现一个 selectRepresentativeStory 方法，把最接近于重心的文章挑选为代表文章。该怎么实现它？需要考虑到你现在比较的是文档，可以尝试使用不同的距离度量。

最后，实现一个使用多个代表文章并通过多数票决决定聚簇归属的 assignTopicToCluster 方法。应该考虑聚簇内所有的成员吗？如果不需要，该如何决定谁才有投票权？

5. 存在一个最佳的聚簇分布吗？标识一个理想的聚类结果在某种程度上是主观的，必然取决于我们对聚类结果的期望。对于我们的默认新闻数据集，你会怎样对它们进行聚类？为了使问题更具体，可以缩小涉及文章的范围，比如所有文件名以 biz-为前缀的新闻，你会把哪些文章聚在一起？为什么？

为了系统地回答这样的问题，你必须为一个聚类结果“好”的程度定义一个度量。自然地就把你带到如下的问题：聚类应该通过什么来度量？一个聚簇要跟什么比较优劣？对于分类问题，有一套结构（一般来说是树形的）与训练集构成的参考框架，通过它可以度量分类的质量。对于聚类问题，我们必须设计其他意义上的聚类质量度量，因为并不存在一个参考框架。所以这种度量必须依赖于内部信息，应该采用聚类结构的什么属性用于计算度量值？

6. ROCK 使用技巧。图 7.6 中的聚类分布结构的第二个问题在于大量单例（只有一个单元的聚簇）的存在。我们知道不可能存在那么多的单例，因为数据是我们通过手工生成的。

你认为 ROCK 算法中哪一部分会对单例数量有最大的影响？聚类数量的设定对这个问题的影响有多大？连接阈值呢？

如果我们另编写一个 ROCKAlgorithm 的实现又会怎样？这个算法还有几个隐含的参数。也许，用于构造相似度矩阵的相似性度量的选择是最基本的因素。还有什么别的合适的相似性度量？是否应该为我们的新闻门户创建一个完全自定制的方

316

Download at http://www.pin5i.com/

法？比如，考虑使用朴素贝叶斯分类器来设定一个连接概率，而非相似性度量。

7. 新闻报道的层次聚类。把信息表示成树形结构是信息组织的一种更有效的方式。树形结构的使用很常见，你计算机中的文件夹与文件就是以层次（树形）结构展示的。在线商店的分类结构同样是组织成树状的。相对于平坦的列表或简单的一层结构，这种结构方式更加引人注目，对用户也更友好。

可以通过多次使用聚类算法来得到层次的聚类结构。实际中，还可以多次使用不同的聚类算法，例如，可以考虑对新闻报道先使用 k-means 算法，然后对 k-mean 生成的每个聚簇使用 ROCK 算法。交换两种算法的顺序，或使用同一种算法是两种可行的变体。

你会看到每种情况各有什么优势？该如何实现这样的方式？不仅从聚类质量的角度来考虑这个问题，还要从性能（计算成本）角度进行考虑。

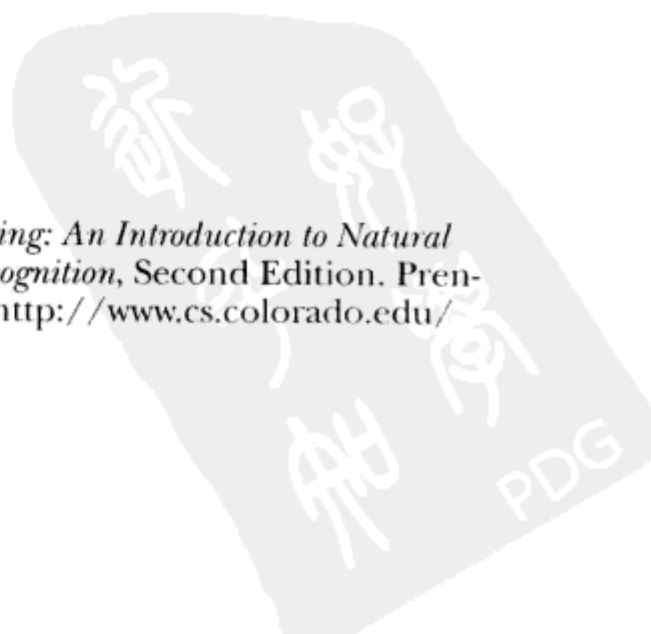
8. 为每个新闻类别细调聚类结果。代码清单 7.14 中描述了 TopicalNewsCluster Builder 的 cluster 方法。通过仔细观察，你可以确信代码里没有特别关键的东西，而只是对于每个新闻类别中的文章进行聚类。如果代码不变，类的名称无疑是一个误导，所以，还是让我们研究一些可能性。

如果用一个整型数组（如 int k[]）表示聚类数量，并用一个双精度数组（double linkThreshold[]）表示连接阈值会怎样？现在你可以对不同的新闻类别使用不同的参数对了，但这些取值应该是什么？我们该如何才能为每个参数选取最合适的值？

如果每个新闻类别又包含多个主题，又可以做些什么修改以改善每个类别中聚类的质量？头脑风暴一下实验你的想法，答案不止一个！

7.9 参考资料

Jurafsky, D., and J. H. Martin (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, Second Edition. Prentice Hall (Series in Artificial Intelligence), 2008, pp 1024. http://www.cs.colorado.edu/~martin/slp.html.



Download at http://www.pin5i.com/

附录 A
BeanShell 简介

为了避免陷入无谓的细节，把精力集中在对算法高层次的理解上，本书使用了 BeanShell。本书附带的 BeanShell 脚本从整体上展示了各种算法。此外，也可以用这些代码做实验，实现交互式学习。在使用 BeanShell 脚本的过程中，我们收获颇多，而且发现其中乐趣无穷。我们还尝试了很多在本书编写过程中诞生的灵感，所以相信你在熟悉 BeanShell 之后一定会发现它的价值。那么这个 BeanShell 到底是什么？

A.1 什么是 BeanShell?

BeanShell 是一门兼容 Java 的轻量级脚本语言。实际上，BeanShell 动态地执行标准 Java 语法，并为其添加了常见的脚本语言特性，例如弱类型、命令、与 Perl 和 JavaScript 中一样的方法闭包等。它由 Pat Niemeyer 编写，是一个开源项目，可以在其网站上找到它的实现（http://www.beanshell.org）。

有人正在致力于 BeanShell 的标准化，并希望能将其纳入将来的 JDK 中。正如 Java Specification Request 274 中说的：

> BeanShell 是一门运行在虚拟机上的语言，能动态地执行所有的 Java 语法和语义，同时可以无障碍地访问 Java 对象和 API。附加的脚本编程和便捷功能使得该语言成为 Java 语言语法的一个严格超集。BeanShell 还试图缩小 Java 代码和脚本语言在语法和运行时上的差异，降低开发难度，简化脚本和静态 Java 间的迁移。

Download at http://www.pin5i.com/

你现在可能会觉得奇怪：这不就是 Java 么？为什么要用 BeanShell？下面来回答这个问题。

A.2　为什么使用 BeanShell？

318

为了本书的写作目的，BeanShell 提供了一个学习和实验智能算法的环境。从学习的角度，你可以从我们已经定义好的 API 和已经写好的脚本开始，随着经验的增长，你可以选择更加深入地了解这些 BeanShell 脚本中的算法。

从研究和开发的角度，你可以用 BeanShell 快速地检验算法调整的效果。此外，还可以在解释器的命令中与算法交互，检验单个输入的反馈。例如，你有一个训练好的分类器，想用它像先知一样判断某个实例的分类。如果事先不知道某个实例的正确分类，就可以交互式地询问分类器。我们希望这些理由能让你有足够的兴趣。接下来让我们看看如何快速地设置和运行 BeanShell。

A.3　运行 BeanShell

只要有 Java 运行环境，并将 BeanShell 的 JAR 文件（bsh.jar）放入 classpath，就能运行 BeanShell 和执行本书中的脚本。本书附带的源代码中提供了 2.0b4 版的 BeanShell，我们还提供了能在 Windows 和*nix 的控制台中加载 BeanShell 解释器的脚本文件。在附带的源代码的 deploy\bin 目录中可以找到这些文件。

在接下来的内容中，我们假设你的工作环境是微软的 Windows 操作系统，在其他有 Java 运行环境的操作系统中一般也可以用类似的方法调整脚本。要说明的是，在这里，假设本书源代码放在 C:\iWeb2 目录中，可以用环境变量%IWEB2_ HOME%引用这个位置，所有的相对路径都是相对这个位置而言的。如果要把源代码放在系统的其他地方，需要对脚本做相应的修改。打开命令行提示符：打开“开始”菜单，单击“运行”，然后在文本框中输入 cmd。切换到 c:\iweb2\deploy\bin 目录，在这里应该能看到名为 bsc.bat 的 Windows 脚本文件。

除了源代码的位置，运行这些脚本之前还要设置%JAVA_HOME%环境变量，可以在命令行提示符环境中查看。如果没有设置该环境变量，至少也要把所选择的 Java 可执行文件添加到 PATH 环境变量中。注意，我们没有在 Java 1.5 或更老的版本上

Download at http://www.pin5i.com/

测试我们的代码，建议你通过 Sun 网站安装最新的 JDK（JDK 6）。

最后，因为执行（严格地说，是“解释”）的是 Java 程序，所以可以在命令行中根据自己的需求和系统配置调整 JVM 的选项。我们建议的 JVM 堆栈设置是 -Xms256M -Xmx1280M，对有些系统而言，这两个值可能太大了。如果遇到问题，要注意根据实际情况调整参数。

A.4 参考资料

The BeanShell open source project. http://www.beanshell.org/home.html.

JSR 274: The BeanShell Scripting Language. Java Specification Requests. http://jcp.org/en/jsr/detail?id=274.

Download at http://www.pin5i.com/

附录 B
网 络 采 集

本附录会介绍网络爬虫组件的概况，简要描述本书中所使用的爬虫的实现细节以及一些由 Java 编写的开源爬虫。

B.1 爬虫组件概况

网络爬虫的作用是探索网络，并下载和存储从中发现的数据。我们在第 2 章中已经见过，网络爬虫只是诸如搜索引擎这样的大型应用的一部分。

典型的网络爬虫包括以下组件：

- 跟踪爬虫所有已知 URL 的存储模块。
- 根据给定的 URL 从网络中获取文件的文件下载模块。
- 负责从 HTML、PDF、微软 Word 等各种格式的文件中提取原始内容的文件解析模块。这个解析器也负责提取文件中的 URL，以及其他在索引中有用的数据，尤其是元数据信息。
- 存放已经采集到的文件的元数据和在爬取过程中从原始文件提取出的内容的存储模块。
- 将 URL 转换到标准形式，以便比较和计算 URL 的规则化模块。
- 避免无效 URL 的 URL 过滤模块。

各个组件的具体设计和实现取决于爬取的内容和爬虫需要处理的数据规模。最

Download at http://www.pin5i.com/

320

简单的例子是从某个已知的网站中采集若干个网页，实现这样的爬虫的源代码的长度不会超过一页纸。对于内网网站，也只需要非常简单的实现就可以满足需求。但如果要处理来自互联网的海量文件，爬虫就会被设计成一组分布在网络中的各个节点上的程序，甚至这些节点的地理放置也是比较靠近数据源的。本节结尾的参考资料介绍了如何实现大规模的采集。

B.1.1 采集的步骤

典型爬虫的运行涉及以下两个步骤：

1．根据给定的 URL 清单（也称为种子 URL）初始化爬虫的 URL 库，然后开始 Web 采集。

2．爬虫加载未访问的种子 URL，判断工作量的大小。

对每一个有效的 URL，Web 爬虫需要做以下操作：

1．获取 URL 的内容。

2．解析所获取的文件，抽取其中的外链和文件内容。

3．保存我们所获得的信息。

4．统一新发现的 URL 的格式。

5．过滤掉应该忽略的 URL。

6．用要爬取的新的 URL 清单更新 URL 库。

7．重复第 2 步，直到获得符合要求的探索深度。

爬虫可以分为两大类：通用型和专用型。通用爬虫收集所有能获得的文件，依赖于 URL 过滤技术来限制爬行的 URL。专用爬虫用于探索与某个话题有关的内容。本书所介绍的技术适用于改进所有的爬虫。

