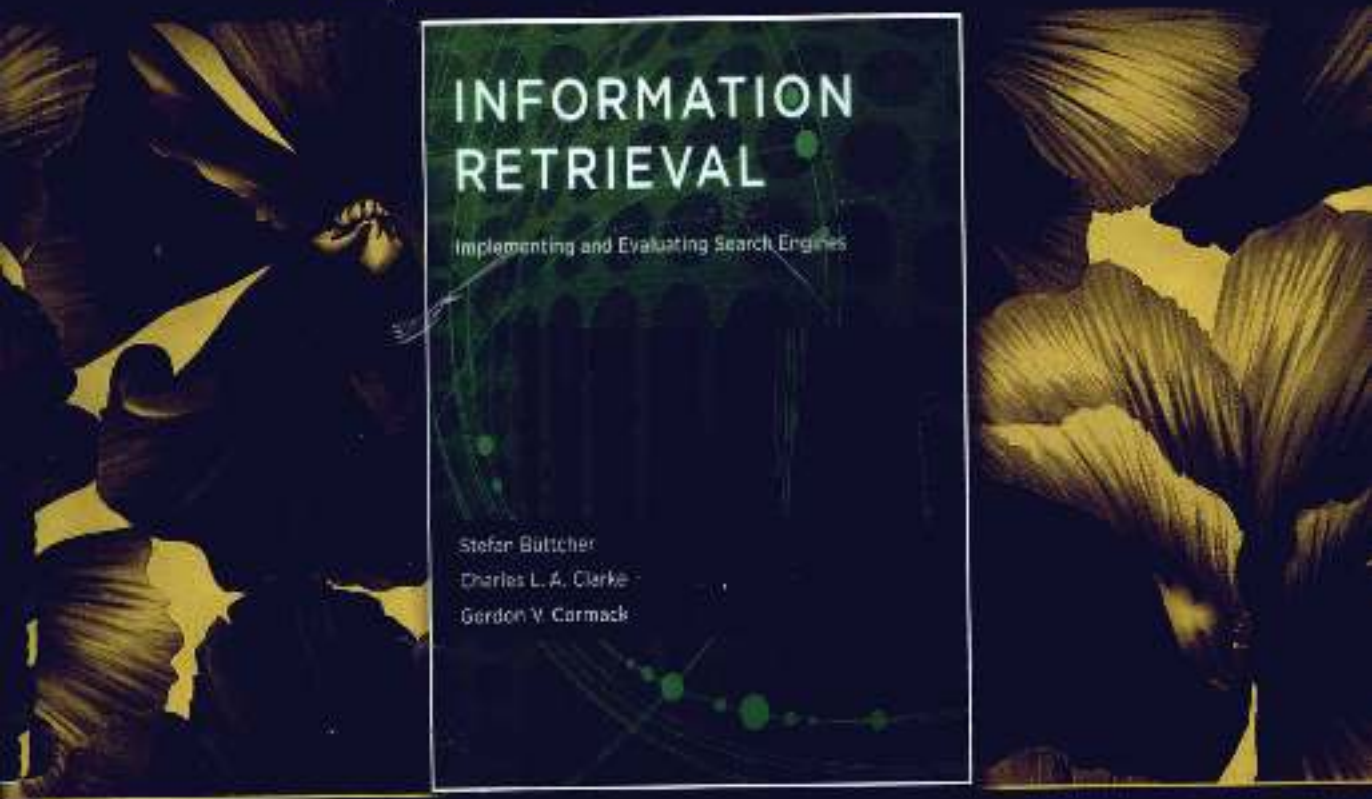
计 算 机 科 学 丛 书

信息检索
实现和评价搜索引擎

（美）Stefan Büttcher（加）Charles L. A. Clarke（加）Gordon V. Cormack 著
陈健 黄青 等译

Information Retrieval
Implementing and Evaluating Search Engines



机械工业出版社
China Machine Press

信息检索 实现和评价搜索引擎

Information Retrieval Implementing and Evaluating Search Engines

"这本书是越来越多的信息检索著作中的一本好书。"

—— Donald H. Kraft，计算机评论

"学术巨匠齐聚一堂编撰了一部信息检索的优秀教材。Stefan Büttcher、Charles L. A. Clarke和Gordon V. Cormack联合计超过50年的研究经验，组成了横跨三代的信息检索研究泰斗组合……这本书是所有信息检索研究者和从业人员的必读教材！"

—— 摘自Amit Singhal撰写的序言

信息检索奠定了现代搜索引擎的基石。本书介绍了现代搜索技术的核心主题，包括算法、数据结构、索引、检索和评价，重点在于实现和实验，每一章都有练习和对学生项目的建议。Wumpus（本书其中一位作者开发的一个多用户开源信息检索系统，可以在网上下载）提供了模型实现，可作为学生练习的一个基础。本书采用的模块化结构使教师可以将此书用于不同水平的研究生课程中，包括从数据库系统角度教授的课程、专注于理论的传统信息检索课程和关于Web检索基础的课程。

对信息检索的基础进行介绍之后，本书分别在相应的部分介绍了3个重要主题——索引、检索和评价。本书的最后一部分借用并扩展了前面部分的基本内容，考虑了以下具体应用：并行搜索引擎、Web搜索和XML检索。除了用于课堂教学，本书对计算机科学、计算机工程和软件工程的专业人员来说也具有很好的参考价值。

作者简介

Stefan Büttcher是Google公司资深网站可靠性工程师。Charles L. A. Clarke和Gordon V. Cormack是滑铁卢大学David R. Cheriton计算机科学学院的计算机科学教授。

上架指导：计算机/信息检索

ISBN 978-7-111-35990-6

9 787111 359906

定价：65.00元

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

计 算 机 科 学 丛 书

信息检索

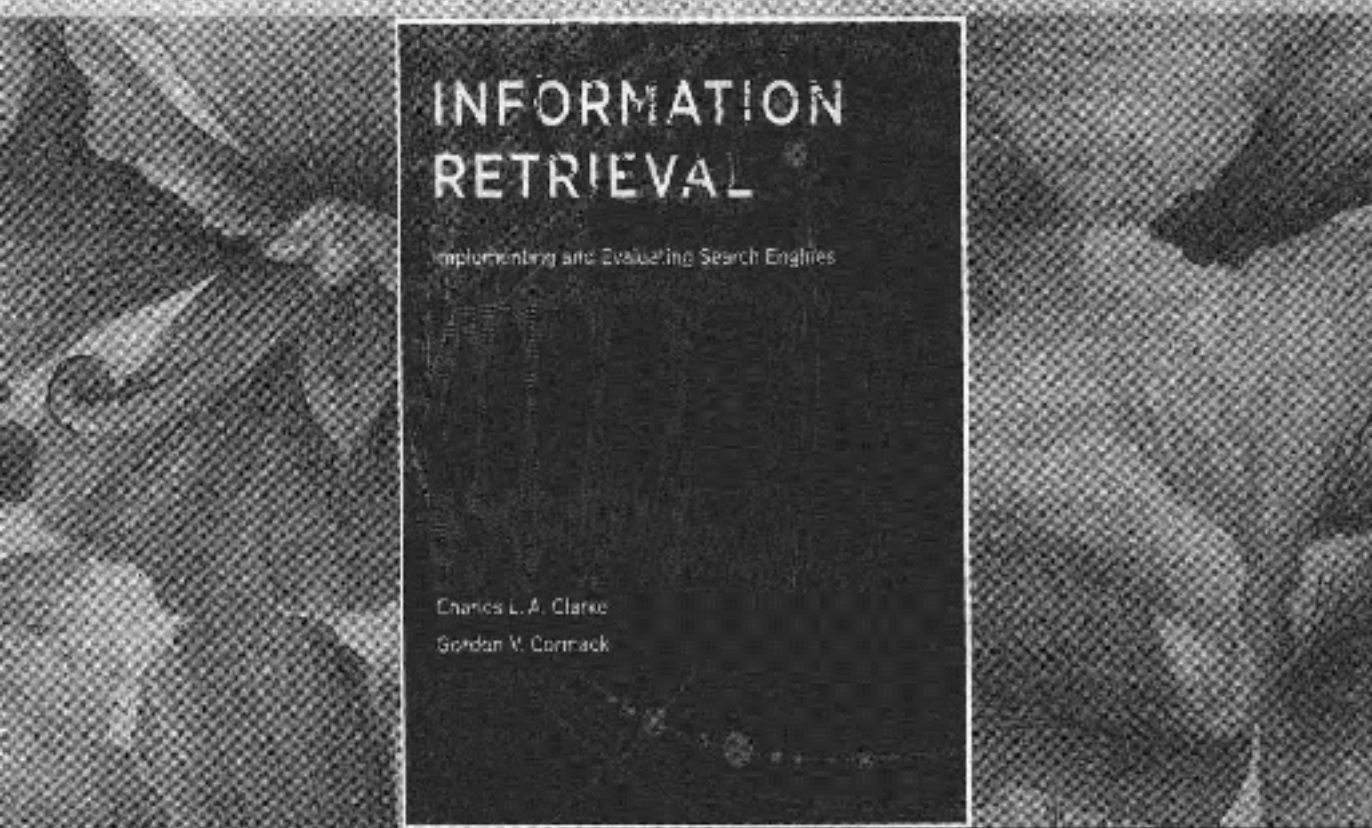
实现和评价搜索引擎

（美）Stefan Büttcher（加）Charles L. A. Clarke（加）Gordon V. Cormack 著

邵健 黄晋 等译

Information Retrieval

Implementing and Evaluating Search Engines



机械工业出版社
China Machine Press

本书从多个视角对信息检索技术进行了深入讲解，内容涵盖了信息检索系统的架构、基础技术、词条和词项、静态和动态倒排索引、查询处理、索引压缩技术、概率模型、语言模型、分类和过滤、融合和元学习、评价方法以及并行信息检索、Web检索和XML检索等具体应用。本书以模块化的方式进行组织，理论性强，体系完整，同时强调实践。作者以认真严谨的态度实现了书中绝大部分的主要方法，并详尽地描述了各种方法的适用环境以及取得的效果。

本书可作为高等院校信息管理与信息系统、计算机科学与技术、情报学、图书馆学以及电子商务等专业的高年级本科生和研究生的教材和参考书，对于从事信息检索与网络分析等实际工作的从业人员也具有较高的参考价值。

Stefan Büttcher, Charles L. A. Clarke, Gordon V. Cormack: Information Retrieval: Implementing and Evaluating Search Engines (ISBN 978-0-262-02651-2).

Original English language edition Copyright © 2010 by Massachusetts Institute of Technology.

Simplified Chinese Translation Copyright © 2012 by China Machine Press.

Simplified Chinese translation rights arranged with MIT Press through Bardon-Chinese Media Agency.

No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system, without permission, in writing, from the publisher.

All rights reserved..

本书中文简体字版由 MIT Press 通过 Bardon-Chinese Media Agency 授权机械工业出版社在中华人民共和国境内（不包括中国香港、澳门特别行政区及中国台湾地区）独家出版发行。未经出版者书面许可，不得以任何方式抄袭、复制或节录本书中的任何部分。

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2010-8044

图书在版编目（CIP）数据

信息检索：实现和评价搜索引擎/（美）布切尔（Büttcher, S.）等著；陈健等译. —北京：机械工业出版社，2011.12

（计算机科学丛书）

书名原文：Information Retrieval: Implementing and Evaluating Search Engines

ISBN 978-7-111-35990-6

Ⅰ. 信…　Ⅱ. ①布…　②陈…　Ⅲ. 情报检索　Ⅳ. G252.7

中国版本图书馆 CIP 数据核字（2011）第 195664 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码 100037）

责任编辑：朱秀英

北京诚信伟业印刷有限公司印刷

2012 年 1 月第 1 版第 1 次印刷

185mm×260mm · 26.75 印张

标准书号：ISBN 978-7-111-35990-6

定价：65.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991；88361066

购书热线：（010）68326294；88379649；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S.Tanenbaum，Bjarne Stroustrup，Brain W.Kernighan，Dennis Ritchie，Jim Gray，Afred V.Aho，John E.Hopcroft，Jeffrey D.Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：(010) 88379604
联系地址：北京市西城区百万庄南街 1 号
邮政编码：100037



华章教育

华章科技图书出版中心

译 者 序

Information Retrieval: Implementing and Evaluating Search Engines

由于手机、个人电脑、互联网等信息工具的快速发展和进化，个人可获取和管理的信息量呈爆发式增长，如何快速准确地找到所需的信息成为信息处理中的一个难题。信息检索技术是解决该问题的主要方法，其最初来源于图书内容的索引和检索，近些年来由于互联网的发展，以此为基础的搜索引擎技术使其受到了广泛的关注和研究。国内无论是高等院校相关专业方向的研究生，还是对搜索技术感兴趣的研究者和开发人员，都迫切需要一本全面专业的信息检索书籍。

国内引进了多本信息检索领域的书籍，本书是其中较新较有特色的一本。它以模块化的方式进行组织，从多个视角对信息检索技术进行了深入的解析，并补充了相关学科的基本知识，例如通用的符号数据压缩技术、统计分析、机器学习、数据库、Web 结构、XML 等等，使读者免去了查阅大量资料和其他书籍的麻烦。这本书理论性强，体系完整，同时也很强调实践。作者以认真严谨的态度对书中绝大部分的主要方法给出了实现细节和分析，并通过实验对比了这些方法，详尽地描述了各种方法的适用环境以及取得的效果，为信息检索在具体环境下的应用提供了很好的参考。在每一章最后的延伸阅读和参考文献部分，读者还可以了解到该章相关知识点的研究历史、发展和目前最新状况，也可据此对相关内容进行更深入的了解和研究。课后练习也经过了精心的设计，各章习题彼此关联、循序渐进，能够帮助读者更好地理解各章的知识点。

感谢原著作者无私地分享了他们在信息检索领域内的独特见解和研究成果。在过去几个月中，胡清兰、吴灿荣、李仕钊、黄锦捷、李蕾、黄蕉平、黄琎都参与了部分翻译、审校工作。感谢徐亚波老师及其学生给出的宝贵意见。当然，本书的翻译工作得以顺利完成，还要感谢机械工业出版社的王春华编辑和其他所有工作人员在各方面的支持和帮助。最后，对于给予我们无私帮助的那些人致以诚挚的谢意。

由于译者水平有限，书中疏漏在所难免，敬请读者批评指正。

陈健、黄晋

2011 年 6 月 29 日

学术巨匠齐聚一堂编撰了一部信息检索的优秀教材。Stefan Büttcher、Charles Clarke和Gordon Cormack以合计超过五十年的研究经验，组成了横跨三代的信息检索研究泰斗组合。Büttcher是Clarke的博士生，而Clarke是Cormack的博士生。他们三人都以对信息检索的深入洞察和建立实用搜索系统的热情而闻名，这种组合在一个充满世界级的研究专家的领域中是很少见的。

本书涵盖了搜索引擎的各个重要组成部分，从爬虫到索引到查询过程。大部分章节用于介绍索引、检索方法和评价的核心主题。重点放在实现和实验上，以让读者了解到信息检索系统的底层细节，包括索引压缩和索引更新策略，同时让读者理解在实际中哪一种方法效果更好。关于评价的两章提供了评价搜索引擎的方法论和统计学基础，使得读者能够知道：例如改变搜索引擎的排名公式是否对检索结果的质量有一个正面的影响。关于分类的一章介绍了对高级搜索操作非常有用的机器学习技术，例如如何将查询限制在某种特定语言书写的文档中，或者如何过滤搜索结果中的不良信息。关于并行信息检索和Web搜索的章节描述了从一个基本的信息检索系统变为一个涵盖数十亿文档并同时为成千上万的用户服务的大规模检索服务系统时所必须做出的改变。

通过引用数以百计的研究文献，作者对当今信息检索研究状况给出了指导性的概述，这个概述的高度远远超过了那些一般的综述。通过使用一个运行样例集和一个通用框架，他们具体描述了在每个环节中的重要方法——为什么这些方法行得通，它们是如何实现的，以及它们是如何工作的。为了写这本书，作者几乎实现和测试了每一个重要的方法，进行了数百次实验，并增加了对实验结果的阐述。每一章最后的练习题鼓励读者自己动手去建立系统并进行探索。

这本书是所有信息检索研究者和从业人员的必读教材！

Amit Singhal，Google Fellow

前　言

Information Retrieval: Implementing and Evaluating Search Engines

信息检索奠定了现代搜索引擎的基石。在这本教材中，我们针对计算机科学、计算机工程和软件工程的研究生以及专业人员介绍了信息检索。选择的主题引起了大部分读者的兴趣，涵盖了算法、数据结构、索引、检索和评价的核心主题，为读者今后的学习提供广博的基础。同时考虑 Web 搜索引擎、并行系统和 XML 检索在已有和新的应用场景的特性。

我们的目的是在理论与实践之间取得平衡，稍微偏向于实践，强调实现和实验。只要有可能，本书中的方法都通过实验进行了对比和验证。每一章都包含了练习和学生项目。本书其中一位作者开发的一个多用户开源信息检索系统 Wumpus，提供了模型实现，可作为学生练习的基础。可以通过 www.wumpus-search.org 获取 Wumpus。

本书组织

本书以模块化结构组织，可分为 5 个部分。第一部分提供了介绍性的材料。第二至第四部分，每部分专注于一个重要主题领域：索引、检索和评价。阅读完第一部分后，第二至第四部分都可以分别单独阅读。第五部分主要基于前面部分的内容来介绍具体的应用领域。

第一部分涵盖了信息检索的基础知识。第 1 章讨论基本概念，包括信息检索系统的架构、术语、文本特征、文档格式、词项分布、语言模型和测试集。第 2 章介绍 3 个重要主题（索引、检索和评价）的基础。这 3 个主题稍后在各自所属的部分（第二至第四部分）有详细介绍。这一章也为读者可以独立阅读每个主题或多或少地提供了基础。第一部分的最后一章，即第 3 章，继续介绍了在第 1 章中引入、在第 2 章中结束的部分主题。它涉及的问题与具体的自然（即人类）语言相关，特别是**分词**（tokenization）——为了进行索引和检索而将一个文档转化成一个词项序列的过程。一个信息检索系统必须能够处理由多种自然语言混合的文档，而这一章就是从这方面讨论几种主要语言的重要特性。

第二部分主要讨论倒排索引的创建、访问和维护。第 4 章讨论建立和访问**静态**（static）索引的算法，这种索引适用于不常变动的文档集，即当文档发生变动时，有足够的时间来重新从头建立索引。第 5 章讨论索引访问和查询过程，这一章介绍一种轻量级的方法来处理文档结构，并使用这种方法来支持布尔约束。第 6 章介绍索引压缩。第 7 章提出用于维护**动态**（dynamic）文档集的算法，也就是文档的更新相对于查询次数是频繁的，同时要求更新必须迅速。

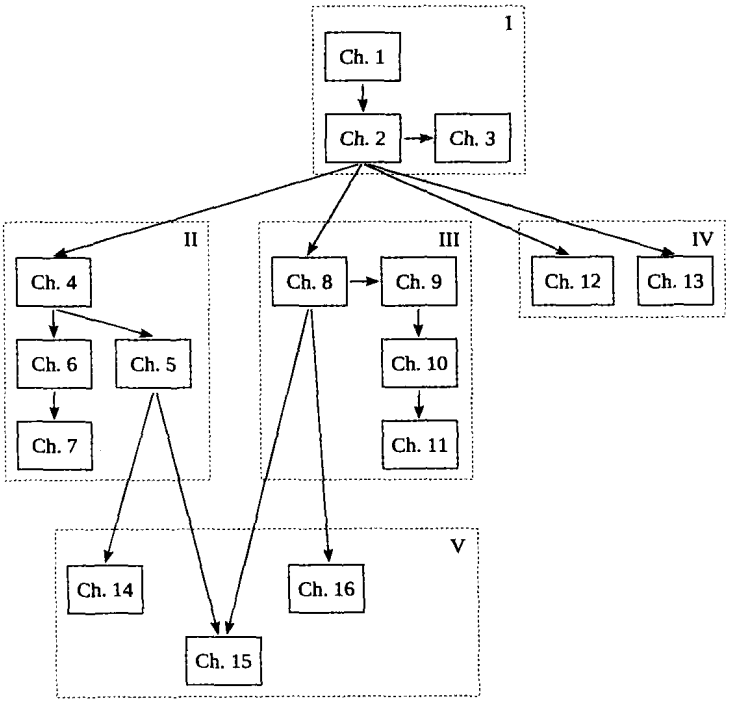
第三部分介绍了检索方法和算法。第 8 章和第 9 章介绍并比较两种基于文档内容的重要排名检索方法：概率模型和语言模型。通过使用文档结构、反馈和查询扩展，可考虑利用一些显式的相关信息来提高这些方法的有效性。我们讨论了每种方法的细节。第 10 章介绍用于文档分类和过滤的技术，包括用于分类的基本的机器学习算法。第 11 章介绍将证据和参数调整进行整合的技术，以及元学习算法及其在排名中的应用。

信息检索评价是第四部分的主题，用独立的章节分别介绍了有效性和效率。第 12 章给出了基本的有效性度量指标，探讨了用于评价有效性的统计基础，并讨论了一些在最近 10 年里提出的度量指标，它们已经超出了传统信息检索评价方法的范围。第 13 章介绍了从响应时间和吞吐量来评价信息检索系统性能的方法。

第五部分是全书的最后一部分，内容涉及一些具体的应用领域，借用并扩展了来自前四个部分的一些基本内容。第 14 章介绍了并行搜索引擎的架构和操作。第 15 章讨论了关于 Web 搜索引擎的一些主题，包括链接分析、抓取和重复检查。第 16 章介绍了 XML 文档集上的信息检索。

书中的每一章都包含了一个小节为深入阅读提供了参考文献，还提供了一组练习题。练习题一般偏向于考查和扩展相应章节介绍的概念。有些练习只需用铅笔和纸花上几分钟就能做好；有些则是需要大量编程的项目。这些参考文献和练习题同时也为我们提供了机会来学习一些在该章的正文部分没有涵盖的重要概念和主题。

下面的示意图展示了本书的各章和各部分之间的关系。箭头表示各章之间的依赖关系。本书的组织使得读者可以关注主题的不同方面。从数据库系统实现的观点来教授的课程可以包括第 1～2、4～7 和 13～14 章。专注于理论的传统信息检索课程可以包括第 1～3、8～12 和 16 章。关于 Web 检索基础的课程可以包括第 1～2、4～5、8 和 13～15 章。每一种涵盖的章节数约占全书的 1/2～2/3，可以在一个 3～4 个月的研究生课程中完成。



本书的组织。各章之间的箭头表示它们之间的依赖关系

背景

我们假设读者拥有计算机科学、计算机工程、软件工程或相关学科的本科相当的基本背景知识，包括：(1) 基本数据结构的概念，例如链表数据结构、B-树和哈希函数；(2) 算法和时间复杂度分析；(3) 操作系统、磁盘设备、内存管理和文件系统。另外，我们假设一些读者熟悉初等概率论和统计学，包括如随机变量、分布和概率群分布函数等概念。

致谢

我们的很多同事花费了大量的时间帮助我们审阅了与其专业领域相关的章节的草稿。我们在这里特别感谢 Eugene Agichtein，Alina Alt，Lauren Griffith，Don Metzler，Tor Myklebust，Fabrizio Silvestri，Mark Smucker，Torsten Suel，Andrew Trotman，Olga Vechtomova，William Webber 和 Justin Zobel 为我们提出了很多宝贵的意见。同时感谢匿名审稿人为我们提供了积极的意见和反馈。

有几个班的研究生起草了早期的一些材料。我们感谢他们的耐心和忍耐。4 个学生——Mohamad Hasan Ahmadi，John Akinyemi，Chandra Prakash Jethani 和 Andrew Kane——非常严谨地审阅了草稿，帮助我们找出和解决了很多问题。另外 3 个学生——Azin Ashkan，Maheedhar Kolla 和 Ian Mackinnon——志愿帮助我们在 2007 年秋季学期进行了一次课内评价，对第一部分中的很多练习有很大的贡献。Jack Wang 校对了第 3 章中关于 CJK 语言的材料。Kelly Itakura 提供了日文输入。

Web 站点

本书的作者维护了一个关于本书材料的 Web 站点，包括勘误以及引用文章的链接，参见 ir.uwaterloo.ca/book。

为了便于参考，以下列表总结了本书中常见的符号。其他必要的符号也在表中列出了。

C	文本集
d	文档
$E[X]$	随机变量 X 的期望值
$f_{t,d}$	词项 t 出现在文档 d 中的次数
l_{avg}	文档集中所有文档的平均长度
l_C	文档集C的大小，以词条数来衡量
l_d	文档 d 的长度，以词条数来衡量
l_t	t 的位置信息列表的长度（即出现次数）
$\mathcal{M}$	概率分布；通常是一个语言模型或一个压缩模型
N	文档集中的文档数
N_t	包含词项 t 的文档数
n_r	相关文档数
$n_{t,r}$	包含词项 t 的相关文档数
$\Pr[x]$	事件 x 的概率
$\Pr[x\|y]$	给定 y，事件 x 的条件概率
q	查询
q_t	词项 t 在查询 q 中出现的次数
t	词项
V	文本集的词汇表
$\vec{x}$	向量
$[\vec{x}]$	向量 $\vec{x}$ 的长度
$[\chi]$	集合 χ 的势（集合 χ 的大小——译者注）

目　录

Information Retrieval: Implementing and Evaluating Search Engines

出版者的话
译者序
序
前言
符号

第一部分　基础知识

第 1 章　绪论 …… 1
1.1　什么是信息检索 …… 1
1.1.1　Web 搜索 …… 1
1.1.2　其他搜索应用 …… 2
1.1.3　其他信息检索应用 …… 2
1.2　信息检索系统 …… 3
1.2.1　信息检索系统基础架构 …… 3
1.2.2　文档及其更新 …… 5
1.2.3　性能评价 …… 5
1.3　使用电子文本 …… 6
1.3.1　文本格式 …… 6
1.3.2　英文文本中的分词 …… 9
1.3.3　词项分布 …… 10
1.3.4　语言模型 …… 11
1.4　测试集 …… 16
1.5　开源信息检索系统 …… 18
1.5.1　Lucene …… 19
1.5.2　Indri …… 19
1.5.3　Wumpus …… 19
1.6　延伸阅读 …… 20
1.7　练习 …… 21
1.8　参考文献 …… 22
第 2 章　基础技术 …… 23
2.1　倒排索引 …… 23
2.1.1　延伸例子：词组查找 …… 24
2.1.2　实现倒排索引 …… 27
2.1.3　文档和其他元素 …… 31
2.2　检索与排名 …… 36
2.2.1　向量空间模型 …… 38
2.2.2　邻近度排名 …… 42
2.2.3　布尔检索 …… 44
2.3　评价 …… 46
2.3.1　查全率和查准率 …… 46
2.3.2　排名检索的有效性指标 …… 47
2.3.3　创建测试集 …… 51
2.3.4　效率指标 …… 52
2.4　总结 …… 53
2.5　延伸阅读 …… 54
2.6　练习 …… 55
2.7　参考文献 …… 56
第 3 章　词条与词项 …… 58
3.1　英语 …… 58
3.1.1　标点与大写 …… 59
3.1.2　词干提取 …… 60
3.1.3　停词 …… 62
3.2　字符 …… 63
3.3　字符 n-gram …… 64
3.4　欧洲语言 …… 65
3.5　CJK 语言 …… 66
3.6　延伸阅读 …… 67
3.7　练习 …… 68
3.8　参考文献 …… 69

第二部分　索　引

第 4 章　静态倒排索引 …… 71
4.1　索引的组成部分和索引的生命周期 …… 71
4.2　词典 …… 72
4.3　位置信息列表 …… 75
4.4　交错词典和位置信息列表 …… 78
4.5　索引的构建 …… 81
4.5.1　基于内存的索引构建法 …… 82
4.5.2　基于排序的索引构建法 …… 85
4.5.3　基于合并的索引构建法 …… 87
4.6　其他索引 …… 90

4.7 总结 …… 90
4.8 延伸阅读 …… 91
4.9 练习 …… 91
4.10 参考文献 …… 92
第5章 查询处理 …… 94
5.1 排名检索的查询处理 …… 94
5.1.1 document-at-a-time 查询处理 …… 95
5.1.2 term-at-a-time 查询处理 …… 99
5.1.3 预计算得分贡献 …… 103
5.1.4 影响力排序 …… 104
5.1.5 静态索引裁剪 …… 105
5.2 轻量级结构 …… 109
5.2.1 广义索引表 …… 110
5.2.2 操作符 …… 111
5.2.3 例子 …… 112
5.2.4 实现 …… 113
5.3 延伸阅读 …… 115
5.4 练习 …… 116
5.5 参考文献 …… 117
第6章 索引压缩 …… 119
6.1 通用数据压缩 …… 119
6.2 符号数据压缩 …… 120
6.2.1 建模和编码 …… 121
6.2.2 哈夫曼编码 …… 123
6.2.3 算术编码 …… 126
6.2.4 基于符号的文本压缩 …… 129
6.3 压缩位置信息列表 …… 130
6.3.1 无参数间距压缩 …… 131
6.3.2 参数间距压缩 …… 133
6.3.3 上下文感知的压缩方法 …… 137
6.3.4 高查询性能的索引压缩 …… 139
6.3.5 压缩效果 …… 142
6.3.6 解码性能 …… 145
6.3.7 文档重排 …… 146
6.4 压缩词典 …… 147
6.5 总结 …… 151
6.6 延伸阅读 …… 152
6.7 练习 …… 152
6.8 参考文献 …… 153
第7章 动态倒排索引 …… 155
7.1 批量更新 …… 155
7.2 增量式索引更新 …… 157
7.2.1 连续倒排列表 …… 158
7.2.2 非连续倒排列表 …… 163
7.3 文档删除 …… 165
7.3.1 无效列表 …… 165
7.3.2 垃圾回收 …… 166
7.4 文档修改 …… 170
7.5 讨论及延伸阅读 …… 171
7.6 练习 …… 172
7.7 参考文献 …… 172

第三部分 检索和排名

第8章 概率检索 …… 174
8.1 相关性建模 …… 174
8.2 二元独立模型 …… 176
8.3 Robertson/Spärck Jones 权重公式 …… 177
8.4 词频 …… 179
8.4.1 Bookstein 的双泊松模型 …… 180
8.4.2 双泊松模型的近似 …… 182
8.4.3 查询词频 …… 183
8.5 文档长度：BM25 …… 183
8.6 相关反馈 …… 184
8.6.1 词项选择 …… 185
8.6.2 伪相关反馈 …… 186
8.7 区域权重：BM25F …… 187
8.8 实验对比 …… 189
8.9 延伸阅读 …… 189
8.10 练习 …… 190
8.11 参考文献 …… 191
第9章 语言模型及其相关方法 …… 194
9.1 从文档中产生查询 …… 194
9.2 语言模型和平滑 …… 196
9.3 使用语言模型排名 …… 198
9.4 Kullback-Leibler 距离 …… 200
9.5 随机差异性 …… 202
9.5.1 一个随机模型 …… 203
9.5.2 精华性 …… 204
9.5.3 文档长度规范化 …… 204

9.6 段落检索及排名 …… 205
9.6.1 段落评分 …… 206
9.6.2 实现 …… 206
9.7 实验对比 …… 207
9.8 延伸阅读 …… 207
9.9 练习 …… 208
9.10 参考文献 …… 208
第 10 章 分类和过滤 …… 210
10.1 详细示例 …… 212
10.1.1 面向主题的批过滤 …… 212
10.1.2 在线过滤 …… 215
10.1.3 从历史样本中学习 …… 216
10.1.4 语言分类 …… 217
10.1.5 在线自适应垃圾邮件过滤系统 …… 220
10.1.6 二元分类的阈值选择 …… 223
10.2 分类 …… 225
10.2.1 比值和比值比 …… 226
10.2.2 构造分类器 …… 228
10.2.3 学习模型 …… 229
10.2.4 特征工程 …… 230
10.3 概率分类器 …… 231
10.3.1 概率估计 …… 231
10.3.2 联合概率估计 …… 235
10.3.3 实际考虑 …… 237
10.4 线性分类器 …… 239
10.4.1 感知器算法 …… 241
10.4.2 支持向量机 …… 241
10.5 基于相似度的分类器 …… 242
10.5.1 Rocchio 法 …… 242
10.5.2 基于记忆的方法 …… 243
10.6 广义线性模型 …… 243
10.7 信息理论模型 …… 246
10.7.1 模型比较 …… 246
10.7.2 序列压缩模型 …… 247
10.7.3 决策树与树桩 …… 249
10.8 实验对比 …… 251
10.8.1 面向主题的在线过滤器 … 251
10.8.2 在线自适应垃圾信息过滤 …… 253
10.9 延伸阅读 …… 254
10.10 练习 …… 255
10.11 参考文献 …… 256
第 11 章 融合和元学习 …… 258
11.1 搜索结果融合 …… 259
11.1.1 固定临界值合成 …… 260
11.1.2 排名和得分合成 …… 261
11.2 叠加自适应过滤器 …… 262
11.3 叠加批分类器 …… 263
11.3.1 holdout 验证 …… 264
11.3.2 交叉验证 …… 264
11.4 bagging …… 265
11.5 boosting …… 266
11.6 多类排名和分类 …… 267
11.6.1 文档得分与类别得分 …… 267
11.6.2 文档排名融合与类别排名融合 …… 268
11.6.3 多类方法 …… 269
11.7 学习排名 …… 272
11.7.1 什么是学习排名 …… 272
11.7.2 学习排名的方法 …… 273
11.7.3 优化什么 …… 273
11.7.4 分类的学习排名 …… 274
11.7.5 排名检索的学习 …… 274
11.7.6 LETOR 数据集 …… 275
11.8 延伸阅读 …… 276
11.9 练习 …… 277
11.10 参考文献 …… 277