B.1.2 我们的简单爬虫

我们的这个简单爬虫可以从网络下载网页，也可以从本地文件系统获取网页。除了 HTML 文件，还可以处理微软的 Word 文件。前面也说过，这只是一个演示用的爬虫，目的是为了试验本书中介绍的智能技术。代码清晰简洁，你可以清楚地看到处理过程的每一步具体发生了什么。每个文件都放在单独的文件中，而且使用的是纯文本格式，所以你可以方便地查看其中的内容，可以在 iweb2.ch2.webcrawler 包中找到与这个爬虫有关的所有的源代码。

Download at http://www.pin5i.com/

在之前的代码清单中我们已经演示了 Web 爬虫的工作方式。爬虫每次运行时都会新建一个名为“crawl-<时间戳>”的目录，用来存放所有的相关数据。在该目录中会创建以下子目录或文件：

- <crawl-dir>/knownurls/knownurlsdb.dat——这个文件存放爬虫已知的 URL 清单。 321
- <crawl-dir>/fetched/——该目录中存放爬虫下载的原始文件。文件是批量下载和处理的，默认的批量处理数量是 10。
- <crawl-dir>/fetched/<batch-number>/<doc-id>.fetched——这里的每个文件中保存的是相应的被搜索文件的原始内容。
- <crawl-dir>/fetched/<batch-number>/<doc-id>.meta——这里的每个文件中保存的是相应的被搜索文件的元数据。不同的传输协议有不同类型的元数据信息。
- <crawl-dir>/processed/——该目录保存着处理后的文件。每一批处理后的 URL 都有一个单独的子目录。
- <crawl-dir>/processed/<batch-number>/content/<doc-id>.content——这些文件中保存着经过解析后的内容。如果更换解析器，这些文件的内容也会发生变化。
- <crawl-dir>/processed/<batch-number>/outlinks/<doc-id>.outlinks——每个文件中保存着文档解析器在相应的母文件中找到的所有链接。
- <crawl-dir>/processed/<batch-number>/properties/<doc-id>.properties——这些文件保存的是被搜索文件的属性，例如文件类型、标题、URL 等。
- <crawl-dir>/processed/<batch-number>/txt/<doc-id>.txt——每个文件只含有解析器从采集到的文件中提取出来的文本内容。
- <crawl-dir>/pagelinks/pagelinkdb.dat——这个文件保存了所有已处理链接的外链。只有经过 URL 过滤器处理的外链 URL 才会存放在这个文件中。

B.1.3　开源 Web 爬虫

有两个主要的基于 Java 的项目——Nutch 和 Heritrix，它们提供了 Web 爬虫的实现。Nutch 是一个 Apache Lucene 的子项目，网址是：http://lucene.apache.org/nutch/。在该项目的网站上有教程会一步步地教您如何安装和运行这个程序，其中当然包括 Web 爬虫。跟我们的爬虫不一样，Nutch 的每个数据文件会保存多个网页，这样就

Download at http://www.pin5i.com/

很难查看小规模爬行的结果，但对于大规模的爬行能有更好的 I/O 性能。

Heritrix 是 Internet Archive 的开源 Web 爬虫，它已用于归档互联网上的大部分网页，该项目的网站是http://crawler.archive.org/。Heritrix项目专注于大规模采集，遵循自由的 LGPL 授权。

另一个可能会让你感兴趣的开源库是 Apache Tika 项目，网址是 http://lucene.apache.org/tika/。Tika 是一组利用现有的解析器库从各类文件中探测和提取元数据及结构化文本内容的工具集。

322

在 Heydon 和 Najork、Gomes 和 Silva 等人的论文中可以找到各方面更详细的信息。Boswell、Shkapenyuk 和 Suel 等人在论文中介绍了设计高性能可扩展的 Web 爬虫时要注意的问题。而关于采集的更多细节，可以参考 Chakrabarti 等人的论文。

B.2 参考资料

Boswell, D. "Distributed High-performance Web Crawlers: A Survey of the State of the Art." 2003. http://www.cs.ucsd.edu/~dboswell/PastWork/WebCrawlingSurvey.pdf.

Gomes, D. and M. Silva. "The Viuva Negra crawler: An experience report." *Software—Practice & Experience*, Volume 38 (2), pp. 161-188, 2006.

Heydon, A. and M. Najork. "Mercator: A Scalable, Extensible Web Crawler." Compaq Systems Research Center, 1999.

Chakrabarti, S., M. Berg, and B. Dom. "Focused crawling: a new approach to topic-specific Web resource discovery." WWW8 International World Wide Web Conference, vol. 31, pp. 1623-1640. Toronto, 1999. http://www8.org/w8-papers/5a-search-query/crawling/index.html.

Shkapenyuk, V. and T. Suel. "Design and Implementation of a High-Performance Distributed Web Crawler." Polytechnic University: Brooklyn, NY, 2001. http://cis.poly.edu/suel/papers/crawl.pdf.

Download at http://www.pin5i.com/

附录 C
数学知识回顾

纵观全书，我们对读者的数学知识要求很低。附录 C 会介绍一些我们在算法中用到但没有明确给出标准形式的数学公式。William H. Pres 等人撰写的经典书籍 *Numerical Recipes: The Art of Scientific Computing* 中不仅介绍了数学知识，还讲解了有关的数值算法。智能算法开发过程中的很多问题都能在本书中找到参考资料，因此，这是一本必备书籍。

C.1 向量和矩阵

在初等数学和物理中，向量是一个箭头的数学表示。最常见且最直观的例子就是连接平面上 A、B 两点的箭头。在我们使用的公式中，通常用一维数组表示向量。换句话说，就是一个有序的数字集合。我们通常用加粗的拉丁字母表示向量，例如 $\boldsymbol{x}$。一个 10 维向量可以表示成数字 x_i 的有序集合，其中 i 表示数字 1 到 10（含 10），如果从 0 开始计数，则是 0 到 9（含 9）。

二维数组则可以视为一个有行和列的表格。矩阵中的每个元素都对应表格中的一个单元格。在我们使用的公式中，通常用二维数组表示矩阵。我们通常用加粗的斜体大写拉丁字母表述矩阵，例如 $\boldsymbol{A}$。跟向量类似，矩阵可以表示成其各个元素的有序集合。换句话说，矩阵 $\boldsymbol{A}$ 也可以写成 $\boldsymbol{A}_{ij}$，两个下标可以有不同的范围。交换这两个下标的位置，称为矩阵的转置（transpose）。即矩阵 $\boldsymbol{A}_{ij}$ 的转置矩阵就是 $\boldsymbol{A}_{ji}$，反

Download at http://www.pin5i.com/

之也是成立的。

324

矩阵不一定都是二维的，还可以是 n 维的，其中 n 可以是任意的自然数。当然，n 维矩阵就有 n 个下标。可以把向量视为是一维矩阵的特殊情况。矩阵有很多有趣的性质，而且可以符号化地执行很多操作，就像我们用 x、y 在基础代数中表示未知量一样。在符号操作中，可以将矩阵视为数字，但跟普通的数字不一样，两个矩阵间的乘法交换律可能是不成立的。所以一般情况下，对于两个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，有可能 $\boldsymbol{AB} \neq \boldsymbol{BA}$。

总之，从实用的角度看，可以将向量视为一维 Java 数组，而将矩阵视为二维或多维 Java 数组。有大量关于向量和矩阵的文献，也有大量的教科书以不同的方式深入介绍了有关知识。在本附录的参考资料中也引用了一些有关的经典教材。

C.2 距离的度量

在第 3 章中，我们看到有很多测量 A、B 两点间距离的方法。现在让我们看看如何将这些不同的度量方法用向量的方式表示成公式。如果用向量表示点（向量 $\boldsymbol{a}$ 表示 A 点，向量 $\boldsymbol{b}$ 表示 B 点），那么两个向量的差也是一个向量，假设是 $\boldsymbol{x}$，$\boldsymbol{x}$ 的大小就是一种度量两点间距离的方法。

最常见的距离度量方法是欧氏距离（Euclidean distance，也称欧几里德距离），也可以称为 L^2 距离或者 L^2 范数（L^2 norm）。从其名称就能看出，欧氏距离是一种通用的距离公式 L^p 范数的一种特殊情况，其中 p 可以是任意大于或等于 1 的实数。L^p 范数的公式如下：

$$\|x\|_p = \left(\sum_{i=1}^{n} \|x_i\|^p \right)^{\frac{1}{p}}$$

将 p 设为 2，就得到了欧氏距离。将 p 设为 1，就得到了第 3 章 To Do 的第一项内容中所讨论的“城市街区”距离或“计程车”距离。当然，在上面的公式中，向量 $\boldsymbol{x}$ 就是我们要测距的两个向量的差，即 $x=b-a$。

另一个广泛使用的距离公式是余弦相似度（cosine similarity）。为了解释余弦相似度的公式，需要先解释一下向量的内积（也经常被称为点积，dot product）。两个向量的内积表示一个向量在另一个向量上的投影。图 C.1 中是同一个平面上的两个

Download at http://www.pin5i.com/

二维向量。当两个向量相互垂直时，一个向量在另一个向量上的投影最短（等于 0）。当两者平行时，投影最大。

我们用一个免费的小程序画出了这个图。读者可以在网上尝试运行这个程序，及其他有关程序，网址是 http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/linear_algebra.html。两个 n 维向量 a、b 的内积是：

325

$$\mathrm{a}\cdot\mathrm{b}=\sum_{i=1}^{n}a_ib_i=a_1b_1+a_2b_2+\cdots+a_nb_n$$

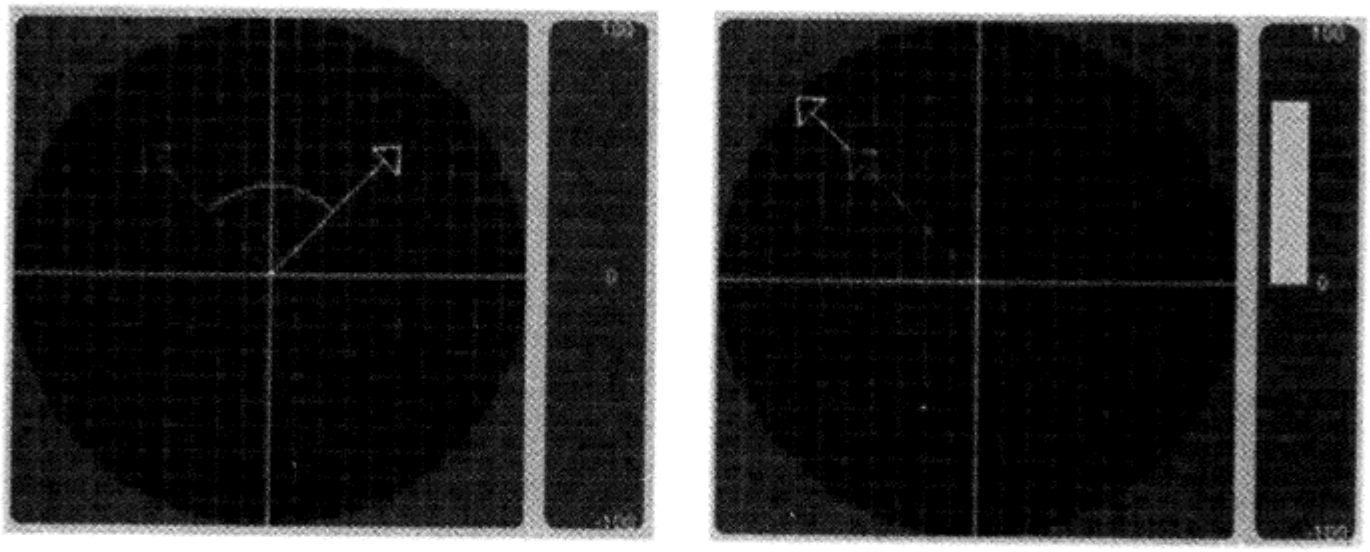


图 C.1　两个向量的内积：垂直（正交）的情况（左图）和平行的情况（右图）

在内积的基础上，我们可以通过如下公式定义余弦相似度：

$$\cos(\theta)=\frac{\mathrm{a}\cdot\mathrm{b}}{(\mathrm{a}\cdot\mathrm{a})(\mathrm{b}\cdot\mathrm{b})}$$

其中，θ 是向量 a 和 b 之间的夹角。这个公式的代码实现在 iweb2.ch3.collaborative.similarity.CosineSimilarityMeasure 类中。有些人也直接用角度本身作为两向量间的相似度，此时只需要在上述公式右侧加上反余弦函数（Math.acos）即可。如果要强调是相似度，后者则更加方便。如果所涉及的数值必须要满足像 L^p 范式那样的数学范式，距离表示方法的选取就变得尤其重要了。