第四部分 评 价

第 12 章 度量有效性 …… 279
12.1 传统的有效性指标 …… 279
12.1.1 查全率和查准率 …… 280
12.1.2 前 k 个文档的查准率 (P@k) …… 280
12.1.3 平均查准率 …… 281
12.1.4 排名倒数 …… 281
12.1.5 算术平均与几何平均 …… 281
12.1.6 用户满意度 …… 282
12.2 TREC …… 282
12.3 在评价中使用统计 …… 283
12.3.1 基础和术语 …… 284

12.3.2 置信区间 …… 286
12.3.3 比较评价 …… 292
12.3.4 被认为有害的假设检验 … 294
12.3.5 配对和未配对差值 …… 295
12.3.6 显著性检验 …… 296
12.3.7 统计检验的效度和检验力 …… 299
12.3.8 报告指标的查准率 …… 302
12.3.9 元分析 …… 303
12.4 最小化判定工作 …… 304
12.4.1 为判定选择合适的文档 … 305
12.4.2 对池进行抽样 …… 309
12.5 非传统的有效性指标 …… 311
12.5.1 分级相关性 …… 311
12.5.2 不完整判定和偏差判定 … 313
12.5.3 新颖性和多样性 …… 314
12.6 延伸阅读 …… 318
12.7 练习 …… 319
12.8 参考文献 …… 320
第 13 章 度量效率 …… 324
13.1 效率标准 …… 324
13.1.1 吞吐量和延迟 …… 325
13.1.2 汇总统计和用户满意度 … 327
13.2 排队论 …… 327
13.2.1 肯德尔符号 …… 328
13.2.2 $M/M/1$ 排队模型 …… 329
13.2.3 延迟量和平均利用率 …… 330
13.3 查询调度 …… 331
13.4 缓存 …… 332
13.4.1 三级缓存 …… 332
13.4.2 缓存策略 …… 334
13.4.3 预取搜索结果 …… 335
13.5 延伸阅读 …… 335
13.6 练习 …… 335
13.7 参考文献 …… 336

第五部分 应用和扩展

第 14 章 并行信息检索 …… 338
14.1 并行查询处理 …… 338
14.1.1 文档划分 …… 339
14.1.2 词项划分 …… 341
14.1.3 混合方案 …… 343
14.1.4 冗余和容错 …… 343
14.2 MapReduce …… 345
14.2.1 基本框架 …… 345
14.2.2 合并 …… 347
14.2.3 辅助关键字 …… 347
14.2.4 机器失效 …… 347
14.3 延伸阅读 …… 348
14.4 练习 …… 349
14.5 参考文献 …… 349
第 15 章 Web 搜索 …… 351
15.1 Web 的结构 …… 351
15.1.1 Web 图 …… 352
15.1.2 静态与动态网页 …… 353
15.1.3 暗网 …… 353
15.1.4 Web 的规模 …… 354
15.2 查询与用户 …… 355
15.2.1 用户意图 …… 355
15.2.2 点击曲线 …… 357
15.3 静态排名 …… 357
15.3.1 基本 PageRank …… 358
15.3.2 扩展的 PageRank …… 362
15.3.3 PageRank 的性质 …… 366
15.3.4 其他链接分析方法：HITS 和 SALSA …… 369
15.3.5 其他静态排名方法 …… 371
15.4 动态排名 …… 371
15.4.1 锚文本 …… 372
15.4.2 新颖性 …… 373
15.5 评价 Web 搜索 …… 373
15.5.1 指定页面发现 …… 374
15.5.2 用户隐式反馈 …… 375
15.6 Web 爬虫 …… 376
15.6.1 爬虫的组成 …… 377
15.6.2 抓取顺序 …… 380
15.6.3 重复与近似重复 …… 381
15.7 总结 …… 383
15.8 延伸阅读 …… 384
15.8.1 链接分析 …… 384
15.8.2 锚文本 …… 385
15.8.3 隐式反馈 …… 386

15.8.4 Web 爬虫 …… 386
15.9 练习 …… 386
15.10 参考文献 …… 387
第 16 章 XML 检索 …… 392
16.1 XML 的本质 …… 393
16.1.1 文档类型定义 …… 395
16.1.2 XML 模式 …… 396
16.2 路径、树和 FLWOR …… 396
16.2.1 XPath …… 396
16.2.2 NEXI …… 397
16.2.3 XQuery …… 398
16.3 索引和查询处理 …… 399
16.4 排名检索 …… 401
16.4.1 排名元素 …… 402
16.4.2 重叠元素 …… 403
16.4.3 可检索元素 …… 404
16.5 评价 …… 404
16.5.1 测试集 …… 404
16.5.2 有效性指标 …… 405
16.6 延伸阅读 …… 405
16.7 练习 …… 407
16.8 参考文献 …… 407

第六部分 附录

附录 A 计算机性能 …… 410

第一部分 基础知识

第1章
Information Retrieval: Implementing and Evaluating Search Engines

绪 论

1.1 什么是信息检索

信息检索（Information Retrieval，IR）被认为是对大规模电子文本和其他人类语言数据进行表示、搜索和处理的技术。信息检索系统和服务现在已经非常普遍了，成千上万的人每天都使用它们来方便地进行商务、教育和娱乐。Google、Bing等Web搜索引擎，是目前为止最普遍和大量使用信息检索服务的形式，提供获取最新技术信息、搜索人和组织、总结新闻和事件以及简化比较购物的途径。电子图书馆系统帮助医学界和学术界的研究人员了解他们研究领域内最新的期刊文章和会议报告。消费者使用本地搜索服务来找到提供所需产品和服务的零售商。在大型公司中，企业搜索系统作为电子邮件、备忘录、技术报告和其他业务文档的存储库，通过保存这些文档和提供相应的手段获得文档蕴涵的知识来提供企业记忆。桌面搜索系统则允许用户搜索他们的个人电子邮件、文档和文件。

1.1.1 Web搜索

对Web搜索引擎的一般用户而言，通常希望只要在一个文本框里输入一个简短的查询——几个简单的词，然后点击一下搜索按钮，马上就可以得到问题的精确答案。在这简单直观的界面后面是一组计算机集群，包括成千上万台协同工作的机器，用来产生最有可能满足查询中所包含信息的网页排名列表。这些机器要识别包含查询词的网页集合，计算每个网页的得分，消除重复和多余的页面，生成余下页面的摘要，最后将摘要和链接返回给用户以便浏览。

为了达到期望的亚秒级响应时间目标，Web搜索引擎结合多层缓存和复制机制、和用最常见的查询，并并发并行处理机制，使它们能够随着快速增长的网页和用户数量同步扩展。为了得到精确的查询结果，它们存储Web的一个“快照”（snapshot）。这个快照必须由运行在成千上万台机器构成的集群上的Web爬虫（crawler）不断采集和更新，定期下载每个页面的最新副本，周期也许是每周一次。包含快速变化信息的高品质网页，例如新闻服务，则可能每天或每小时都会被更新。

来看一个简单的例子。假设你身边有一台可以连接到Internet的计算机，请你用一分钟的时间打开一个浏览器并尝试在一个主流商业Web搜索引擎上输入“information retrieval”进行查询，搜索引擎通常在一秒钟之内就会响应。在一些时间来看看前10个结果。每一个结果都列出一个网页的URL，通常还提供了一个标题和一个从网页正文提取的简短的文字片段。总的来说，结果来自多个不同的网站，包括与主要教科书、期刊、会议和其他研究者

相关的网站。类似这样常见的**信息性**（informational）查询，Wikipedia[⊖]的文章可能出现在结果集中。前 10 个结果是否包含了不合适的内容？它们的顺序是否还可以再改进呢？让我们看一下紧接的后 10 个结果，看看其中的一个是否可以替代前 10 个中的一个。

现在，我们考虑数以百万计的包含了“information”和“retrieval”这两个词的网页集合。这个集合包含了很多与信息检索主题相关的网页，但这些网页不如排在最前面的 10 个结果那么一般化，例如学生主页和个别研究论文。另外，这个集合还包括很多恰好包含了这两个词但却与这个主题没有任何直接联系的网页。搜索引擎的排名算法基于一系列的特征，从这上百万个可能的网页中选择出排在最前面的网页，这些特征包括网页的内容和结构（例如标题）、网页之间的关系（例如它们之间的链接关系）以及整个 Web 的结构和内容。对于某些查询，用户的某些特征，例如她所在的地理位置或以往的搜索行为，也可能会起到一定的作用。权衡这些特征，以便以用户所期望的查询相关性来对网页进行排名，是**相关性排名**（relevance ranking）的一个例子。在不同的背景和要求下有效实现和评价相关性排名算法是信息检索的核心问题，也是本书的中心议题。

1.1.2 其他搜索应用

桌面和文件系统搜索是信息检索广泛应用的另一个实例。桌面搜索引擎提供对存储在本地硬盘，甚至内部网络上的其他硬盘内的文件进行搜索和浏览的手段。与 Web 搜索引擎相比，桌面搜索引擎系统需要对文件格式和创建时间有更多的了解。例如，用户可能希望只搜索他们的电子邮件，或只知道文件创建或下载的大致时间。因为文件可能变化很快，所以这些系统必须与操作系统的文件系统层直接交互，并且要设计得能够处理高负荷的更新。

介于桌面和普通的 Web 搜索之间，企业级信息检索系统为企业和其他组织提供文档管理和搜索服务。这些系统的实现细节千差万别。有一些基本上就是将 Web 搜索引擎运用到企业内部网，仅抓取组织内部可见的网页并提供类似标准 Web 搜索引擎那样的搜索界面。另一些则提供更一般化的文档和内容管理服务，可以方便地进行显式更新、版本控制和访问控制。在许多行业，这些系统能满足有关电子邮件和保持其他商务联系的监管要求。

数字图书馆和其他专业信息检索系统支持对高品质资料集的访问，这些资料集常常是专有的，可包括报纸文章、医学期刊、地图、图书等，由于版权的问题不能放在一个普通的网站里。考虑到这些资料的编辑质量和有限范围，通常可利用结构化的特征——作者、标题、日期和其他出版数据——来缩小查询需求和提高检索效率。此外，电子图书馆可能包含由**光学字符识别**（Optical Character Recognition，OCR）系统从印刷资料中提取的电子文本，由 OCR 输出带来的字符识别错误会给检索过程带来更复杂的情况。

1.1.3 其他信息检索应用

搜索是信息检索领域的中心任务，但信息检索还涵盖了存储、处理和检索人类语言数据等各种相互关联的问题：

- **文档路由、过滤和选择性传播**（routing，filtering，and selective dissemination）是典型信息检索过程的反过程。一个典型的搜索应用是在一组给定的文档集合上评价一个输入查询，而一个路由、过滤或传播系统则将最新创建或发现的文档与一组预先由用户提供的固定查询进行比较，最符合给定查询的文档被确定为用户最有可能感兴

⊖ en.wikipedia.org/wiki/Information_retrieval

趣的文档。例如一个新闻聚合器，就可能使用一个路由系统将当天的新闻分为“商业”、“政治”和“生活”类，或将按兴趣划分的头条新闻发送到特定的订阅者那里。一个电子邮件系统可以运用一个垃圾邮件过滤器来拦截不需要的消息。正如我们将要看到的那样，尽管这两个问题在具体应用和实现细节上不一样，但它们在本质上是一样的。

- **文本聚类**（clustering）和**分类**（categorization）系统根据共有属性将文档分组。聚类和分类之间的差异源于对系统提供的信息。我们提供**训练数据**（training data）给分类系统来说明不同的类别。例如将“商业”、“政治”和“生活”类的文章样例提交给分类系统，接下来分类系统就能将未标记的文章放进相应的类别中。与此相反，聚类系统不需要训练样本，它将文档按照它自己发现的模式分到不同的组别里。
- **摘要**（summarization）系统将文档缩减为一些能描述它们内容的关键段落、句子或词组。Web 搜索结果中的文本片段就是其中的一个例子。
- **信息提取**（information extraction）系统识别命名实体，例如场所和日期，并将这些信息合并为描述这些实体之间的关系的结构化记录——例如，从 Web 数据中创建书籍和它们的作者列表。
- **话题检测与跟踪**（topic detection and tracking）系统从新闻流和其他类似的信息源中识别事件，并跟踪相关的事件。
- **专家搜索**（expert search）系统找出组织里谁是某一特定领域的专家。
- **问答**（question answering）系统整合了多个来源的信息，并为特定问题提供简明的回答。它们往往合并和扩展了其他的信息检索技术，包括搜索、摘要和信息提取。
- **多媒体信息检索**（multimedia information retrieval）系统将相关的排名和其他信息检索技术扩展到图像、视频、音乐和语音中去。

很多信息检索问题涵盖了图书馆和信息科学领域，还有很多计算机科学中的其他主题领域，例如自然语言处理、数据库和机器学习。

在以上列出的主题中，分类和过滤技术已经被广泛使用，并且我们对这些领域进行了介绍。由于篇幅有限，我们无法对其他的主题进行深入。但是，它们都依赖于和扩展了本书介绍的基本技术。

1.2 信息检索系统

大部分信息检索系统都有一个共同的基础组织架构，这个基础组织架构能够根据具体的应用需求做调整以达到适用。本书讨论的大部分概念都基于这个基础架构来介绍。此外，与其他技术领域一样，信息检索也有自己的专业术语。当从某一特定技术观点出发时，词的解释往往与其普通的意思有很大的不同。为了避免混淆以及对书中后面部分的介绍提供一些背景知识，我们将简述该主题涉及的基本术语和技术名词。

1.2.1 信息检索系统基础架构

图 1-1 展示了信息检索系统的主要组成部分。在执行搜索之前，用户会有一个**信息需求**（information need），这种信息需求能够引发并驱动一次查询处理。我们有时把这个信息需求称为**主题**（topic），特别是当其以书面形式被提出来作为一个信息检索系统性能评价的测试集的一部分时。在信息需求的驱动下，用户构造一个**查询**（query）并提交给信息检索系统。一般来说，这个查询由多个**词项**（term）组成，对于网络搜索，大部分查询包含 2～3 个词项。这

里我们使用“词项”而不用“**词**”（word），主要原因是在应用中一个查询词项可能根本不是一个词。根据实际的信息需求，查询词项可能会是日期、数字、音符或者**词组**（phrase），甚至还可以是通配符或者其他的部分匹配符。例如，词项“inform＊”可以匹配任何以“inform”为前缀的词（如“inform”、“informs”、“informal”、“informant”和“informative”等）。

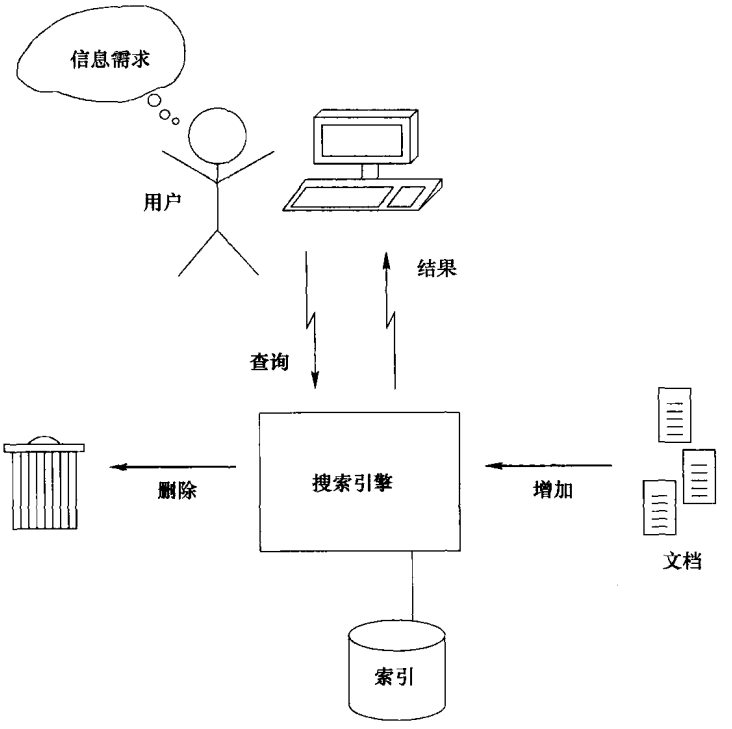


图 1-1 信息检索系统的组成

虽然用户往往提交的是简短的关键字查询，但是信息检索系统通常也支持丰富的查询语法，通常包含复杂的布尔表达式和模式匹配操作符（第 5 章）。这些查询手段可以用于将搜索限制在某一特定网站上，或是在一些字段，如作者和标题上指定一些约束条件，又或者是使用其他的**过滤器**（filter）将搜索范围锁定到部分文档集上。用户界面是用户和信息检索系统之间的接口，使用查询界面提供的丰富查询手段，可以简化查询的创建过程。

用户查询由**搜索引擎**（search engine）处理，搜索引擎可运行在用户本地机器上，也可运行在远程的大型机器集群上，或者介于二者之间的任何地方。搜索引擎的一个主要任务就是为**文档集**（document collection）维护和处理**倒排索引**（inverted index）。这个索引便是引擎完成搜索和相关性排名的主要数据结构。作为一个基本功能，倒排索引给出了词项及它在文档集中出现位置的一个映射。由于倒排列表与文档集本身具有相同规模，所以必须确保索引访问和更新操作是高效执行的。

为了支持相关性排名算法，搜索引擎还维护与索引相关的文档统计信息，比如包含每个词项的文档数和每个文档的长度。此外，为了能把有意义的结果返回给用户，搜索引擎通常还要能访问到文档原始内容。

利用倒排索引、文档集统计信息和其他一些数据，搜索引擎接受用户查询，处理这些查询，最后返回一个有序结果列表。为了完成相关性排名，搜索引擎为每个文档计算一个**得分**

(score)，有时也称为**检索状态值**（Retrieval Status Value，RSV）。按得分将文档排序后，搜索引擎对结果列表作进一步处理，比如删除重复或多余的结果。例如，Web 搜索引擎有可能仅仅从一个主机或域名中返回一个或两个结果，而删除了其他来源的相同页面。如何根据用户的查询为文档评分是这个领域内最基本的问题之一。

1.2.2 文档及其更新

本书中，我们用“**文档**”（document）这个通用术语来代指那些能够返回给用户作为搜索结果的任何独立的单元。实际上，这个特殊文档可能是电子邮件、网页、新闻文章，甚至视频。一个大对象中预先定义好的组成部分也可能作为单独的搜索结果返回给用户，比如一本书的某些页或某些段落，我们称这些组成部分为**元素**（element）。当任意的文本段落、视频片段或者类似的资料从大对象中作为搜索结果返回给用户时，我们称这些为**片段**（snippet）。

对于大部分应用来说，更新模型相对简单。文档总是作为一个整体被添加或删除。一旦一个文档被添加到搜索引擎，它的内容就不会被更新。这种更新模型足以满足大多数的信息检索应用。例如，当 Web 爬虫发现一个网页被修改了，就通过删除旧网页和增加新网页来完成更新。即便是在文件系统搜索环境中，一个文件的某个部分被进行了任意的修改后，当用户保存修改时，大部分词处理程序或是其他应用都会通过重写整个文件来处理文本数据。一个特例是电子邮件应用，通常是将新到来的邮件追加到收件箱底端。因为收件箱通常比较大且增长迅速（所以不可能每次稍有改动就重写整个文件夹——译者注），因此支持追加操作就显得非常重要了。

1.2.3 性能评价

一般从两个基本的方面来衡量信息检索系统的性能：**效率**（efficiency）和**有效性**（effectiveness）。效率的评价可以通过时间（如每次查询的执行时间）和空间（如每个文档所占的字节数）来衡量。效率最直观的反映就是**响应时间**（response time），通常也称为**延迟**（latency），即用户提交一个查询到获得结果所经历的时间。当需要支持大量用户的并发操作时，查询的**吞吐量**（throughput），用每秒钟能处理的查询数来衡量，就成为影响系统性能的重要因素。对于一个通用的 Web 搜索引擎，所需的吞吐量可能每秒钟远远超出数以万计的查询。效率也要考虑存储空间，一般通过存储索引和其他数据结构所需的磁盘空间或者内存大小来衡量。此外，当有上千台机器为产生一个搜索结果同时协同工作时，它们的耗电量和碳排放量也是一个重要的考虑因素。

查询结果是否有效完全取决于用户判断，所以衡量有效性比衡量效率要困难得多。现在评价有效性的关键思想是利用**相关性**（relevance）的概念：如果一个文档内容能够（完全或部分）满足给定查询的信息需求，就认为该文档与此查询相关。为了决定文档的相关性，一个评价员（assessor）审阅一个文档/主题对，并给出一个相关性得分。这个得分有可能是**二值**（binary）的（“相关”或“不相关”），或是**分级**（graded）的（如，“太好了”、“很好”、“好”、“不错”、“还行”、“不相关”、“太糟了”）。

相关性排名的基本目标通常可以通过**概率排名原则**（Probability Ranking Principle，PRP）体现，具体描述如下：

> 如果一个信息检索系统对查询的响应是按文档集中文档的相关性概率递减排名的，那么对用户来说，系统的整体有效性就达到最大。

这个假设在信息检索领域是得到公认的，并且构成了标准信息检索系统评价方案的基

础。但是，它忽略了相关性的重要性，在实际中，相关性是一定要考虑的。特别是，相关性的基本概念还要扩展到返回文档的大小和范围。文档的**特异度**（specificity）反映了文档内容与信息需求关联的密切程度。特异度高的文档应主要由与信息需求相关的材料组成。在一个特异度很低的文档中，其大部分材料都与查询主题无关。文档的**详尽性**（exhaustivity）反映了其内容覆盖了多少与需求相关的信息。一个详尽性高的文档覆盖了所有与需求相关的信息，而详尽性低的文档可能只覆盖了有限几个方面。特异度和详尽性是两个独立的指标。一个大文档可能有很高的覆盖率，但是由于包含了太多无关的内容致使其特异度很低。

当相关性通过一个已排名文档列表来展现的时候，人们就开始关注**新颖性**（novelty）了。当用户浏览过排名靠前的文档并了解了相关内容后，她的信息需求可能会变化。尽管有些文档排名相当靠前（例如第二），但如果这个文档只包含了一点点或根本没有包含新颖信息时，这个文档就与已改变的信息需求毫不相关了。

1.3 使用电子文本

以电子文本形式存在的人类语言数据，是信息检索最原始的素材。构建信息检索系统必须同时理解电子文本格式以及不同文本的编码特征。

1.3.1 文本格式

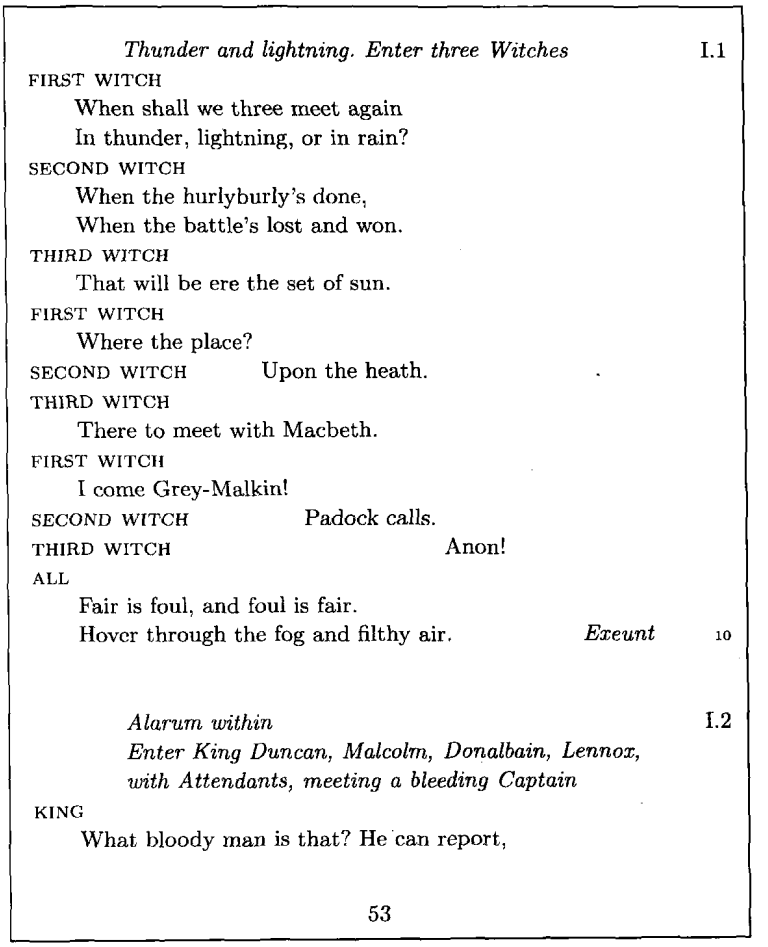
威廉·莎士比亚（William Shakespeare）的著作集提供了大篇幅英文文本现成的例子，网上有许多可以免费下载的电子版本。莎士比亚一生的著作包括 37 部戏剧，一百多首十四行诗和诗歌。图 1-2 是戏剧《Macbeth》剧本第一幕的开头部分。

这幅图看起来就像是剧本打印在一张纸上。从信息检索系统的角度来看，当这一页剧本以电子文本的形式存在，并最终需要在系统中为其建立索引时，需要考虑两个方面。首先是该页的**内容**（content），也就是从这一页中按正常顺序读出来的一串词，如“Thunder and lightning. Enter three Witches First Witch When shall we...”。其次是该页的**结构**（structure），也就是行和页中的间隔、演员台词的标记、舞台指导、幕次和场次，甚至页码。

电子文本的内容和结构能够以各种词处理程序和桌面排版系统支持的种种文档格式进行编码。这些格式包括 Microsoft Word、HTML、XML、XHTML、LaTeX、MIF、RTF、PDF、PostScript、SGML 等。在某些譬如文件系统检索这样的环境中，e-mail 格式，甚至是程序源代码格式都可以看成是文档格式。尽管这些格式的细节内容已经超出了本书范围，但是能基本了解它们对索引和检索的影响仍然是非常重要的。

我们对两种格式特别感兴趣。第一种是 **HTML**（HyperText Markup Language，超文本标记语言），它是网页的基本形式。特别值得注意的是，这种格式内在地支持超链接，这使得网页之间的关系可以被明确地表示出来，并且 Web 搜索系统可以对这些关系进行利用。锚文本往往伴随着一个超链接，部分描述了链接网页的内容。

第二种格式是 **XML**（eXtensible Markup Language，可扩展标记语言），从严格意义上来说，它不是一种文档格式，而是一种定义文档格式的**元语言**（metalanguage）。后面我们会详细讨论 XML（第 16 章），现在先直接使用它。XML 具有易用的特性，用它来构造易读编码的理由是不言自明的。因此，这本书后面的所有例子中，特别是在讨论文档结构的时候，都是采用 XML 格式。HTML 和 XML 一样，都是从 20 世纪 80 年代出现的**标准通用标记语言**（Standard Generalized Markup Language，SGML）发展过来的，并且它们标记文档结构的方法类似。



Thunder and lightning. Enter three Witches I.1

FIRST WITCH
When shall we three meet again
In thunder, lightning, or in rain?
SECOND WITCH
When the hurlyburly's done,
When the battle's lost and won.
THIRD WITCH
That will be ere the set of sun.
FIRST WITCH
Where the place?
SECOND WITCH Upon the heath.
THIRD WITCH
There to meet with Macbeth.
FIRST WITCH
I come Grey-Malkin!
SECOND WITCH Padock calls.
THIRD WITCH Anon!
ALL
Fair is foul, and foul is fair.
Hover through the fog and filthy air. *Exeunt* 10

Alarum within I.2
Enter King Duncan, Malcolm, Donalbain, Lennox,
with Attendants, meeting a bleeding Captain
KING
What bloody man is that? He can report,

53

图 1-2 莎士比亚文集《Macbeth》第一幕开头部分

图 1-3 是莎士比亚文集某个版本的《Macbeth》开头部分的 XML 编码形式。这个编码是由 Jon Bosak 完成的，他是 XML 标准的主要创始人之一。形如〈name〉的标签代表一个结构化元素的开始，形如〈/name〉的标签代表一个结构化元素的结束。标签也可以采用其他形式，并且包括了用来定义包含文本特性的属性，这些细节留待第 16 章介绍。本书中大量的例子中用到的标签都是类似于这幅图中简单的标签。

如果按 XML 的传统意思，这种编码只能表示出文档的**逻辑结构**（logical structure）——演员、台词、（戏剧）幕、（戏剧）场次和剧本行。而文档的**物理结构**（physical structure）——字体、加粗、布局、页面间隔等，直到这一页被显示的时候，目标显示媒体才需要知道这些细节。

遗憾的是，许多文档格式对逻辑结构和物理结构并未保持一致的区分。此外，某些格式妨碍了我们确定文档的内容，或是作为检索请求的结果，我们不能返回比整个文档少的任何内容（换言之，我们不能获得文档摘要——译者注）。这种格式大多是所谓的**二进制格式**（binary format），因为它们包含了内部指针和其他复杂的组织结构，也不能视为字符流来处理。

举个例子来说，PostScript 格式的文档内容是使用编程语言 Forth 的某一版本来编码的。一个 PostScript 文档实质上就是一个可执行的程序，用于打印或显示的时候调整文档。尽管使用编程语言来编码文档能够获得最大的灵活性，但是却使得建立索引所需的文档内容

难以获取。PDF 格式是 PostScript 的进化版，虽然没有包含一种完整的编程语言，但是仍保持许多旧格式的灵活性和复杂性。PostScript 和 PDF 最初是由 Adobe 公司开发的，但是现在都是开放的标准，可以使用许多包括开源工具在内的第三方工具来创建和操作这种格式的文档。

```
<STAGEDIR>Thunder and lightning. Enter three Witches</STAGEDIR>
<SPEECH>
<SPEAKER>First Witch</SPEAKER>
<LINE>When shall we three meet again</LINE>
<LINE>In thunder, lightning, or in rain?</LINE>
</SPEECH>
<SPEECH>
<SPEAKER>Second Witch</SPEAKER>
<LINE>When the hurlyburly's done,</LINE>
<LINE>When the battle's lost and won.</LINE>
</SPEECH>
<SPEECH>
<SPEAKER>Third Witch</SPEAKER>
<LINE>That will be ere the set of sun.</LINE>
</SPEECH>
<SPEECH>
<SPEAKER>First Witch</SPEAKER>
<LINE>Where the place?</LINE>
</SPEECH>
<SPEECH>
<SPEAKER>Second Witch</SPEAKER>
<LINE>Upon the heath.</LINE>
</SPEECH>
<SPEECH>
  <SPEAKER>Third Witch</SPEAKER>
  <LINE>There to meet with Macbeth.</LINE>
</SPEECH>
```

图 1-3 莎士比亚文集《Macbeth》部分剧本的 XML 编码

许多转换工具可以从 PostScript 和 PDF 文档中抽取内容，对它们进行压缩并形成索引。每一个这样的转换工具都使用启发式方法分析文档并推测文档内容。尽管使用标准工具能为文档生成很好的输出，但当面对一些不寻常来源的文档时，效果可能并不好。一种极端的情况就是，有可能要将整个文档放入内部缓冲区，然后使用模式匹配算法去识别字符和词。

另外，即便为 PostScript 和 PDF 文档确立最简单的逻辑结构，这也将是一个很大的挑战。甚至一个文档的标题也要通过字体、大小、文档中的位置以及其他物理结构上的特征才能确立。对于 PostScript 文档，因为执行单个页面的程序将会影响到后面的页面，所以抽取单个页面的内容也可能会有问题。PostScript 的这个特点使得信息检索系统无法从一个很大的 PostScript 文档中仅返回几个页面作为检索结果，例如，无法从一个很长的技术手册中返回一个简单章节作为结果。

其他文档格式是专用的，也就是说，它们只能与某个软件制造商的产品关联起来。这些专有格式包括微软的“doc”格式。直到最近，由于微软 Office 在市场上的主导地位，这种格式才广泛用于文档交换和协作。尽管这种专有格式的技术说明通常是可以获取的，但是它们很复杂，而且可能随着版本更新不断修改，而这一切完全是由制造商决定的。微软以及其他制造商现在都把注意力转向到基于 XML 的格式（例如 OpenDocument 格式或者微软的 OOXML），制造商的这种转变也许能降低对文档建立索引的复杂度。

实际上，HTML 可能也会有二进制格式的那些问题。许多 HTML 页面中包含 JavaScript脚本或者 Flash 编程语言。这些脚本可能会重写整个网页的内容，在屏幕上显示任意内容。对于那些由脚本产生内容的网页，实际上 Web 爬虫和搜索引擎很难抽取和索引其有意义的内容。

1.3.2 英文文本中的分词

无论文档的格式如何，为了构建用于检索查询请求的倒排索引，每一个文档都必须先转化为一个**词条**（token）序列。对于英文文档，一个词条通常对应于一个字母数字组成的字符（A～Z 和 0～9）序列，也可能是编码结构信息，例如 XML 中的标签，或者文档的其他特征信息。分词是索引过程中关键的一步，因为它有效限制了系统能够处理的查询类别。

作为分词之前的预处理步骤，二进制格式的文档要被转换为**原始文本**（raw text）——字符流。这个转换过程一般是将字体信息以及其他更低层的物理格式信息丢弃，但通过向原始文本中重新插入适当的标签来保留更高层的逻辑格式。高层次的格式可能包括标题、段落边界以及其他类似的元素。XML 或 HTML 文档不需要这个预处理步骤，因为它们已经包含了建立索引所需的词条，从而大大减少了处理成本。从本质上来讲，这些格式的文档本来就是原始文本。

对于英文文档，原始文本中的字符被编码为 7 位 ASCII 值。然而对于其他语言，ASCII 编码方式变得不适用。对于这些语言，每个字符不能被编码为单独的一个字节，因此必须使用其他的编码方式。由 Unicode 表示的 UTF-8 是一种解决这些语言编码问题的流行的方法（见 3.2 节）。对于生活中使用到的大多数语言，甚至许多已经灭绝的语言，例如腓尼基语（Phoenician）和苏美尔（Sumerian）的楔形文字，UTF-8 也能够为这些语言的每个字符提供 1～4 字节的编码。UTF-8 能够向后兼容 ASCII，所以 ASCII 编码的文本顺其自然地也是 UTF-8 的文本。

对图 1-3 的 XML 文档进行分词，每一个 XML 标签和连续字符串都被视为词条。为了简化匹配过程，我们将标签外的大写字母转换为小写，“FIRST”、“first”和“First”将被同等对待。分词的结果如图 1-4 所示。每一个词条旁边用一个数字来指示这个词在莎士比亚 37 部戏剧集中出现的位置，这个位置从戏剧《Antony and Cleopatra》开始计算，从 1 开始，以《The Winter's Tale》结束，位置为1 271 504。这个简单的分词方法在余下的第 1 章和第 2 章部分对于我们要介绍的内容已经足够了，在必要的时候我们就假定使用这样的分词方法。在第 3 章中，将会重新检视对英文以及其他语言进行分词的方法。

文本集中不同词条或者**符号**（symbol）的集合被称为**词汇表**（vocabulary），用$\mathcal{V}$来表示。莎士比亚文集的词汇表中共有$|\mathcal{V}|=22\ 987$ 个不同的符号。

$$\mathcal{V}=\{\texttt{a},\texttt{aaron},\texttt{abaissiez},\cdots,\texttt{zounds},\texttt{zwaggered},\cdots,\texttt{<PLAY>},\cdots,\texttt{<SPEAKER>},\cdots,\texttt{</PLAY>},\cdots\}$$

在莎士比亚文集中，我们将任何连续的字母数字串视为一个词，则词汇表共包含 22 943 个词和 44 个标签。本书将词汇表中的符号视为“词项”，因为它们构成了与查询中词项匹配的基础。另外，我们将一个词条视为是一个词项的一次“出现”。虽然这种用法有助于加强文档词条与查询词项的联系，但它可能会掩盖符号和词条之间的关键区别。符号是抽象的，词条是抽象的实例化。在哲学上，这种差别称为**“型值之差”**（type-token distinction）。在面向对象编程中，就如同类和对象之间的区别。

745396	745397	745398	745399	745400
<STAGEDIR>	thunder	and	lightning	enter
745401	745402	745403	745404	745405
three	witches	</STAGEDIR>	<SPEECH>	<SPEAKER>
745406	745407	745408	745409	745410
first	witch	</SPEAKER>	<LINE>	when
745411	745412	745413	745414	745415
shall	we	three	meet	again
745416	745417	745418	745419	745420
</LINE>	<LINE>	in	thunder	lightning
745421	745422	745423	745424	745425
or	in	rain	</LINE>	</SPEECH>
745426	745427	745428	745429	745430
<SPEECH>	<SPEAKER>	second	witch	</SPEAKER>
		...		