如果要构建自己的范式，根据自己的需求计算距离，就要阅读泛函分析方面的书籍。我们推荐 A.N. Kolmogorov 和 S.V. Fomin 撰写的 *Elements of the Theory of Functions and Functional Analysis*。从数学的角度看，距离的度量涉及度量空间，还需要一些集合论的知识才能理解序列的覆盖度、有限点、开集、闭集等概念。关于泛函分析的书籍数不胜数，我们推荐 Kolmogorov 的书是因为它简明、完整，而且价

Download at http://www.pin5i.com/

格比较实惠。

326

C.3 高级矩阵方法

如果已经把所遇到的问题“翻译”成了矩阵语言，你就能使用大量已有的方法和技术来解决这个问题。这样就能非常仔细地研究系统的各种性质，这种精确性只有数学才能提供。当然，实际的情况可能并没有听上去这么简单。可能只有部分方法才适用于您的具体情况，此外，在脑海里把矩阵跟问题的特性联系在一起更像是艺术，而不是科学。尽管如此，如果你对数学有兴趣且有科学冒险精神，我们还是强烈建议您在数据挖掘中使用高级的矩阵方法。

Lars Eldén 最近出版的一本书籍中就涉及了智能算法，书名叫 *Matrix Methods in Data Mining and Pattern Recognition*。这本书介绍了向量和矩阵的基础知识、线性方程组和最小二乘法、向量和矩阵的正交、各种矩阵分解（QR 和 SVD，即上三角分解和奇异值分解）以及非负矩阵分解，其中最能引起读者兴趣的应该是这本书的 9.1 节，它介绍了 k-means 算法。此书其他部分用矩阵代数工具解决了手写数字分类、文本挖掘、PageRank 算法（本书第 2 章也有介绍）、关键字和关键句的自动提取、基于张量 SVD 的面部识别等问题。此书中还提供了大量关于科学计算的参考文献。所以，对想深入学习本书介绍的算法的读者，我们强烈推荐此书。

C.4 参考资料

Eldén, L. *Matrix Methods in Data Mining and Pattern Recognition.* (Series: Fundamentals of Algorithms). SIAM: Society for Industrial and Applied Mathematics, 2007.

Kolmogorov, A.N., and S.V. Fomin. *Elements of the Theory of Functions and Functional Analysis.* Dover Publications, Inc. New York, 1961.

Lay, D.C. *Linear Algebra and Its Applications (Third Edition).* Addison Wesley, 2005.

Press, W.H., S. A. Teukolsky, W.T. Vetterling, and B.P. Flannery (2007). *Numerical Recipes: The Art of Scientific Computing (Third Edition).* Cambridge University Press, 2007.

Trefethen, L.N., and D. Bau. *Numerical Linear Algebra.* SIAM: Society for Industrial and Applied Mathematics, 1997.

Download at http://www.pin5i.com/

附录 D
自然语言处理

在本书中多次用到自然语言处理（Natural Language Processing，NLP）。NLP 指的是用于处理书面或口头（通常是人类）语言的技术和方法。从实践的角度看，它有助于我们处理文本和音频记录，实现对其内容的分析。可以想象得到，这个领域很有意思，也很庞大。

人工智能领域很早就开始致力于自然语言处理的研究。实际上，著名的图灵测试就是考验计算机在电话线上与人类交流的能力的，在没有人参与的情况下判断电话线的另一端是否是人类，Saygin 等人的论文是一篇很好的关于图灵测试的综述。在这个历史悠久的领域，可以找到很多从不同角度看待这个问题的分支。因此，像计算语言学和语音合成等研究领域所研究的问题实际上与自然语言处理所研究的问题是一样的（或者是非常接近的）。

Daniel Jurafsky 和 James Martin 撰写的 *Speech and Language Processing* 一书是自然语言处理的绝佳参考资料。此书的作者将自然语言工程学分成如下部分：

- 语音学和音系学——研究单词发音以及如何根据人声识别单词。
- 语言构词学——研究人类语言中单词的构成，以及它们是如何表达各种含义的。例如，“电话（phone）”和“音韵学（phonology）”之间的关系，“水疗（hydro）”和“水文学（hydrology）”之间的关系。
- 语言语法学——研究自然语言中如何构造句子。有一个很有名的预言能说明语法的重要性，这是先知 Delphi 赐给一位即将上战场的战士的，其中的单词

Download at http://www.pin5i.com/

328

（古希腊语）是这样的“ηξειζαψψηξειζθνηξειζ”。这些单词能有两种解释，乐观的解释是，先知说“勇敢地去吧，你将从战场平安归来。”另一种解释则是，先知说“勇敢地去吧，你会光荣地战死沙场。”知道正确的语法是解释阿波罗语言的关键。

- 语义学——研究单词在特定上下文中的含义。例如，在美式英语中，“cool”可以解释成“不冷也不热”或“自信，无须担心”，还可以根据上下文解释成“表现得冷漠和不屑”。
- 语用学——研究单词的含义和交流目的之间的关系。
- 话语学——研究比句子更大的语言单元。

Jurafsky 和 Martin 对所有的这些内容都做了介绍。此外，他们还回顾了语音合成、语音识别、统计语法分析、时间表达分析（temporal expression analysis），甚至还涉及了交谈代理（conversational agent）和机器翻译。

从实践方面说，现在已经有很多关于 NLP 的开源项目，读者如果对文字或其他语言表现形式感兴趣，可以尝试做这些项目。UIMA 是 IBM 在十年前发起的，现已被 OASIS 标准化（http://www.oasisopen.org/）。UMIA 的全称是 Unstructured Information Management Applications，指的是用于分析海量非结构化信息（例如各种格式的商业报告、分析和合同）的软件系统。现在该项目由 Apache 的项目孵化器维护（http://incubator.apache.org/uima/）。经过长时间的发展，该项目已稳定，其开发也非常活跃。

另一个稳定的项目是 GATE（General Architecture for Text Engineering），该项目由谢菲尔德大学自 1995 年起开始开发，有很好的支持文档。如果需要一些方便上手的例子来学习 Jurafsky 和 Martin 的书中涉及的各种内容，这个项目无疑是最佳选择。根据项目的文档，该项目由以下三个元素构成：

- 描述如何将各种单独的组件组装成语言处理系统的架构。
- Java 编写的框架，提供了底层的 API。
- 图形开发环境。项目的网址是：http://gate.ac.uk/。

你可能也会对主要由 William Cohen 开发的 MinorThird 项目感兴趣。William Cohen 是卡内基梅隆大学机器学习系的教授。项目发起者对该项目的描述是：“MinorThird 是用于存储、注释文本，学习从中提取内容和文本分类的 Java 类的集合。”该项目托管在http://sourceforge.net网站上，以 BSD 协议发布，在将其嵌入到产

Download at http://www.pin5i.com/

品中之前请先阅读第三方库的注意事项。尽管有法律方面的一些注意事项，但这个项目中丰富的 NLP 方法还是对分析文本非常有帮助的。其他著名的开源 NLP 库有 MinorThird 的前辈 SecondString，以及一个名为 openNLP 的大型项目。这两个项目都可以在http://sourceforge.net上找到。 329

D.1　参考资料

Cohen, W.W. "Minorthird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data." http://minorthird.sourceforge.net, 2004.

Jurafsky, D. and J. Martin. *Speech and Language Processing,* Second Edition. Prentice Hall (Series in Artificial Intelligence), 2008.

Saygin, A.P., I. Cicekli, and V. Akman. "Turing Test: 50 Years Later." *Minds and Machines* 10 (4) pp. 463-518, 2000.

Turing, A. "Computing Machinery and Intelligence." *Mind,* 59 (236), pp. 433-460, 1950.

Download at http://www.pin5i.com/

附录 E
神 经 网 络

在第 5 章中，我们介绍了神经网络的核心思想，同时还演示了如何在实际环境中构建、训练和使用一个简单的神经网络。神经网络有广泛的应用范围，我们所看到的只是冰山一角而已。很难在不涉及数学知识的情况下来讨论神经网络的基础知识。我们希望自己能很好地解释有关的基本概念，但深入地理解其中的工作原理需要仔细阅读有关的专业文献。

为了降低从本书（即《智能 Web 算法》）转移到非常专业的神经网络技术的难度，我们向读者推荐两本通俗易懂的介绍神经网络中数学知识的书。其一是 Tom Mitchell 的 *Machine Learning*，这是一本非常优秀的机器学习入门书籍，我们强烈推荐将其作为常用的参考书。尤其是在介绍人工神经网络的一章中，作者还介绍了反向传播算法及其数学推导过程，一个详细的人脸识别的例子、各种误差函数、各种最小化误差的过程，以及网络结构的动态改变。

对于通用的文献，我们首先推荐的是 David MacKay 的 *Theory, Inference, and Learning Algorithms* 一书。该书是介绍更加先进和复杂的神经网络的，它首先介绍了神经网络的概况，接着详细介绍了作为分类器的单个神经元，以及单个神经元的作用，并从概率的角度揭示了神经网络的学习过程。另外还介绍了名为 Boltzmann 机（对具有物理背景的读者来说，这是一个很精彩的话题）的随机 Hopfield 网络。贝叶斯神经网络也被用来作为介绍 Gaussian 过程的跳板。Gaussian 过程是一个已被广泛使用的统计模型。还应该提及的是，MacKay 书中所有的算法都是用名为 Octave 的自由、开源的数据编程语言实现的。该语言可以在所有的主流操作系统中运行，因

Download at http://www.pin5i.com/

此，可以方便地试验这些代码。

在这类图书中，关于神经网络研究的一本必读书是神经网络领域的大师 Tuevo Kohonen 的 *Self-Organizing Maps* 一书。该书不仅在无监督学习中介绍了神经网络，还有一章简明扼要地介绍了必要的数学知识，并且以最佳的形式介绍了学习矢量量化（learning vector quantization）技术。这本内容丰富、针对研究生写作的书不仅有很强的原创性，其阅读体验也非常优秀。

顾名思义，Paul D. McNelis 的 *Neural Networks in Finance: Gaining Predictive Edge in the Market* 关注的是神经网络在金融领域的应用。在入门章节之后，该书介绍了几个金融问题的建模，例如 Black-Sholes 定价模型、企业债券建模、时间序列预测、信用卡银行风险以及通货膨胀和通货紧缩的建模等。

还有一本要提及的书是 Akira Hirose 的 *Complex-valued Neural Networks*。该书系统地介绍了近年来发展迅速的复值神经网络（complex-valued neural network）。这类网络在自适应雷达系统、数字海拔地图处理、语音合成等领域有重要的意义。由于这类网络是用复数表示的，所以任何能用复值量化表示的问题都能从中获益。显然，这不是一本入门书，但它可以为读者带来新的想法，扩展读者在设计神经网络时的思路。该书还介绍了一般神经网络设计中所面临的挑战。下面的参考资料中提供了更多有关神经网络的参考书籍。

E.1　参考资料

Arbib, M.A., S. Amari, NetLibrary, Inc., and P. H. Arbib. *The Handbook of Brain Theory and Neural Networks*. MIT Press, 2003.

Bishop, C.M., *Neural Networks for Pattern Recognition*. Oxford University Press, 1996.

Dreyfus, G. *Neural Networks*. Springer, 2005.

Haykin, S. *Neural Networks: A Comprehensive Foundation, Second Edition*. Prentice Hall, 1998.

Kohonen, T. *Self-Organizing Maps*, Third Edition. Springer, 2000.

MacKay, D.J.C. *Information Theory, Inference, & Learning Algorithms*. Cambridge University Press, 2003.

Maier, K.D., C. Beckstein, R. Blickhan, W. Erhard, and D. Fey. "A multi-layer-perceptron neural network hardware based on 3D massively parallel optoelectronic circuits." Proceedings of the *6th International Conference on Parallel Interconnects*, pp. 73-80, 1999.

Mandic, D.P., and J. A. Chambers. *Recurrent Neural Networks for Prediction: Learnning algorithms, architecture, and stability*. John Wiley & Sons, Inc., 2001.

Mitchell, T. *Machine Learning*. McGraw Hill Higher Education, 1997.

Neapolitan, R.E.. *Learning Bayesian Networks*. Prentice Hall, 2003.

Ripley, B.D. *Pattern Recognition and Neural Networks*. Cambridge University Press, 2008.