图 1-4 莎士比亚文集的《Macbeth》部分剧本的分词结果

表 1-1 莎士比亚文集 Boask 版 XML 格式文档中出现最频繁的 20 个词

排名	频率	词条	排名	频率	词条
1	107 833	〈LINE〉	11	17 523	of
2	107 833	〈/LINE〉	12	14 914	a
3	31 081	〈SPEAKER〉	13	14 088	you
4	31 081	〈/SPEAKER〉	14	12 287	my
5	31 028	〈SPEECH〉	15	11 192	that
6	31 028	〈/SPEECH〉	16	11 106	in
7	28 317	the	17	9344	is
8	26 022	and	18	8506	not
9	22 639	i	19	7799	it
10	19 898	to	20	7753	me

1.3.3 词项分布

表 1-1 中列举了莎士比亚戏剧集 XML 格式文档中出现最频繁的 20 个词。在这些词项中，排在前六的是标记剧本行、演员和台词的标签。至于一般的英文文本，“the”是最频繁出现的词，其次是很多的代词、介词和其他功能词。超过三分之一（8336）的词项，例如，“abaissiez”和“zwaggered”，仅出现过一次。

标签出现的频率是由文档集本身的结构约束所决定的。每一个开始标签〈name〉都会对应一个结束标签〈/name〉。每一部戏剧恰好有一个戏剧名。每一句台词至少对应一个演员，但当一组角色齐声发言的时候，也有少数台词是对应多个演员的。平均而言，每一行台词平均有 8 或 9 个词。

虽然在不同的文档集中，标签的类型和相对频率不一样，但是英语文本中词的相对频率通常遵循一个统一的模式。不考虑标签，图 1-5 描绘出了莎士比亚文集中词项的频率排名。x 轴和 y 轴都是对数坐标。图中的点基本上落在斜率为 −1 的直线上，除了最频繁的词和最不频繁的词落在该直线下方。在这个图中，对应“the”的点出现在最左上角，对应“zwaggered”这样只出现一次的词的点都出现在最右下角。

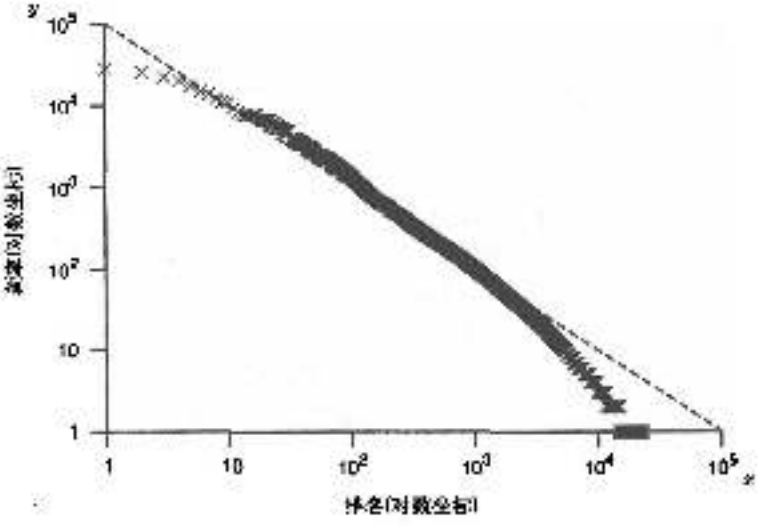


图 1-5 莎士比亚文集词频的排名顺序。虚线符合 $\alpha=1$ 的齐夫定律

这条反映词频与排名关系的直线被称为齐夫定律（Zipf's law），语言学家 George Zipf 在 20 世纪 30 年代发现了这个关系，并且用它来模拟在社会科学一些领域内数据的相对频率（Zipf，1949）。数学上，这种关系可以表示为：

$$\log(\text{frequency}) = C - \alpha \cdot \log(\text{rank}) \tag{1-1}$$

或等价于

$$f_i \propto \frac{1}{i^{\alpha}} \tag{1-2}$$

其中，f_i 是指第 i 个频繁词的词频。对于不同的英文文本，α 取值各有不同，但通常都逼近 1。齐夫定律也适用于其他自然语言以及其他类型数据的相对词频。在第 4 章中，这个定律促成了使用某种数据结构来索引自然语言文本的想法。在第 13 章中，我们使用齐夫定律来对搜索引擎查询的相对频率建模。

1.3.4 语言模型

莎士比亚戏剧集的 Bosak 版本 XML 中，除去标签共有 912 052 个词条（但包含了一些原版没有的序言）。如果随机地从里面选出一个词条，选到“the”的概率是 28 317/912 052≈3.1%。选到“zwaggered”的概率为 1/912 052≈0.000 11%。现在，假设发现了一个先前不为人知的莎士比亚戏剧，能否从已经知道的剧本来猜测这个剧本的内容？在进行这些猜测之前，我们先重新定义不包含标签的词汇表 V。因为一个没被发现过的莎士比亚的剧本不太可能已经进行了 XML 编码（至少在第一次发现的时候是这样）。

关于未知文本的内容预测，可以使用一种特殊类型的概率分布方法，称为**语言模型**（language model）。最简单的语言模型就是一个在词汇表中的符号的固定的概率分布 $M(\sigma)$：

$$\sum_{\sigma \in \mathcal{V}} \mathcal{M}(\sigma) = 1 \tag{1-3}$$

一个语言模型常基于已知文本来建立。例如，可做如下定义：

$$\mathcal{M}(\sigma) = \frac{\text{frequency}(\sigma)}{\sum_{\sigma' \in \mathcal{V}} \text{frequency}(\sigma')} \tag{1-4}$$

其中，frequency(σ) 是词项 σ 在莎士比亚文集中出现的次数。我们有，$\mathcal{M}$(“the”)≈3.1%，$\mathcal{M}$(“zwaggered”) ≈ 0.000 11%。

如果随机地从莎士比亚已知的戏剧中抽一个词条，抽到词项 σ 的可能为$\mathcal{M}(\sigma)$。基于这个认识，从这个未知的戏剧中随机抽取一个词条，我们也假设抽到 σ 的可能为$\mathcal{M}(\sigma)$如果开始阅读这个未知的戏剧，我们可以使用语言模型来预测文本中的下一个词项，即下一个词项是 σ 的可能为$\mathcal{M}(\sigma)$。在这个简单的语言模型中，我们分开考虑每个词项。该语言模型对第一个词项、下一个词项、下下个词项都使用相同的独立预测，依此类推。基于此语言模型，以下六个词项 “to be or not to be” 出现的概率是：

2.18%×0.76%×0.27%×0.93%×2.18%×0.76%=0.000 000 000 069%

公式 (1-4) 被称为是这类简单语言模型的**最大似然估计** (Maximum Likelihood Estimate, MLE)。一般来说，给定一个数据集，最大似然是估算一个概率分布中未知参数的标准方法。这里，一个参数对应一个词项——代表了在未知文本中接下来会出现这个词项的概率。可以粗略地说，最大似然估计是使得数据集最像（给定数据集——译者注）的参数值。在这种情况下，莎士比亚文集就是必需的数据集。公式 (1-4) 就是使未知文本最像是莎士比亚作品的参数概率。假定未知文本与当前文本相似，则最大似然模型提供了一个预测文本内容的好起点。

语言模型可以用于量化一个新的文本片段与已知文本集的相似程度。假定现在分别有表征莎士比亚作品和表征英国剧作家 John Webster 作品的语言模型。对于一个先前未知的刚发现的新戏剧，专家们争论谁可能是它的作者。假定上述两个语言模型能够捕捉两位作家的显著特点，例如他们各自更喜欢的词汇，那么就可以应用这两个语言模型来计算这个新戏剧的属向概率。概率高的模型就指示了这个新文本的作者。

然而，语言模型并非一定得使用最大似然估计。词项构成的词汇表上的任何概率分布都可视为是一个语言模型，例如，考虑以下概率分布：

$\mathcal{M}$(“to”) = 0.40　$\mathcal{M}$(“be”) = 0.30　$\mathcal{M}$(“or”) = 0.20　$\mathcal{M}$(“not”) = 0.10

基于这个分布，以下六个词 “to be or not to be” 出现的概率是：

0.40×0.30×0.20×0.10×0.40×0.30=0.029%

同理，基于这个模型，以下六个词 “the lady doth protest too much” 出现的概率为 0。

实际上，未知文本中可能含有未知的词。兼顾考虑到这些未知的词，可通过在词汇表中增加一项 “UNKNOWN” 来代表 “**词汇表外**” (out-of-vocabulary) 的词项。

$$\mathcal{V}' = \mathcal{V} \cup \{\text{UNKNOWN}\} \tag{1-5}$$

相应的扩展语言模型$\mathcal{M}'(\sigma)$将为 “UNKNOWN” 赋予一个正的概率值：

$$\mathcal{M}'(\text{UNKNOWN}) = \beta \tag{1-6}$$

其中，$0 \leqslant \beta \leqslant 1$。$\beta$ 值代表的是在$\mathcal{M}$模型中下一个词项不出现在当前文档集中的概率。对于其他的词项，作如下定义：

$$\mathcal{M}'(\sigma) = \mathcal{M}(\sigma) \cdot (1 - \beta) \tag{1-7}$$

其中，$M(\sigma)$是最大似然语言模型。β值取决于现有文档特征。例如，β可能是已知文本中某个词项出现概率的一半：

$$\beta = 0.5 \cdot \frac{1}{\sum_{\sigma' \in V} \text{frequency}(\sigma')} \tag{1-8}$$

幸运的是，词汇表外的词项通常不会成为信息检索系统的一个问题，因为文集完整的词汇表是在建立索引的时候确定的。

索引和文档压缩（第6章）是语言模型应用的另外的两个重要领域。当使用到压缩算法，词汇表中的词项通常是单个字符或字节，而不是整个词的形式。语言模型当被转换为压缩形式后，通常也称为**压缩模型**（compression model），这个术语在文献中很常见。其实，压缩模型只是语言模型的特殊化。在第10章中，压缩模型将被运用于检测垃圾邮件以及其他一些过滤问题。

语言模型可以用来生成文本以及预测未知的文本。例如，基于概率分布$M(\sigma)$可以随机地产生一些词项序列，从而生成一个“随机莎士比亚作品”：

strong die hat circumstance in one eyes odious love to our the wrong wailful would all sir you to babies a in in of er immediate slew let on see worthy all timon nourish both my how antonio silius my live words our my ford scape

1. 高阶模型

上述随机文本读起来不像是莎士比亚的作品，倒更像是不出名的作家写的。由于每个词项都是单独产生的，因此生成“our”之后生成“the”的概率仍然是3.1%。但是在实际的英文文章中，所有格形容词“our”后面几乎都是跟一个常见名词。尽管词组“our the”包含的是两个频繁出现的词，但是这样的词组很少出现在英文文章中，莎士比亚文集中也从未出现过。

高阶语言模型结合具体的上下文背景。**一阶**（first-order）语言模型由条件概率组成，后一个符号出现的概率由前一个符号来决定，例如：

$$M_1(\sigma_2 | \sigma_1) = \frac{\text{frequency}(\sigma_1\sigma_2)}{\sum_{\sigma' \in V} \text{frequency}(\sigma_1\sigma')} \tag{1-9}$$

词项的一阶语言模型等价于使用相同技术估计的**二元**（bigrams）零阶模型（例如MLE）：

$$M_1(\sigma_2 | \sigma_1) = \frac{M_0(\sigma_1\sigma_2)}{\sum_{\sigma' \in V} M_0(\sigma_1\sigma')} \tag{1-10}$$

更一般地，每个n阶语言模型都可以用一个$(n+1)$元零阶模型来表示：

$$M_n(\sigma_{n+1} | \sigma_1 \ldots \sigma_n) = \frac{M_0(\sigma_1 \ldots \sigma_{n+1})}{\sum_{\sigma' \in V} M_0(\sigma_1 \ldots \sigma_n \sigma')} \tag{1-11}$$

举个例子，考虑莎士比亚文集中的词组“first witch”。这个词组共出现23次，而“first”共出现1349次。根据最大似然二元模型，可以得到以下概率：

$$M_0(\text{“first witch”}) = \frac{23}{912\,051} \approx 0.002\ 5\%$$

（注意分母是912 051而不是912 052，因为二元数比词条数要少一个）。在一阶模型中对应的概率为：

$$M_1(\text{“witch”} \mid \text{“first”}) = \frac{23}{1349} \approx 1.7\%$$

使用公式（1-9）和（1-10），不考虑词条数和二元数的区别，“our the”的最大似然估计

值为：

$$\mathcal{M}_0(\text{“our the”})=\mathcal{M}_0(\text{“our”})\cdot\mathcal{M}_1(\text{“the”}\mid\text{“our”})=0\%$$

正如我们期望的那样，这个概率值为 0，因为这个词组从未在文章中出现。遗憾的是，这个模型对于很多没有出现在莎士比亚文集中的、但合理的二元词组，也会赋予 0 概率，例如“fourth witch”。由于在莎士比亚文集中“fourth”出现了 55 次，“witch”出现了 92 次，可以想象在未知戏剧中也有可能包含这个二元词组。此外，我们也应为例如“our the”这样的二元词组赋予一个比较小的正概率，以适应那些不常见的用法，包括古老的拼写和重音。举个例子，《The Merry Wives of Windsor》中包含了大量显然毫无意义的二元词组“a the”，这是法国医生 Caius 的台词“If dere be one or two, I shall make-a the turd.”。一旦看到了更多的上下文，词组的意义就变得清晰了。

2. 平滑

解决上述问题的一个方法是使用对应的零阶模型$\mathcal{M}_0$来**平滑**（smooth）一阶模型$\mathcal{M}_1$。平滑处理之后的模型$\mathcal{M}_1'$是$\mathcal{M}_0$和$\mathcal{M}_1$的线性组合：

$$\mathcal{M}_1'(\sigma_2\mid\sigma_1)=\gamma\cdot\mathcal{M}_1(\sigma_2\mid\sigma_1)+(1-\gamma)\cdot\mathcal{M}_0(\sigma_2)\tag{1-12}$$

同样地，

$$\mathcal{M}_0'(\sigma_1\sigma_2)=\gamma\cdot\mathcal{M}_0(\sigma_1\sigma_2)+(1-\gamma)\cdot\mathcal{M}_0(\sigma_1)\cdot\mathcal{M}_0(\sigma_2)\tag{1-13}$$

其中，γ是平滑参数（$0\leqslant\gamma\leqslant1$）。例如，使用最大似然估计且设$\gamma=0.5$，有：

$$\begin{aligned}\mathcal{M}_0'(\text{“first witch”})&=\gamma\cdot\mathcal{M}_0(\text{“first witch”})+(1-\gamma)\cdot\mathcal{M}_0(\text{“first”})\cdot\mathcal{M}_0(\text{“witch”})\\&=0.5\cdot\frac{23}{912\ 051}+0.5\cdot\frac{1349}{912\ 052}\cdot\frac{92}{912\ 052}\approx0.001\,3\%\end{aligned}$$

和

$$\begin{aligned}\mathcal{M}_0'(\text{“fourth witch”})&=\gamma\cdot\mathcal{M}_0(\text{“fourth witch”})+(1-\gamma)\cdot\mathcal{M}_0(\text{“fourth”})\cdot\mathcal{M}_0(\text{“witch”})\\&=0.5\cdot\frac{0}{912\ 051}+0.5\cdot\frac{55}{912\ 052}\cdot\frac{92}{912\ 052}\approx0.000\,000\,30\%\end{aligned}$$

一阶模型可以用零阶模型来平滑化，二阶模型可以用一阶模型来平滑，依次类推。显然，对于零阶模型，这样的方法不适用。然而，我们可以采用与“词汇表外的词项”相同的方法（公式（1-5））。一般地（也比较常见），对于小型文集S，可以使用从大型文集L中建立的零阶模型$\mathcal{M}_{S,0}$来进行平滑：

$$\mathcal{M}_{S,0}'=\gamma\cdot\mathcal{M}_{S,0}+(1-\gamma)\cdot\mathcal{M}_{L,0}\tag{1-14}$$

其中，L是任意的（但大型）的英文文集。

3. 马尔可夫模型

图 1-6 描述了另一种描述词项分布的重要方法，**马尔可夫模型**（Markov model）。马尔可夫模型本质上是一个具有转移概率的增强型有限自动机。当用于表达语言模型时，每一个转移都由一个词项和一个概率标识。通过这些转移关系，可以预测词项或产生词项。例如从状态 1 出发，沿着状态序列 1→2→3→4→1→2→3，可以得到“to be or not to be”这个字串，并得到对应的概率：

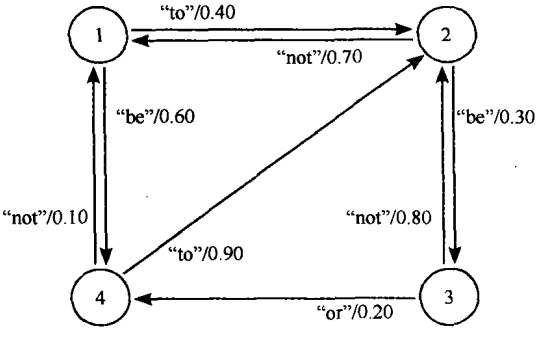


图 1-6 马尔可夫模型

$$0.40\times0.30\times0.20\times0.10\times0.40\times0.30 = 0.029\%$$

没有转移（例如状态1和状态4之间）相当于转移概率为0。因为这样的转移不会发生，因此无需将其与某个词项关联。为简单起见，假定两个状态之间最多只存在一个转移。这种假定不会丢掉任何信息。任何一个在相同的状态对中存在多个转移的马尔可夫模型，都会有一个在两个状态中不存在多个转移的马尔可夫模型与之对应（见练习1.7）。

由马尔可夫模型预测的概率依赖于初始状态。如果从状态4开始，产生“to be or not to be”的概率为：

$$0.90\times0.30\times0.20\times0.10\times0.40\times0.30=0.065\%$$

第10章会介绍到用于过滤的文档压缩模型的基础就是马尔可夫模型。

n 个状态的马尔可夫模型可用 $n\times n$ 的**转移矩阵**（transition matrix）M 来表示，其中 $M[i][j]$ 是从状态 i 转移到状态 j 的概率。图1-6的马尔可夫模型的转移矩阵如下：

$$M=\begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \\ 0.70 & 0.00 & 0.30 & 0.00 \\ 0.00 & 0.80 & 0.00 & 0.20 \\ 0.10 & 0.90 & 0.00 & 0.00 \end{pmatrix} \tag{1-15}$$

注意到矩阵中所有的值都位于[0,1]范围内，M 中每一行和为1。具有这种性质的 $n\times n$ 矩阵被称为**随机矩阵**（stochastic matrix）。

给定一个转移矩阵，通过将这个转移矩阵乘以一个表征当前状态的**状态向量**（state vector）可以得出这个转移的结果。例如，可以用向量（1000）来表示初始状态1。这步之后，我们有：

$$\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \\ 0.70 & 0.00 & 0.30 & 0.00 \\ 0.00 & 0.80 & 0.00 & 0.20 \\ 0.10 & 0.90 & 0.00 & 0.00 \end{pmatrix}=\begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \end{pmatrix}$$

也就是说，上一步后，正如期望的那样，进入状态2的概率为0.40，进入状态4的概率为0.60。再乘一次，可以得到：

$$\begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \end{pmatrix}\begin{pmatrix} 0.00 & 0.40 & 0.00 & 0.60 \\ 0.70 & 0.00 & 0.30 & 0.00 \\ 0.00 & 0.80 & 0.00 & 0.20 \\ 0.10 & 0.90 & 0.00 & 0.00 \end{pmatrix}=\begin{pmatrix} 0.34 & 0.54 & 0.12 & 0.00 \end{pmatrix}$$

这个计算结果告诉我们，两步之后进入状态2的概率为0.54。一般地，状态向量是各元素之和为1的任何的 n 维向量。将状态向量乘以转移矩阵 k 次，即可得出 k 步之后，进入每一个状态的概率。

一个随机矩阵和一个初始状态向量就构成了**马尔可夫链**（Markov chain）。第15章中会介绍到马尔可夫链可用于Web链接分析算法的表达。马尔可夫链——作为马尔可夫模型的延伸——是以俄罗斯统计学家安德烈·马尔可夫（Andrey Markov，1856—1922）的名字命名的，他提出并证明了很多马尔可夫链的性质。

1.4 测试集

尽管戏剧《Macbeth》以及莎士比亚的其他戏剧都为解释简单的信息检索概念提供了绝好的范例，但为了评价的目的，研究者人员开发了更多的测试集。许多测试集都是作为**TREC**[⊖]（Text REtrieval Conference，文本检索会议）的一部分被创建的，这是由**美国标准与技术研究所**（National Institute of Standards and Technology，NIST）从1991年以来每年都会进行的一系列实验评价结果。TREC为研究者们提供了一个论坛，供他们在广泛的问题上测试自己的信息检索系统。例如，来自学术界、工业界和政府的超过100个组织参加了TREC 2007。

通常，TREC实验由6～7个不同的**专题**（track）组成，每个专题关注信息检索的一个不同领域。近几年，TREC专题涵盖了企业搜索、基因信息检索、法律发现、垃圾邮件过滤和博客搜索。每个专题又分成若干任务，测试这个领域的不同方面。例如，TREC 2007的企业搜索专题就包括了在给定主题上的电子邮件讨论搜索任务和专家发现任务。每个专题通常会持续三年或更长时间。

TREC为信息检索研究人员至少做出了两个重要贡献。第一，它向研究者提供了一个平台，供他们在相同问题、相同数据上展示和讨论他们的工作，并进行不同系统间的直接比较，谁优谁劣一目了然。作为结果，往往在引进TREC的某个专题之后，都能够看到显著的进步和性能提升。第二，TREC致力于创建**可重用**（reusable）的测试集，不仅可供会议参与者验证他们的改进工作，未参加会议者也可用此来评价自己的工作。另外，TREC自成立以来，就为全世界许多类似的实验性能评估提供了灵感。其中包括了欧洲的INEX侧重XML检索，CLEF侧重跨语言信息检索，日本的NTCIR侧重亚洲语言信息检索，还有印度的FIRE等。

TREC任务

基本搜索任务——即系统根据先前未知的主题从静态文本集中返回一个排名列表——在TREC专业的词汇中，这个任务被称做**“特别检索”**（特定）任务。除了一组文档集，特定任务的测试集还包括主题集（查询从中产生），以及一个相关性判断集（分别称为“qrel files”或“qrels”），指示了文档是否和每个主题相关。纵观TREC历史，特定任务已经成为了不同研究议题的一部分，比如Web检索或基因信息检索。尽管议题不同，特定任务的组织和实施在不同的专题里基本是相似的，这点从TREC最开始出现的时候就没太大变化。

较早的TREC（2000年前）中，特定任务的文档集来源于发给TREC参会者的5张CD中的160万个文档。包括了报纸和从出版物中选出来的通讯等，来源如**华尔街日报**（Wall Street Journal）、**洛杉矶时报**（LA Times），还有由美国联邦政府公布的文档如**联邦登记册**（Federal Register）、**美国国会记录**（Congressional Record）等。这些文档大多是由专业人士撰写和编辑，报告事实或描述事件。

图1-7是TREC第五张CD一篇文档的简短摘录。这篇文档是1990年5月19日洛杉矶时报上的一篇新闻报道。为了便于TREC实验，这篇文章被编码为XML格式。尽管各TREC集标签的规则不一样，但是在这里，所有的TREC文档都采用相同的标签来定义文档之间的边界和文档标识符。每一个TREC文档由标签〈DOC〉…〈/DOC〉包围，〈DOCNO〉…

⊖ trec.nist.gov

〈/DOCNO〉标记一个唯一的文档。在进行相关性判定时，在 qrels 文档集中还需要用到这个标识符。这个规定简化了索引过程，并且使得集合能够比较容易地合并。很多信息检索系统为满足这个规定的文档提供了直接的便利。

```
<DOC>
<DOCNO> LA051990-0141 </DOCNO>
<HEADLINE> COUNCIL VOTES TO EDUCATE DOG OWNERS </HEADLINE>
<P>
The City Council stepped carefully around enforcement of a dog-curbing
ordinance this week, vetoing the use of police to enforce the law.
</P>
...
</DOC>
```

图 1-7 TREC第五张 CD 中的文档例子（LA051990-0141）

新的 TREC 的特定任务所需的文档集通常都从 Web 中得到。到 2009 年为止，其中最大的是 426 GB 的 GOV2 集，包含了由 2004 年初开始从美国政府的 gov 域名站点上抓取的2500万个网页。网络爬虫从 gov 域名中尽可能多地抓取网页，因此这个集合可视为这段时间内该域名的合理快照。GOV2 包含了各种长度和格式的文档，如 PDF 格式的长技术报告，甚至只包含一些链接的 HTML 网页等。从 TREC 2004 直至 2006 结束之前，GOV2 共构建了大小为 TB 级的专题。同时也为 TREC 2007 和 2008 百万查询专题提供了文档集。

虽然 GOV2 集合大大超过了以往任何的 TREC 集，但对比由商业 Web 搜索引擎能够处理的文集而言，就不只小一两个数量级了。TREC 2009 引入了一个有一亿多个 Web 网页的集合，即 ClueWeb09，给信息检索研究者提供了一个与商业 Web 搜索规模相当的工作平台。⊖

每年，NIST 会为一个专题中的特定任务创建 50 个新的主题。参与者在没有下载完这些主题之前是不能运行系统的。下载完这些主题之后，参与者从中创建查询，并在这个文集上运行这些查询，然后把排名后的结果返回给 NIST 以供评价。

图 1-8 显示了一个 TREC 1999 创建的典型的 TREC 特定任务的主题。像大多数 TREC 主题一样，它由三个部分构成，每一部分以若干格式描述了潜在的信息需求。**标题**（title）字段是设计成作为关键字查询，类似于提交给搜索引擎的查询一样。**描述**（description）字段以完整句子或问题的形式提供了比较长的主题要求描述。它也有可能作为一个查询，特别是在那些将自然语言处理技术作为检索的一部分的研究系统中。**叙述**（narrative）字段，它可能是很长的一段，作为其他两个字段的补充，提供一些指定相关文档特性的额外信息。叙述字段主要在人工评价的时候会用到，用来确定检索出的文档是否与主题相关。

这本书的大多数检索实验都在 4 个 TREC 测试集上进行，基于两个文档集，一个较小，一个较大。较小的文档集包含了上述 TREC CD 中第四、第五个 CD 中的文档，不包括**美国国会记录**（Congressional Record）中的文档。这个文档集涵盖了**金融时报**（Financial Times）、美国**联邦公报**（Federal Register）、美国对外广播情报处、**洛杉矶时报**（LA Times）中的文档。这个文档集，称为 TREC45，用于 TREC 1998 和 1999 主要的特定任务。

⊖ boston.lti.cs.cmu.edu/Data/clueweb09

```
<top>

<num> Number: 426
<title> law enforcement, dogs

<desc> Description:
Provide information on the use of dogs worldwide for
law enforcement purposes.

<narr> Narrative:
Relevant items include specific information on the
use of dogs during an operation. Training of dogs
and their handlers are also relevant.

</top>
```

图 1-8 TREC 中第 426 个主题

1998 年和 1999 年，基于这个文档集，NIST 分别创建了 50 个用于相关性判断的主题。1998 年的主题编号为 351～400，1999 年的主题编号为 401～450。因此，基于 TREC45 这个文档集，我们有两个测试集，分别为 TREC45 1998 和 TREC45 1999。虽然我们的实验过程跟 TREC 中所对应的实验过程（在这里我们不详述）存在一些细微差别，但在我们在这些文集上的实验结果可以和 TREC 1998 和 1999 公布的结果相媲美。

我们的实验中较大的那个文档集是上述的 GOV2 文集。我们将这个集合与 2004 年（主题编号 701～750）和 2005 年（主题编号 751～800）TREC TB 级专题中的主题和判断一起，构成了 GOV2 2004 和 GOV2 2005 文集。在这些文集上的实验结果报告可与 TREC 2004 和 2005 公布的 TB 级专题的结果相媲美。

表 1-2 书中大部分实验所用的测试集

文档集	文档数	大小（GB）	年份	主题
TREC45	500 000	2	1998	351～400
			1999	401～450
GOV2	25 200 000	426	2004	701～750
			2005	751～800

表 1-2 概括了我们的四个测试集。其中 TREC45 集可从 NIST 标准参考数据产品网页上的专题数据库 22 和 23 得到。[㊀]GOV2 集由格拉斯哥大学发布。[㊁]这些文集上的主题和查询集可从 TREC 数据库中获得。[㊂]

1.5 开源信息检索系统

现在有很多开源信息检索系统可供你使用，来完成本书的练习，并开始进行自己的信息检索实验。在 Wikipedia[㊃]中总是可以找到开源信息检索系统的（非详尽）清单。

由于可用系统的清单太长，我们不能全部都详尽地介绍。在这里，我们只对三个特定的系统进行简要的概述，选这三个系统是因为它们的知名度、对信息检索研究的影响力或它们

㊀ www.nist.gov/srd
㊁ ir.dcs.gla.ac.uk/test_collections
㊂ trec.nist.gov
㊃ en.wikipedia.org/wiki/List_of_search_engines

与本书内容密切相关。这三个系统都可以从 Web 上下载，根据它们各自的许可证都可以免费使用。

1.5.1 Lucene

Lucene 是一个用 Java 语言实现的索引和检索系统，同时为其他编程语言提供了开放的接口。这个项目在 1997 年由 Doug Cutting 开始。从那以后，它由一个单开发者的成果发展壮大成现在多个国家的数百个开发者共同参与的全球项目。这个项目目前由 Apache Foundation[㊀]主持。Lucene 是迄今为止最成功的开源搜索引擎。运行它的最大的系统如 Wikipedia，所有输入到 Wikipedia 搜索框中的查询都由 Lucene 处理。使用 Lucene 索引和搜索功能的项目列表可在 Lucene 的“PowerdBy”页面[㊁]中找到。

以模块化和可扩展性闻名的 Lucene 允许开发者定义他们自己的索引以及检索规则和公式。在底层，Lucene 的检索框架基于字段（field）这一概念：每一个文档都是字段的集合，比如文档标题、内容、URL 等。这使得它易于指定结构化搜索请求，并对于文档的不同部分可以给予不同的权重。

由于其大受欢迎，有很多书和教程可以帮助你快速学习 Lucene。你只需在喜欢的 Web 搜索引擎中试一下查询“Lucene tutorial”。

1.5.2 Indri

Indri[㊂]是用 C++语言实现的学术信息检索系统。它由马萨诸塞大学的研究者开发，是 Lemur 项目[㊃]的一部分，Lemur 项目是马萨诸塞大学和卡耐基梅隆大学的共同研究成果。

Indri 因它的高检索效率而著名，总能在 TREC 搜索引擎排行榜中的前几位找到它。它的检索模型是第 9 章介绍的多种语言模型方法的结合。与 Lucene 一样，Indri 可以处理文档的多个字段，如标题、内容、锚文本，这对于 Web 搜索是非常重要的（第 15 章）。通过伪相关反馈机制，Indri 支持自动查询扩展，自动查询扩展是一种在初始查询请求中增加相关查询词的技术，这种技术的实现是基于初始查询结果内容的（见 8.6 节）。Indri 还支持独立于查询的文档评分，例如给搜索结果排名时，可偏向比较新的文集（见 9.1 节和 15.3 节）。

1.5.3 Wumpus

Wumpus[㊄]是用 C++语言实现的学术搜索引擎，由滑铁卢大学开发。不像多数其他的搜索引擎，Wumpus 没有“文档”这个概念，在建立索引的时候也不知道每一个文档的开始和结束。相反，文档集的每个部分都有可能作为一个检索单元，这主要取决于查询中指定的结构化搜索条件。这对于某些搜索任务来讲具有相当的吸引力，在这些搜索中，理想的搜索结果不是一个完整的文档而是文档的一节、一段，或者几个顺序段落。

Wumpus 支持很多不同的检索方法，包括第 2 章的近似排名函数、第 8 章的 BM25 算法和第 9 章讨论到的语言模型和随机性差异的方法。此外，它还可以进行实时索引更新（例如，添加/删除文档到/从索引）并且为多用户的安全限制提供支持，因为当系统有多个用户

㊀ lucene.apache.org
㊁ wiki.apache.org/lucene-java/PoweredBy
㊂ www.lemurproject.org/indri/
㊃ www.lemurproject.org
㊄ www.wumpus-search.org

时，每个用户只能搜索索引中的一部分，这种安全限制就很有用了。

除非有明确说明，本书中所展示的性能图都使用 Wumpus 得到。

1.6 延伸阅读

本书并不是信息检索这个主题的第一本书。之前的书都对信息检索进行了一般介绍，有几本需要提及。Salton (1968) 和 van Rijsbergen (1979) 撰写的经典书籍继续介绍了这个领域的基本问题。Grossman 和 Frieder (2004) 提出的重要主题的处理方法仍然在使用。Witten 等人 (1999) 提供了一些本书中没有涉及的相关主题上的背景资料，包括文本和图像压缩。

近几年，有几个很好的介绍性的教材。Croft 等人 (2010) 的教科书主要是面向本科学生来介绍这个领域。Baeza-Yates 和 Ribeiro-Neto (2010) 提供了这个领域上的一个较全面的调查报告，不同的专家就各自专长的领域负责不同的章节。Manning 等人 (2008) 提供了另外一个可读的调查报告。

特定主题上的调查文章会定期地出现在信息检索基础与趋势 (Foundations and Trends in Information Retrieval) 系列杂志上，数据库系统百科全书 (Encyclopedia of Database Systems) (Özsu and Liu, 2009) 也收录了很多与信息检索相关主题的介绍性文章。Hearst (2009) 给出了用于信息检索应用的用户接口设计简介。自然语言处理领域，特别是统计自然语言处理的子领域，跟信息检索领域密切相关。Manning and Schütze (1999) 提供了该领域的详细介绍。

在该领域顶级的研究会议是 ACM 举办的国际信息检索大会 (SIGIR)，现在已经举办四十多年了。其他顶级的研究会议和研讨会包括 ACM 信息知识和组织会议 (ACM Conference on Information and Knowledge Management, CIKM)，数字图书馆联合会议 (Joint Conference on Digital Libraries, JCDL)，欧洲信息检索会议 (European Conference on Information Retrieval, ECIR)，ACM 网络检索与数据挖掘国际会议 (ACM International Conference on Web Search and Data Mining, WSDM)，字符串处理及信息检索研讨会 (String Processing and Information Retrieval, SPIRE)，信息检索会议 (Text REtrieval Conference, TREC)。重要的信息检索研究成果也会经常出现在相关领域的顶级出版物中，比如说万维网会议 (World Wide Web Conference, WWW)，神经信息处理系统年会 (Annual Conference on Neural Information Processing Systems, NIPS)，人工智能会议 (Conference on Artificial Intelligence, AAAI) 和知识发现和数据挖掘会议 (Knowledge Discovery and Data Mining Conference, KDD)。最好的信息检索刊物是 ACM 信息系统会刊 (ACM Transactions on Information Systems)。其他较好的刊物包括信息检索 (Information Retrieval) 和之前的信息处理与管理 (Information Processing & Management)。

我们能得到很多致力于学习和使用 XML 的书籍和网站。一个重要的资源是万维网联盟 (World Wide Web Consortium, W3C)[⊖] 的 XML 主页，万维网联盟是负责定义和维护 XML 技术规范的组织。除了大量的参考资料，这个网站上还有链接指向介绍性教程和指南。Jon Bosak 的个人主页[⊜] 包含很多 XML 的早期发展和应用相关的文章和其他信息，包括莎士比亚的戏剧。

⊖ www.w3.org/XML/

⊜ www.ibiblio.org/bosak

1.7 练习

练习 1.1 记录下来你将要提交的十个查询（如果你的搜索引擎支持，你可以查看网页浏览记录）。注意与之前提交的查询相比有哪些是完善的地方，例如通过增加或删除一些查询词来减小或增大搜索范围。然后选择三个商业搜索引擎，包括你最常用的那个，然后在这三个引擎上重新做这些查询。检查每个引擎返回结果中的前 5 个结果。用－10 到＋10 对每个结果进行评分，其中＋10 表示这个结果很好，－10 表示这是一个误导结果（垃圾）。计算每个搜索引擎在所有十个查询和结果上所得到的平均分。你最常用的那个是否得到了最高分？你认为这个练习中的结果是否真实反映了搜索引擎的相对质量？对这个实验提出三个可能的改进方法。

练习 1.2 下载并安装一个开源信息检索系统，如 1.5 节列出的那些。从你的 E-mail 或其他资料里创建一个小的文档集，几个就够了。为它们建立索引，并试着做一些查询。

练习 1.3 在图 1-6 的马尔可夫模型中，从状态 3 开始，生成“not not be to be”的概率是多少？

练习 1.4 在图 1-6 的马尔可夫模型中，从未知状态开始生成“to be”。生成之后会到达哪个或哪些状态？

练习 1.5 在图 1-6 的马尔可夫模型中，是否存在有限数 n，使得无论从哪个状态开始，都可以在生成 n 个或大于 n 个的字符串之后，总能得知当前状态？

练习 1.6 给定一个马尔可夫模型，假设存在有限数 n，使得无论从哪个状态开始，可以在生成 n 个或大于 n 个的字符串之后，总能得知当前状态。描述一个把马尔可夫模型转换成 n 阶有限上下文模型的过程。

练习 1.7 假设我们扩展了马尔可夫模型使得每对状态之间存在多个转移，其中每个转换上面标识一个不同的词项。对于任意这样的模型，证明存在一个等价的马尔可夫模型，使得任意状态对中不超过一个转移。提示：分割目标状态。

练习 1.8 描述将 n 阶有限上下文语言模型转换为马尔可夫模型的步骤。总共需要用多少个状态？

练习 1.9（项目练习） 这个练习基于 Wikipedia 创建了一个测试集，在第一部分后面的很多练习中都会用到它。

首先，下载一个最新的英文版 Wikipedia。在撰写本文时，它的下载版本包括一个大文档。Wikipedia 本身包含了描述这个下载格式的文档[⊖]。

除了文章本身，下载还包括了**重定向记录**（redirection record），替代了文章的标题。预处理这个下载文集，去除这些记录和其他任何无关的信息，只留下一组独立的文档集合。Wikipedia 本身的格式也应该去掉，或者你也可以用 XML 风格的标签来替换。给每篇文章添加一个标识符。添加〈DOC〉和〈DOCNO〉标签。结果应该与 TREC 规则相一致，如 1.4 节文字和图 1-7 所描述的那样。

练习 1.10（项目练习） 根据图 1-8 的样式，创建 3～4 个主题在英文的 Wikipedia 上测试检索性能。避免那种一篇“好”文章就能完全满足信息需求的主题。相反地，应尽可能创建那些需要多篇文章才能覆盖相关信息的主题。（注意：这个练习适合作为一个班级项目的基础，创建一个 Wikipedia 测试集，每个学生贡献足够多的主题使得主题总数达到 50 或更多）。（更多细节参考练习 2.13）

练习 1.11（项目练习） 下载并安装一个开源信息检索系统（见练习 1.2）。用这个信息检索系统索引你在练习 1.9 中建立的测试集。用你在练习 1.10 中创建的主题标题作为查询。对每个主题，判断前 5 个返回的文档是否是相关的。该系统对所有主题是否都适用？

练习 1.12（项目练习） 根据 1.3.2 节介绍的过程，对你在练习 1.9 中创建的文集进行分词。在这个练习中，除去所有的标签，即只包含词（即字母数字串）。Wikipedia 文本已进行了 UTF-8 编码，但是在这个练习中，你可以把它当做是 ASCII 文本。画一个词频-排名对数坐标图，就像图 1-5 那样。数据是否符合齐夫定律？如果符合，α 的近似值是多少？

⊖ en.wikipedia.org/wiki/Wikipedia：Database_download

练习 1.13（项目练习） 根据练习 1.12 的分词结果，创建一个三元语言模型。利用这个语言模型，实现一个 Wikipedia 文本随机生成器。你将如何扩展这个文本生成器来生成大小写和标点符号，使生成文本更像英语？你又如何扩展这个生成器来生成包含标签和链接的 Wikipedia 随机文本？

1.8 参考文献

Baeza-Yates, R. A., and Ribeiro-Neto, B. (2010). *Modern Information Retrieval* (2nd ed.). Reading, Massachusetts: Addison-Wesley.

Croft, W. B., Metzler, D., and Strohman, T. (2010). *Search Engines: Information Retrieval in Practice.* London, England: Pearson.

Grossman, D. A., and Frieder, O. (2004). *Information Retrieval: Algorithms and Heuristics* (2nd ed.). Berlin, Germany: Springer.

Hearst, M. A. (2009). *Search User Interfaces.* Cambridge, England: Cambridge University Press.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval.* Cambridge, England: Cambridge University Press.

Manning, C. D., and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* Cambridge, Massachusetts: MIT Press.

Özsu, M. T.; and Liu, L., editors (2009). *Encyclopedia of Database Systems.* Berlin, Germany: Springer.

Salton, G. (1968). *Automatic Information Organziation and Retrieval.* New York: McGraw-Hill.

van Rijsbergen, C. J. (1979). *Information Retrieval* (2nd ed.). London, England: Butterworths.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images* (2nd ed.). San Francisco, California: Morgan Kaufmann.

Zipf, G. K. (1949). *Human Behavior and the Principle of Least-Effort.* Cambridge, Massachusetts: Addison-Wesley.

基础技术

作为本书余下部分的基础，本章将对第1章提及的信息检索的各方面内容做一个概述，包括索引、检索和评价的基础。索引和检索的内容构成了前两节，它们紧密联系，为这些主题提供了一个统一的观点。第三节主要讨论评价，即对前两节中介绍的算法的效率和有效性进行评价。

2.1 倒排索引

倒排索引（有时称为倒排文件（inverted file））几乎是每一个信息检索系统的核心数据结构。最简单地，倒排索引提供了文档集C中词项t与其出现的位置之间的映射。倒排索引的基本组成可由图2-1解释。它给出了莎士比亚戏剧集（图1-2和图1-3）文本上的一个索引。词典（dictionary）列出了文档集的词汇表V中包含的词项。每一个词项与一个表征它出现位置的位置信息列表（posrings list）相关联，这个列表与图1-4中的位置编号是一致的。

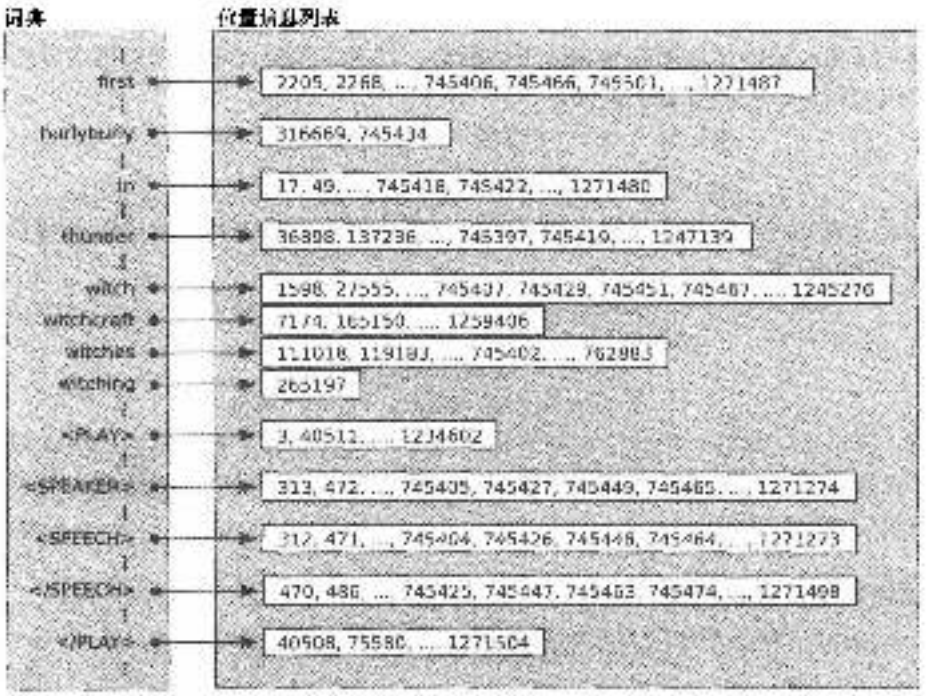


图2-1 莎士比亚戏剧集的模式独立倒排索引。这个词典建立了词项到其出现位置之间的映射

如果你之前接触过倒排索引，你可能会惊讶于图2-1展示的索引不包含文档标识符。只简单地包含每个词项在文档集中出现的位置。这种类型的索引称为模式独立索引（schema-independent index），因为这种索引不对文档的底层结构作任何假定（数据库领域通常称为模式（schema））。这种索引是最简单的，所以本章中绝大部分例子都使用这种类型的索引。2.1.3节将会概述其他的索引类型。

不管使用的是哪一种特定类型的索引，索引的组成部分——词典与位置信息列表——都可以存储在内存、磁盘，或者同时存在于二者中。现在，我们特意把精确的数据结构模糊化。将倒排索引定义为一个具有四个方法的抽象数据类型（ADT）。

- first(t) 返回词项 t 在文档集中第一次出现的位置；
- last(t) 返回词项 t 在文档集中最后一次出现的位置；
- next(t, *current*) 返回在 *current* 位置之后词项 t 第一次出现的位置；
- prev(t, *current*) 返回在 *current* 位置之前词项 t 最后一次出现的位置。

并且，我们定义 l_t 为词项 t 在文档集中出现的总次数（即该词项位置信息列表的长度），定义 l_C 为文档集的长度，因此有 $\sum_{t \in \mathcal{V}} l_t = l_C$（$\mathcal{V}$ 指的是文档集的词汇表）。

从图 2-1 中的倒排索引中，我们可以得到：

first("hurlyburly") = 316669　　last("thunder") = 1247139
first("witching") = 265197　　last("witching") = 265197
next("witch", 745429) = 745451　　prev("witch", 745451) = 745429
next("hurlyburly", 345678) = 745434　　prev("hurlyburly", 456789) = 316669
next("witch", 1245276) = ∞　　prev("witch", 1598) = $-\infty$

$l_{\text{<PLAY>}} = 37$　　$l_C = 1271504$
$l_{\text{witching}} = 1$

符号$-\infty$和∞分别是文档开始和结束的标记，代表了超出词项序列头尾之外的位置。作为一个实用的约定，我们有如下定义：

$$\text{next}(t, -\infty) = \text{first}(t) \qquad \text{next}(t, \infty) = \infty$$
$$\text{prev}(t, \infty) = \text{last}(t) \qquad \text{prev}(t, -\infty) = -\infty$$

倒排索引的方法允许顺序或随机地访问位置信息列表，其中每个位置信息列表的顺序扫描都是一个简单循环。

```
current ← −∞
while current < ∞ do
    current ← next(t, current)
    do something with the current value
```

然而，许多算法需要对位置信息列表随机访问，包括接下来要介绍的词组查找算法。通常，这些算法将某个词项的函数调用的输出结果作为另外一个词项的函数调用的参数，而不是顺序访问位置信息列表。

2.1.1 延伸例子：词组查找

许多商用 Web 搜索引擎以及其他的信息检索系统，将以双引号括起来（"…"）的词项列表视为是一个词组。为了处理包含词组的查询，信息检索系统必须要能确定词组在文档集中出现的位置。这个信息在检索过程中用于过滤和排名——那些不包含这个词组的准确匹配的文档将可能被去除。

词组查找算法很好地展示了算法在倒排索引上是如何操作的。假如我们要在莎士比亚戏剧集中找到词组“first witch”所有出现的位置。也许我们正想知道这个角色要讲的所有台词。通过肉眼扫描图 2-1 的位置信息列表，我们可以在 745406～745407 之间找到它的一次出现。通过类似的方式，即查找到“first”的后面紧跟“witch”，我们也可以定位这个词组在文档集中其他出现的位置。本节我们给出一个实现这一过程的算法，在我们定义的倒排索

引 ADT 的辅助下，它可以高效地定位给定词组所有出现的位置。

我们用一段区间 $[u, v]$ 来表示一个词组的位置，其中 u 代表词组出现的起始位置，v 代表结束位置。除 [745406, 745407] 外，词组“first witch”还出现在 [745466, 745467]、[745501, 745502] 和其他位置。词组查找算法的目的就是确定文档集中给定词组所有出现位置的 u 和 v 值。

本书我们就采用如上所述的区间表示法来确定检索结果。在某些上下文中，这种区间表示法也可以很方便地表示那个地方出现的文本。例如，区间 [914823, 914829] 表示以下文本：

O Romeo, Romeo! wherefore art thou Romeo?

给定一个包含 n 个词项的词组“$t_1\ t_2 \cdots t_n$”，算法将从左到右扫描位置信息列表，为每个词项调用 next 方法，然后从右到左扫描，为每个词项调用 prev 方法。经过这样一次往返，算法就计算出一个区间，区间中词项按顺序出现并且位置尽可能靠近。然后，算法检查词项是否是紧密相邻的。如果是，则找到了该词组的一个出现；否则，算法继续执行。

图 2-2 给出了给定位置之后查找词组下一次出现位置的函数 nextPhrase 的算法核心。第 2～3 行循环调用倒排索引中的方法，按顺序定位词项。循环结束后，如果词组出现在区间 $[position, v]$ 中，那么，词组在位置 v 处结束。第 7～8 行的循环在保持词项顺序的前提下将区间尽可能地缩小。最后，第 9～12 行验证这些词项是否是紧密相邻的。如果不毗邻，函数将执行一个尾递归调用。在第 12 行中，注意到 u（而不是 v）作为递归调用的第二个参数被传递。如果词组中每一个词项都不同，那么 v 也可作为这个参数被传递。当词组中有两个词项 t_i 和 t_j 相同时（$1 \leqslant i < j \leqslant n$），传递 u 才能够正确处理这一情况。

```
    nextPhrase (t1t2...tn, position) ≡
1       v ← position
2       for i ← 1 to n do
3           v ← next(ti, v)
4       if v = ∞ then
5           return [∞, ∞]
6       u ← v
7       for i ← n − 1 down to 1 do
8           u ← prev(ti, u)
9       if v − u = n − 1 then
10          return [u, v]
11      else
12          return nextPhrase(t1t2...tn, u)
```

图 2-2 查找在给定位置之后一个词组第一次出现的位置的函数，函数调用倒排索引 ADT 的 next 和 prev 方法，返回文档集中的一段位置区间作为结果

举个例子，假如我们想要找出词组“first witch”第一次出现的位置：nextPhrase (“first witch”, $-\infty$)。算法首先确定“first”第一次出现的位置：

$$\text{next}(\text{“first”}, -\infty) = \text{first}(\text{“first”}) = 2205$$

如果这次出现的“first”是这个词组的一部分，那么紧跟着就应该出现“witch”。然而，

$$\text{next}(\text{“witch”}, 2205) = 27555$$

也就是说，“witch”并没有紧随其后。我们现在可以断定这个词组第一次出现的结束位置不可能在位置 27555 之前，我们计算

$$\text{prev}(\text{“first”}, 27555) = 26267$$

在“first”的位置信息列表中，我们可以发现从 2205～26267 之间“first”出现了 15 次。因为区间 [26267，27555] 的长度为 1288，而不是符合要求的长度 2，所以算法继续寻找“first”的下一次出现位置：

$$\text{next}(\text{“first”}, 26267) = 27673$$

注意到算法第 8 行中对 prev 方法的调用不是必需的（见练习 2.2），但是它们能够帮助我们分析这个算法的复杂度。

如果我们想获得词组所有的出现位置而不仅是一次，则需要多一个循环，为词组的每一次出现都调用一次 nextPhrase：

```
u ← −∞
while u < ∞ do
    [u,v] ← nextPhrase(“t1t2...tn”, u)
    if u ≠ ∞ then
        report the interval [u,v]
```

循环会报告产生的每一个区间。依具体的应用而定，报告 [u，v] 的同时可能还需返回包含这个词组的文档给用户，或者是用数组或者其他数据结构来保存这些区间以便进一步处理。与图 2-2 中的代码类似，u（而不是 v）是 nextPhrase 的第二个参数。从而，在以下这段著名的 Monty Python[⊖]的歌词中，该函数能正确地定位词组“spam spam spam”出现了六次：

Spam spam spam spam

Spam spam spam spam

为了分析这个算法的时间复杂度，首先观察到，每次调用 nextPhrase 都 $O(n)$次调用了倒排索引中的 next 和 prev 方法（n 次调用 next，接着 $n-1$ 次调用 prev）。算法执行到第 8 行后，区间 [u，v] 包含了词组中按顺序出现的所有词项，并且不存在一个更小的区间包含以上按顺序出现的词项。接下来注意到词项 t_i 在文档集中的每一次出现，在算法 1～8行计算出来的区间中不会超过一次。即使词组中包含两个相同的词项 t_i 和 t_j，与 t_i 匹配的文档集中的词条也最多出现在一个区间中，尽管这个词条可能会作为 t_j 的匹配出现在另外一个位置区间中。因此，算法的时间复杂度由词组里所有词项中最短位置信息列表的长度来决定：

$$l = \min_{1 \leqslant i \leqslant n} l_{t_i} \tag{2-1}$$

综合这些观察，在最坏的情况下算法需要 $O(n \cdot l)$次调用 ADT 中的方法来定位词组所有出现的位置。如果词组同时包含常见词项和非常见词项（“Rosencrantz and Guildenstern are dead”），那么调用次数决定于最不频繁的词项（“Guildenstern”）而不是最频繁的词项（“and”）。

我们强调一下，$O(n \cdot l)$代表的是调用方法的次数，而不是算法的步数，且每个方法调用所需的时间依赖于方法的具体实现细节。根据算法对数据结构的访问模式，存在一个简单高效的实现方法，且对于同时含有频繁词项和非频繁词项的词组有很好的性能。我们在下一节会详细介绍。

⊖ Monty Python（巨蟒）小组是英国六人喜剧团体，成立于 20 世纪 60 年代后期。他们的电视喜剧系列《Monty Python's Flying Circus》在 20 世纪 70 年代风靡一时。科学家（尤其是计算机科学家）在命名新事物时经常会从 Python 上寻找灵感，网络术语“spam”（垃圾邮件）来自 Python 的著名电视剧集中的一集，他们在其中以“spam”为歌词不停地唱着；而编程语言“Python”更是一种直接的借用。——译者注

虽然在最坏的情况下算法需要 $O(n \cdot l)$次调用方法，但是实际次数依赖于词项在文档集中的相对位置。例如，假定我们要在以下编排的文档集中搜索“hello world”这个词组：

hello … hello … hello … hello world … world … world … world

所有的“hello”都出现在“world”前面。那么无论这个文档有多大或是每个词项出现多少次，这个算法都只需调用四次方法，就能确定这个词组在文档中只出现了一次。尽管这个例子有些极端且不太实际，但它解释了算法内在的**自适应性**（adaptive）——它的实际执行时间由数据特性决定。其他一些信息检索问题也可以用自适应的算法来解决，我们也会为了提高效率而尽可能地使用这些自适应的算法。

为了更清晰地说明这个算法的自适应性，我们介绍一种可以决定调用方法次数的数据特征的指标。考虑图 2-2 中第 9 行检验代码之前获得的区间 $[u, v]$。这个区间包含了词组中按顺序出现的所有词项，并且不存在一个更小的区间包含按顺序出现的所有词项。我们将具有这种特点的区间称为词项的**候选词组**（candidate phrase）。定义 κ 为这个给定文档集中的所有候选词组的个数，那么为了获取该词组所有出现的位置，所需调用方法的次数为$O(n \cdot \kappa)$。

2.1.2 实现倒排索引

> It moves across the blackness that lies between stars, and its mechanical legs move slowly. Each step that it takes, however, crossing from nothing to nothing, carries it twice the distance of the previous step. Each stride also takes the same amount of time as the prior one. Suns flash by, fall behind, wink out. It runs through solid matter, passes through infernos, pierces nebulae, faster and faster moving through the starfall blizzard in the forest of the night. Given a sufficient warm-up run, it is said that it could circumnavigate the universe in a single stride. What would happen if it kept running after that, no one knows.
>
> ——罗杰·泽拉兹尼，《光与暗的生灵》

当文档集不发生改变且小到足可以全部放在内存中时，倒排索引可以用非常简单的数据结构来实现。词典可以存储在哈希表或类似结构中，每一个词项 t 的位置信息列表可以存放在一个长度为 l_t 的固定数组 $P_t[]$ 中。对于莎士比亚文集中的词项“witch”，这个数组可以如下表示：

1	2		31	32	33	34		92
1598	27555	…	745407	745429	745451	745467	…	1245276

我们定义的倒排索引 ADT 的 first 和 last 方法能够在常数时间内分别返回 $P_t[1]$和 $P_t[l_t]$，next 和 prev 方法则可以使用二分查找来实现，时间复杂度为 $O(\log(l_t))$。next 方法的实现细节如图 2-3 所示，prev 方法的实现细节也是类似的。

回顾 2.1.1 节介绍的词组查找算法在最坏情况下需要调用 $O(n \cdot l)$次 next 和 prev 方法。如果我们定义：

$$L = \max_{1 \leqslant i \leqslant n} l_{t_i} \tag{2-2}$$

那么算法的时间复杂度变为 $O(n \cdot l \cdot \log(L))$，因为每次调用一个方法最多需要$O(\log(L))$。当使用符号 κ 来表示候选词组数时，时间复杂度变为 $O(n \cdot \kappa \cdot \log(L))$。

当词组中同时包含频繁词项和非频繁词项时，这种实现方法具有非常好的性能。例如，

在莎士比亚文集中，词项“tempest”仅出现了49次。正如我们在1.3.3节中所见，词项“the”出现了28 317次。然而，当要查找词组“the tempest”时，通过最多2×49=98次的二分查找，我们需要访问“the”的位置信息列表数组不到2000次。

```
    next (t, current) ≡
1       if l_t = 0 or P_t[l_t] ≤ current then
2           return ∞
3       if P_t[1] > current then
4           return P_t[1]
5       return P_t[binarySearch (t, 1, l_t, current)]

    binarySearch (t, low, high, current) ≡
6       while high − low > 1 do
7           mid ← ⌊(low + high)/2⌋
8           if P_t[mid] ≤ current then
9               low ← mid
10          else
11              high ← mid
12      return high
```

图2-3 通过一个独立的二分查找函数实现的next方法。数组$P_t[]$（长度为l_t）保存了词项t的位置信息列表。binarySearch函数假设$P_t[low] \leqslant current$和$P_t[high] > current$。算法的第1～4行保证了这一假设，第6～11行的循环用一个变型维持了这一假设

可是，当词组中包含出现频率接近的词项时，反复的二分查找是非常耗时的。词组“two gentlemen”中的两个词项在莎士比亚文集中都出现了几百次（具体分别是702和225次）。定位这个词组的所有出现需要超过2000次访问“two”的位置信息列表数组。在这种情况下，可能通过同时顺序访问两个位置信息列表数组并进行比较反而会比较有效。可以通过修改next和prev方法的定义让词组查找算法这样执行。

首先，我们注意到词组查找算法会为某个给定词项t_i连续调用next方法，包括递归调用在内，作为第二参数传递给nextPhrase的值是严格递增的。在为给定词组定位所有出现的过程中，算法会l次调用next方法（和前面一样，这里的l是最短的位置信息列表的长度）：

$$\text{next}(t_i, v_1),\ \text{next}(t_i, v_2),\ \cdots,\ \text{next}(t_i, v_l)$$

其中，

$$v_1 < v_2 < \cdots < v_l$$

并且，这些调用的返回结果也是严格递增的：

$$\text{next}(t_i, v_1) < \text{next}(t_i, v_2) < \cdots < \text{next}(t_i, v_l)$$

例如，当在莎士比亚文集中查找“first witch”时，为“first”产生的连续调用如下：

next(“first”,—∞),next(“first”,26267),next(“first”,30608),…

这些方法返回的值如下：

2205<27673<32995<…

当然，v_1，v_2，……的具体值以及next方法调用的实际次数还取决于这个词组中另一个词项的位置。但是我们可以知道这些值是依次递增的，并且在最坏的情况下有l个。

为了实现线性扫描，我们记录（或缓存）为给定词项调用next方法的返回值。当（为同一个词项）再次调用这个方法时，我们从缓存的位置继续扫描。图2-4给出了方法的实现细节。变量c_t缓存了上一次调用返回的值在数组中的偏移量，用另一个独立的值来缓存词组中的每个词项（如c_{first}和c_{witch}）。因为算法中的方法可能不会按严格递增的顺序来处理位置信息列表，所以一旦假设没有被满足（第6～7行），我们必须小心地重置c_t。

```
    next (t, current) ≡
1     if l_t = 0 or P_t[l_t] ⩽ current then
2       return ∞
3     if P_t[1] > current then
4       c_t ← 1
5       return P_t[c_t]
6     if c_t > 1 and P_t[c_t − 1] > current then
7       c_t ← 1
8     while P_t[c_t] ⩽ current do
9       c_t ← c_t + 1
10    return P_t[c_t]
```