Download at http://www.pin5i.com/

索 引†

数字

10-fold cross-validation (10 重交叉验证) 221
χ^2 distribution (卡方分布) 255–256
χ^2 statistic (卡方统计) 251, 274–275

A

abstract syntax tree. See AST (抽象语法树，见 AST)
Accuracy (准确性) 221
accuracy differences (准确性差异) 251
activation rule (激活原则) 203
AdaBoost 234, 265, 267, 276
adaptive resampling (自适应重采样) 265, 276
adjacency (邻接)
  list (列表) 62
  matrix (矩阵) 134
adjustable mortgages (浮动利率贷款) 234
adjustment (调整)
  primitivity (原始性) 36
  stochasticity (随机性) 36
Aggarwal, C.C. 160
Agglomerative hierarchical algorithms (凝聚层次算法) 129
aggregated content (聚合内容) 5
aggregating classifiers (聚集分类器) 263
AI (人工智能) 16
  utility problem (实用性问题) 190
Aitken extrapolation (Aitken 外推法) 63
algorithms (算法) 6
agglomerative hierarchical (凝聚层次化) 129
applicability limitations (应用限制) 18
application context (应用环境) 278
arc-x4 269
average-length (平均长度) 137
average-link (平均链接) 138
BIRCH 131
Borvka_fs 141
classification (分类) 164
clustering (聚类) 302
computational time estimation (计算时间估计) 19
constrained clustering (受约束聚类) 130
density-based (基于密度) 151
divisive hierarchical (分割层

†：索引所列页码为英文版页码，请参照“▭”表示的原书页码。

Download at http://www.pin5i.com/

次化) 129
Expectation-Maximization (最大期望) 161
gradient-descent learning (梯度下降学习) 218
graph theoretic (图论的) 139
KISS 19
k-means (k 均值) 142
Kruskal_fs 141
link-based (基于链接) 132
NaiveBayes (朴素贝叶斯) 172
nearest neighbor (最近邻居) 129
parallelization (并行) 18
partitional (分割) 129, 142
ranking (排序) 286
regression (回归) 172
Rete 192
ROCK 147, 300
scalability (可扩展性) 18
single-link (单链接) 135
SQLEM 161
structural (结构化) 171
alpha 62
Amazon.com (亚马逊网站) 108–109
among first with recommendations (第一个推荐系统) 3
item to item approach (商品到商品的方法) 92
analysis (分析)
link (链接) 22
user click (用户点击分析) 22, 33
analyzer (分析器)
lexical (词法) 31
analyzing (分析) 30, 282
and (和) 227
Android (Android 操作系统) 287
Anscombe's quartet (安斯库姆四重奏) 112
ANTLR 193
Apache Axis 15
Apache CXF supports numerous standards (Apache CXF 支持多种标准) 15
Apache POI project (Apache POI 项目) 30, 283
Apache Shindig 9
application intelligence (应用智能) 279
arc-fs 276
arcing 265
arc-x4 234, 265, 267, 276
arc-x4 algorithms (arc-x4 算法) 269
crux (难题) 271
fourth power (四次方) 271
Armstrong, Lance 23, 175
arrays sorting (数组排序) 125
Arthur, David 145
artificial data (人造数据) 241
generation (生成) 239
artificial intelligence. See AI (人工智能。见 AI)
assignTopicToCluster 297–298
assignTopicToStory 297, 299
assumption of independence (独立性假设) 254
AST 193
Atom 13
not RFD-based (不基于 RFD) 14
syndication format (聚合格式) 14
Attribute (属性) 175
attribute selection (属性选择) 273
news portal (新闻门户网站) 313
automatic categorization (自动分类) 174
average distance (平均距离) 137
average-link algorithm (平均链接算法) 137–138
AverageLinkAlgorithm 134

B

back propagation algorithms termination conditions (回溯算法终止条件) 218
backward chaining (回溯链) 189
BadUserType 239
bagging 257, 263
algorithm (算法) 258
data diversity (数据多样性) 258
independent classifiers (独立分类器)265
premise of (前提) 258
tweaks (调优) 260
tweaks and tips (调优和提示) 276
BaggingCreditClassifier 258–260
accuracy and execution time (精确度和执行时间) 259
bankruptcy significance (破产显

Download at http://www.pin5i.com/

著性) 235
base recommenders enhancement (基本推荐器增强) 107
BaseConcept 297
BaseInstance 178, 297
BaseLayer 216
BaseNN 208, 213–214
basic search (基础搜索)
load, index, search (加载、索引、搜索) 24
BasicWebCrawler 27, 282
Bayes theorem (贝叶斯理论) 49, 172, 182, 227
attribute independence assumption (属性独立性假设) 183
conditional probability (条件概率) 182
evidence (证据) 182
likelihood (释然率) 182
naive assumption (朴素假设) 51
posterior probability (后验概率) 182
prior probability (先验概率) 182
Bayes theorem formula output (贝叶斯理论公式输出) 184
Bayes theorem input probability (贝叶斯理论输入概率)
evidence (证据) 48
likelihood (释然率) 48
prior probability (先验概率) 48
Bayes theorem output probability (贝叶斯理论输出概率)
posterior probability (后验概率) 49
Bayesian (贝叶斯)
belief networks (信仰网络) 182
networks (网络) 172
neural networks (神经网络) 330
BeanShell (BeanShell 脚本语言) 317
Bernoulli process (贝努利过程) 221
Beyer, Kevin 160
bias vs. generalization (偏见与泛化) 19
binary classification (二分类) 174, 178
BIRCH
algorithm (算法) 131
clustering algorithm (聚类算法) 158
black box trap (黑盒陷阱) 205
Black-Sholes option pricing model (Black-Sholes 定价模型) 331
Boltzmann machines (Boltzmann 机器) 330
Boorah 13
BoostCreditClassifier 267
boosting 234, 265
computational performance (计算性能) 267
main idea (主要想法) 265
strategy (策略) 67
BoostingARCX4Classifier 268–269
BoostingCreditClassifier 268
bootstrap (引导程序) 222, 257
aggregating (聚合) 234, 257
process (过程) 258
BootstrapTrainingSetBuilder 261, 276
Borvka_fs algorithm (Borvka_fs 算法) 141
Borvka, Otaker 161
Bradley, Paul S. 131
Breiman, Leo 265
Brin, Sergey 33
BSD license (BSD 授权) 30
building intelligence (构建智能) 12
business news (商业新闻) 284

C

C5.0 171
calculation of similarity (计算相似度) 84
car ownership (车辆所有权) 236
categorical data (分类数据) 130
categories (分类) 164
internet newsgroups (网络新闻组) 166
newspaper articles (报纸文章) 166
restaurant menu (餐馆菜单) 166
categorization (分类)
automatic (自动地) 174
email (电子邮件) 187
Celtix by IONA. See Apache CXF (IONA 的 Celtix，见 Apache CXF)
centroid (中心点) 142
initial selection (初始选择) 145

Download at http://www.pin5i.com/

role (角色) 143
Cereghini, Paul 161
CF 80
item based (基于商品的) 89
requirements (需求) 80
CFOI 166
chain effect (链式反应) 142
Chebyshev polynomial (切比雪夫多项式) 63
Chi2 251
chi-square (卡方) 251, 274
chronological age (按年龄排序) 235
Cinematch 3, 107
city block metric (城市区块测度) 118
CJK tokenizing (中日韩语分词) 31
classification (分类) 48, 289
algorithms (算法) 164–165
binary (二分类) 169, 174, 178
content noise reduction (内容降噪) 293
continuous values (连续值) 169
correct (正确) 243
cost (代价) 221
cross-referencing (交叉引用) 304
data noise effects (数据噪声的影响) 206
discrete values (离散值) 169
distance-basedalgorithms (基于距离的算法) 169
email (电子邮件) 228
erroneous (错误的) 243
flat class structure (平面分类结构) 169
forecasting (预测) 169
generalization vs. specialize-tion (通用化和专业化) 178
groups of instances (实例的群组) 294
hierarchical class structure (层次化分类结构) 169
majority vote rule (多数投票规则) 295
multiclass (多分类) 169, 174, 178
neural networks (神经网络) 165, 169
news categoies (新闻分类) 294
news groups (新闻组) 294
order effect (次序的影响) 288
overview (概览) 169
performance characteristics (性能特点) 250
region of influence (影响的区域) 233
regression algorithms (回归算法) 169
classification (continued) (分类（续）)
representative news story (代表性新闻) 299
rule-based (基于规则) 188
rule-based algorithms (基于规则的算法) 169
runtime (运行时间) 250
runtime performance (运行时间性能) 224
specialization vs. generalization (专业化和通用化) 178
statistical algorithms (统计算法) 169, 172
strategy (策略) 288
structural algorithms (结构化算法) 169–170
training time (训练时间) 250
utility problem (实用性问题) 225
wrong decision impact (错误决策的影响) 222
classification accuracy (分类准确性)
statistically insignificant (统计显著的) 258
classification attributes (分类属性)
large number effect (大量分类属性的影响) 166
classification system (分类系统)
CFOI 166
ICD-10 166
Library of Congress (国会图书馆) 166
Linnaean (林奈) 166
OIIC 166
Schatzker 166
SOII 166
classification training (分类训练)
attribute value coverage (属性

Download at http://www.pin5i.com/

值覆盖率) 223
representative data (代表性数据) 223
scaling characteristics (缩放特征) 224
statistical assessment (统计评价) 224
ClassificationStrategy 297, 314
ClassificationStrategyImpl 294–295, 297, 299, 314
classifier ensemble (分类器组合)
incremental growth (增量式增长) 265
classifier training (分类器训练)
user clicks (用户点击) 48
ClassifierEnsemble 260–261, 263, 268, 271
ClassifierResults 251
classifiers (分类器) 48
aggregating (聚合) 263
combination (合并) 232
comparison (比较) 233
decision tree (决策树) 266
ensembles (组合) 263
fusion (融合) 232
lifecycle stages (生命周期阶段) 173
metaclassfier scheme (元分类器策略) 173
pair-wise comparisons (逐对比较) 250
selection (选择) 232, 256, 275
sensitivity (敏感性) 207
stable (稳定) 245
training stage (训练阶段) 173
unstable (不稳定) 245, 258
validation stage (验证步骤) 173
classify-cluster approach (先分类再聚类方法) 293
Classify-Cluster-DS 292
classifyClusters 299
classifyStories 299
cleansing news stories (净化新闻) 285
Clearspring 9
Clementine171
CLIPS 170
cluster (聚类)
discovery (发现) 125
formations (形成) 140
invariance (不变性) 306
structure (结构) 128
cluster centroid (聚类中心点)
center of mass analogy (与质心的类比) 143
cluster formation goodness (聚类形式优良度) 315
cluster-classify approach (先聚类后分类方法) 293
Cluster-Classify-DS 292
clustering (聚类) 289
by age (按年龄) 127
agglomerative hierarchical algorithms (凝聚层次算法) 129
algorithm (算法) 302
arbitrary objects (任意对象) 128
array sorting (数组排序) 125
average distance (平均距离) 137
average-link algorithm (平均链接算法) 137
BIRCH algorithm (BIRCH 算法) 131
book example (书籍例子) 122
by cluster structure (按聚类结构) 129
categorical data (分类数据) 130
categorization (分类) 128
centroid (中心点) 142
computational complexity (计算复杂度) 157
conceptual modeling (概念建模) 129
constrained algorithms (受约束算法) 130
curse of dimensionality (维度灾难) 159
data normalization (数据规范化) 127
by data size (按数据大小) 131
data squashing (数据挤压) 158
by data structure (按数据结构) 130
by data type (按数据类型) 129–130
DBSCAN 151
dendrograms (树状图) 132
density-based algorithms (基