图 2-4 采用线性扫描实现 next 方法。这种实现方法为每个词项 t 维护一个缓存索引偏移量 c_t，这里 $P_t[c_t]$表示为这个词项调用 next 方法返回的最后一个有限数结果（即最后一个合理的位置——译者注）。如果这个位置存在，则算法从这个缓存偏移量开始扫描，否则在算法第 6～7 行缓存偏移量被重置（为 1）

如果我们采用类似的方法实现 prev 方法，也维护相应的缓存值，那么词组查找算法为词组中的词项扫描位置信息列表，在有限次数内（$O(1)$）访问位置信息列表数组中的每一个元素。因为算法也许需要完全扫描最长的位置信息列表（长度为 L），并且所有位置信息列表都有可能是这个长度，所以算法的整体时间复杂度为 $O(n \cdot L)$。这种情况下算法的自适应特性没有产生任何好处。

现在我们有两种实现 next 和 prev 的方法，使得 nextPhrase 也有两种实现方法。第一种实现方法整体时间复杂度为 $O(n \cdot l \cdot \log(L))$，特别适用于最短位置信息列表比最长位置信息列表短很多的情况（$l \ll L$）。第二种实现方法的时间复杂度为 $O(n \cdot L)$，适用于所有位置信息列表长度近似的情况（$l \approx L$）。

对于这两种方法，我们可以想象在运行时通过比较 l 和 L 来选择合适的算法。然而，通过结合以上两种算法的特征，还是有可能定义第三种实现方法，其时间复杂度显然取决于最长和最短列表的相对长度（L/l）。第三种算法基于**跳跃式搜索**（exponential 或 galloping search）。思想是从一个缓存位置之后以指数级增加步长（“跳跃”）向前扫描直至跳过答案。此时，由最后两步构成的区间就恰好是答案所在的区间，用二分查找即可。图 2-5 给出了这个算法的细节。

```
    next (t, current) ≡
1     if l_t = 0 or P_t[l_t] ⩽ current then
2       return ∞
3     if P_t[1] > current then
4       c_t ← 1
5       return P_t[c_t]
6     if c_t > 1 and P_t[c_t − 1] ⩽ current then
7       low ← c_t − 1
8     else
9       low ← 1
10    jump ← 1
11    high ← low + jump
12    while high < l_t and P_t[high] ⩽ current do
13      low ← high
14      jump ← 2 · jump
15      high ← low + jump
16    if high > l_t then
17      high ← l_t
18    c_t ← binarySearch (t, low, high, current)
19    return P_t[c_t]
```

图 2-5 通过跳跃式搜索实现的 next 方法。算法第 6～9 行使用缓存值（如果有）确定 *low* 的初始值，使它满足 $P_t[low] \leqslant current$。第 12～17 行以指数级步长向前搜索，直到找到一个 *high* 值，满足 $P_t[high] > current$。最后结果通过二分查找来确定（如图 2-3 所示）

图 2-6 展示并对比了在莎士比亚文集中使用三种方法调用 prev（“witch”，745429）的情形。使用二分查找（图 2-6a）将访问数组 7 次，一开始算法从位置 1 和 92 的区间开始二分查找（没有在图中显示），依次是位置 46，23，34，28 和 31。使用线性扫描（图 2-6b）则从初始的缓存偏移位置 1 开始，将访问数组 34 次，包括为了检查边界条件而访问的次数（没有在图中显示）。跳跃式搜索（图 2-6c）在锁定二分查找范围之前将会依次访问位置 1，2，4，8，16 和 32，然后访问位置 24，28，30 和 31，总共 12 次访问位置信息列表数组（包括边界条件检查）。在线性扫描和跳跃式搜索结束后，缓存数组偏移都被更新为 31。

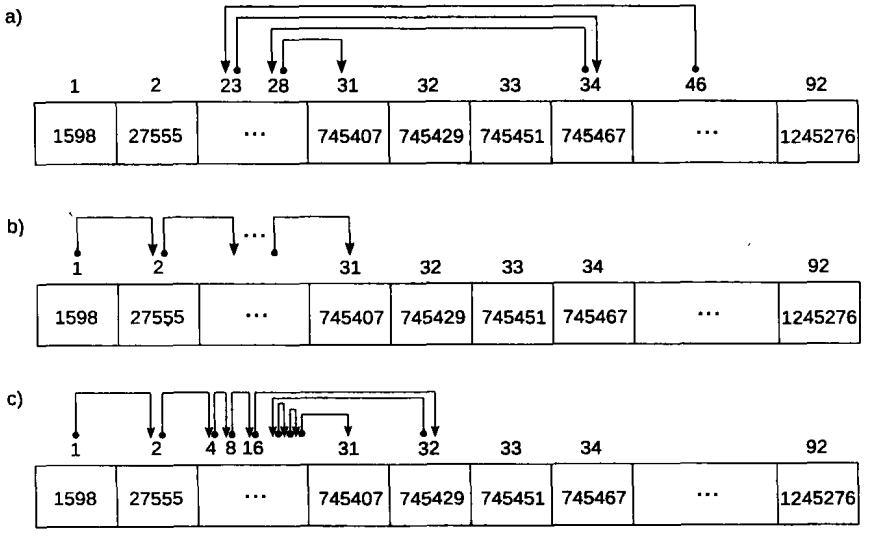


图 2-6 执行 prev（“witch”，745429）=745407 三种方法的访问模式：a）二分查找，b）线性扫描，c）跳跃式搜索。b）和 c）算法都从初始缓存位置 1 开始执行

为了分析跳跃式搜索的时间复杂度，我们回过头来看看当初由线性扫描算法启发得到的 next 方法调用序列。令 c_t^j 表示在处理给定词组查找过程中为词项 t 第 j 次调用 next 后的缓存值。

$$P_t[c_t^1] = \text{next } (t, v_1)$$
$$P_t[c_t^2] = \text{next } (t, v_2)$$
$$\cdots$$
$$P_t[c_t^l] = \text{next } (t, v_l)$$

对于跳跃式搜索来说，某一次调用 next 所需的工作量取决于两次调用之间缓存值的变化大小。假设缓存值变化了 Δc，那么一次调用所需的工作量为 $O(\log(\Delta c))$。这样，如果我们定义

$$\Delta c_1 = c_t^1$$
$$\Delta c_2 = c_t^2 - c_t^1$$
$$\cdots$$
$$\Delta c_l = c_t^l - c_t^{l-1}$$

则为词项 t 调用 next 所需总工作量为

$$\sum_{j=1}^{l} O(\log(\Delta c_j)) = O\left(\log\left(\prod_{j=1}^{l} \Delta c_j\right)\right) \tag{2-3}$$

我们知道一组非负数的算术平均数总是大于它的几何平均数：

$$\frac{\sum_{j=1}^{l} \Delta c_j}{l} \geqslant \sqrt[l]{\prod_{j=1}^{l} \Delta c_j} \tag{2-4}$$

并且，因为 $\sum_{j=1}^{l} \Delta c_j \leqslant L$，我们有

$$\prod_{j=1}^{l} \Delta c_j \leqslant (L/l)^l \tag{2-5}$$

因此，对于词项 t 所需调用 next（或者 prev）的总工作量为

$$O\left(\log\left(\prod_{j=1}^{l} \Delta c_j\right)\right) \subseteq O\left(\log\left((L/l)^l\right)\right) \tag{2-6}$$

$$= O(l \cdot \log(L/l)) \tag{2-7}$$

因此查找一个 n 个词项的词组，跳跃式搜索的整体时间复杂度为 $O(n \cdot l \cdot \log(L/l))$。当 $l \ll L$ 时，这个性能与二分查找接近；当 $l \approx L$ 时，与线性扫描的性能接近。如果还考虑这个算法的自适应性，根据类似的推理过程，可以得到时间复杂度为 $O(n \cdot \kappa \cdot \log(L/\kappa))$。

尽管在我们的讨论中主要关注用倒排索引来实现词组查找，但我们可以看到，对于其他问题，跳跃式搜索也是适用的一种技术。本书的第二部分将这些思路扩展到存储在磁盘上的数据结构上。

2.1.3 文档和其他元素

大部分信息检索系统和算法都在一个检索的基本单位上进行操作，那就是文档。正如在第 1 章讨论的那样，特定应用环境的需求决定了文档的组成元素。根据这些需求，一个文档可以是 e-mail 邮件、Web 页面、新闻或者类似的元素。

在很多应用环境中，一个文档的定义是相当自然的。然而，在少数环境中，例如一组书集，一个自然单元（即一整本书）有时会太大了，特别是与查询相关的材料只局限于小部分文本的时候，不能返回相对合理的结果。其实，这时期望的返回结果可能是一章、一节或者是一小节，甚至是几页。

在莎士比亚戏剧集作为文档集的这个例子中，最自然的做法可能是将每一部戏剧都视为一个文档，但视具体的环境而言，每一幕、每一场、每段台词或行都有可能是合适的检索单元。举一个简单的例子，假设我们现在对台词感兴趣，并想找出所有由“first witch”所讲的台词。

词组“first witch”第一次出现在 [745406，745407]。可以很直接地找出所有包含这个词组的台词段。使用我们定义的倒排索引 ADT 中的方法，可以立即确定位于这个词组之前的台词段的开始处在：

prev(“〈SPEECH〉”，745406)=745404

这段台词结束于：

next(“〈/SPEECH〉”，745404)=745425

一旦我们确定区间［745406，745407］包含在区间［745404，745425］内，就可以知道这段台词包含这个词组。确认包含关系是有必要的，因为词组不一定总是作为台词的一部分出现。如果我们想找到所有包含“first witch”的台词段，可以在词组下一个出现位置重复以上过程。

仍然有一个小问题。尽管我们知道这个词组出现在台词中，但我们不知道找到的“first witch”是否代表角色。因为这个词组可能出现在某句台词里。幸运的是，确认 witch 是一个角色只需再调用倒排索引中的另外两个方法（练习 2.4）。实际上，简单地调用这些方法足以解决很多结构化的关系，列举如下：

1）任何一个 witch 所说的台词。

2）说“To be or not to be”的那个角色。

3）提到 witch 和 thunder 的戏剧的名称。

从一个更广义的上下文来讲，这种可以指定检索单元和通过简单的包含关系进行过滤的灵活方法有着许多的应用。在 Web 检索系统中，简单的过滤可以使检索结果只限于某个域内。在企业搜索中，在发送者字段中运用一些限制可以使我们选择信息只由某个特定的人发送。在文件系统检索中，运用结构化限制能够确定文件的权限和安全限制是否允许用户检索一个目录。

因为在信息检索应用中往往要求轻量级结构，因此我们采用一个简单统一的方法来支持这种结构，即直接将这种结构合并到倒排索引中去，使得它成为倒排索引所提供的基础功能之一。上面的例子体现了我们的方法。具体细节会在第 5 章中介绍，这种方法也是实现高级搜索的基础，而高级搜索广泛应用于如法律检索这样的特殊领域的信息检索中。这种方法也可用于 8.7 节介绍的不同字段加权方法中，也就是认为一个查询词项出现在文档标题中会比出现在正文里能体现更强的查询相关性。这种方法也是 XML 检索中所需的复杂索引结构的实现基础（见第 16 章）。

尽管有些环境需要轻量级的数据结构，但是大多数信息检索研究仍然假定文本集是被自然划分为文档的，这也是检索的最小基本单元。在 e-mail 检索系统中，邮件便是这个基本的检索单元。在文件系统中，基本检索单元是文件；在 Web 中则是网页。除了是一个检索的自然单元之外，对于来自于单点或 Web 网站的所有文档集，文档这种形式还是文本的自然划分，从而允许一组文集可被划分为多个子集以进行并行检索，或重排序文档以提高系统效率。

面向文档的索引

由于文档检索代表了一个如此重要的特殊地位，因此常常围绕它来进行索引优化。为了进行优化，文档集中的位置标号分为两个部分：文档编号与该文档内的偏移。

我们用标记 $n:m$ 来表示面向文档的索引内的位置信息，其中，n 为文档标识符（或者叫**文档编号**（docid）），m 为**偏移**（offset）。图 2-7 展示了一个为莎士比亚戏剧集建立的倒排索引，这里，戏剧就是文档。我们沿用之前介绍的倒排索引 ADT 中的方法，但是它们接受**文档编号**：**偏移**作为参数，并也以这种方式返回。

文档内的偏移范围从 1 到该文档的长度。我们仍然用$-\infty$和∞来标记文档的开始和结束，忽略写法上略微的区别，$-\infty$记为$-\infty:-\infty$，∞记为$\infty:\infty$。当词项的位置表示为这种形式时，便可使用文档编号作为主键来比较它们的值。

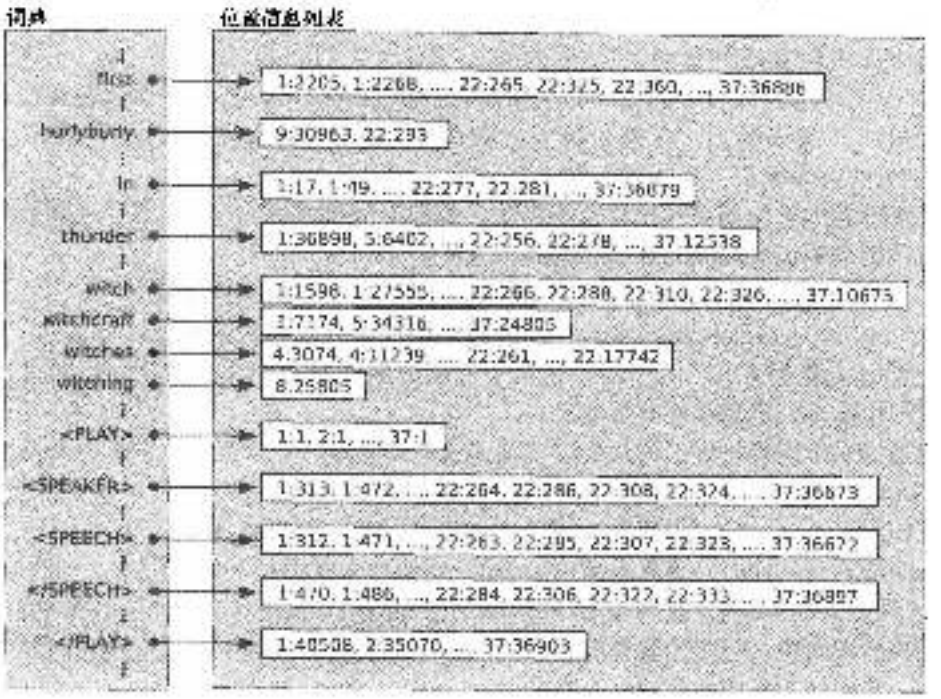


图 2-7 莎士比亚文集上面向文档的索引，等价于图 2.1 中的索引。每一个位置表示为：文档编号：文档内偏移

例如，

first("hurlyburly") = 9:30963　　last("thunder") = 37:12538

first("witching") = 8:25805　　last("witching") = 8:25805

next("witch", 22:288) = 22:310　　prev("witch", 22:310) = 22:288

next("hurlyburly", 9:30963) = 22:293　　prev("hurlyburly", 22:293) = 9:30963

next("witch", 37:10675) = ∞　　prev("witch", 1:1598) = -∞

$n<n' \cdot m<m'$ 当且仅当（$n<n'$ 或者（$n=n'$ 且 $m<m'$））

我们将可以支持面向文档检索的结构优化后的索引称为模式依赖（schema dependent）的倒排索引。因为将文本划分为检索单元（即它的模式）的同时也决定了其倒排索引的建立方式。

没有优化的索引称为模式独立（schema-independent）的倒排索引。一个模式独立的倒排索引允许在查询阶段才指定文档的定义，对比模式依赖的倒排索引而言，这样做有可能会带来额外的时间代价。

为了能够支持排名算法，模式依赖的倒排索引通常要维护各种面向文档的统计值。我们使用如下符号来表示这些统计值：

N_t	文档频率	在文档集中包含词项 t 的文档数
$f_{t,d}$	词频	词项 t 在文档 d 中出现的次数
L_d	文档长度	文档中包含的词条数
L_{ave}	平均长度	整个文档集的文档平均长度
N	文档数	文档集的总文档数

请注意：$\sum_{d \in C} l_d = \sum_{t \in V} l_t = l_C$ 和 $l_{avg} = l_C / N$。

在整个莎士比亚戏剧集中，$l_{avg} = 34363$。若 $t =$ "witch"，$d = 22$（即 $d =$ Macbeth），那么，$N_t = 18$，$f_{t,d} = 52$，$l_d = 26805$。在模式依赖的索引中，这些统计值通常作为索引数据结构的组成部分来维护。在模式独立的索引中，就需要在查询的时候再使用倒排索引 ADT 中的方法去计算了（见练习 2.5）。

为了帮助我们为文档级的操作写出算法，我们为倒排索引 ADT 定义了一些新方法。首先是能将位置分割为文档编号和偏移的两个方法：

docid(*position*)　　返回 *position* 所在的文档编号

offset(*position*)　　返回 *position* 在文档内的偏移

当一个位置信息表示成 $[u:v]$ 时，这两个方法分别返回 u 和 v。这些方法同样能在模式独立的索引中实现，但效率会略低些。

我们还为基本的倒排索引方法定义了面向文档的版本，在本章的余下部分会使用到：

firstDoc(t)　　返回包含词项 t 的第一个文档的编号

lastDoc(t)　　返回包含词项 t 的最后一个文档的编号

nextDoc(t, *current*)　返回当前(*current*)文档后包含词项 t 的第一个文档的编号

prevDoc(t, *current*)　返回当前(*current*)文档前包含词项 t 的最后一个文档的编号

在模式依赖的索引中，很多位置会有相同的文档编号前缀。我们可以将这些相同的文档编号分离出来，产生如下形式的位置：

$$(d, f_{t,d}, \langle p_0, \cdots, p_{f_{t,d}} \rangle)$$

其中，$\langle p0, \cdots, p_{f_{t,d}} \rangle$ 是词项 t 在文档 d 中出现 $f_{t,d}$ 次分别的偏移位置信息列表。除了能够删除不必要的重复信息之外，这种标记方法更好地反映了在模式依赖索引实现过程中，词项的位置实际上是如何表示的。使用这种标记，词项 "witch" 的位置信息列表可以表示成如下形式：

$$(1,3,\langle 1598, 27555, 31463\rangle), \cdots, (22,52,\langle 266, 288, \cdots\rangle), \cdots, (37,1,\langle 10675\rangle)$$

在特殊的应用环境中，可能不需要在倒排索引中保存位置偏移。对于一些应用而言，基本的关键字搜索就足够了，文档级的统计信息就能够进行有效的排名。对那些最简单的排名和过滤技术来说，甚至连这些文档级的统计信息都不需要。

根据位置信息列表中的信息类型，我们可以把倒排索引分为 4 种类型，前 3 种属于模式依赖索引：

- **文档编号索引**（docid index）是最简单的索引类型。对于每个词项，它包含词项所在的所有文档的标识符。尽管它很简单，但这种索引类型足以支持包括基本布尔查询（2.2.3 节）和称为**协调级排名**（coordination level ranking）的简单的相关性排名（练习 2.7）在内的过滤操作。
- 在**词频索引**（frequency index）中，倒排列表中的每一项包含两个组成部分：文档 ID 和词频。词频索引中的每一个位置形式为（d，$f_{t,d}$）。词频索引足以支持许多有效的排名方法（2.2.1 节），但是却不能完成词组查找和高级过滤。
- **位置索引**（positional index）由形如（d，$f_{t,d}$，$\langle p_1, \cdots, p_{f_{t,d}} \rangle$）的位置组成。位置索引支持所有需要词频索引的搜索操作。此外，还可以用于实现词组查询，邻近度排名（2.2.2 节），以及其他需要使用到每个查询词项在文档中的精确位置的查询类型，包括所有类型的结构化查询。
- **模式独立索引**（schema-independent index）没有像位置索引那样的面向文档的优化，

但二者有时可以互换。

表 2-1 莎士比亚文集的《Romeo and Juliet》第一场第一幕中的文本片段

文档 ID	文档内容
1	Do you quarrel, sir?
2	Quarrel sir! no, sir!
3	If you do, sir, I am for you; I serve as good a man as you.
4	No better.
5	Well, sir.

表 2-2 表 2-1 中词项的位置信息列表。在每一个例子中，在实际列表的起始处添加了列表的长度

词项	文档编号列表	位置信息列表	模式独立信息列表
a	1; 3	1; (3, 1, ⟨13⟩)	1; 21
am	1; 3	1; (3, 1, ⟨6⟩)	1; 14
as	1; 3	1; (3, 2, ⟨11, 15⟩)	2; 19, 23
better	1; 4	1; (4, 1, ⟨2⟩)	1; 26
do	2; 1, 3	2; (1, 1, ⟨1⟩), (3, 1, ⟨3⟩)	2; 1, 11
for	1; 3	1; (3, 1, ⟨7⟩)	1; 15
good	1; 3	1; (3, 1, ⟨12⟩)	1; 20
i	1; 3	1; (3, 2, ⟨5, 9⟩)	2; 13, 17
if	1; 3	1; (3, 1, ⟨1⟩)	1; 9
man	1; 3	1; (3, 1, ⟨14⟩)	1; 22
no	2; 2, 4	2; (2, 1, ⟨3⟩), (4, 1, ⟨1⟩)	2; 7, 25
quarrel	2; 1, 2	2; (1, 1, ⟨3⟩), (2, 1, ⟨1⟩)	2; 3, 5
serve	1; 3	1; (3, 1, ⟨10⟩)	1; 18
sir	4; 1, 2, 3, 5	4; (1, 1, ⟨4⟩), (2, 2, ⟨2, 4⟩), (3, 1, ⟨4⟩), (5, 1, ⟨2⟩)	5; 4, 6, 8, 12, 28
well	1; 5	1; (5, 1, ⟨1⟩)	1; 27
you	2; 1, 3	2; (1, 1, ⟨2⟩), (3, 3, ⟨2, 8, 16⟩)	4; 2, 10, 16, 24

表 2-1 是莎士比亚戏剧《Romeo and Juliet》的一个片段。这里，每一行被视为是一个文档——为了使这个例子长度合理，我们去掉了片段中的标签。表 2-2 展示了这个片段中的所有词项对应的位置信息列表，分别给出了文档编号列表、位置信息列表和模式独立信息列表的实例。

在四种不同的索引类型中，文档编号索引总是最小的，因为它包含最少的信息。对于一般文档集而言，位置索引和模式独立索引所占用空间是最大的，是词频索引的 2～5 倍，是文档编号索引的 3～7 倍。实际的倍数取决于文档集中文档的长度、词项分布的偏斜情况，以及压缩比率。表 2-3 展示了我们的三个文档样例集在这四种不同的索引类型下，索引所占的空间大小。索引压缩会对索引大小有显著的影响，在第 6 章会详细讨论。

表 2-3 压缩前后，三个测试文档集在不同索引类型下的索引大小。在每一个例子中，第一个数表示索引中的每一个数据简单地以 32 位整数保存下来所占的容量，第二个数表示索引中的数据采用字节对齐的编码方法压缩保存时所占的容量

	莎士比亚文集	TREC	GOV2
文档编号索引	n/a	578 MB/200 MB	37 751 MB/12 412 MB
词频索引	n/a	1110 MB/333 MB	73 593 MB/21 406 MB
位置信息索引	n/a	2255 MB/739 MB	245 538 MB/78 819 MB
模式独立索引	5.7 MB/2.7 MB	1190 MB/533 MB	173 854 MB/65 960 MB

在面向文档的索引的介绍中，我们大大扩展了在本章开始部分介绍的四种基本方法的倒排索引的相关符号。表 2-4 给出了符号列表，可供本书后面内容作方便的参考。

表 2-4 倒排索引的符号列表

基本倒排索引方法	
first(*term*)	返回词项 *term* 在文档集中第一次出现的位置
last(*term*)	返回词项 *term* 在文档集中最后一次出现的位置
next(*term*, *current*)	返回在 *current* 位置之后词项 *term* 第一次出现的位置
prev(*term*, *current*)	返回在 *current* 位置之前词项 *term* 最后一次出现的位置
等价于以上基本方法的面向文档的方法	
firstDoc(*term*), lastDoc(*term*), nextDoc(*term*, *current*), lastDoc(*term*, *current*)	
模式独立索引位置	
$n:m$	n=文档编号，m=偏移
docid(*position*)	返回 *position* 所在的文档编号
offset(*position*)	返回 *position* 在文档内的偏移
文档和词项统计符号	
l_t	t 的位置信息列表长度
N_t	在文档集中包含词项 t 的文档数
$f_{t,d}$	词项 t 在文档 d 中出现的次数
l_d	文档中包含的词条数
l_{avg}	整个文档集的文档平均长度
N	文档集的总文档数
位置信息列表的结构	
文档编号索引	$d_1, d_2, \cdots, d_{N_t}$
词频索引	$(d_1, f_{t,d_1}), (d_2, f_{t,d_2}), \cdots$
位置索引	$(d_1, f_{t,d_1}, \langle p_1, \cdots, pf_{t,d_1}\rangle), \cdots$
模式独立索引	$p_1, p_2, \cdots, p_{lt}$

2.2 检索与排名

基于上一节介绍的数据结构，这一节将介绍三种简单的检索方法。前两种方法产生排名的结果，根据文档与查询的相关性将文档集中的文档进行排序。第三种检索方法在文档集上应用布尔过滤，找出与谓词匹配的文档。

排名检索的查询通常用**词项向量**（term vector）来表示。当在信息检索系统中输入一个查询时，通过空格区分的词项来表达这个向量。例如，输入到商业 Web 搜索引擎中的查询

william shakespeare marriage

也许是希望获得莎士比亚与安娜·海瑟薇的婚姻情况的一组排名网页。为了使查询的这种特性更加明显，我们将词项向量明确地用符号 $\langle t_1, t_2, \cdots, t_n\rangle$ 来表示。前面的那个查询可以写成

〈“william”, “shakespeare”, “marriage”〉

你可能会想为什么把查询表达成向量而不是集合。当查询中存在重复词项或词项出现的顺序很重要的时候，向量的表达方式就很有用了（如果我们假设不是一个定长向量空间，则**列表**（list）也可以）。在排名的公式中，我们使用符号 q_t 来代表词项 t 出现在查询中的

次数。

布尔谓词由标准的布尔操作符（AND，OR，NOT）构成。布尔查询的结果集由那些匹配谓词的文档组成。例如，布尔查询：

"william" AND "shakespeare" AND NOT ("marlowe" OR "bacon")

是要找出那些包含词项"william"和"shakespeare"但不包含"marlowe"或"bacon"的文档。在后面的章节里，我们会扩展标准的布尔操作符集，使得我们能够对结果集再指定一些限制条件。

排名检索中的词项向量和布尔检索中的谓词在传统理解上有一个关键的不同点。布尔谓词通常被理解为严格的过滤器——如果文档不能够匹配这个谓词，那么这个文档就不能作为结果返回。相反，词项向量常被理解为描述信息需求的概要。文档不需要包含所有的查询词项才能返回。例如，如果我们对莎士比亚的生平和著作感兴趣，我们可能会试图列举尽可能多的相关词，从而构造一个详细的（这通常也是很累人的）查询请求来描述我们的信息需求。例如：

william shakespeare stratford avon london plays sonnets poems tragedy comedy poet playwright players actor anne hathaway susanna hamnet judith folio othello hamlet macbeth king lear tempest romeo juliet julius caesar twelfth night antony cleopatra venus adonis willie hughe wriothesley henry ...

尽管多数相关网页会包含这些查询词中的若干个，但很少会全部都包括。这也正是排名检索方法的任务，它决定哪些没有输入的词项会对最终文档的排序结果造成影响。

布尔检索与排名检索可自然地结合成一个两步检索过程。首先用布尔谓词限定只返回文档集中的一个子集。子集中的文档再按给定主题进行排名。商业 Web 搜索引擎一般都是这样的两步检索过程。直到最近，大部分的系统才将这个查询

william shakespeare marriage

解释为既是一个词项的布尔连接式——"william" AND "shakespeare" AND "marriage"——同时也是一个可用于排名的词项向量——〈"william", "shakespeare", "marriage"〉。对于一个将被作为结果返回的页面来说，每一个词项都必须出现在页面文本内容或是指向该页面的锚文本中。

当向查询中输入更多的词项时，由此过滤掉那些缺少一个或多个词项的相关页面对系统性能的影响有好有坏。原则上，添加词项可以更好地表达信息需求，因此对性能提高是有帮助的。尽管一些商业 Web 搜索引擎现在使用更少的限制过滤器来使排名结果中可以出现更多的页面，但基本还是遵循这个两步检索过程。这些系统处理长查询的能力很差，有些情况下只能返回很少的结果甚至是没有。

为了确定一个合适的文档排名，基本的排名检索方法会比较文档之间的一些简单特征。这些特征中最重要的一项是词频，$f_{t,d}$，也就是查询词项 t 出现在文档 d 中的次数。给定两个文档 d_1 和 d_2，如果一个查询词项出现在 d_1 的次数比出现在 d_2 的次数要多，那么在其他条件一样的情况下，就有可能表示 d_1 排在 d_2 前面。对于查询〈"william", "shakespeare", "marriage"〉来说，词项"marriage"在一个文档中的重复出现表明这个文档比那些仅包含这个词项一次的文档排名要更靠前。

另外一个很重要的特征就是**词项邻近度**（term proximity）。如果查询词项在文档 d_1 中出现的位置比在文档 d_2 中的更接近，那么在其他条件一样的情况下，这可能表明 d_1 应该排在 d_2 的前面。在一些情况中，词项构成了词组（如"william shakespare"）或其他一些

组合，但是紧凑程度的重要性不等同于文档中是否包含匹配的词组。例如，“william”，“shakespeare”和“marriage”同时出现在下面这样一个片段中：

> ... while no direct evidence of the **marriage** of Anne Hathaway and **William Shakespeare** exists, the wedding is believed to have taken place in November of 1582, while she was pregnant with his child ...

如果这些词项距离很远，那可能它们根本不存在什么关系。

其他一些特征也能帮助我们在各项竞争因素中做出权衡。例如，一个有千余字的文档包含了4次“william”，5次“shakespeare”，2次“marriage”，与一个包含3次“william”，2次“Shakespeare”，7次“marriage”的五百字的文档相比，是排在后面还是前面呢？这些特征包括与平均文档长度（l_{avg}）相关的文档长度（l_d），还有与文档集中文档总数（N）相关的包含词项 t 的文档数（N_t）。

尽管上述特征构成了多数检索模型和排名方法的核心，包括本章讨论的那些，但是其他特征同样很有用。在一些应用领域，例如Web搜索，探索一些其他的特征对于一个搜索引擎的成功有至关重要的作用。

一个重要的特征是文档结构。例如，查询词项出现在文档标题还是出现在正文应区别对待。通常文档之间的关系也很重要，例如Web文档之间的链接。在Web搜索的应用中，分析网页之间的链接使我们可以为它们赋予一个与查询无关的序或**静态排名**（static rank），而这将是影响检索的一个因素。最后，当有一群人定期使用一个企业内部或Web上的信息检索系统时，观察他们的行为习惯也有助于提升系统性能。例如，一个Web站点总是比其他站点获得更多的点击率，则意味着用户更偏好这个站点——在其他条件都一样的情况下——可就此改进排名的结果。在后面的章节将详细讨论这些其他特征。

2.2.1 向量空间模型

向量空间模型（vector space model）是本书中详细介绍的信息检索模型中最古老和最有名的一个。这个模型始于20世纪60年代，一直发展到20世纪90年代，由Gerald Salton提出并最终发表出来，他是早期信息检索研究者中最具影响力的一位。因此，向量空间模型与包括排名检索在内的许多已应用于信息检索问题中的领域是密切相关的，包括文档聚类与分类，并且会一直发挥重要作用。近年来，向量空间模型很大程度上被概率模型、语言模型和机器学习方法（见本书第三部分）超过了。然而，这个模型底层蕴含的简单道理以及它悠久的历史使得它仍然是介绍排名检索的理想工具。

基本思想是很简单的。查询和文档都表示为高维空间中的向量，而每个分量对应文档集词汇表中的一个词项。这种查询向量表示方法与前一节介绍的词项向量表示方法相比是有差别的，词项向量里只包含出现在查询中的词项。给定一个查询向量和一个文档向量集，每一个文档对应一个文档向量，我们通过计算查询向量和每个文档向量之间的夹角来获得它们的相似性，以此对文档进行排名。夹角越小，向量越相似。图2-8使用只包含两个分量（A 和 B）的向量解释了这一基本思想。

线性代数为我们提供了一个很简单的公式来计算两个向量之间的夹角 θ。给定两个 $|\mathcal{V}|$ 维的向量 $\vec{x}=\langle x_1, x_2, \cdots, x_{|\mathcal{V}|}\rangle$，$\vec{y}=\langle y_1, y_2, \cdots, y_{|\mathcal{V}|}\rangle$ 我们有：

$$\vec{x}\cdot\vec{y}=|\vec{x}|\cdot|\vec{y}|\cos(\theta) \tag{2-8}$$

其中，$\vec{x}\cdot\vec{y}$ 代表向量之间的**点积**（dot product）（也叫做**内积**（inner product）或**数量积**（scalar product））。$|\vec{x}|$和$|\vec{y}|$分别代表向量的长度。点积如下定义：

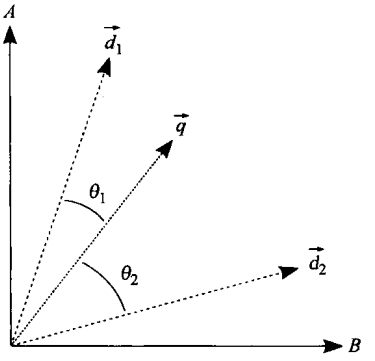


图 2-8 向量空间模型中的文档相似度。分别计算查询向量 $\vec{q}$ 和两个文档向量 $\vec{d}_1$，$\vec{d}_2$ 之间的夹角。因为 $\theta_1<\theta_2$，因此 d_1 应排在 d_2 前面

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{|\mathcal{V}|} x_i \cdot y_i \tag{2-9}$$

向量的长度可用欧几里得距离公式来计算

$$|\vec{x}| = \sqrt{\sum_{i=1}^{|\mathcal{V}|} x_i^2} \tag{2-10}$$

对上面的公式做替换，我们可以得到如下公式：

$$\cos(\theta) = \frac{\vec{x}}{|\vec{x}|} \cdot \frac{\vec{y}}{|\vec{y}|} = \frac{\sum_{i=1}^{|\mathcal{V}|} x_i y_i}{\left(\sqrt{\sum_{i=1}^{|\mathcal{V}|} x_i^2}\right)\left(\sqrt{\sum_{i=1}^{|\mathcal{V}|} y_i^2}\right)} \tag{2-11}$$

我们可以用反余弦来算出 θ 的大小，但是由于余弦值随着角 θ 单调递减，这个时候我们就可以停止计算了，直接用余弦值来作为相似性指标。如果 $\theta=0°$，$\cos(\theta)=1$，那么这两个向量是共线的，也就是说这两个向量极其相似。如果 $\theta=90°$，$\cos(\theta)=0$，那么这两个向量是正交的，这两个向量是极不相似的。

也就是说，给定一个文档向量 $\vec{d}$ 和一个查询向量 $\vec{q}$，余弦相似度 $\mathrm{sim}(\vec{d},\vec{q})$ 可以通过下面的公式来计算：

$$\mathrm{sim}(\vec{d},\vec{q}) = \frac{\vec{d}}{|\vec{d}|} \cdot \frac{\vec{q}}{|\vec{q}|} \tag{2-12}$$

文档向量和查询向量的点积可被归一化到单位长度。假定向量的各个分量都是非负的，那么这个**余弦相似度**（cosine similarity measure）的值范围为 0～1，且相似度越高，值越大。

自然地，哪怕是一个适中规模的文档集，向量空间模型也会产生几百万个维度的向量。乍看之下，高维可能会使效率变低，但是在许多情况下，查询向量是很稀疏的，只有少数几个非零的分量。例如，与〈“william”，“shakespeare”，“marriage”〉这个查询相对应的向量仅有 3 个非零分量。为了计算这个向量的长度，或者是它与某个文档向量的点积，我们仅需考虑与这三个词项相对应的分量。另一方面，文档中可能会包含成千上万的词项，而文档向量中的每一个非零向量就与一个词项一一对应。然而，文档向量的长度是与查询无关的。那么，文档向量的长度就可以预计算，与其他的文档描述信息一起保存到词频索引或者位置索引中，或者可以事先将文档向量归一化，用归一化之后的向量来取代位置信息列表中的

词频。

尽管查询请求通常都很短，但是在向量空间模型中，文档和查询请求被同一对待，这使得可以把整篇文档视为查询。公式（2-12）就可看做是计算两篇文档相似度的公式。把文档视为查询也是实现商业搜索引擎中“相似页面”和“更多相似页面”功能的一个方法。

将余弦相似度作为一种排名方法，直观上易理解且自然又简单。如果我们能够用向量恰当地表示查询和文档，那么可以根据文档与查询之间的余弦相似度来进行排名。将文档或查询表示为向量时，一个词项被赋予一个**权重**（weight），代表了对应的向量分量的值。纵观向量空间模型的发展历史，已经提出并评价了很多的权重分配公式。除少数例外，这些方法都可归纳为一个称为**“TF-IDF 权重”**（TF-IDF weight）的方法系列。

当给一个文档向量分配权重时，通过计算一个词频（$f_{t,d}$）的函数和一个逆文档频率（$1/N_t$）的函数的乘积来计算 TF-IDF 权重。当为一个查询向量分配权重时，本质上是把查询也看做是一个小文档，查询中词项的词频（q_t）用 $f_{t,d}$ 来代替。也可以用不同的 TF 和 IDF 函数来计算文档向量和查询向量的权重（这是很常见的）。



我们强调 TF-IDF 权重是词频与逆文档频率的**函数**（function）的乘积。很常见的一个错误就是直接使用 $f_{t,d}$ 来代指词频分量，这样做将会导致效率低下。

多年以来，已提出和评价了大量变形的 TF 和 IDF 函数。IDF 函数通常将文档频率和文档集中文档总数（N）关联起来。IDF 函数背后的含义就是，出现在很多文档中的词项比出现在少数文档中的词项的权重要低。在这两个函数中，IDF 函数更趋向于一种“标准形式”，

$$\mathrm{IDF}=\log(N/N_t) \tag{2-13}$$

其中用 N_t 和 N 分数的对数形式作为 IDF 函数的变形结构。

各种 TF 函数背后的含义就是，在同一文档中多次出现的词项比少数几次出现的词项的权重要高。另一个隐藏在 TF 函数定义下的考虑因素是：它的值不必随着 $f_{t,d}$ 的增长而线性增长。尽管出现两次的词项比出现一次的应赋予更高的权重，但也不必是两倍那么多。以下这个函数满足这些要求并多次出现在 Salton 后期的著作里：

$$\mathrm{TF}=\begin{cases}\log(f_{t,d})+1 & \text{如果 } f_{t,d}>0\\ 0 & \text{否则}\end{cases} \tag{2-14}$$

当这个公式用于查询向量时，通常使用词项 t 在查询 q 中的词频 q_t 来代替 $f_{t,d}$。我们使用这个公式，连同公式（2-13），来计算下面例子中的文档和查询中词项的权重。

考虑表 2-1 中的《Romeo and Juliet》文档集，以及表 2-2 给出的相应的位置信息列表。因为文档集中有 5 个文档，“sir”这个词在文档集中出现了 4 次，因此“sir”的 IDF 权重值为：

$$\log(N/f_{\mathrm{sir}}) \;=\; \log(5/4) \;\approx\; 0.32$$

在这个式子和其他用到对数的 TF-IDF 计算式中，对数的底通常是不重要的。必要时，对于该例和书中的其他例子，我们都假定底为 2。

因为“sir”在文档 2 中出现了两次，那么与“sir”对应的分量的 TF-IDF 值为

$$(\log(f_{\mathrm{sir},2})+1)\cdot(\log(N/f_{\mathrm{sir}})) \;=\; (\log(2)+1)\cdot(\log(5/4)) \;\approx\; 0.64$$

为其他分量和其他文档计算 TF-IDF 值，可以得出下面的向量：

$\vec{d_1} \approx \langle 0.00, 0.00, 0.00, 0.00, 1.32, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 1.32, 0.00, 0.32, 0.00, 1.32 \rangle$

$\vec{d_2} \approx \langle 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 1.32, 1.32, 0.00, 0.64, 0.00, 0.00 \rangle$

$\vec{d_3} \approx \langle 2.32, 2.32, 4.64, 0.00, 1.32, 2.32, 2.32, 4.64, 2.32, 2.32, 0.00, 0.00, 2.32, 0.32, 0.00, 3.42 \rangle$

$\vec{d_4} \approx \langle 0.00, 0.00, 0.00, 2.32, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 1.32, 0.00, 0.00, 0.00, 0.00, 0.00 \rangle$

$\vec{d_5} \approx \langle 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.32, 2.32, 0.00 \rangle$

其中，分量的顺序是按照它们对应的词项的字母顺序进行排列的。将这些向量除以它们的长度进行归一化，得到：

$\vec{d_1}/|\vec{d_1}| \approx \langle 0.00, 0.00, 0.00, 0.00, 0.57, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.57, 0.00, 0.14, 0.00, 0.57 \rangle$

$\vec{d_2}/|\vec{d_2}| \approx \langle 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.67, 0.67, 0.00, 0.33, 0.00, 0.00 \rangle$

$\vec{d_3}/|\vec{d_3}| \approx \langle 0.24, 0.24, 0.48, 0.00, 0.14, 0.24, 0.24, 0.48, 0.24, 0.24, 0.00, 0.00, 0.24, 0.03, 0.00, 0.35 \rangle$

$\vec{d_4}/|\vec{d_4}| \approx \langle 0.00, 0.00, 0.00, 0.87, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.49, 0.00, 0.00, 0.00, 0.00, 0.00 \rangle$

$\vec{d_5}/|\vec{d_5}| \approx \langle 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.14, 0.99, 0.00 \rangle$

如果我们想根据查询〈“quarrel”，“sir”〉对这五个文档排名，首先要构造查询向量并用相应的长度做归一化：

$\vec{q}/|\vec{q}| \approx \langle 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.97, 0.00, 0.24, 0.00, 0.00 \rangle$

计算这个查询向量和每一个文档向量的点积，得到以下余弦相似度值：

文档编号	1	2	3	4	5
相似度值	0.59	0.73	0.01	0.00	0.03

最终的文档排名为：2，1，5，3，4。

向量空间模型的查询处理过程是很直接的（如图 2-9 所示），实质上是将查询词项的位置信息列表执行合并操作。文档编号和对应的得分记录在一个数组中，每次计算出一个得分就进行累加。每次计算一个文档。在 while 循环的每一次迭代中，算法计算文档 d 的得分（对应的文档向量为 $\vec{d}$），并将文档编号及得分保存在 *Result* 数组中，然后确定下一次迭代要处理的文档编号。这个算法不会为不包含任何查询词项的文档显式地计算得分，只是给它们隐式地赋予 0 分。最后，根据 score 值对 *Result* 排名，并返回排在最前面的 k 个文档。

```
    rankCosine (⟨t_1, ..., t_n⟩, k) ≡
1       j ← 1
2       d ← min_{1≤i≤n} nextDoc (t_i, −∞)
3       while d < ∞ do
4           Result[j].docid ← d
5           Result[j].score ← (d⃗/|d⃗|) · (q⃗/|q⃗|)
6           j ← j + 1
7           d ← min_{1≤i≤n} nextDoc (t_i, d)
8       sort Result by score
9       return Result[1..k]
```

图 2-9 向量空间模型中的排名检索查询处理过程。给定词项向量〈t_1，…，t_n〉（与查询向量 $\vec{q}$ 对应），该函数返回前 k 个文档

对于许多检索应用而言，是不需要得到所有文档的排名列表的。相反，我们返回最多 k 个文档，这里 k 的取值根据应用环境的需要而定。例如，Web 搜索引擎在第一页仅可能返回前 $k=10$ 或者 20 个结果。当只需返回前 k 个文档时，为每个包含任何查询词项的文档，

有些甚至只包含权重很低的单个词项，都计算得分是十分低效的。这个缺点促使了很多适用于向量空间模型和其他信息检索模型的改进查询处理方法的提出。第 5 章将讨论这些查询处理方法。

在本节一开始还列出了其他的一些文档特征——词频、词项邻近度、文档频率、文档长度——向量空间模型仅仅显式地用到了词频和文档频率。当把向量归一化到单位长度时，也隐式地用到了文档长度。当一个文档长度是另外一个文档长度的两倍，但以相同的比重包含相同的词项时，它们的归一化向量是一样的。向量空间模型没有考虑词项邻近度。这一特点使得它（以及其他具有相同特点的信息检索模型）被形象地称为**“词袋”**（bag of words）模型（即对于文本，这种模型忽略其词序、语法、句法，将其仅仅看做是一个词项的集合，就像装满词项的袋子一样——译者注）。

这一节介绍到的向量空间模型的版本基于 Salton 后期著作中关于向量空间模型的介绍。实际上，为了提高效率，向量空间模型的实现通常忽略长度归一化和文档向量中的 IDF 因子。而且，余弦相似度中内在的欧几里得长度归一化已被证明不适用于处理那些既包含长文档又包含短文档的文档集，需要进行大量的调整来支持这些文档集的处理。2.3 节将会评价这些方法的效率和有效性。

向量空间模型可能由于其完全启发式的性质而遭到批判。除了直观、简单的数学形式和 2.3 节中的实验，我们不再为其进行更多的辩护。后面章节（第 8、第 9 章）介绍到的信息检索模型具有更坚实的理论框架作为支撑。因此，这些模型适应性更强，也更容易扩展，以便可以适应更多的文档特征。

2.2.2 邻近度排名

上一节介绍的向量空间排名方法很明显地只取决于 TF 和 IDF。相反，这一节介绍的方法明显依赖词项邻近度。这种方法隐式地使用了词频；而文档频率、文档长度和其他一些文档特征在这里不起任何作用。

当词项向量 $\langle t_1, t_2, \cdots, t_n \rangle$ 中的各个分量在一个文档中出现的位置较为接近，而在其他文档出现的位置相隔甚远时，则表明该文档更有可能是相关的。给定一个词项向量 $\langle t_1, t_2, \cdots, t_n \rangle$，我们定义向量的一个**覆盖**（cover）为文档区间 $[u, v]$，它包含了所有的词项，并且不存在一个更小的区间 $[u', v']$，$u \leqslant u' \leqslant v' \leqslant v$，也包含了所有的词项。在 2.1.2 节介绍到候选词组中要求所有词项必须按顺序出现，这是覆盖的一个特例。

在表 2-1 中的文档集中，区间 [1:2, 1:4]，[3:2, 3:4]，[3:4, 3:8] 都是词项向量 〈“you”, “sir”〉 的覆盖。尽管区间 [3:4, 3:16] 也包含这两个词项，但它不是一个覆盖，因为它包含了 [3:4, 3:8] 这个覆盖。同理，词项向量 〈“quarrel”, “sir”〉 存在两个覆盖：[1:3, 1:4] 和 [2:1, 2:2]。

注意到覆盖有可能会交叠。但是，匹配词项 t_i 符号最多出现在 $n \cdot l$ 个覆盖中，这里 l 是向量中词项的最短位置信息列表长度。为了明白词项向量 $\langle t_1, t_2, \cdots, t_n \rangle$ 最多存在 $n \cdot l$ 个覆盖，考虑这样一个文档集，所有的词项以相同的顺序出现同样多次：

$$\cdots t_1 \cdots t_2 \cdots t_3 \cdots t_n \cdots t_1 \cdots t_2 \cdots t_3 \cdots t_n \cdots t_1 \cdots$$

我们将为何有不超过 $n \cdot l$ 个覆盖的证明留到练习 2.8。一个新的覆盖开始于向量中一个词项的出现位置。因此一个词项向量的覆盖总数受 $n \cdot l$ 的限制，而不是依赖于最长位置信息列表 L 的长度。在讨论邻近度排名的时候，我们定义 κ 为文档集中词项向量的覆盖数，且 $\kappa \leqslant n \cdot l$。

也许这并不奇怪，计算覆盖的算法与图 2-2 介绍的词组查找算法很相似。图 2-10 的函数能够计算出在给定位置之后下一个覆盖出现的位置。第 1 行，算法确定最小的位置 v，使得区间 $[position, v]$ 包含向量中的所有词项。u 之后开始的覆盖不可能早于这个位置结束。第 4 行，算法收缩以 v 结尾的区间，调整 u 使得不存在其他更小的以 v 结尾的区间也包含所有词项。第 5 行检查 u 和 v 是否位于同一个文档。如果不是，将递归调用 nextCover。

```
nextCover (⟨t1, ..., tn⟩, position) ≡
1    v ← max1≤i≤n(next(ti, position))
2    if v = ∞ then
3        return [∞, ∞]
4    u ← min1≤i≤n(prev(ti, v + 1))
5    if docid(u) = docid(v) then
6        return [u, v]
7    else
8        return nextCover(⟨t1, ..., tn⟩, u)
```

图 2-10 为词项向量⟨t_1,…,t_n⟩计算给定位置之后下一个覆盖出现的位置的函数

最后的检查是必要的，因为覆盖最终会影响一个文档的排名得分。技术上来说，区间 [1:4, 2:1] 是词项向量 〈“quarrel”, “sir”〉 一个完全可以接受的覆盖。但在模式依赖索引中，一个跨文档的覆盖不太有实际意义。

排名基于两个假设：(1) 覆盖越短，包含覆盖的文档内容越可能是相关的；(2) 一个文档包含的覆盖越多，就越有可能是相关的。这些假设来源于直觉。第一个假设暗示了一个独立覆盖的得分基于它的长度。第二个假设暗示一个文档的得分可以通过累加文档中包含的独立覆盖的得分来得到。结合这些想法，我们用如下公式对一个包含覆盖 $[u_1, v_1]$, $[u_2, v_2]$, $[u_3, v_3]$ ……的文档 d 计算得分：

$$\text{score}(d) = \sum \left(\frac{1}{v_i - u_i + 1} \right) \tag{2-15}$$

图 2-11 展示了采用邻近度排名时的查询处理过程。第 5～13 行，调用 nextCover 函数来获取覆盖，并在 while 循环中逐个处理。在第 13 行，覆盖数 k 恰好等于 nextCover 的调用次数。当跨越了一个文档的边界时（第 6 行），得分和文档编号就存入记录数组 *Result* 中（第 8～9 行）。当所有的覆盖都被处理之后，最后一个文档的信息存入数组（第 14～17 行），根据得分排列数组（第 18 行），返回排在最前面的 k 个文档（第 19 行）。

当 rankProximity 函数调用 nextCover 函数时，传进 nextCover 函数的第二参数的位置信息是严格递增的。同理，当 nextCover 函数连续调用 next 和 prev 方法时，next 和 prev 的第二参数也是严格递增的。正如 2.2.1 节中词组查找算法那样，我们利用这个性质用跳跃式搜索来实现 next 和 prev 方法。根据同样地推理，当使用跳跃式搜索时，rankProximity 算法整体时间复杂度为 $O(n^2 l \cdot \log(L/l))$。

注意，这个时间复杂度是词项向量大小 n 的二次方，因为在最坏情况下会有 $O(n \cdot l)$ 个覆盖。幸运的是，这个算法的自适应性使得问题复杂度减少了些，为 $O(n \cdot \kappa \cdot \log(L/\kappa))$。

对于得分非 0 的文档，它必须包含所有的词项。鉴于此，邻近度排名与许多商业搜索引擎是类似的。当根据查询 〈“you”, “sir”〉 对表 2-1

```
rankProximity (⟨t1, ..., tn⟩, k) ≡
1    [u, v] ← nextCover(⟨t0, t1, ..., tn⟩, −∞)
2    d ← docid(u)
3    score ← 0
4    j ← 0
5    while u < ∞ do
6        if d < docid(u) then
7            j ← j + 1
8            Result[j].docid ← d
9            Result[j].score ← score
10           d ← docid(u)
11           score ← 0
12       score ← score + 1/(v − u + 1)
13       [u, v] ← nextCover(⟨t1, ..., tn⟩, u)
14   if d < ∞ then
15       j ← j + 1
16       Result[j].docid ← d
17       Result[j].score ← score
18   sort Result[1..j] by score
19   return Result[1..k]
```

图 2-11 邻近度排名查询处理过程。图 2-10 的 nextCover 函数用来获取每一个覆盖

的文档集进行排名时，邻近度排名赋予文档 1 的得分是 0.33，文档 3 是 0.53，剩下的文档都为 0。

当查询为〈“quarrel”，“sir”〉时，为同样地文档集排名，邻近度排名方法为文档 1 和文档 2 赋予的得分为 0.50，文档 3 到文档 5 都为 0。与余弦相似度不一样，文档 2 中“sir”的第二次出现并没有使文档得分增加。单个词项的词频不是邻近度排名要考虑的因素；相反，词项共同出现的频率及邻近度才是影响因素。一个文档虽然包含了所有词项的许多匹配但仅包含一个覆盖的情况是有可能的，也就是查询词项被分散到了文档的各个地方。

2.2.3 布尔检索

除了在 Web 搜索引擎中会隐式地用到布尔过滤，在一些特别的应用领域中，如数字图书馆和法律领域内的搜索，显式支持布尔查询是很重要的。与排名检索相比，布尔检索返回的结果是文档集合而不是有序列表。在布尔检索模型中，词项 t 用于指定那些包含它的文档。标准的布尔操作符（AND、OR 和 NOT）用于构建布尔查询，因此布尔查询就可以表达为如下集合上的操作：

A AND B　　A 和 B 的交集（$A \cap B$）

A OR B　　A 和 B 的并集（$A \cup B$）

NOT A　　文档集中 A 的补集（$\overline{A}$）

其中，A 和 B 可以是词项或其他的布尔查询。例如，在表 2-1 的文档集上，查询：

（“quarrel” OR “sir”） AND “you”

就指定了集合 {1，3}，而查询

（“quarrel” OR “sir”） AND NOT “you”

指定了集合 {2，5}。

我们用来解决布尔查询的算法是图 2-2 词组查找算法和图 2-10 覆盖查找算法的一个变形。算法查找出符合布尔查询的**候选结果**（candidate solution），其中每个候选结果都由一**系列**满足这个布尔查询的文档组成，并且使得没有一个更小的文档集合也满足这个查询。当一个候选结果系列的长度为 1 时，这个文档满足查询且应该包含在结果集中。

相同的操作方法在之前的两个算法中做了介绍。在词组查找算法中，第 1～6 行确定了一个范围，包含了所有按顺序出现的词项，并且使得不存在一个更小的范围满足这个条件。在覆盖查找算法中，第 1～4 行同样确定所有的词项并使其尽可能地靠近。两个算法中都运用了一个附加的限制。

为了简化布尔搜索算法的定义，我们在布尔查询的操作上定义两个函数，扩展了模式依赖倒排索引中的 nextDoc 和 prevDoc 方法。

docRight(Q, u)——从文档 u 后找到满足 Q 的第一个候选结果的结束文档

docLeft(Q, v)——在文档 v 前找到满足 Q 的最后一个候选结果的起始文档

对于词项，我们定义：

$$\text{docRight}(t,u) \equiv \text{next Doc}(t,u)$$
$$\text{docLeft}(t,v) \equiv \text{prev Doc}(t,v)$$

对于 AND 和 OR 操作，我们定义：

$$\text{docRight}(A \text{ AND } B,u) \equiv \max(\text{docRight}(A,u),\text{docRight}(B,u))$$
$$\text{docLeft}(A \text{ AND } B,v) \equiv \min(\text{docLeft}(A,v),\text{docLeft}(B,v))$$
$$\text{docRight}(A \text{ OR } B,u) \equiv \min(\text{docRight}(A,u),\text{docRight}(B,u))$$

$$\text{docLeft}(A \text{ OR } B, v) \equiv \max(\text{docLeft}(A, v), \text{docLeft}(B, v))$$

为了确定一个给定查询的结果，这些定义可以递归使用，例如：

$$\begin{aligned}
&\text{docRight}((\text{“quarrel” OR “sir”}) \text{ AND “you”}, 1)\\
&\quad \equiv \max(\text{docRight}(\text{“quarrel” OR “sir”}, 1), \text{docRight}(\text{“you”}, 1))\\
&\quad \equiv \max(\min(\text{docRight}(\text{“quarrel”}, 1), \text{docRight}(\text{“sir”}, 1)), \text{nextDoc}(\text{“you”}, 1))\\
&\quad \equiv \max(\min(\text{nextDoc}(\text{“quarrel”}, 1), \text{nextDoc}(\text{“sir”}, 1)), 3)\\
&\quad \equiv \max(\min(2, 2), 3)\\
&\quad \equiv 3
\end{aligned}$$

$$\begin{aligned}
&\text{docLeft}((\text{“quarrel” OR “sir”}) \text{ AND “you”}, 4)\\
&\quad \equiv \min(\text{docLeft}(\text{“quarrel” OR “sir”}, 4), \text{docLeft}(\text{“you”}, 4))\\
&\quad \equiv \min(\max(\text{docLeft}(\text{“quarrel”}, 4), \text{docLeft}(\text{“sir”}, 4)), \text{prevDoc}(\text{“you”}, 4))\\
&\quad \equiv \min(\max(\text{prevDoc}(\text{“quarrel”}, 4), \text{prevDoc}(\text{“sir”}, 4)), 3)\\
&\quad \equiv \min(\max(2, 3), 3)\\
&\quad \equiv 3
\end{aligned}$$

NOT 操作符的定义更复杂些，在给出主要的算法后会介绍。

图 2-12 给出了 nextSolution 函数，在给定位置后确定布尔查询的下一个结果。这个函数调用 docRight 和 docLeft 来获得候选结果。在第 4 行之后，区间 $[u, v]$ 包含了这个候选结果。如果候选结果只包含一个文档，那么就返回。否则，这个函数被递归调用。基于这个函数，布尔查询 Q 的所有候选结果可按如下过程产生：

```
    nextSolution (Q, position) ≡
1       v ← docRight(Q, position)
2       if v = ∞ then
3           return ∞
4       u ← docLeft(Q, v + 1)
5       if u = v then
6           return u
7       else
8           return nextSolution (Q, v)
```

图 2-12 在给定位置后确定布尔查询 Q 的下一个候选结果的函数。函数 nextSolution 调用 docRight 和 docLeft 来获得候选结果。依赖于查询的结构，这些函数可能会被递归调用

```
u ← −∞
while u < ∞ do
    u ← nextSolution(Q, u)
    if u < ∞ then
        report docid(u)
```

使用 nextDoc 和 prevDoc 的跳跃式搜索实现，这个算法的时间复杂度为 $O(n \cdot l \cdot \log(L/l))$，其中 n 为查询中词项的个数。如果使用了文档编号索引或词频索引，并且索引中没有记录位置信息，l 和 L 分别为由文档数目度量的查询中词项的最短和最长位置信息列表长度。这个算法的时间复杂度的推理过程与词组查找算法和邻近度排名算法的推理过程相似。当考虑候选结果数 k 时，体现了算法的自适应性，时间复杂度变为 $O(n \cdot k \cdot \log(L/k))$。注意，算法第 4 行对 docLeft 方法的调用是可以去掉的（见练习 2.9），但是为了帮助我们分析算法的复杂度，我们还是给出了关于候选结果的明确定义。

在 docRight 和 docLeft 方法的定义中，我们忽略了 NOT 操作符。实际上，不必为了实现 NOT 操作而去专门实现这些函数的一般化版本。因此，德摩根定律可用于变换查询，将任何 NOT 操作内移，直至它直接与查询词项相连：

$$\text{NOT}(A \text{ AND } B) \equiv \text{NOT } A \text{ OR NOT } B$$
$$\text{NOT}(A \text{ OR } B) \equiv \text{NOT } A \text{ AND NOT } B$$

例如，查询：

"william"AND "shakespeare" AND NOT("marlowe" OR "bacon")

可以转换为：

"william"AND "shakespeare" AND (NOT "marlowe"AND NOT "bacon")

这个转换没有改变查询中 AND 和 OR 操作符的数量，因此不会改变查询中词项的数量（n）。在适当的使用德摩根定律之后，我们得到只包含 NOT t 这样形式的查询，其中 t 是一个词项。为了能够处理包含这种形式的表达式的查询，我们需要 docRight 和 docLeft 的相应定义。可以利用 nextDoc 和 prevDoc 来给出这些定义。

```
docRight(NOT t, u) ≡
    u' ← nextDoc(t, u)
    while u' = u + 1 do
        u ← u'
        u' ← nextDoc(t, u)
    return u + 1
```

然而，这种方法效率可能很低。当包含词项 t 的文档不多时，这个定义还是可以获得可接受的性能的，但是当有很多文档包含词项 t 时，性能就变得很难接受了，此时几乎就是线性扫描位置信息列表了，这是我们使用跳跃式搜索可以避免的。并且，docLeft(NOT t, v)的等价实现需要在位置信息列表中进行**反向扫描**（backward），违背了跳跃式搜索所需条件，因此无法利用其优点。

因此，我们直接在 2.1.2 节介绍的数据结构上实现 NOT 操作符，用 nextDoc(NOT t, u)和 prevDoc(NOT t, v)来扩展倒排索引支持的方法。我们将细节留到练习 2.5。

2.3 评价

我们对余弦相似度和邻近度排名方法的陈述大部分依赖于直觉。我们用直觉来说明这些做法都是正确的——将文档和查询表示为向量，通过比较向量之间的夹角来决定相似度，当词项出现得更为频繁或更靠近时我们对其赋予更高的权重。只有当这些方法在实践中是有效的时候，这种对直觉的依赖才是可接受的。并且，检索算法的实现必须在用户认为合理的时间内对典型查询计算出结果，还必须在有效性和效率之间达到平衡。比起一个立即就能得到的结果，用户不会为了得到更好一点的结果而去等更长的时间——哪怕只是几分几秒。

2.3.1 查全率和查准率

检索方法的有效性度量取决于人们对相关性的评估。有些情况下，这些评估隐式地体现在用户的行为中。例如，用户点击了一个查询结果，却又很快退了出来，我们由此可推测这个结果并不好。然而，许多公开发表的信息检索实验都是为了实验目的而进行的人工评估，例如第 1 章介绍的 TREC 实验。这些评估通常是二值的——评估者阅读文档并判断它与某个主题是**相关**（relevant）或**不相关**（not relevant）。TREC 实验通常就是使用这样的二值评判，一个文档只要有任何部分与主题相关，则可判断它是相关的。

例如，给定 TREC 主题 426 描述的信息需求（图 1-8），用户可以构造以下布尔查询（("law" AND "enforcement") OR "police") AND ("dog" OR "dogs")。

在TREC45文档集上执行这个查询会产生881个文档，约占拥有50万文档的文档集的0.17%。

为了确定类似这样的一个布尔查询的有效性，我们比较两个集合：(1) 查询返回的文档集 *Res*，(2) 文档集中与这个主题相关的文档子集 *Rel*。根据这两个集合，我们计算两个标准有效性指标：**查全率**（recall）和**查准率**（precision）。

$$\text{查全率}=\frac{|Rel \cap Res|}{|Rel|} \tag{2-16}$$

$$\text{查准率}=\frac{|Rel \cap Res|}{|Res|} \tag{2-17}$$

概括地说，查全率反映了有多少相关文档出现在结果集中，而查准率反映了结果集中相关文档的比重。

根据官方NIST的评价，TREC45测试集中有202个文档与主题426相关。我们的查询返回了其中的167个，即查准率为0.190，而查全率为0.827。用户可能会觉得这样的结果是可以接受的。只有35个相关文档没有被返回。然而，为了找到一个相关文档，用户必须平均阅读4.28个不相关文档。

有时候会看到查全率和查准率被组合为一个简单的值，称为 ***F*-测度**（F-measure）。最简单的 *F*-测度——F_1，是查全率和查准率的调和平均：

$$F_1 = \frac{2}{\frac{1}{R}+\frac{1}{P}} = \frac{2 \cdot R \cdot P}{R+P} \tag{2-18}$$

其中，R 代表查全率，P 代表查准率。与算术平均值$((R+P)/2)$相比，调和平均能够在查全率和查准率之间达到一个平衡。例如，如果我们返回整个文档集作为查询结果，那么查全率将为1，查准率则几乎接近0。对于这个结果，算术平均值大于0.5，而调和平均值为$2P/(1+P)$，当 P 接近0时，这个值接近0。

这个式子也可一般化为**加权调和平均**（weighted harmonic mean），使得可以更加侧重查全率或查准率。

$$\frac{1}{\alpha\frac{1}{R}+(1-\alpha)\frac{1}{P}} \tag{2-19}$$

其中，$0 \leqslant \alpha \leqslant 1$。$\alpha=0$ 时，这个测度等价于查准率；$\alpha=1$ 时，等价于查全率。当 $\alpha=0.5$ 时，等价于公式（2-18）。根据标准实验（van Rijsbergen，1979，第7章），我们设置 $\alpha=1/(\beta^2+1)$，而且定义F-测度为：

$$F_\beta=\frac{(\beta^2+1) \cdot R \cdot P}{\beta^2 \cdot R+P} \tag{2-20}$$

其中，β 可为任何实数。因此，F_0 为查全率，F_∞ 为查准率。$|\beta|<1$ 时侧重查全率；$|\beta|>1$ 时侧重查准率。

2.3.2 排名检索的有效性指标

如果用户只想阅读一两个相关文档，那么排名检索比布尔检索更能提供一个有用的结果集。为了将查全率和查准率的概念扩展到排名检索算法返回的有序列表中，我们考虑由查询返回的前 k 个文档 $Res[1..k]$，且有如下定义：

$$\text{recall@}k=\frac{|Rel \cap Res[1..k]|}{|Rel|} \tag{2-21}$$

$$\text{precision@}k=\frac{|Rel\cap Res[1..k]|}{|Res[1..k]|} \tag{2-22}$$

其中，precision@k 通常也记为 P@k。如果我们将主题 426 的标题看做是一个用于排名检索的词项向量，〈“law”，“enforcement”，“dogs”〉，用邻近度排名方法可得到 P@10＝0.400，recall@10＝0.019 8。如果用户从上而下阅读列表，那么在前 10 个文档里面会发现 4 个相关文档。