Download at http://www.pin5i.com/

于密度的算法) 151
divisive hierarchical algorithms (分割层次算法) 129
epsilon neighborhood (ε 领域) 154
Euclidian distance (欧氏距离) 127
fine tuning (微调) 316
goodness measure (优度度量) 150
hierarchical (层次) 316
hierarchical algorithms (层次算法) 129
high dimensionality (高维度) 158
human expert (人类专家) 127
in high dimensions (在高维度中) 157
iterative optimization (迭代优化) 129
k-means algorithm (k-means 算法) 129, 142
lack of normalization (缺乏规范化) 134
large databases (大型数据库) 131
link-based algorithms (基于链接的算法) 134
many dimensions (多个维度) 128
mean value (平均值) 142
MST 139
news articles (新闻) 129
objective (目标) 150
and ordering (和排序) 124
overview (概述) 128
partitional algorithms (分割算法) 129
performance characteristics (性能特点) 157
point density (数据点密度) 151
proximity threshold (相似度阈值) 135
ROCK 147
R-trees (R 树) 158
single-link algorithm (单链接算法) 135
singletons (单节点聚类) 140
Sourceforge-like case study (类似于 Sourceforce 的实例研究) 123
spectral methods (谱方法) 130
SQL limitations (SQL 的局限) 125
SQLEM algorithm (SQLEM 算法) 125
threshold parameter (阈值参数) 129
very large datasets (超大规模数据集) 157
visual identification (视觉识别) 124
VLDB 131
wavelet methods (小波方法) 130
with SQL (用 SQL) 124
clustering algorithm (聚类算法)
BIRCH 158
combinations (组合) 162
clustering applications (聚类的应用)
creation of social networks (构建社交网络) 122
like-minded individuals (志趣相投的人) 122
targeted advertisement (目标定位广告) 122
clustering attribute (聚类属性)
age (年龄) 123
education level (教育水平) 123
income range (收入范围) 123
paid participation (付费参与) 123
professional skills (专业技能) 123
social relationship rating (社会关系评价) 123
clustering distribution optimality (聚类分布最优性) 315
Cochran's Q test (Cochran Q 检验) 233, 250, 255–256
Codehaus XFire. See Apache CXF (Codehaus XFire。见 Apache CXF)
collaboration as opposed to intelligence (协作与智能的区别) 4
collaborative filtering. See CF (协同过滤。见 CF)

Download at http://www.pin5i.com/

collaborative platforms (协作平台) 5
collective intelligence (集体智慧) 4
collective knowledge capture (集体智慧获取) 4
combination of classifiers (分类器融合) 232
  computational robustness (计算健壮性) 233
  representational advantage (典型优点) 233
  risk reduction (降低风险) 233
combining classifiers bagging (分类器融合)
  bagging 234
  boosting 234
comparator (比较符) 126
complexity (复杂性)
  multiclass classification (多分类) 224
computational (计算的)
  cluster (聚类) 63
  complexity (复杂性) 157
  cost (代价) 316
  linguistics (语言学) 327
  nodes (节点) 63
Concept 175, 182
CONCEPT_LABEL_FRAUD 211
CONCEPT_LABEL_VALID 211
ConceptMajorityVoter 264
conceptPriors 50
  map (图) 183
conditional probabilities (条件概率) 50, 183
  user clicks (用户点击) 48
confidence interval (置信区间) 221
conflict resolution (冲突解决) 196–198
confusion matrix (混淆矩阵) 220, 243, 259, 274
constrained clustering algorithms (受约束聚类算法) 130
content (内容) 13
  aggregator (聚合器) 6, 9
  annotation (注释) 9
  cleansing (净化) 283
  field (域) 27, 31
  impurities (杂质) 283
  reconciliation (调整) 7
content aggregation (内容聚合)
  digg.com 99
content similarity (内容相似性)
  case study (实例研究) 93
  normalization (规范化) 103
  text analysis sensitivity (文本分析敏感性) 96
content-based (基于内容)
  accumulation and analysis (积累和分析) 80
  recommendation (推荐) 70
Corcho, Oscar 165
correlation (相关)
  complete negative (完全为负) 111
  complete positive (完全为正) 111
cosine similarity (余弦相似性) 95, 324
CosineDistance 152
CosineSimilarity 149, 187
CosineSimilarityMeasure 95
cost (代价)
  function (函数) 223, 230
  matrix (矩阵) 230
craigslist 13
crawler (爬虫) 13
  collecting data (采集数据) 22
  custom (自定义) 281
  fetched documents (获取文档) 24
  known URLs (已知 URL) 24
  page links (页面链接) 24
  processed documents (已处理文档) 24
crawling (爬行) 23, 30, 281–282
  Apache Tika 321
  custom Web crawler (自定义 Web 爬虫) 320
  depth of (深度) 13
  Heritrix 321
  Nutch 321
  retrieved content structure (获取的文档结构) 282
  stages of (步骤) 320
CrawlResultsNewsDataset 284
createClusters 300–301
createClustersWithinTopics 300, 305

Download at http://www.pin5i.com/

credibility of classification (分类可信度) 219
credit (信用)
  risk (风险) 233
  score (评分) 236
credit card activity (信用卡活动) 236
credit worthiness (信用价值)
  attributes (属性) 235
  case study (实例研究) 233
  overview (概况) 234
CreditErrorEstimator 244, 266, 274
criminal record (犯罪记录) 236
cross product calculation (叉乘计算) 111
cross-referencing (交叉引用) 304
curse of dimensionality (维度灾难) 159, 166
CustomAnalyzer 95
Cutting, Doug 22

D

DAG (DAG) 172, 202
damping factor (阻尼系数) 36
DangerousUserType 239
dangling node (悬挂节点) 62
  heuristic (启发式) 67
data (数据)
  diversity (多样性) 265
  incongruent (不一致) 17
  missing values (缺失值) 17
  noisy (噪声) 259
  normalization (规范化) 17, 156, 204
  preprocessing (预处理) 204
  reliability (可信度) 17
  renormalization (再规范化) 115
  representation inaccuracies (表示不准确) 17
  size issues (大小问题) 18
  squashing (挤压) 158
  understanding (理解) 207
  understanding importance (理解重要性) 279
  variability (可变性) 17
data normalization (数据规范化) 110
  PearsonCorrelation 113
DataGenerator 240
DataPoint 146, 154
Datapoint 134
dataset (数据集)
  dimensionality (维度) 156
DatasetAdapter 311
DBSCAN 151
  algorithm (算法) 162
  border point (边界点) 154
  core point (核心点) 154
  directly density reachable (密度直接可达) 154
  eps variable (eps 变量) 154
  ink drops analogy (滴墨模拟) 151
  minPoints variable (minPoints 变量) 154
DBSCANAlgorithm 152–154
decision tree (决策树) 170, 245, 258, 273
  accuracy (准确度) 246
  algorithms (算法) 171
  classifier (分类器) 234, 266–267
decision tree classification (决策树分类)
  instability (不稳定性) 247
  interpretation (解释) 247
decisionTree
  printTree 247
declarative programming (申明式编程) 188
default analyzer (默认分析器) 31
degree (度)
  of belief (信仰) 81
  of credibility (可信) 223
  of freedom (自由) 251, 255–256
Delphi (Delphi 推荐引擎) 310–311
  Dataset interface (Dataset 接口) 81
  inner workings (内部原理) 86
  recommend (推荐) 87
  recommendation engine (推荐引擎) 80–81
  similarity between users (用户间相似性) 82
DelphiUC 103
delphiUC 103
DelphiUR 103
Dendrogram 132
dendrogram (树状图)

Download at http://www.pin5i.com/

data structure (数据结构) 132–133
initialized (初始化的) 138
two linked hash maps (双链哈希表) 132
visual representation of (可视化表示) 132
density-based algorithms (基于密度的算法) 151
spatial clustering of applications with noise. See DBSCAN (空间聚类在噪声数据中的应用。见 DBSCAN)
dez 165
Dhillon, Inderjit S. 145
diagnosis (诊断)
of diseases (疾病) 166
of injuries (伤害) 166
Diff2PropTest 253
difference of proportions test (比例差异测试) 233, 250, 253
Digg
API 99, 146
RESTful services (RESTful 服务) 14
Digg stories (Digg 新闻)
blood donors (献血者) 146
CSV file (CSV 文件) 146
Facebook 146
DiggCategory 100
DiggDelphi
findSimilarUsers (findSimilarUsers 方法) 103
getTopNFriends (getTopNFriends 方法) 103
inner workings (原理) 102
recommend (推荐) 103–104
dimensionality (维度)
curse of (灾难) 157
directed acyclic graphs. See DAG (有向无环图。见 DAG)
directed graph (有向图) 34
discourse (话语) 288, 328
Distance 154
distance (距离)
city block (城市块) 324
Euclidean (欧式) 324
L2 324
properties (属性) 73
symmetry (对称) 74
taxi cab (出租车) 324
triangle inequality (三角不等式) 74
distributed computing fallacies (分布式计算设施) 17–19
distribution of clusters (聚类分布) 305
divisive hierarchical algorithms (分割层次算法) 129
docid field (docid 域) 27
DocRank (DocRank 算法) 55–56, 280, 286
inner workings (原理) 57
matrix builder (矩阵构造器) 57
relational tables (关系表) 61
values reused (值重用) 61
doctype field (doctype 域) 27
document (文档)
distance (距离) 92
heuristic importance (启发式重要性) 59
terms (术语) 286, 288
document collection (文档集合)
business news (商业新闻) 23
Lance Armstrong 23
U.S. politics (美国政治) 23
world news (世界新闻) 23
domain of discourse (谈话的领域) 5
dot (inner) product (点积(内积)) 96
Drools 165, 170, 189, 193
ReteOO 190
Drools attribute (Drools 属性)
no-loop (无循环) 197
ruleflow-group (规则流组) 197
salience (显著) 197
DTCreditClassifier 246, 259, 266
Dunham, Margaret 158

E

ECLiPSe 189
Eisner, Jason 161
elements of intelligence (智能要素)
synergy (协同) 100
EM algorithm (EM 算法)
E-step (E 步骤) 161
M-step (M 步骤) 161
email categorization (电子邮件分组) 174, 178, 187

Download at http://www.pin5i.com/

email classification (电子邮件分类)
blacklists (黑名单) 175
header tests (头部测试) 175
idiosyncracies (特点) 175
real-time blackhole lists (实时黑洞名单) 175
whitelists (白名单) 175
email concept (电子邮件概念)
NOT SPAM 178
SPAM 178
email content (电子邮件内容)
congressional elections (国会选举) 175
global warming (全球变暖) 175
Lance Armstrong 175
marathon (马拉松) 175
newspaper advertisement (报纸广告) 175
Nicaragua elections (尼加拉瓜选举) 175
NVidia stock (NVidia 股票) 175
Ortega 175
spam (垃圾) 175
U.S. politics (美国政治) 175
world news (世界新闻) 175
email messages sorting (电子邮件排序) 174
EmailClassifier 175–176, 178, 184
EmailData 176
EmailDataset 176
getTrainingSet 178
setBinary(false) 187
EmailInstance 178
EmailRuleClassifier 192
embedding intelligence (嵌入智能) 11
Engage 9
ensembles (组合)
accuracy (准确度) 260
of classifiers (分类器组合) 263
Epictetus (埃皮克提图) 232
epsilon neighborhood (ε 领域) 154
selecting value (选择值) 156
error (错误)
type I (I 类) 220
type II (II 类) 220
ESPN 314
Ester, Martin 151
estimateUserBasedRating 88
Euclidean distance (欧氏距离) 77, 127, 130, 145, 160, 324
EuclideanDistance 127
evaluation (评价)
10-fold cross-validation (10 重交叉验证) 221
evaluation for recommendations (推荐评价) 116
ExcellentUserType 239
Expectation-Maximization algorithm (最大期望算法) 161

F

F distribution (F 分布) 256
F statistic (F 统计) 256
F test (F 检验) 233, 250, 255
Facebook 2, 6
RESTful API 14
fact checking (事实检验) 2
fallacies (误解)
intelligent applications (智能应用) 17
Fan, James 145
FASTCLUS 145
Fawcett, Tom 222
Fayyad, Usama M. 131
feed formats (源格式)
Atom 13
RSS 13
FetchAndProcessCrawler 23, 282
addUrl 27
purpose of class (类的用途) 24
fetched (获取的) 282
field content (content 域)
indexed (被索引的) 27
stored (被存储的) 28
unstored (未存储的) 27
Fielding, Roy T. 14
FileListNewsDataset 285, 292, 297
filesystem analogy (文件系统类比)
Attributes 48
Concepts 48

Download at http://www.pin5i.com/

Instances 48
financial (金融)
assets (资产) 238
turbulence (动荡) 234
findSimilarUsers 81
fine tuning clustering (优化聚类) 316
Fisher, Ronald A. 255
Fisher-Snedecor distribution (分布) 256
flat reference structures (平面参考结构) 167
floating-point arithmetic (浮点算术) 119
FN rate (假阴性率) 230
folksonomy (大众分类法) 5
FoodieBytes 13
forecasting (预测)
example (例子) 169
foreclosures (取消抵押品赎回权) 234
Forgy, Charles 190
Forgy, E.W. 144
forward chaining (前向链) 170, 189
FP rate (假阳性率) 221, 230
fraud (欺诈)
benefit application forms (受益的应用形式) 199
detection (检测) 229
internet auction (在线拍卖) 199
purchasing transactions (购买交易) 199
telecommunications (电信) 199
TenUsersSample 200
transactional data (交易数据) 200
fraud detection (欺诈检测) 199
biases (偏差) 214
hidden layer (隐藏层) 214
use case (用例) 199
FraudErrorEstimator 203, 205
fraudulent transactions (欺诈交易)
identify (识别) 204
frequency (频率)
of occurrence (出现) 59
of terms (术语) 95
Friedman, Jerome 172
Friendster 9
F-score 221
FTest 256
functional analysis (功能性分析) 325

G

Gabow, Harold 161
Galil, Zvi 161
games (游戏)
online (在线) 10
garbage collection (垃圾收集) 61
GATE 328
gating network (门网络) 275
Gaussian (高斯)
distribution (分布) 200, 262
processes (过程) 182, 330
generalization (泛化) 227
generated-test-txns.txt 201
geometries (集合)
flat vs. curved (平面与曲面) 79
getNoisyType 241
goodness measure (优良测度) 150
GoodUserType 239
Google 33
began it all (开创新时代) 2
Finance (财经) 3
maps (地图) 13
matrix (矩阵) 35
News (新闻) 3, 8, 279–280, 286
Google PageRank. See PageRank (Google PageRank。见 PageRank)
Gospodnetic, Otis 22
gradient-descent learning algorithm (梯度下降学习算法) 218
grammar-based tokenizer (基于语法的分词器) 31
graph (图)
directed (有向) 34
theoretic algorithms (理论算法) 139
group classification (组分类)
representative news story (有代表性的新闻) 294
grouping (分组)
discrepancy (矛盾) 292
GroupLens 107
groups of news groups (新闻组

Download at http://www.pin5i.com/

分组) 304
Guan, Yuqiang 145
Guha, Ramanathan V. 161

H

Hadoop 63
Hadoop distributed filesystem. See HDFS (Hadoop 分布式文件系统。见 HDFS。)
Hastie, Trevor 172
Hatcher, Erik 22
HDFS 63
health news (健康新闻) 284
Hebbian learning (Hebbian 学习) 229
hexagon (六边形) 143
hi5 9
hierarchical (层次)
agglomerative algorithm (凝聚算法) 130
clustering (聚类) 129
reference structures (参考结构) 167
hierarchical clustering (层次聚类) 129
news stories (新闻) 316
high dimensionality space (高维度空间)158
high-dimensional clustering (高维度聚类) 157
HITS 34
Hits object (Hits 对象) 29
hm 141
home equity (房屋净值)
lines of credit (信贷额度) 236
HousingMaps 12
HTML 30, 283
parser (解析器) 30
hyperbolic tangent function (双曲正切函数)59, 78
Hypertext Induced Topic Search. See HITS (超级文本感知主题搜索。见 HITS)
Hyves 9

I

IBM DeveloperWorks (IBM DeveloperWorks 网站) 7
IETF (互联网工程任务组) 14
if-then clauses (if-then 子句) 170
imeem 9
income (收入) 237
indexDocument 27
indexing (索引) 22, 30, 282
searching beyond (搜索不仅仅是索引) 32
stage (步骤) 31
IndexSearcher 29
IndexWriter 27
inference (推测) 2
response time limits (反馈时间限制) 18
information content (信息内容) 227
information retrieval. See IR (信息获取。见 IR)
inner product (内积) 324
applet (小工具) 324
Instance 175, 182
intelligence (智能)
building (构建) 12
collective (收集) 4
embedding (嵌入) 11
milestone (里程碑) 33
as opposed to collaboration (跟协作的区别) 4
triangle of (三要素) 5
intelligent (智能的)
combination potential (组合潜力) 304
crawling (爬行) 312
document system (文档系统) 4
sanity check note (注意健全性检查) 81
searching (搜索) 288
intelligent application (智能应用)
elements (元素) 5
fallacies (误解) 17
prerequisites (前提条件) 11
internet (互联网)
behavioral characteristic (行为特征) 33
structural characteristic (结构特征) 33
Internet Engineering Task Force. See IETF (互联网工程任务组。见 IETF)
intrapoint distance statistics (内部点距离统计) 156
IR (信息获取) 21, 64
traditional steps (传统步骤)

Download at http://www.pin5i.com/

22
Item 71
item similarity (项目相似度) 89
efficiency (效率) 97
large size data (大规模数据) 92
ItemBasedSimilarity
calculate 90
Items (Items 类) 308
ithm 161

J

Jaccard
coefficient (系数) 130, 146, 186
metric (度量) 84, 91
similarity (相似性) 79, 117, 149
similarity measure (相似性度量) 130
JaccardCoefficient 149, 187, 213, 229
jaccardThreshold 187
JANINO 194
Janino
embedded Java compiler (嵌入式 Java 编译器) 194
JAR (JAR) 81
Java embedded compiler (Java 嵌入式编译器)
Janino 194
Java Archive. See JAR (Java 档案。见 JAR)
JavaScript 14
JavaScript Object Notation. See JSON (JavaScript 对象概念。见 JSON)
JAX-WS 2.0 15
JAX-WSA 15
JBoss
Drools 165
Rules 170, 189
Jess 170, 189
JFlex 31
job classes (工作类别) 237
JSON 1, 14
JSR-181 15

K

K nearest neighbors. See kNN (K 最近邻居。见 kNN。)
Karger, David R. 161
Kendall's tau (肯得尔 tau) 112, 119
Klein, Philip N. 161
Kleinberg, Jon 34
k-means algorithm (k-means 算法) 129, 142, 162, 304
centroids (中心点) 144
core algorithm (核心算法) 143
pickInitialMeanValues 144
k-means clustering (k-means 聚类) 129
k-means++ 145
kNN 171
knowledge representation (知识表示) 165
knownurls 282
Kruskal_fs algorithm (Kruskal_fs 算法) 141
Kullback-Leibler divergences (Kullback-Leibler 发散度) 145

L

L2 norm (L2 范式) 324
labels (标签)
top-level (顶层) 6
land property (土地财富) 238
language (语言)
morphology (语形论) 327
syndication-specific (聚合专用) 14
syntax (语法) 327
language detection (语言探测) 284
Langville, Amy 36
large databases (大型数据库)
algorithm properties (算法属性) 131
large-scale cleansing effectiveness (大规模净化效率) 286
large-scale crawling efficiency (大规模爬行效率) 286
large-scale searching (大规模搜索)
computing constraints (计算限制) 61
data structures (数据结构) 62
PageRank accuracy (Page Rank 精度) 62
learning rule (学习规则) 203

Download at http://www.pin5i.com/

learning vector quantization. See LVQ (学习向量量化。见 LVQ。)
leave-one-out (留一验证法) 222
legitimate transactions (合法交易) 204
level of significance (显著性水平)
statistical test (统计测试) 251, 253, 255
lexical analyzer (词法分析器) 31
lexicographic ordering (字典排序) 130
LGPL 55
library (图书馆)
call number (查询编号) 166
linear (线性)
correlation coefficient (相关系数) 110
regression (回归) 171
link algorithms (链接算法)
comparison (比较) 139
visualization (可视化) 135
link analysis (链接分析) 3, 34
documents (文档) 55
link-based algorithms (基于链接的算法) 132, 134
LinkedIn 9
LinkMatrix 149
links (链接) 201
linkThreshold 303
Linnaean classification (林奈分类) 166
Lloyd, S.P. 144
loan defaults (贷款拖欠) 234
logistic (逻辑)
function (函数) 172
regression (回归) 172
lookup table (查询表)
problems with approach (问题和方法) 16
Lp norm (Lp 范式) 324
Lucene 22, 283, 286
boosting 31
Document class (Document 类) 27
Document object (Document 对象) 30
document score (文档分数) 286
Field class (Field 类) 27
index files (索引文件) 25
Query (Query) 32
query expression (查询表达式) 31
QueryParser class (QueryParser 类) 29
searching (搜索) 28
searching with (用 Lucene 搜索) 22–32
StandardAnalyzer class (Standard Analyzer 类) 31
Lucene analyzers (Lucene 分析器) 30
non-English languages (非英语语言) 31
stop words (停止词) 31
synonyms (同义词) 31
text (文本) 30
Lucene and PageRank combining scores (Lucene 和 PageRank 组合分数) 43–45
Lucene Documents
removal and update (删除和更新) 31
variety (变化) 31
Lucene text (Lucene 文本)
analyzers (分析器) 31
Lucene, PageRank, and naive Bayes combining scores (Lucene、PageRank 和朴素贝叶斯组合分数) 46, 51
LuceneIndexBuilder 25, 66
LucenePDFDocument 30, 283
LVQ 171

M

machine accuracy (机器精度) 119
MapReduce 63
Markov chain theory (马尔科夫链理论) 35
mashups (混搭)
aggregated content (聚合的内容) 7
defined (定义的) 7
mathematical formulas (数学公式) 323
matrices (矩阵) 323
sparse (稀疏) 35
matrix (矩阵)
adjacency (邻接) 134

Download at http://www.pin5i.com/

confusion（混淆）220, 243, 274
cost (代价) 230
similarity (相似度) 134
transition probability (转换概率) 35
matrix H (矩阵 H)
basic link contribution (基础链接贡献) 40
dangling node (悬孤节点)
contribution (贡献) 40
substochastic version (随机版本) 40
symmetric reordering (对称重排序) 62
teleportation contribution (跳转贡献) 40
Word documents (Word 文档) 57
maxBatchSize 27
McNemar 253
test (检验) 233, 250, 274
McNemar test (McNemar 检验)
bagging vs. boosting (bagging 对比 boosting) 267
McNemarTest 251
mean value. See centroid (均值。见中心点。)
media-sharing sites (文件分享网站)
binary format (二进制格式) 9
MegaUpload 9
mergeClusters 139
MergeGoodnessMeasure 149
meta-algorithm (元算法) 279
metaclassifier scheme (元分类器策略) 174
metadata (元数据)
Web page (网页) 24
metric (度量) 76
spaces (空间) 325
Meyer, Carl 36
Microsoft (微软) 15
Microsoft OLE 2 Compound Document (微软 OLE 2 Compound Document) 30
Microsoft Word (微软 Word) 30
97, 2000, XP, and 2003 (97、2000、XP 和 2003) 283
documents (文档) 55
documents parsing (文档解析) 55
parser (解析器) 30
Microsoft Word 2000 (微软 Word 2000) 30
Microsoft Word 2003 (微软 Word 2003) 30
Microsoft Word 97 (微软 Word 97) 30
Microsoft Word XP (微软 Word XP) 30
minimum spanning tree. See MST (最小生成树。见 MST)
mining opportunity（挖掘机会）280
MinorThird 328
misclassification (错误分类)
cost (代价) 243
misclassified (错误分类)
news stories (的新闻) 293
missing attribute value (缺失属性值) 184
mixture of experts (专家的混合) 256, 275
module (模块)
pattern-matching（模式匹配）189
mortgage (按揭) 236
application (申请) 234
down payment (首付) 237
financing (融资) 234
mortgage mess (房贷)
United States (美国) 234
mortgage rates (房贷利率)
teaser (体验) 234
motorcycle ownership (摩托车所有权) 237
MovieLens 107
MovieLens dataset (MovieLens 数据集) 108
large (大型) 116
RMSE 116
small (小型) 110
MovieLensData
createDataset 108
MovieLensDelphi 108, 113, 116
MovieLensItemSimilarity 113
MST 129, 139
algorithms (算法) 161
Borvka_fs algorithm (Borvka_fs 算法) 141
chain effect (链式效应) 142

Download at http://www.pin5i.com/

findMinimumEdge 142
Kruskal_fs algorithm (Kruskal_fs 算法) 141
randomized algorithm (随机算法) 161
MST class (MST 类) 140
Edge 141
MST link algorithm (MST 链接算法)
time complexity (时间复杂度) 142
MSTSingleLinkAlgorithm 134
MSWordDocumentParser 55
multiclass classification (多分类) 174, 178
complexity (复杂度) 224
multidimensional data (多维数据)
ordering (排序) 126
multilingual text (多语言文本) 284
MusicData 81
MusicUser 71, 73
getSimilarity 72, 76
plot 74
MyDiggSpace.com 99
case study (案例研究) 99
data statistics (数据统计) 100
Find friends (找朋友) 100
MyDiggSpaceDataset 146
MySearcher 28, 31, 43, 51
MySpace 2, 6, 9
MySQL 124