通过定义，我们可以看出 recall@k 随着 k 单调增加。相反，如果排名检索方法遵循第 1 章介绍的概率排名原则（即按照相关性从大到小对文档进行排名），那么 P@k 随着 k 的递增而减小。对于主题 426，由邻近度排名得到：

k	10	20	50	100	200	1000
P@k	0.400	0.450	0.380	0.230	0.115	0.023
recall@k	0.020	0.045	0.094	0.114	0.114	0.114

因为只有 82 个文档包含了查询中的所有词项，因此邻近度排名方法不能返回 200 或者 1000 个得分大于 0 的文档。为了能与其他排名方法做比较，我们假定排在后面的文档都是不相关的，从而计算查全率和查准率。从整体来看，用户也许更满意上一小节中布尔查询的结果。但是这样比较当然是不公平的。布尔查询包含不在词项向量中的词项，而且可能需要更多的工作量。对于同样的词项向量，由余弦相似度排名方法得到：

k	10	20	50	100	200	1000
P@k	0.000	0.000	0.000	0.060	0.070	0.051
recall@k	0.000	0.000	0.000	0.030	0.069	0.253

余弦相似度排名方法的查询结果相当糟糕，用户不太可能会对这样的结果满意。

通过改变 k 值大小，我们可以调节查准率和查全率，为获得更多相关文档而接受一个低查准率或是相反。信息检索实验可以考虑一系列的 k 值，当用户希望很快得到结果并且只需要少量结果时，可取较低的 k 值。当用户希望发现尽可能多的信息并愿意对结果列表进行深入探究时，可取较高的 k 值。第一种情况在 Web 搜索中很常见，通常用户只会查看最前面的 1～2 个结果就去尝试别的了。第二种情形通常发生在法律领域内，也许会有蛛丝马迹令案件峰回路转，因此彻底的搜索是非常必要的。

从上面的例子中我们可以看到，即便是 k＝1000，查全率也还是明显低于 100%。除了提高 k 值，我们还需深入分析结果集来尝试显著地提高查全率。而且，因为许多相关文档也许根本不包含查询中的任何词项，因此如不返回那些得分为 0 的文档，查全率一般是不可能达到 100%的。为了简单而且一致，信息检索的实验通常只考虑一定数量的文档，通常取前 k＝1000 个结果。在更高层次的处理中，我们简单地认为查准率为 0。当在做实验的时候，我们将这个值传递给 k 作为检索函数的参数，如图 2-9 和 2-11 所示。

为了研究查全率和查准率之间的平衡关系，我们绘制**查全率-查准率曲线**。图 2-13 展示了邻近度排名的三个例子。图中描绘的是主题 426 和其他两个取自 1998 TREC 特定任务的主题：主题 412（“airport security”）和主题 414（“Cuba，sugar，exports”）的曲线。11 个查全率点以 10%的增量从 0%递增到 100%，曲线给出了在某个查全率水平上可达到的最大查准率。当查全率为 0%时意味着在任何查全率水平上都可获得最高的查准率。因此，主题

412可获得的最高查准率为80%，主题414可获得的最高查准率为50%，主题426可获得的最高查准率为100%。当查全率为20%或者更高时，主题412的邻近度排名的查准率高达57%，主题414的查准率为32%，但主题426的查准率为0%。这种在某一给定查全率水平或之上给出最大查准率的技术称为**插值查准率**（interpolated precision）。插值的方法具有能够产生单调递减曲线的特性，可以很好地描述查全率和查准率之间的平衡关系。

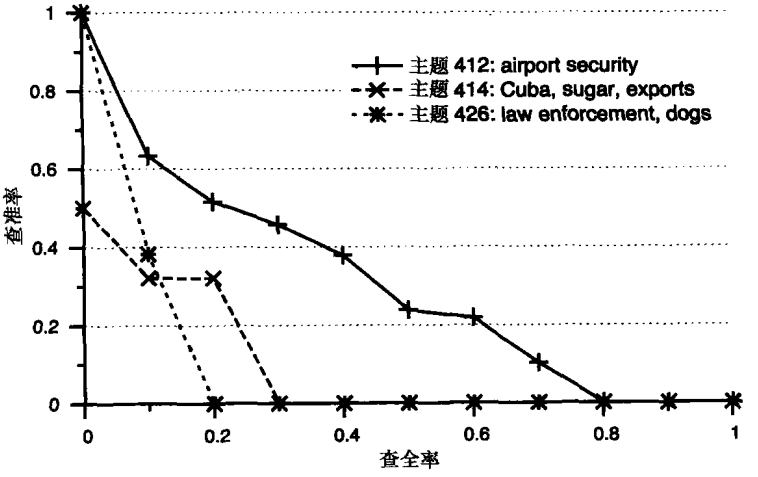


图2-13 在TREC45文档集上，3个TREC主题得到的11个点的插值查全率-查准率曲线。结果通过邻近度排名产生

为了描述检索在一系列查全率上的效果，我们可以计算**平均查准率**（average precision）。定义如下：

$$\frac{1}{|Rel|}\cdot\sum_{i=1}^{k}\text{relevant}(i)\times \text{P@}i \tag{2-23}$$

其中，如果排名为 i 的文档是相关的（即如果 $Res[i]\in Rel$），则 relevant(i)=1，否则为0。平均查准率代表了位于（非插值）查全率-查准率曲线下方区域的一个近似。在主题426中返回的前1000个文档中，邻近度排名可以达到0.058的平均查准率；余弦相似度可达到0.016的平均查准率。

目前为止，我们仅考虑了单个主题上的有效性指标。自然地，单个主题上的性能不足以说明什么，一个典型的信息检索实验要考察50个或更多个主题。在一组主题上计算有效性指标的标准过程，是计算单个主题上的指标然后取它们的算术平均。在研究信息检索的文献中，P@k、recall@k 和其他指标，包括查全率-查准率曲线在内，都是一组主题上的平均值。除非作者想要讨论这些主题的某些指定特征，否则很少会看到单个主题上的值或是曲线。在使用平均查准率时，为了避免可能与平均P@k值混淆，明确地称一组主题上的算术平均为**平均查准率均值**（Mean Average Precision，MAP）。

因为MAP综合考察了系统在全系列查全率上的性能，且频繁地被使用在TREC和其他一些评价论坛中，所以直到几年前在信息检索文献的检索实验里关于MAP值的报告几乎无处不在。最近，随着MAP许多缺点不断地变得明显以及其他的指标开始逐步普及，MAP逐渐退为其次。

遗憾的是，由于MAP是平均值的平均，因此从用户能体验的实际性能上很难对MAP给出明确、直观的解释。尽管如P@10这样的指标提供的关于整体系统性能的信息并不多，

但它更好理解。因此在本书中，我们在实验中将同时报告 P@10 和 MAP。在第Ⅲ部分，我们探讨其他的有效性指标，并将其与查准率、查全率和 MAP 进行比较。

为了简单起见，同时也为了与已发表的结果保持一致，我们不建议自己写代码来计算有效性指标。NIST 提供了一个程序 trec_eval[⊖]，可计算一系列标准指标，包括 P@k 和 MAP。这个程序是计算 TREC 报告的实验结果的标准工具。Chris Buckley 是 trec_eval 的创建者和维护者，会定期更新这个程序，也会包括在文献中出现的新指标。

有效性结果

表 2-5 展示了在四个测试集上不同的检索方法的 MAP 和 P@10 值。第一行是 2.2.1 节介绍的余弦相似度排名方法的评价结果。第 2 行是 2.2.2 节介绍的邻近度排名方法的评价结果。

表 2-5 本书中讨论的几个检索方法的有效性指标

方法	TREC45				GOV2			
	1998		1999		2004		2005	
	P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
余弦 (2.2.1)	0.264	0.126	0.252	0.135	0.120	0.060	0.194	0.092
邻近度 (2.2.2)	0.396	0.124	0.370	0.146	0.425	0.173	0.562	0.230
余弦 (原始TF值)	0.266	0.106	0.240	0.120	0.298	0.093	0.282	0.097
余弦 (TF文档)	0.342	0.132	0.328	0.154	0.400	0.144	0.466	0.151
BM25 (Ch. 8)	0.424	0.178	0.440	0.205	0.471	0.243	0.534	0.277
LMD (Ch. 9)	0.450	0.193	0.428	0.226	0.484	0.244	0.580	0.293
DFR (Ch. 9)	0.426	0.183	0.446	0.216	0.465	0.248	0.550	0.269

正如我们在 2.2.1 节介绍的那样，历年来已有许多余弦相似度方法的变体。接下来两行是其中的两个的评价结果。第一个变体是用原始的 TF 值，即词项出现的频率 $f_{t,d}$ 来代替公式（2-14）中的 TF 值。在 TREC45 文档集中，这种改变降低了性能，但是在 GOV2 这个文档集上，又显著地提升了性能。第二个变体（对应表中第 4 行）忽略文档长度归一化和文档 IDF 值（在查询向量中仍考虑）。在这种变体中我们简单地通过计算非归一化文档向量和查询向量的内积来计算文档得分：

$$\text{score}(q,d)=\sum_{t\in(q\cap d)} q_t\cdot\log\left(\frac{N}{N_t}\right)\cdot(\log(f_{t,d})+1) \tag{2-24}$$

也许令人惊讶的是，这个改变能显著提升性能，使之几乎达到邻近度排名方法的水平。

我们如何解释这种性能提升呢？向量空间模型的出现和发展是在这样一个时期：文档都差不多长，通常都是短的书籍摘要或者是科学文章。将文档表示为向量的想法是这个模型蕴含的重要灵感。一旦我们将文档想象为向量，那么就不难想出下一步，即在向量上使用标准的数学操作，包括加法、归一化、内积和余弦相似度。很遗憾的是，当文档集中的长度各不相同时，向量归一化就不太适用了。因为这个原因，到了 20 世纪 90 年代早期，最后一个向量空间模型的变形就成为了标准。

本章之所以介绍向量空间模型更多的是因为它比其他模型历史更悠久。由于它的长期影响，任何信息检索的教科书都不会少了它。在后面的章节里，我们会介绍其他具有不同理论基础的排名检索方法。表 2-5 中的最后 3 行给出了其中三种方法的评价结果：基于概率模型

⊖ 参见 trec.nist.gov/trec_eval。

的方法（BM25），基于语言模型的方法（LMD）和基于随机多样性的方法（DFR）。表 2-6 给出了这些方法的计算公式。从表中可以看出，这三种方法仅仅依赖于在 2.2 节开始部分介绍的简单特征。它们都表现出比余弦相似度和邻近度排名更好的性能。

表 2-6 后面章节要讨论的排名公式。在这些公式中，q_t 代表查询词项的频率，即词项 t 出现在查询中的次数。BM25 中的 b 和 k_1 以及 LMD 中的 μ 都是可变的参数。在我们的实验中，$b=0.75$，$k_1=1.2$，$\mu=1000$

方法	公式
BM25 (Ch. 8)	$\sum_{t\in q} q_t \cdot (f_{t,d} \cdot (k_1+1))/(k_1 \cdot ((1-b)+b \cdot (l_d/l_{\text{avg}})) + f_{t,d}) \cdot \log(N/N_t)$
LMD (Ch. 9)	$\sum_{t\in q} q_t \cdot (\log(\mu + f_{t,d} \cdot l_C/l_t) - \log(\mu + l_d))$
DFR (Ch. 9)	$\sum_{t\in q} q_t \cdot (\log(1+l_t/N) + f'_{t,d} \cdot \log(1+N/l_t))/(f'_{t,d}+1)$， 其中 $f'_{t,d} = f_{t,d} \cdot \log(1+l_{\text{avg}}/l_d)$

图 2-14 描绘了 1998 年 TREC 特定任务的 11 点插值查全率-查准率曲线图，与表 2-5 中的第 4 行和第 5 行对应。LMD 和 DFR 方法看上去略好于 BM25，而 BM25 又好于邻近度排序方法和 TF docs 版本的余弦相似度方法。

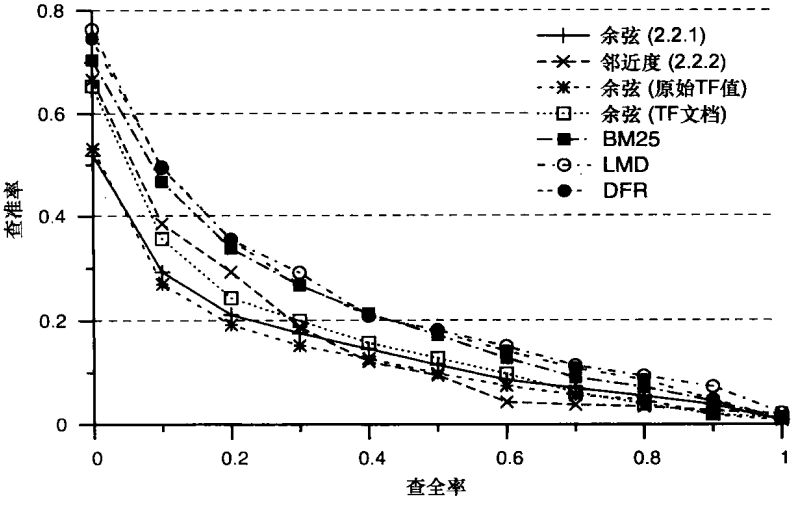


图 2-14 1999 TREC 特定任务（TREC45 文档集中主题 401～450）上多个基本检索方法的 11 点插值查全率-查准率曲线图

2.3.3 创建测试集

检索方法的有效性评价取决于人们对相关性的评估。给定一个主题和一个文档集合，评价者阅读每篇文档然后做出判断：相关或是不相关？如果主题很好理解，评价可以进行得很快。一旦加快速度，评价者就可在 10 秒甚至更短时间内做出判断。遗憾的是，就算以每 10 秒做出一个判断的速度，除休息日和假期评价者每天工作 8 小时，在 TREC45 这个拥有 50 万文档的集合上完成一个主题的判断也要将近 1 年的时间，如果换成大小为 2500 万的 GOV2 文档集，那么评价者可能要耗上她的全部职业生涯。

考虑到在整个文档集中判断一个主题的困难性，TREC 和其他的检索实验依赖于一种称为**文档池**（pooling）的技术来限制所需的判断次数。标准的 TREC 的实验流程是接受每个参与特定任务的各个小组的一个或多个**运行实例**（runs）。每一个运行实例包含与某个主题

相关的前 1000 或 10 000 个文档。文档池方法从每个运行实例中取出前 100 个左右的文档**组成**（pooled）一个文档池，让评价者随机抽取进行评价。就是有几十个参与者，文档池中每个主题的文档数量也不会超过 1000，因为不同的系统有可能返回同一个文档。即便评判进行得非常仔细，每一个判断平均都需要好几分钟，一个评价者也还是有望在不到一个星期内完成工作。通常所花的时间还更少。

文档池方法的目标就是在保证得到合理的查全率和查准率的情况下，减少评价的工作量。对于在池中加入了文档的运行实例，P@k 和 recall@k 值至少在池子里是准确。这个池子外的许多文档还是需要被评估的，因为其他的运行实例对超出池子深度外的文档进行了排名。MAP 仍然在每个运行实例的 1000 或 10 000 个文档上计算，**没有被评估的文档被认为是不相关的**（unjudged documents treated as not relevant）。因为排在前面的文档对 MAP 值的影响更大，所以用这种方法来进行估计的准确性仍然是可接受的。

根据文档池方法，TREC 一个测试集的创建和类似的效果评价可按以下步骤进行：

1）从公开的数据源，或者通过与文档集的拥有者达成协定，来获取一个恰当的文档集。

2）形成至少 50 个主题，这个最小值通常对于一个有意义的评价来说是可接受的。

3）将主题分发给任务参与者，接收他们实验的每个运行实例的结果。

4）建立缓冲池，判断文档与主题的相关性，将结果返回给参与者。

当生成主题时，在标准的信息检索系统上对主题做一次预先测试，可以在靠前的那些返回结果中看出相关文档和不相关文档的大致合理的分布。如果相关文档太多，会很容易使系统的 MAP 值达到 1.0，这样就很难区别系统之间的差别。如果相关文档太少，很多系统难以使 MAP 值大于 0.0，一样难以区别不同系统。借用《The Story of the Three Bears》中 Goldlocks 的标准，我们希望不要太多，不要太少，合适就行。

理想情况下，测试集应该是可重用的，研究者开发了新的检索方法后，可用统一的文档、主题和评判方法来决定如何将新方法和旧方法进行比较。尽管文档池方法使得我们不用花上一辈子的时间去创建这么一个测试集，但你还是会对产生这样一个可重用的测试集的能力有一些合理的担心。一个真正新颖的检索方法可能会检索到很多的相关文档，它们既不是原来缓冲池的一部分也没有被评判过。如果未被评判的文档因为计算评价指标而被视为是不相关的，这个新颖的方法也许会得到过分苛刻的得分。更悲剧的是，它因此甚至会被研究者抛弃而从此消失在科学界。解决这个问题及相关问题的另一个有效性指标将在第 12 章介绍。

2.3.4 效率指标

除了需要评价一个系统的检索有效性，评价系统的效率通常也是有必要的，因为它影响系统运行的代价以及用户对系统的满意程度。从用户的角度看来，唯一有用的效率指标是**响应时间**（response time），即从输入查询到收到结果之间的时间。**吞吐量**（throughput）是指给定时间段内系统处理查询的平均数量，这是搜索引擎运营商关心的主要指标，特别是当搜索引擎同时被多个用户使用并需要每秒中处理成千上万个查询的时候。必须要有足够的资源来应付由多个用户产生的查询负载，以确保不影响每个用户需要等待的响应时间。难以忍受的长时间等待会让用户不满，最终会导致用户流失。

实际测量的吞吐量以及它和响应时间之间的平衡需要对查询负载进行详细的模拟。为了简化效率的度量，我们主要关注响应时间和处理单个用户在一个无负载的搜索引擎上发起的一次查询的性能。第 13 章更广泛地讨论了系统的查询效率，并更详细地讨论了吞吐量。

测量响应时间的一个简单但合理的过程是执行一个完整的查询集，捕捉第一个查询恰好

开始被处理的明确时刻和最后一个结果产生后的结束时间。这个在整个集合上的时间除以查询的数量就等于每个查询的**平均响应时间**（average response time）。结果在产生之后就丢弃了，不需要将其进行存储或传输到网络中去，因为这些操作的开销很大（从而影响响应时间的计算——译者注），特别对于小的文档集更是如此。

在执行查询集之前，应重启或重置信息检索系统，将为上一个查询进行的预计算或存储在内存中的信息清除，操作系统的 I/O 缓存也应刷新。为了提高测量的准确性，查询集会被执行多次，每一次系统都要重置，多次执行时间的平均值就用于计算平均响应时间。

作为一个例子，表 2-7 用 Okapi BM25 排名函数（见表 2-6）的 Wumpus 实现，比较了模式独立索引和词频索引的平均响应时间。我们在这个例子中用到了 Okapi BM25，是因为这个排序函数的 Wumpus 实现为了效率进行了明显的调整。

表 2-7 Okapi BM25 的 Wumpus 实现（第 8 章）的查询平均响应时间，使用了两种不同的索引类型和四个不同的查询集

索引类型	TREC45		GOV2	
	1998	1999	2004	2005
模式独立索引	61 ms	57 ms	1686 ms	4763 ms
词频索引	41 ms	41 ms	204 ms	202 ms

使用词频索引具有明显的效率优势，特别是在大的 GOV2 文档集上。使用模式独立索引需要在线时间计算文档和词项的统计信息，而这些在词频索引中是预计算好的。对于用户来说，202 ms 的响应时间是很快的，而 4.7s 的响应时间就显得慢了。然而使用词频索引，词组查找或排名函数都不能用于其他的文档（因为一个文档上预计算的统计信息对于别的文档就不适用了——译者注）。

表中给出的效率指标以及本书中的吞吐量，都是在 AMD Opteron 处理器（2.8 GHz），2 GB RAM 的机架式服务器上运行得到的结果。附录 A 给出了这个计算机系统详细的性能参数。

2.4 总结

本章涵盖了很多主题，而且都介绍得很详细。几个重点内容如下：

- 我们将倒排索引视为是一种抽象数据类型，可通过表 2-4 中的方法和定义来访问。它有四种重要的变形——文档编号索引、词频索引、位置索引以及模式独立索引——区别在于它们存储的信息类型和格式。
- 多个检索算法——如本章介绍的词组查找算法、邻近度排名算法和布尔查询处理算法——都可以使用跳跃式搜索高效地实现。这些算法都是自适应的，它们的时间复杂度取决于数据的特征，如候选词组的个数。
- 排名检索和布尔过滤在当今的信息检索系统中都非常重要。排名检索的合理方法依赖于简单的文档和词项统计信息，如词项频率（TF）、逆文档频率（IDF）以及词项邻近度。著名的余弦相似度方法将文档和查询表示成向量，根据文档向量和查询向量之间的夹角大小对文档进行排名。
- 查全率和查准率是广泛使用的两个有效性指标。它们之间是相互牵制的，例如查全率的提高会导致查准率的下降。在一系列的查全率水平上评价系统的整体有效性时，平均查准率均值（MAP）是一个标准的方法。整本书中 MAP 和 P@10 是最主要的

有效性指标。

- 响应时间代表了用户能感知的信息检索系统的效率。我们通过顺序处理查询，即一次读入一个查询，在开始下一个查询之前报告这个查询的结果，来对最小响应时间做出合理估计。

2.5 延伸阅读

倒排索引长期以来都是信息检索系统（Faloutsos，1985；Knuth，1973，第 552 页-第 554 页）底层实现的标准数据结构。也有其他的数据结构被提出，通常是为特殊的检索操作提供更有效的支持。然而，这些数据结构没有像倒排索引那样的灵活性和一般性，因此几乎都不再使用了。

长期以来签名文件被看做是倒排索引最重要的竞争对手，特别当磁盘和内存很珍贵的时候（Faloutsos，1985；Faloutsos 和 Christodoulakis，1984；Zobel 等人，1998）。签名文件通过快速消除不匹配查询的文档来高效地支持布尔查询。它们用一个方法实现了如 2.2 节讲的两步检索过程的过滤步骤。遗憾的是，签名文件会找到错误的匹配，而且不能通过简单的扩展来支持词组查询和排名检索。

后缀树（Weiner，1973）是一棵查找树，从根到叶子的一条路径对应文档集中一个唯一的后缀。后缀数组（Gonnet，1987；Manber 和 Myers，1990）是一个指针数组，指向文档集中按字母顺序存放的后缀。后缀树和后缀数组都是为了支持高效的词组查找，以及例如字母顺序范围搜索和结构化检索等操作而设计的（Gonnet 等人，1992）。遗憾的是，这两种数据结构对排名检索的支持都很差。

跳跃式搜索最早由 Bentley 和 Yao 提出（1976）。本章介绍的词组查找、邻近度排名和布尔检索都可看做是这种算法的变形，它们在有序列表上计算集合操作。“galloping”这个词由 Demaine 等人（2000）创造，他提出和分析了求集合并、交、差的自适应算法。Barbay、Kenyon（2002）和 Baeza-Yates（2004）提出了其他在有序列表上求交集的算法。我们通过少数简单操作将倒排索引抽象成 ADT 是源于 Clarke 等人（2000）的想法。本章介绍的邻近度排名算法是他们论文中算法的一个简单版本。

向量空间模型发展到本章介绍的版本经历了一个漫长的过程，至少可以追溯到 Luhn 在 20 世纪 50 年代的一些工作（Luhn，1957，1958）。这个模型长期的成功和影响很大程度上是由于 Salton 和他在 Cornell 大学的研究小组的努力，并且最终成为了他们 SMART IR 系统的关键元素，这个系统的第一版于 1964 年的秋天开始运营（Salton，1968，422 页）。直到 20 世纪 90 年代早期，SMART 仍然是信息检索研究领域中为数不多的可用平台之一，在这个时期，许多非 Salton 小组的研究者都将其作为自己研究工作的基础。

直到 20 世纪 90 年代早期第一次 TREC 实验的时候，向量空间模型已经发展到很接近本章介绍的这种形式（Buckley 等人，1994）。一个更进化的版本，于 1995 年第四次 TREC 会议中被提出，是文档长度调整方法的一个补充称为**旋转文档长度归一化**（pivoted document length normalization（Singhal 等人，1996））。这个版本被排名检索接受为其向量空间模型的核心部分，但不适用于其他应用，例如聚类和分类。1995 年 Salton 去世之后，SMART 系统在 Buckley 的指导下继续发展，并在十余年后仍是一个非常具有竞争力的研究系统（Buckley，2005）。

潜在语义分析（Latent Semantic Analysis，LSA）是向量空间模型一个重要的且众所周知的扩展（Deerwester 等人，1990），我们在本书中没有介绍。LSA 应用线性代数中的奇异

值分解，来降低词项向量空间的维度。目的是为了减少同义词带来的副作用——很多词项都是同样的意思——可以将相关的词归并到同一维上。概率潜在语义分析（Probabilistic Latent Semantic Analysis，PLSA），是一个使用概率模型达到同样目的的相关技术（Hofmann，1999）。遗憾的是，由于难以高效实现，LSA 和 PLSA 在信息检索系统上的广泛应用受到了限制。

Spam 歌有可能是对计算机术语影响最大的一首歌。这首歌首次出现在《Monty Python's Flying Circus》第二季的第 12 集中，于 1970 年 12 月 15 日由 BBC 电视台播出。

2.6 练习

练习 2.1 简化 next（"the"，prev（"the"，next（"the"，$-\infty$）））。

练习 2.2 考虑图 2-2 中的词组查找算法的以下版本：

nextPhrase2 $(t_1 t_2 ... t_n, position) \equiv$
  $u \leftarrow$ **next**$(t_1, position)$
  $v \leftarrow u$
  for $i \leftarrow 2$ **to** n **do**
    $v \leftarrow$ **next**(t_i, v)
  if $v = \infty$ **then**
    return $[\infty, \infty]$
  if $v - u = n - 1$ **then**
    return $[u, v]$
  else
    return nextPhrase2$(t_1 t_2 ... t_n, v - n)$

这个算法与图 2-2 的版本相比，调用 next 方法的次数一样但不需要调用 prev 方法。解释这个算法是如何运行的。它如何在给定位置之后找到一个词组的下一次出现？如果结合跳跃式搜索来查询一个词组的所有出现，你认为是否会比原版算法获得更高的效率？解释一下。

练习 2.3 使用倒排索引 ADT 中的方法写一个算法，查找台词段（〈SPEECH〉…〈/SPEECH〉）所在的位置区间。使用如图 2-1 的模式独立索引，剧本如图 1-3 所示。

练习 2.4 使用倒排索引 ADT 中的方法写一个算法，查找莎士比亚戏剧集中 witch 的台词所在的位置区间。使用如图 2-1 的模式独立索引，剧本如图 1-3 所示。

练习 2.5 假定在莎士比亚戏剧集上有一个模式独立的倒排索引，我们将每一部戏剧视为一个文档。

（a）仅使用 first、last、next 和 prev 方法写一个算法计算以下统计量：（i）N_t，文档集中包含词项 t 的文档数；（ii）$f_{t,d}$，文档 d 中词项 t 出现的次数；（iii）l_d，文档 d 的长度；（iv）l_{avg}，文档的平均长度；（v）N，总文档数。

（b）仅使用 first、last、next 和 prev 方法写出 docid、offset、firstDoc、lastDoc、nextDoc 以及 prevDoc 的实现算法。

为简单起见，你可将每个〈PLAY〉标签的位置当做是文档编号，因为文档从这个位置开始；这样就不需要顺序地为文档编号赋值了。例如，《Machbeth》的文档编号为 745142（因为"fisrt Witch"第一次出现在[745142 : 265, 745142 : 266]）。

练习 2.6 当位置信息列表存储在磁盘而不是完全存储在内存中时，倒排索引方法调用的时间很大程度大依赖于访问磁盘所需的开销。当执行一个词组查找如"Winne the Pooh"时，词组同时包括频繁词项和不频繁词项，为 winne 的第一次方法调用（next（"winnie"，$-\infty$）），可能需要将整个位置信息列表从磁盘中读入内存数组。其他的调用可在该数组中用跳跃式搜索返回结果。同样地，"pooh"的位置信息列表也会在它的第一次方法调用的时候被读入内存。可是，"the"的位置信息列表可能无法整个放入内存，目标词组的出现位置也许只是这些位置中的一小部分。

理想情况下，"the"的所有位置不需要从磁盘中读取。调用方法可以一次读取位置信息列表中的一小部分，遍历列表会产生多次磁盘访问，在合适的地方可以跳过列表的一部分。

因为每个为“the”进行的方法调用都会产生一个磁盘访问，因此一个可取的方法是查找词组“Winnie the Pooh”时，一旦定位“winnie”的出现位置后立即检查“pooh”的出现位置。如果“pooh”没有出现在预期位置，我们就丢弃“winnie”的这次出现并继续寻找它的下一次出现，这样就不需要检查“the”是否出现在这两个词项中间。只有当我们确定“winnie”和“pooh”出现在正确的位置（“winnie___pooh”），我们才需要为“the”调用方法。

一般说来，给定一个词组“$t_1\,t_2\cdots t_n$”，搜索算法可以从最不频繁的词项逐个处理到最频繁的词项，一旦发现漏了某个词项，就立即放弃目前这次搜索，这样才可能将为最频繁的词项进行的方法调用减少到最小次数。用倒排索引 ADT 中的基本定义和方法，完善这个算法的细节。这个算法调用 next 和 prev 方法的时间复杂度是多少？如果使用了跳跃式搜索来实现这些方法，那么整体时间复杂度又是多少？

这个算法是自适应的吗？（提示：考虑在 2.1.1 节最后的文档集上搜索“hello world”，要调用多少次方法？）

练习 2.7 协调水平（Coordination level）是指向量〈t_1，…，t_n〉出现在一个文档中的词项个数。它的取值范围从 1～n。使用倒排索引 ADT 中的方法，写一个根据文档的协调水平对文档进行排名的算法。

练习 2.8 证明词项向量〈t_1，…，t_n〉最多有 $n \cdot l$ 个覆盖（见 2.2.2 节），其中 l 是向量中词项最短位置信息列表的长度。

练习 2.9 根据练习 2.2 的启示，为图 2-10 中的算法设计一个新的版本，使得新算法中无需调用 docLeft 方法。

练习 2.10 如果 $\alpha=1/(\beta^2+1)$，证明公式（2-19）和公式（2-20）是等价的。

练习 2.11（项目练习） 为 2.1.2 节介绍的倒排索引实现一个面向内存基于数组的版本，包括 next 方法的三个版本：二分查找、线性扫描和跳跃式搜索。使用你实现的倒排索引去实现 2.1.1 节的词组查找算法。

为了测试你的词组查找算法，我们建议使用 256 MB 或稍微大些的文档集，以便能够放得进你的计算机内存中。练习 1.9 产生的文档集（或者是它的子集）大小合适。从你的文档集中选出至少 10 000 个长度不同的词组，来验证你的算法能正确定位这些词组。你的选择应包括长度为 2～3 的短词组，长度为 4～10 的稍长的词组，以及长度为 100 或更长的长词组。还应包括频繁和不频繁的词项。至少一半的词组应该至少包含一个非常常用的词项，例如冠词或介词。

根据 2.3.4 节的指引，用你实现的三个版本的 next 方法来比较词组查找的效率。计算平均响应时间。为三个版本对着词组长度绘制响应时间。为了更清晰，你应该将线性扫描的结果与其他两个方法分开绘制。

选择另外一个所有长度都为 2 的词组集合。包括频繁和不频繁的词项。为三个版本对着 L（最长位置信息列表的长度）绘制响应时间。为三个版本对着 l（最短位置信息列表的长度）绘制响应时间。

选做：实现练习 2.6 中描述的词组查找算法，用这个新算法重复以上性能评价过程。

练习 2.12（项目练习） 实现 2.2.1 节中描述的余弦相似度排名算法。用练习 2.13 中产生的测试文档集或任何其他可用测试集，如 TREC 文档集，来测试你实现的算法。

练习 2.13（项目练习） 作为一个班级项目，基于 Wikipedia 构造一个测试文档集，进行一个类似于 TREC 风格的特定检索实验。继续练习 1.9 和 1.10，每个学生都贡献足够的主题，使得主题总数达到 50 个或更多。每个学生可以实现他自己的检索系统，可从一个开源的信息检索系统开始，使用后面章节介绍的技术再逐步扩展它，或是使用学生自己发明的新技术。每个学生都应在他自己的系统上将主题题目作为查询来运行。将这些结果保存进缓冲池并进行判断，每个学生都判断他创建的那个主题。交互判断界面的设计和实现也可以作为一组感兴趣学生的子项目。用 trec_eval 来比较运行结果和技术。

2.7 参考文献

Baeza-Yates, R. (2004). A fast set intersection algorithm for sorted sequences. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching*, pages 400–408. Istanbul, Turkey.