N

naive Bayes algorithms (朴素贝叶斯算法) 284, 288
robustness (鲁棒性) 51
naive Bayes classifier (朴素贝叶斯分类器) 243, 267, 294
naive similarity (朴素相似度) 78
beta 78
NaiveBayes (朴素贝叶斯) 48, 50, 175–176, 179, 227, 244, 294, 297
classification (分类) 172
classifier (分类器) 46, 48
TrainingSet (训练集) 182
natural language elements (自然语言元素)
high-level (高层) 288
natural language processing. See NLP (自然语言处理。见 NLP)
NBCreditClassifier 244, 259,266
NBLanguageDetector 284
NBStoryClassifier 294–295,297, 299
Neapolitan, Richard E. 173
nearest neighbor (最近邻居)
algorithms (算法) 129
NekoHTML 30, 283
NetFlix
Cinematch 3
Netflix 3, 108–109
movies selection (电影选择) 107
Netflix prize (Netflix 奖励)
competition (竞赛) 118
RMSE 116
Netflix.com 92
Netscape Rich Site Summary 14
network (网络)
topology (拓扑) 213
neural network (神经网络) 169, 171, 234, 258
architecture (架构) 203
BaseNode (BaseNode) 217
calculateWeightAdjustments 219
connectFully 217
credit classifier (信用分类器) 247
design complexity (设计复杂度) 249
disadvantages (缺点) 172
essential elements (必要元素) 202
feedback (反馈) 202
feedforward (前馈) 202
fireNeuron 217
fireNeuronDerivative 217
fully connected (全连通) 202
layers (层) 202
learning rate (学习率) 217
learningRate 217
LinearNode 216
links (链接) 217
overview (全局概况) 201
SigmoidNode 216
structure (结构) 216

Download at http://www.pin5i.com/

training phase (训练阶段) 202
updateWeights 219
neural network classifiers accuracy (神经网络分类器准确率) 248
neural networks (神经网络) 330
complex valued (复数的) 331
neurons (神经元) 201
news (新闻)
content (内容) 280
portal (门户) 279–280
topic (话题) 297
News Alerts (新闻提醒) 281
news browser (新闻浏览器)
create and display (创建和显示) 291
window (窗口) 286
news categories (新闻分类) 297, 305
assignment (分配) 288
news clustering (新闻聚类)
analysis (分析) 302
news group (新闻组)
clustering robustness (聚类鲁棒性) 293
groups of (分组) 304
juxtaposition (并列) 292
news stories (新闻) 279
arrangement (安排) 290
misclassified (错误分类) 293
searching (搜索) 279
NewsCategory 297
NewsClusterBuilder 302–303
NewsCrawler 281–282, 312
NewsDataset 284, 292, 301
NewsProcessor 286, 295, 299
training phase (训练阶段) 294
NewsStory 297
Niemeyer, Pat 317
Ning 9
NLP 97, 281, 283
NNCreditClassifier 248, 259, 266
NNFraudClassifier 203–204, 208, 210
nodes (节点)
dangling (悬挂) 36, 62
hidden (隐藏) 201
noise (噪声)
elements (元素) 153
levels (水平) 240, 242–243
noisy data (噪声数据) 259
nonparametric (非参数)
correlation (相关) 112, 119
method (方法) 257
normal distribution (正态分布) 242
null hypothesis (零假设) 250
numerical representation (数值表示) 130

O

OASIS 328
Object Management Group. See OMG (对象管理组。见 OMG)
Octave 331
OIIC 166
OLE 2 283
OMG 168
online games (网络游戏) 10
ontology (本体) 165, 167
analogy with OOD (跟 OOD 类比) 167
attributes (属性) 167
concepts (概念) 167
engineering (工程) 165
example (例子) 167
instances (实例) 167
management (管理) 165
semantic (语义) 167
OpenSocial
premise (前提条件) 8
Oracle 9, 15
order of operations (操作顺序) 289
ordering and clustering (排序和聚类) 124
food (食物) 2
Ordonez, Carlos 161
Orkut 79
orkut 9
outlinks (外链) 24, 35
overfitting (过拟合) 178, 227

P

Package 194
Page, Larry 34
pagelinks 282
PageRank 33–45, 280, 286
acceleration techniques (加速技术) 63
Aitken extrapolation (Aitken

Download at http://www.pin5i.com/

外插法) 63
alpha 36
alpha coefficient (alpha 系数) 38
alpha effect on convergence (alpha 系数对收敛的影响) 38
alpha selection (alpha 选择) 38
approximate aggregation technique (近似聚合技术) 63
calculation (计算) 36
convergeness and uniqueness (收敛性和唯一性) 35
damping factor (阻尼系数) 36
dangling nodes (悬孤节点) 36, 67
direct methods (solvers) (直接方法（解决器）) 38
epsilon effects (epsilon 的影响) 42
hyperlink matrix (超链接矩阵) 34
key idea (关键思路) 34
power method (幂方法) 34–35, 38
primitivity adjustment (原始性调整) 36
quadratic extrapolation technique (二次外插技术) 63
random surfer (随机浏览) 36
scaling (缩放) 67
score scaling (分数缩放) 45
stochasticity adjustment (随机性调整) 36
teleportation effect (跳转效果) 36, 38, 67
vector (向量) 34
PageRankMatrixH 38
pair-wise classifier comparisons (分类器逐对比较) 250
parsing (解析) 30, 282
partitional algorithm (划分算法) 142
pattern recognition (模式识别) 9
PDF 30, 283
documents (文档) 55
indexing (索引) 30
PDFBox 30, 283
Pearson's r (皮尔逊 r 系数) 110
counterexample (反例) 112
Pearson's r calculation (计算皮尔逊 r 系数)
singular case (奇异情况) 111
PearsonCorrelation 113
roundoff error (舍入误差) 119
pecuniary aid (金钱援助) 234
perceptions (人知) 164
personalization (个性化) 46, 280
temporal effects (时间效应) 47
vector (向量) 67
phonetics (语音学) 327
phonology (音系学) 327
PhraseQuery 32
slope 32
Plaxo 9
point density (点密度) 151
pointwise deviations (各点的偏差) 111
politics news (政治新闻) 284
portal feature (门户特点)
In the News (新闻中) 280
portals (门户)
aggregated content (聚合的内容)
dispersed (传播) 8
PorterStemFilter 178
posterior probability (后验概率) 184
heuristics (启发) 50
power laws (幂次定律) 45
power method (幂方法)
acceleration (加速) 38
number of iterations (迭代次数) 62
pragmatics (语用) 288, 328
Precision 221
precision (精确度) 64–65
PredictedNewsStoryRating 311
prediction (预测)
degree of belief (可信度) 81
PredictWallStreet 3
preprocessing stage (预处理阶段) 23
prerequisites (前提要求)
for intelligent applications (智能应用) 11
prior probability heuristics (先验概率启发) 51
probabilities (概率)
conditional (条件) 50, 182–183
posterior (后验) 50
prior (先验) 50, 182–183

Download at http://www.pin5i.com/

probability (概率) 182
distribution (分布) 251
of linkage (链接) 316
processed (经过处理的) 282
processing (处理)
natural-language (自然语言) 22
production rules (生成规则) 170
product-moment correlation coefficient (积差相关系数) 110
ProgrammableWeb 7
programming (编程)
declarative (声明式) 188
imperative (指令式) 188
Prolog 189
proximity (亲近度)
relative (相对的) 134
threshold (阈值) 134–135

Q

quality assurance (质量保证) 293
query (查询)
context (上下文) 286
"google ads" 47
terms (词) 288
QueryParser 32

R

Random 242
random (随机)
samples (样本) 257
surfer (浏览) 35
Rank 40
error evaluation (误差评价) 42
rank correlation (排序相关性) 119
ranking algorithms (排序算法) 286
RapidShare 9
Rating 71
rating (评价)
value range note (值区间的注意事项) 81
rating storage (评价存储)
advantages (优点) 86
RatingCountMatrix 84, 91
Ratings 308
ratings (评价)
artificial bias (人为偏差) 104
range (区间) 71
RDF 13
Site Summary (Site Summary) 13
Recall (Recall) 221
recall (覆盖率) 64–65
recommendation engine (推荐引擎) 70–71
based on CF (基于 CF 的) 82
basic concepts (基本概念) 308
code optimizations (代码优化) 89
combinations (组合) 99–100
content based item-item (基于条目内容的) 99
data normalization (数据规范化) 108
ensemble methods (集成方法) 118
neighbor selection (邻居选择) 108
score normalization (分数规范化) 103
similarity (相似性) 71
types (类型) 79
user based (基于用户的)80
recommendation example (推荐实例)
music song online store (音乐歌曲在线商店) 70
online music store (在线音乐商店) 80
recommendation heuristics (推荐启发) 103
recommendation ratings (推荐评分)
value range (值区间) 118
recommendation system (推荐系统)
news stories (新闻报道) 308
size issues (大小问题) 115
recommendations (推荐) 3
accuracy evaluation (准确度评价) 116
ancient proverb (古谚语) 80
based on content (基于内容) 92
cost evaluation (代价评价) 105
evaluation for (评价) 116
large systems (大型系统) 108
live update (实时更新) 115
quality (质量) 115
real-time (实时) 115
response time (响应时间) 110
roundoff-error minimization (最小化舍入误差)118

Download at http://www.pin5i.com/

RecommendationType
ITEM_PENALTY_BASED 311
recommender combination (推荐器组合)
based on averaging (基于平均) 105
based on voting (基于投票) 105
reference encoding (参考编码) 62
reference structures (参考结构) 5, 165
dictionaries (字典) 5
knowledge bases (知识库) 5
ontologies (本体) 5
regression algorithms (回归算法) 172
Reina, Cory A. 131
relative (相对)
proximity (亲近度) 134
ranking (排名) 102
relevance (相关性)
subjectivity (主观性) 46
relevance score (相关性分数) 29, 286
combination (组合) 68
generalization (泛化) 92
repository of knowledge (知识库) 9
representational motivation (表达性的动机) 233
Representational State Transfer. See REST (见 REST)
representations of knowledge (知识表示) 165
Resource Description Framework. See RDF (见 RDF)
REST 14
Rete 170
algorithm (算法) 189, 192
ReteOO 190
retirement (退休) 238
RFC 4287 14
Richardson, Leonard 14
RMSE 116
RMSEEstimator 116
Robust Clustering Using Links. See ROCK (使用链接的健壮聚类。见 ROCK)
ROC
curves (曲线) 222
graphs (图) 222
ROCK 130, 146
algorithm (算法) 161–162, 300, 304
algorithm details (算法细节) 148
algorithms tweaks (算法优化) 315
formula explanation (公式解释) 150
goodness measure (优良测度) 149
initialization stage (初始阶段) 149
key idea (关键思路) 147
link structure (链接结构) 149
termination criteria (终止条件) 149
ROCKAlgorithm 146–147, 303, 315
ROCKClusters class 149
root mean square error. See RMSE(均方根误差。见 RMSE)
roundoff error (舍入误差) 111
magnitude of (量级) 119
minimize (最小化) 118
RSS 13
RSS 2.0 14
R-trees 158
Ruby, Sam 14
rule engine (规则引擎) 189
authoring (编写) 193
runtime (运行时) 193
rule-based classification (基于规则的分类) 188
RuleEngine 228
RuleQuest 171
rules (规则)
AccumulateFunction 194
attribute salience (salience 属性) 197
ChainedProperties 194
ClassificationResult 191
Dialect 194
Email 191
engine (引擎) 165
global statement (全局声明) 191
isSpamEmail 191

Download at http://www.pin5i.com/

Package instance (Package 实例) 194
PackageBuilder 194
PackageBuilderConfiguration 194
RuleBase 194
RuleEngine 193
StatefulSession 194
WorkingMemory 194

S

SAAJ 15
Salesforce 9
salience (salience 属性) 198
sampling (采样)
distribution estimation (分布估计) 257
with replacement (有放回) 257
SAP 15
SAS 145
Scalable Vector Graphics. See SVG (可缩放矢量图。见 SVG)
ScanScout 9
Schatzker classification system (Schatzker 分类系统) 166
score (分数)
relevance (相关性) 29
scoring (打分)
index page (索引页) 45
screen scraping (屏幕抓取) 13
scripting (脚本) 317
search engines (搜索引擎)
tuning (优化) 64
turning point in Web history (网络史上的转折点) 3
search personalization (搜索个性化)
user clicks (用户点击) 45
search quality (搜索质量) 64
metrics (度量) 64
search validation (搜索验证)
precision (精确度) 64
precision/recall plot (精确度/查全率图) 65
recall (查全率) 64
searching (搜索) 21, 30, 282
analysis stage (分析阶段) 31
basic stages (基本步骤) 29
beyond indexing (不仅仅是索引) 32
credibility (可信度) 65
efficient data structures (高效的数据结构) 61
intention (意图) 33
large-scale issues (大规模问题) 61
link analysis (链接分析) 32
Lucene and PageRank (Lucene 和 PageRank)
scores (分数) 43
PageRank 34
reference encoding (参考编码) 62
relevance (相关性) 46
relevance score (相关性分数)29
SearchResult 29
second order effects (二阶效应) 101
See5 171
selectBestMatchingTopic 299
selection strategies (选择策略) 264
selectLongestStory 299, 315
selectRepresentativeStory 315
self-organizing maps (自组织地图) 229
semantic ontology (语义本体) 167
semantics (语义) 288, 328
semiotics (符号学) 198
serialized PHP (序列化 PHP) 14
setTopTerms 297
SFDataset 134
Sheikholeslami, Gholamhosein 130
shortest path metric (最短路径度量) 130
similarity (相似度) 71
ad hoc threshold (ad hoc 阈值) 104
best empirical results (最佳经验结果) 79
between friends (好友间) 81
calculation of (计算) 76, 84
code optimization (代码优化) 82
common misconception (常见误解) 79
compares proximity (亲近度比较) 71

Download at http://www.pin5i.com/

cosine (余弦) 95
evaluation facade (外观评价) 87
formula selection (公式选择) 79
formulas (公式) 77
heuristic (启发) 78
hybrid models (混合模型) 99
item (条目) 89
Jaccard metric (Jaccard 度量) 84
large scale comparisons of (大规模比较) 79
linear correlation coefficient (线性相关系数) 110
measures (计量) 73
metrics (度量) 117
music songs (音乐歌曲) 89
naive (朴素) 78
normalization (正规化) 89, 104
of content (内容) 92
plotting (绘图) 74
ratio of the common items (共有条目比例) 78
related to cognition (与感觉相关的) 74
shortcomings (缺点) 77
symmetric matrix (对称矩阵) 84
symmetrical property (对称属性) 84
visual representation (可视化表示)74
similarity matrix (相似度矩阵) 134
sparsity (稀疏性) 84
upper triangular form (上三角形式) 84
Simple Access Object Protocol. See SOAP (简单访问对象协议。见 SOAP)
single-link algorithms (单链接算法) 135
chain effect (链式效应) 142
computational complexity (计算复杂度) 136
single-link algorithms (continued) (单链接算法（续）)
MST 139
proximity threshold (亲近度阈值) 135
SingleLinkAlgorithm 134
singletons (单例) 140, 146, 294,301, 305
large number (大量) 303
Six Apart (六度分割) 9
SOAP 15
SOAP 1.1 15
SOAP 1.2 15
Soar 170
social networking sites (社交网络网站)
two most visited (两个访问量最大的) 6
SOII 166
SortedArrayClustering 126–127
SourceForge.net 123
SourceForge-like data (类 SourceForge 数据) 134
spam (垃圾)
documents (文档) 56
filtering (过滤) 174
pages (网页) 32
Spam Assassin 175
spamRules.drl 191
spamRulesWithConflict.drl 198
spanning tree (生成树) 139
sparse matrices (稀疏矩阵) 35
Spearman rank-order correlation (Spearman 排序相关性) 112
coefficient (系数) 119
specialization (专业化) 227
Specificity 221
spectral (谱)
clustering (聚类) 130
methods (方法) 63
speech synthesis (语音合成) 327
Spencer, Thomas H. 161
spiders (蜘蛛) 13
sports news (体育新闻) 284
SQL
clustering with (聚类) 124
statements (命令) 124
SQL query (SQL 查询)
ORDER BY 124
SQLEM algorithm (SQLEM 算法) 161
Staab, Steffan 165
standard (标准)

Download at http://www.pin5i.com/

deviation (差) 111
normal distribution (正态分布) 254
StandardAnalyzer 30–31, 95, 178, 283
StandardTokenizer 31
statistic choice (统计选择) 251
statistic q (统计量 q) 255
statistically significant (统计显著性) 250
stochasticity adjustment (随机性调整) 36
stock (股票)
forecasting (预测) 3
StoryRecommender 310–311
structural algorithms (结构化算法) 171
functional (功能化) 171
numerical approximation (数值近似) 171
structures (结构)
flat reference (平面参考) 167
hierarchical reference (层次参考) 167
Studer, Rudi 165
supercluster (超级聚簇) 302, 305
avoidance (避免) 308
breakup (分裂) 307
supervised learning (监督学习) 220
semiempirical approach (半经验方法) 224
support vector machines. See SVM (支持向量机。见 SVM)
surfer (浏览)
random (随机) 35
SVG 1
SVM 16
Swing 292
client (客户端) 301–302
synapse (突触) 201–202
weight (权值) 202
syndicated content (聚合的内容) 13
syndication-specific language (聚合专用语言) 14

T

Tanimoto metric (Tanimoto 度量) 117
Tarjan, Robert E. (Tarjan, Robert E.) 161
Taxicab Geometry (Taxicab Geometry) 118
teaser rates (体验利率) 234
technology articles (科技文章)
email spam (垃圾电子邮件) 290
technology news (技术新闻) 284
teleportation mechanism (跳转机制) 34
tells (告诉) 200
term vectors (搜索词向量) 59
termination criteria (终止条件) 129, 149
test (测试)
difference of (差异)
proportions (比例) 233, 250
test-users.txt 243
text (文本)
multilingual (多语言) 284
tokenizing (分词) 30
understanding freely typed (理解自由输入的) 4
text analysis (文本分析) 59
noise reduction (降噪) 95
stemming (找词根) 95
stop words (停用词) 95
word disambiguation (单词歧义) 97
TextMining 30, 55, 283
See also tm-extractor (另见 tm-extractor)
textual analysis (文本分析) 178
textual information (文本信息)
representation (表示) 95
threshold parameter (阈值参数) 129
threshold value (阈值)
statistical test (统计检验) 250
Tianji 9
Tibshirani, Robert 172
timestamp (时间戳) 47
title field (title 字段) 27
tm-extractor 55
token (分词)
frequency of occurrence (出现频率) 59
tokenizer (分词器)
grammar-based (基于语法的) 31
tokenizing (分词)

Download at http://www.pin5i.com/

acronyms (首字母缩略词) 31
alphanumerics (字母数字混合) 31
Chinese, Japanese, and Korean (中日韩文) 31
computer host names (计算机主机名) 31
email addresses (电子邮件地址) 31
text (文本) 30
top N frequent terms (最频繁的 N 个词) 178
Top stories (焦点新闻) 280
TopicalNewsClusterBuilder 307, 316
top-level labels (顶级标签) 6
training (训练)
bootstrap 222
leave-one-out (留一) 222
TrainingSet 48, 50, 176,178, 297
training-txns.txt 204
training-users.txt 243
trainOnAttribute 183
TransactionDataset 204
TransactionInstanceBuilder 210, 213
TransactionNN 204, 208, 210–211, 213–214, 229
transactions (交易)
identify fraudulent (欺诈识别) 204
legitimate (合法的) 204
transition probability (转换概率)
matrices (矩阵) 35
transparent learning (透明学习)
algorithms (算法) 16
trap (陷阱)
black box (黑盒) 205
triangle of intelligence (智能三元素) 5
triangulation effect (三角化效果) 101
type I error (I 类错误) 220
type II error (II 类错误) 220

U

UIMA 328
uncorrelated items (无关项目) 111
underfitting (低拟合) 227
unsolicited bulk email. See spam (群发垃圾邮件，见 spam)
unstable classifiers (不稳定分类器) 258
unsupervised learning (无监督学习) 128
url field (url 域) 27
UseCaseData 258
createUserTypes 241
User 71
user click analysis (用户点击分析) 33
user clicks (用户点击) 287
news portal (新闻门户) 312
user similarity (用户相似度)
reliability (可信度) 97
UserBasedSimilarity 82
UserClick 48
user-clicks.csv 46, 313
user-content similarities (用户内容相似性) 103
UserContentBasedSimilarity 93
UserCreditNN 248–249, 274
UserInstanceBuilder 245, 261
user-item similarities (用户商品相似度)
content-based (基于内容的) 97
user-item-content similarity (用户-商品-内容 相似度) 103
Users 308
UserStatistics 204
UserType
addNoiseLevel 242
utility problem (实用性问题) 225

V

Vassilvitskii, Sergei145
vector (向量)
personalization (个性化) 67
vectors (向量) 323
very large database. See VLDB (超大规模数据库，见 VLDB)
VeryGoodUserType 239
Viadeo (Viadeo 网站) 9
visual pattern recognition (可视化模式识别) 151
visualization (可视化) 243
VLDB 131
voice recognition (语音识别) 9

W

WaveCluster 130

Download at http://www.pin5i.com/

wavelet clustering (小波聚类) 130
Web
semantic interpretation (语义解释) 14
Web crawler (网络爬虫)
categories (分类) 320
components (组间) 319
custom (自定义) 66
Web crawling (Web 爬取) 13, 319
Web services (Web 服务)
providers (供应商) 15
websites (网站)
inference (推测) 2
learning capacity (学习能力) 2
syndication (聚合) 13
WhitespaceAnalyzer 31
WHO 166
wiki (维基)
defined (定义) 9
Wikipedia (维基百科) 9
world news (世界新闻) 284
WS-Addressing 15
WS-I Basic Profile 15
WS-Policy 15
WS-RM 15
WS-Security 15

X

Xerox (施乐)
Palo Alto Research Center (帕罗奥多研究中心) 80
XForms 1
Xignite
financial Web services (金融 Web 服务) 15
XING 9
XML (可扩展标记语言) 14, 30, 283
XML Path Language. See XPath (XML 路径语言。见 XPath)
XML User Interface Language. See XUL (XML 用户界面语言。见 XUL)
XOR gate (异或门) 213
XORNetwork (异或网络) 213
XPath 1
XSL Transformation. see XSLT (XSL 转换。见 XSLT)
XSLT 1
XUL 1

Y

Yahoo! (雅虎)
RSS feeds (RSS 源) 14
YouTube
media sharing (文件分享) 9

Z

z statistic (z 统计量) 254–255

Download at http://www.pin5i.com/

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

智能Web算法

Algorithms of the Intelligent Web

算法是解决问题的一系列步骤。为实现有价值的Web应用（如推荐引擎、智能化搜索、内容组织系统等），本书提供了清晰的、精心组织过的算法模式。利用这些技术，你可以捕获用户原始而重要的信息，并把它们应用于实践中以获取相应的收益。

用户数据中包含大量有价值的关联信息，它们往往无法通过人工观察而直观地获取，对于希望从这些数据中挖掘信息的Web开发者来说，《智能Web算法》是一本很好的手册。作者作为一名Web开发者，拥有丰富的实践经验，加上多年来对机器学习领域技术的专研，使得本书对技术的解释清晰明了，读者可快速将其用于解决自己的问题。同时，本书提供的Java程序展示了如何搭建一个智能的应用，以及如何从用户的行为中进行学习，这是一笔现成的财富。

本书包含的内容

- 搭建一个像Netflix或Amazon那样的推荐系统。
- 实现Google的PageRank算法。
- 发现社交网络站点的意见领袖。
- 分类组织新闻讨论组的帖子。
- 从共享书签中选择感兴趣的主题。
- 基于文本技术进行垃圾邮件过滤及邮件分类。

Haralambos (Babis) Marmanis 博士是一位把机器学习技术应用于工业界的先行者，也是供应管理的世界级专家。Dmitry Babenko曾经为银行、保险、供应链管理与商务智能公司设计过应用与基础架构。

读者可以通过www.manning.com/AlgorithmsoftheIntelligentWeb在线获得作者的信息、样例代码和免费的电子版样章。

绝对出色——这是我读过的所有技术书籍中最好的。

——Robert Hanson
QualityTechnologyServices

你不需要为搭建一个智能网站而专门去修一个PhD学位，读这本书就可以了。

——Ajay Bhandari,
FoodieBytes.com

非常有用……它会带你快速入门。

——Sumit Pal, LeapFrogrx

非常棒……理论与实践的完美结合。

——Carlton Gibson Noumenal
Software

分析当前所知，解锁Web的未来。

——Eric Swanson, AAA

上架建议：Web开发 > 人工智能

MANNING

策划编辑：张春雨
责任编辑：李利健

ISBN 978-7-121-13919-2

9 787121 139192 >

定价：65.00元

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。