Barbay, J., and Kenyon, C. (2002). Adaptive intersection and t-threshold problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 390–399. San Francisco, California.

Bentley, J. L., and Yao, A. C. C. (1976). An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87.

Buckley, C. (2005). The SMART project at TREC. In Voorhees, E. M., and Harman, D. K., editors, *TREC — Experiment and Evaluation in Information Retrieval*, chapter 13, pages 301–320. Cambridge, Massachusetts: MIT Press.

Buckley, C., Salton, G., Allan, J., and Singhal, A. (1994). Automatic query expansion using SMART: TREC 3. In *Proceedings of the 3rd Text REtrieval Conference*. Gaithersburg, Maryland.

Clarke, C. L. A., Cormack, G. V., and Tudhope, E. A. (2000). Relevance ranking for one to three term queries. *Information Processing & Management*, 36(2):291–311.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407.

Demaine, E. D., López-Ortiz, A., and Munro, J. I. (2000). Adaptive set intersections, unions, and differences. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 743–752. San Francisco, California.

Faloutsos, C. (1985). Access methods for text. *ACM Computing Surveys*, 17(1):49–74.

Faloutsos, C., and Christodoulakis, S. (1984). Signature files: An access method for documents and its analytical performance evaluation. *ACM Transactions on Office Information Systems*, 2(4):267–288.

Gonnet, G. H. (1987). PAT *3.1 — An Efficient Text Searching System — User's Manual.* University of Waterloo, Canada.

Gonnet, G. H., Baeza-Yates, R. A., and Snider, T. (1992). New indices for text — PAT trees and PAT arrays. In Frakes, W. B., and Baeza-Yates, R., editors, *Information Retrieval — Data Structures and Algorithms*, chapter 5, pages 66–82. Englewood Cliffs, New Jersey: Prentice Hall.

Hofmann, T. (1999). Probabilistic latent semantic indexing. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 50–57. Berkeley, California.

Knuth, D. E. (1973). *The Art of Computer Programming*, volume 3. Reading, Massachusetts: Addison-Wesley.

Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317.

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2(2):159–165.

Manber, U., and Myers, G. (1990). Suffix arrays: A new method for on-line string searches. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 319–327. San Francisco, California.

Salton, G. (1968). *Automatic Information Organziation and Retrieval.* New York: McGraw-Hill.

Singhal, A., Salton, G., Mitra, M., and Buckley, C. (1996). Document length normalization. *Information Processing & Management*, 32(5):619–633.

van Rijsbergen, C. J. (1979). *Information Retrieval* (2nd ed.). London, England: Butterworths.

Weiner, P. (1973). Linear pattern matching algorithm. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11. Iowa City, Iowa.

Zobel, J., Moffat, A., and Ramamohanarao, K. (1998). Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490.

第3章 |
Information Retrieval: Implementing and Evaluating Search Engines

词条与词项

词条（token）在查询与文档之间建立了联系的桥梁。在索引过程中，信息检索系统将每个文档划分为一系列的词条序列，并且将它们插入为了搜索而建立的倒排索引中。在查询时，查询也要进行相应的分词。分词后的查询词项与倒排索引是否匹配将最终影响检索和排名的效果。

第1章介绍了对原始文本进行分词的一些简单规则：词条是由非字母数字序列隔开的字母数字序列。词条要进行**大小写规范化**（case normalized），即将大写字母转换为小写字母。此外，忽略完整XML的复杂性——简单地将XML标签如〈name〉和〈/name〉视为词条。

尽管这些简单的规则足以用来应付本书中介绍的实验，但实际应用中需要更复杂的分词规则。例如，根据简单的规则，“didn't”将被表示为词条对“didn t”，这无法与搜索词组“did not”的查询匹配。数字“10 000 000”分词后可表示为三个词条“10 000 000”，其中第一部分将错误地与查询词项“10”匹配。

并且，在使用这些简单规则时，明确地假定原始文本是用英文书写的，并采用ASCII编码。在这些规则里，“字母数字”的定义里不包含字母β和数字“八”，因为这些字符都不能用ASCII表示。即便是英文单词“naïve”也不能用ASCII值正确表示。

在大多数实际环境中，信息检索系统必须为英语以外的其他语言提供一定的支持。为了支持大多数人类语言，需要考虑每一种语言的特点，包括它们的字符集和分词规则。粗略地说，分词是将文档划分为**词**（word）的过程。尽管在人类语言中，词的概念本质上是通用的（Trask，2004），但在不同的语言中，词的特征有着明显差异。一个词不一定是一个词条，一个词条也不一定是一个词。

本章先重温一下英文的分词和词项匹配问题，接着将讨论延伸到其他语言。主要关注文档的内容，而将XML和其他文档结构的复杂性留待第5章和第16章中讨论。分词中一些微小的细节对特定查询的检索效果也会产生重大的影响。为了得到正确结果，需要谨慎考虑这些特定查询，以及在整体上衡量检索效果。

3.1节涵盖了英文中与分词紧密相关的两个传统的信息检索主题：**词干提取**（stemming）和**停词**（stopping）。词干提取通过将查询词项缩减为它们的公共词根，使得“orienteering”可以匹配“orienteers”。停词反映了对多数常用词的观察：如“the”和“I”，它们对检索而言没有什么价值。忽略这些**停用词**（stopword）能够提高检索效率并缩小倒排索引的规模。

3.2节讨论了字符编码问题，包括Unicode和UTF-8的基础知识。3.3节介绍了字符n-gram索引，它是信息检索系统能够支持新语言的基础。字符n-gram索引也可作为一种对噪音文本分词的方法，如由OCR（光学字符识别）系统产生的文本。本章最后部分讨论了许多欧洲和亚洲语言的分词方法及与此相关的难题。

3.1 英语

英语为理解分词和词项匹配提供了一个很好的切入点。作为本书的读者，对英语必须要熟悉。它是最广泛使用的语言之一，以英语为母语的人数以亿计。另有好几亿的人将其作为

第二或第三语言来学习。在 Web 上，英文内容占了绝对支配的地位。因此，对英语提供有效支持是许多信息检索系统最基本的需求。

3.1.1　标点与大写

英语存在着许多能够导致微妙的分词问题的特征。其中很多都与标点符号以及大写有关。英文中，一个特定的标点符号可以用于许多毫不相关的场合。例如前面已经提到过的如“I'll”、“it's”中的撇号，合理的分词是“I will”和“it is”，而不是“I ll”和“it s”。然而，撇号同样也可以用在如“o'clock”和“Bill's”这样的结构中。第一个最好将它看做是一个词条，而不是两个，因为与“clock”匹配并不恰当。而对于“Bill's”，如果与“bill”的匹配合理，那么，就应该作为两个词条来索引。

句点通常用于结束句子。然而，也可以出现在缩略语、数字、字母、因特网地址以及其他的内容中。“I.B.M”和“U.S.”这两个缩略语也可以写成“IBM”和“US”。信息检索系统应对这样的缩略语进行一致的识别和分词，使得查询“IBM”可以匹配以上两种形式。标点符号同样也可以作为公司或组织名称的一部分，例如“Yahoo!”和“Panic! At the Disco”。对于小部分的词项，如“C++”、“C#”和“\index”（LaTeX 中的一个命令），不正确地处理词项中的标点符号可能会使整个查询结果变得毫无用处。然而对于大多数的查询来说，可以安全地忽略标点符号。

另一方面，大小写规范化（即将索引和查询中的所有大写字母都转换成小写）会影响几乎每一个查询。在很多语言中，句子中第一个词的首字母往往是大写的，即使用户用小写输入查询（很多用户都是这么做的），也希望能找到对应的词。遗憾的是，对很多词项来说，字母的大写直接决定着这个词项的本意，因此大小写规范化将会导致潜在的危害。例如，缩略语“US”和代词“us”，或缩略语“THE”[⊖]与英语中的定冠词“the”，只有大写能将它们区分开来。

处理缩略语以及类似分词问题的一种简单的启发式方法就是**双路索引**（double index）这些词项。例如，缩略语“US”的每一次出现位置都被索引两次。倒排索引中将同时包含“us”和“US”的位置信息列表。缩略语“US”的每一次出现位置会同时记录在这两个列表中，然而代词“us”的出现位置只记录在第一个列表中（即 us 对应的位置信息列表——译者注）。包含词项“us”的查询将用第一个列表来处理，包含词项“US”（也可能是“U.S.”）的查询将用第二个列表来处理。双路索引也支持正确的人名索引，可以将人名“Bill”与“bill”（即发票）区别开来。

当为大写及标点调整分词过程时，必须非常谨慎。调整分词过程而产生的影响往往是好坏参半的。《终结者》（1984）这部电影的影迷能立即反应出“I'll be back”与“I will be back”的匹配是不恰当的。有时，全部大写的文本（WRITTEN ENTIRELY IN CAPITAL LETTERS）可能是意外或是作者用来表达自己内心情绪的一种方法。这样的文本不能看做是一个由缩略语和缩写组成的长序列。这个句子中的词项“IN”并不是指美国的印第安纳州，这篇文档也不是讲关于印第安纳州首府的。无论使用什么样的信息检索技术，都必须从有效性和效率的整体影响来评价分词。

⊖ THE 是一个操作系统，是多道程序设计的开河先锋，由图灵奖获得者 Edsger Dijkstra 和他的同事于 20 世纪 60 年代中期在埃因霍芬理工大学创建。

3.1.2 词干提取

词干提取使得例如“orienteering”能够匹配“orienteers”，或“runs”能够匹配“running”。词干提取考虑词项的**词法**（morphology）或内部结构，为了方便比较，将每一个词项缩减到词根形式。例如“orienteering”和“orienteers”可以缩减到词根形式“orienteer”。“runs”和“running”可以缩减到“run”。

在信息检索系统中，索引和查询时都会用到**词干提取器**（stemmer）。在索引阶段，每一个词条都经过词干提取器，输出的词根用于建立索引。在查询阶段，每一个查询词项也都经过同一个词干提取器处理，并与索引词项进行匹配。因此查询词项“runs”会与“running”匹配，因为这两个词有共同的词根“run”。

词干提取与语言学中的**词形归并**（lemmatization）概念相关。词形归并将每一个词项缩减到**词位**（lexeme），大致对应了作为字典条目的一个单词。每一个词位都有一个特定形式的单词，或**词元**（lemma），来代表它。词元是指查字典时所用的单词形式。例如“run”通常用来表达一组单词，包括“runs”、“running”和“ran”等。

词干提取与词形归并有时候是等同的，但是这种观点具有一定的误导性。严格地说，词干提取是一个操作过程。当词干提取器将一个词项转换成它的词根形式时，我们不直接关注这种转换的语言学效率，只关心它对特定查询检索效率的影响。

Porter词干提取器是最有名的英文词干提取器之一（Porter，1980），由Martin Porter在20世纪70年代后期提出。图3-1展示了著名的Hamlet独白开场部分在使用了Porter词干提取器前后的对比。在使用词干提取器之前，这一段文本去除了标点符号和进行了大小写规范化。正如你从图中看到的那样，这个词干提取器会产生根本不是英文单词的词根形式。例如，“troubles”词干提取结果为“troubl”。这种做法实际上不会产生什么问题，因为这些词根形式对用户来说是不可见的。查询中的词项“troubling”也会被提取为“troubl”，这样就能产生正确的匹配。

原文

To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to, 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
To sleep: perchance to dream: ay, there's the rub;

规范化后	规范化和词干提取后
to be or not to be that is the question	to be or not to be that is the question
whether tis nobler in the mind to suffer	whether ti nobler in the mind to suffer
the slings and arrows of outrageous fortune	the sling and arrow of outrag fortun
or to take arms against a sea of troubles	or to take arm against a sea of troubl
and by opposing end them to die to sleep	and by oppos end them to die to sleep
no more and by a sleep to say we end	no more and by a sleep to sai we end
the heart ache and the thousand natural shocks	the heart ach and the thousand natur shock
that flesh is heir to tis a consummation	that flesh is heir to ti a consumm
devoutly to be wish d to die to sleep	devoutli to be wish d to die to sleep
to sleep perchance to dream ay there s the rub	to sleep perchanc to dream ay there s the rub

图3-1 Hamlet独白的开始部分。最上面是原始文本，左下角是经过大小写规范化和去除标点后的版本，右下角是经过了Porter词干提取器的版本

Porter 词干提取器有时候又太过头了。“orienteering” 和 “orienteers” 都被会提取为 “orient” 而非 “orienteer”。词项 “oriental” 同样被提取为 “orient”，这样，这些词项就被混为一谈了。此外，该词干提取器不能正确处理不规则动词和复数形式。“runs” 和 “running” 都被提取为 “run”，但不规则动词形式 “ran” 还是被提取为 “ran” （没变）。类似地，“mouse” 被提取为 “mous”，但 “mice” 仍然还是 “mice”。此外，这个词干提取器只处理后缀。如 “un” 和 “re” 的前缀不会被移除。

这个词干提取器使用多组重写规则，分成几步来处理。例如，第一组重写规则（构成步骤 1a）按如下形式处理复数形式：

sses → ss
ies → i
ss → ss
s →

在运用规则时，规则左边的模式与词项的当前后缀做匹配。如果后缀匹配成功，则通过将后缀去除并替代为这个规则右边的模式来重写这个后缀。因此，应用这些规则中的第一条规则，“caresses” 将会被重写成 “caress”，应用第二条规则，“ponies” 将会被重写为 “poni”（而不是 “pony”）。列表中只有第一条匹配规则会起作用。第三条规则看上去似乎什么都没做，实际上它能够阻止那些具有 “ss” 后缀的词（比如 “caress” 被第四条规则重写。最后一条规则简单地移除词项最后的 “s”，将 “cats” 重写为 “cat”。总的来说，这个算法有 5 个主要步骤，有一些会有子步骤，总共有 9 组规则，最长的包含 20 条规则。

词干提取器的应用可提高查全率，从而检索到更多的文档，但不恰当的匹配和对不相关的文档赋予高分都会导致查准率降低。但是，在一个典型的 TREC 实验上的平均数值来看，词干提取通常还是会显著提升查准率的。表 3-1 给出了在第 1 章讨论的测试集上使用 Porter 词干提取器后的影响。表中的值可直接与表 2-5 作比较。对于这四个数据集，词干提取能够提升检索有效性。这些实验结果表明，如果你正在参与一个类似于 TREC 的实验，那么使用词干提取将会是一个很好的想法。

表 3-1　词干提取的影响。表中列出了本书中讨论到的各种检索方法的有效性指标。表中的值可与表 2-5 作比较

方法	TREC45				GOV2			
	1998		1999		2004		2005	
	P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
邻近度 (2.2.2)	0.418	0.139	0.430	0.184	0.453	0.207	0.576	0.283
BM25 (Ch. 8)	0.440	0.199	0.464	0.247	0.500	0.266	0.600	0.334
LMD (Ch. 9)	0.464	0.204	0.434	0.262	0.492	0.270	0.600	0.343
DFR (Ch. 9)	0.448	0.204	0.458	0.253	0.471	0.252	0.584	0.319

当词干提取出现错误时，会给系统带来严重的负面影响。例如，TREC 主题 314 “marine vegetation”（海洋植被）使用 Porter 词干提取器之后变为 “marin veget”。遗憾的是，“marinated vegetables”（腌制蔬菜）也被提取为同样地词根形式。如果信息检索系统将食谱、餐厅评论连同（或者是）海洋植物学方面的文章一同返回作为这个查询的检索结果，用户可能会感到吃惊和迷惑。因为不知道为什么她的查询会检索出这样的文档。在她看来，系统可能坏了或是存在漏洞。即便是她知道导致这个问题的原因，她也不知道如何解决，除非有方法可以明确禁用当前的这个词干提取器。

对于研究而言，词干提取还是很合适的。对于一个如 TREC 实验的参与者来说，几乎

不需要额外再做什么，词干提取就能显著提高性能。然而，在更加现实的环境中，任何有可能会产生令人费解的错误的分词方法，都会被谨慎地处理。在将词干提取器集成到信息检索系统之前，应该认真考虑可能带来的负面作用。

3.1.3 停词

虚词（function word）是那些本身没有明确意义的词；相反，它们用来修饰其他单词或指示语法关系。英文中，虚词包括介词、冠词、代词以及连词。在任何语言中，虚词通常都是出现最频繁的词。图 1-1 中的词（相对于 XML 标签来说）都是虚词。

当文档被视为无结构的“词袋”时，就如在向量空间模型中那样，查询中包含虚词是不必要的。即使在邻近度模型中，给定普通文本中所有虚词的频率，某个特定虚词与其他查询词项非常邻近也是很常见的。因此信息检索系统通常会定义一个**停用词**（stopword）表，其中常常包括虚词。在查询阶段，会从查询中去掉这些停用词，检索只在剩下的词项中进行。

考虑查询中词项“the”的出现。在英文文本中大约 6% 的词条会与这个词项匹配。同样，大约 2% 的词条会与词项“of”匹配。几乎每一个英文文档都包含这两个词项。如果一个用户查找与“the marriage of William Shakespeare”相关的信息，信息需求将简化为查询〈“marriage”，“william”，“shakespeare”〉——“of”和“the”被视为停用词——不会对检索效果产生任何负面影响。

除虚词外，停用词表中还包括单个的字母、数字以及其他常见词项，如 be 动词。在 Web 这个特殊的应用环境中，停用词表中可能还包括如“www”、“com”、“http”这种基本在上下文中没有任何意义的词项。根据不同的信息检索系统和具体的应用环境，停用词表的大小可能从十几个到好几百个。去掉了典型停用词之后，图 3-1 中的文本缩减成如下：

question
ti nobler mind suffer
sling arrow outrag fortun
take arm sea troubl
oppos end die sleep
sleep sai end
heart ach thousand natur shock
flesh heir ti consumm
devoutli wish die sleep
sleep perchanc dream ay rub

因为停用词通常都是频繁词，对应较长的位置信息列表，所以去除这些停用词可以避免处理这些很长的列表，从而能够显著地减少查询的执行时间。而且，如果所有查询中的停用词都去掉，那么它们就不必包含在索引中。在早期的信息检索系统里，消除停用词为减少索引规模带来了好处，一个很重要的考虑是当时的磁盘和内存非常小而且昂贵。

遗憾的是，一小部分的查询还是受到了停用词消除的影响，通常这时停用词是一个词组中的重要部分。在 Hamlet 独白中的“to be or not to be that is the”就是一个很著名的例子。我们立即就能识别出这个引用句全部由典型的停用词构成。乐队“The The”是另外一个例子——也进一步说明了大写的重要性。尽管你可以将这些例子看做是很极端的情况——它们确实也是——但搜索引擎仍应以合理的方式处理它们。

为了处理类似的情况，建议索引中还是包含停用词。依赖具体的检索算法，当这些停用

词的出现所起的正面影响很小的时候，可以考虑选择性地将它们从查询中去除。当索引中包含停用词时，信息检索系统可以具体情况具体分析。现代商业搜索引擎中的排名方法融合了许多排名特征，包括基于词频和邻近度的特征。对于不考虑查询词项间邻近度的特征，可以将停用词去掉。对于考虑查询词项间邻近度的特征，特别需要匹配它们在词组中的出现位置时，最好是保留停用词。

3.2　字符

对原始文本进行分词需要理解字符是如何编码的。目前为止，本书默认地假定文本字符用 7 位的 ASCII 值编码。ASCII 足以用来编码英文文本中的大多数字符，包括大小写字母、数字以及许多标点符号。

尽管对于英文来说，ASCII 编码是可接受的，但是对于多数其他语言来说它还远远不够。1963 年 ASCII 被标准化，在那个时候内存还很昂贵，网络也还是一个梦想，说英语的国家是计算设备的主要市场。支持其他语言作为一个需求已在一个个国家或地区的基础上展开。通常，这些特殊的字符编码都不能完全支持它们的目标语言。对于某些语言，不同的硬件厂商为不同的国家或地区开发了不兼容的编码形式。光是英文，IBM 的扩充的二进制编码的十进制交换码（Extended Binary Coded Decimal Interchange Code，EBCDIC）就是 ASCII 的一个强有力的竞争对手。

进步虽然缓慢，但到了 20 世纪 80 年代晚期，人们还是努力创建了一种统一的字符编码方法，能够容纳所有使用中的语言（甚至最终包括许多消失了的语言）。现在 Unicode[⊖]标准为大部分的语言提供了字符编码方式，并且涵盖的语言越来越多。除了简单实验用的信息检索系统，支持 Unicode 对信息检索系统来说是非常重要的。尽管许多文档仍然使用其他的编码方法，但是它们都能转化为 Unicode。为 Unicode 提供简单的支持使得信息检索系统得以成长，并容纳更多需要的新语言。

Unicode 给每个字符分配一个唯一的值，称为**码点**（codepoint），但并不指定如何用这些值来表示原始文本。码点的形式为 U+*nnnn*，其中 *nnnn* 指的是这个码点的十六进制值。例如，字符 β 用码点 U+03B2 来表示。Unicode 目前支持超过 100 000 个字符，码点值的范围超出了 U+2F800。

UTF-8，一个与 Unicode 相关的标准，是一种使用码点来代表原始文本的简便方法。尽管有许多表示 Unicode 的方法，但 UTF-8 具有几大优势，包括向后兼容 ASCII。UTF-8 用 1～4 字节来表示一个码点。每个出现在 ASCII 字符集中的字符都被编码为一个字节，与对应的 ASCII 编码值一样。因此用 ASCII 编码的原始文本也自动成为 UTF-8 编码的原始文本。

尽管 UTF-8 为每个字符采用一种变长的编码方式，但是这种编码方法却很直观，第一个字节的高位表示了编码的长度。如果最高有效位为 0，那么这个字节的形式为 0xxxxxxx——编码长度为 1 字节。这个字节代表了码点，它的值由后 7 位表示，它同时也代表了具有相同 7 位 ASCII 值的那个字符。例如，UTF-8 中的字节 01100101 代表了字符“e”（U+0065），与“e”的 ASCII 表示形式一样。

如果第一个字节的最高有效位是 1，编码长度就为 2～4 字节，接下来的字节位就指示了用一元编码的长度。因此，两字节的编码的第一个字节形式为 110xxxxx，三字节的编码的

⊖ www.unicode.org

第一个字节形式为 1110xxxx，四字节的编码的第一个字节形式为 11110xxx。一个多字节编码的第二个及其后字节的形式都为 10xxxxxx。通过检测任意字节的两个最高有效位，可以判断该字节是否为一个编码的起始字节。

因为首字节的最高有效位被占用了，所以二至四字节编码的码点值由后面未被占用的位来决定。两字节的编码能表示范围 U+0080～U+07FF 内的码点，三字节的编码能表示范围 U+0800～U+FFFF 内的码点，四字节的编码能表示 U+10000 及以上范围内的码点。例如，三字节编码统一具有以下形式：

1110xxxx 10yyyyyy 10zzzzzz

并代表 16 位值 xxxxyyyyyyzzzzzz。字符“八”的码点为 U+516B，用二进制表示为 01010001 01101011。在 UTF-8 中，这个字符将会被编码为三个字节：11100101 10000101 10101011。

3.3 字符 *n*-gram

将字符串转化为用于索引的词条序列的难度取决于底层语言的细节。词的识别以及提取的过程因语言的不同而千差万别。字符 n-gram（n 元文法）是另外一种复杂的特定于语言的分词技术。本节将以英文为例来讲解这一技术。后面的章节会将这一技术应用到其他语言中。

使用这种技术，简单地将 n 个字符的重叠序列视为词条。例如，当 $n=5$ 时，“orienteering”将会被分割为以下的 5-gram：

orie orien rient iente entee nteer teeri eerin ering ring

下划线“_”指示一个词的开始和结束。索引时为每一个唯一的 n-gram 建立一个位置信息列表。在查询阶段，查询也相应地划分为 n-gram。使用这种技术，一个三字符单词如“the”将被索引为 5-gram“_the_”。对于两字符和一字符的词，用一点小技巧，将“of”索引为“_of_”，“a”索引为“_a_”。

原则上，n-gram 索引方法不用考虑标点、大写、空格以及其他语言特征。尽管 n 的最优取值因语言不同而不同，但是这种方法是独立于特定语言的。只需要将文档划分为 n-gram，建立索引，然后再处理查询就可以了。实际上，n-gram 索引还是会考虑语言的一些基本特征。对英文来说，我们去掉标点并进行大小写规范化。

如果没有词干提取器，n-gram 可以替代词干提取器。对于许多语言来说，两个词项拥有相同的 n-gram 的数量可以反映出它们的词形。“orienteers”与“orienteering”匹配的 5-gram 的数量要比“orienteers”与“oriental”多。

对英文来说，n-gram 索引没有显著的影响。表 3-2 给出了在标准实验文档集中使用 5-gram 索引的效果。表中结果可与表 2-5 和表 3-1 中的结果作比较。与表 2-5 比较起来，结果有些混乱。有些好一点，有些糟一点。从下一节中会看到，在其他语言上的效果会不太一样。

表 3-2 5-gram 索引的影响。该表列出了本书中讨论到的检索方法的有效性评指标。表中的结果可与表 2-5 和表 3-1 中的结果作比较

方法	TREC45				GOV2			
	1998		1999		2004		2005	
	P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
邻近度 (2.2.2)	0.392	0.125	0.388	0.149	0.431	0.171	0.552	0.233
BM25 (Ch. 8)	0.410	0.177	0.446	0.214	0.463	0.226	0.522	0.296
LMD (Ch. 9)	0.416	0.186	0.438	0.222	0.404	0.188	0.502	0.276
DFR (Ch. 9)	0.440	0.203	0.444	0.230	0.478	0.243	0.540	0.284

n-gram 索引的代价是增加了索引规模但降低了效率。表 3-2 中的文档集上建立的压缩索引比相应的基于单词的索引大了 6 倍（在一个典型的英文文本集中，一个单词平均由 6 个字符组成）。查询执行时间由此会增加 30 倍或以上，因为每个查询（分词后）的词项数及其位置信息列表长度都增加了。

n-gram 索引可进一步扩展。在英文中，单词由空格和标点来分隔，n-gram 可以跨越词的边界，使得可以捕捉到词组之间的关系。例如，片段“…perchance to dream…”将被划分为如下的 5-gram：

perc perch ercha rchan chanc hance ance nce_t ce_to e_to_ _to_d to_dr o_dre _drea dream ream_

其中，下划线“_”代表单词与单词之间的间隔。尽管这个技术用于我们的实验中，对英文文档集上的检索方法有略微的负面影响，但对于其他方法和语言却是有利的。

3.4　欧洲语言

本节中对分词的讨论将从英文延伸到其他欧洲语言。我们将隶属于几个不同语系的广泛的语言都归到这一类。例如，法语和意大利语属于罗曼语；荷兰语和瑞典语属于日耳曼语；俄语和波兰语属于斯拉夫语。罗曼语、日耳曼语、斯拉夫语彼此之间相互联系，但芬兰语和匈牙利语属于这三个之外的第四大语系，与前面的语系毫不相关。爱尔兰语和苏格兰盖尔语属于第五大语系。巴斯克语，在西班牙和法国有超过 100 万的人使用，是一**种独立**（isolate）的语言，与其他使用中的语言都不相关。

尽管“欧洲语言”这个类别从语言学角度来说没什么意义，但从信息检索的角度来看，将这些语言视为一组。由于欧洲大陆共同的历史，这些语言在**正字法**（orthography），即书写规则上，有类似的规定。它们都使用一个包含大小写形式的几十个字母组成的字母表。用标点提供句子的结构。字母序列通常由空格和标点分隔成词。

欧洲语言中普遍存在的大量分词问题在英文中是没有的。英文中几乎不存在重音符和其他发音符，除“naïve”是一个很少见的例子外。在许多其他欧洲语言中，发音符对发音来说是非常重要的，也因此可区分不同的单词。在西班牙语中，“cuna”是摇篮的意思，但是“cuña”是楔形物的意思。在葡萄牙语中，“nó”是扣子的意思，但“no”是一个虚词，表示“在”的意思。尽管经过仔细地编辑，发音符都能被正确标识，但在查询和其他更加随意的书写中，这些符号常常不存在，因此会导致无法匹配。

一个解决方案是在分词过程中将这些发音符号都去掉，将“cuna”和“cuña”视为同一个词项。另一个解决方案是双路索引这样的词项，同时保留有发音符和没有发音符的形式。在查询阶段，由信息检索系统来决定发音符是否重要。特殊的查询方法也同时考虑这两种形式，使得可以匹配任何一种方式。最好的解决方案取决于特定语言的细节和信息检索环境。

对于某个语言或语系，还会有其他的分词问题。很多日耳曼语言会把组合名词写成一个单词。如荷兰语中，“bicycle wheel”等于“fietswiel”。在分词过程中，这个组合名词就被**间断**（broken）或**分割**（segmented）为两个词项，使查询词项“fiets”能够得到匹配。同样，德语单词“Versicherungsbetrug”是“Versicherung”（“保险”）和“Betrug”（“欺诈”）的组合。注意，在这种情况下，不能简单地将单词划分为两个部分，还要仔细处理连接两个部分的“s”字符。当信息检索系统包含混合语言文档时还会有其他的问题。只有重音符号才能区分法语中的“thé”（“茶”）和英语中的定冠词“the”。

大多数欧洲语言可以使用词干提取器。特别注意 Snowball 词干提取器，它能够支持包

括土耳其语在内的十多种欧洲语言[⊖]。Snowball 由 Martin Porter 创建，他也是英文词干提取器 Porter 的开发者，这两个提取方法差不多。n-gram 字符分词方法对于欧洲语言也有很好的效果，$n=4$ 或 5 为最佳值。

表 3-3 比较了 8 种欧洲语言上的分词技术，使用的是类似于 TREC 风格的测试集。对于所有的语言来说，Snowball 词干提取器都比未提取词干好。对于荷兰语、芬兰语、德语以及瑞典语，字符 5-gram 方法的效果是最好的。

表 3-3 在几个欧洲语言（基于 McNamee，2008）上比较分词技术。表中列出了四种分词方法的平均查准率均值，分别是未提取词干的单词、Snowball 词干提取器、字符 4-gram、字符 5-gram

语言	单词	词干提取器	4-gram	5-gram
荷兰语	0.416	0.427	0.438	0.444
英语	0.483	0.501	0.441	0.461
芬兰语	0.319	0.417	0.483	0.496
法语	0.427	0.456	0.444	0.440
德语	0.349	0.384	0.428	0.432
意大利语	0.395	0.435	0.393	0.422
西班牙语	0.427	0.467	0.447	0.438
瑞典语	0.339	0.376	0.424	0.427

3.5 CJK 语言

中文、日文、韩文统称为 **CJK 语言**。如欧洲语言那样，尽管不属于同一语系，但共同的历史使这些语言的书写形式都遵循相同的正字法。与欧洲语言相比，它们的字符集相当庞大。一张典型的中文报纸就包含了几千个不同的汉字。并且，中文和日文单词之间是没有空格的。因此，这些语言中如何划分内容就成了分词中最重要的部分。

字符集的复杂性带来了很多问题。日文使用三种主要的书写方式。其中两种方式里，每个字符都代表了口语里的一个音节。第三种方式源于中文汉字，并且具有相同的 Unicode 码点。原则上，每个词都可以用这三种方式中的任何一种进行书写。用一种方式书写的查询词项应该要能匹配用其他方式书写的文档词项。

许多中文汉字有繁体和简体两种形式。由于历史原因，华语世界的不同地区由不同形式占主导地位。例如，在中国香港繁体字很普遍，而在中国的大部分地区简体字才是标准。因此，为一种形式的查询返回另一种形式的文档也是对的。很多人能同时阅读两种形式的文本。一些软件工具如浏览器插件也能自动将一种形式转换为另一种形式。

中文的每个字都有具体的意义，这是很不常见的，但是这个性质并不意味着可将每个汉字都视为一个单词。而且，一个词的意思与词中单个汉字的意思也许根本不一样；一个词不等同于其各部分之和。例如，中文词“**危机**”由“**危**”和“**机**”组成（Gore，2006）。然而，第二个汉字在这里意思更接近“关键点”，而在其他上下文中通常是指机器。同一个字（机）也是“机场”的第一个字，它的第二个字是“场”。将词的各部分正确地连接对保持它的本意非常重要。“**机场**”不是按字面意思指“机器的场所”。“**自由**”也不是指“自己的原由”。

尽管有些中文词，包括许多虚词，只有一个汉字，但大多数词会更长。大部分的词都是二元的：包括两个汉字。将一个汉字序列划分为词是一个困难的过程。就像以下没有空格的

⊖ snowball.tartarus.org

英文句子“Wegotogethertogether.”一样。正确的划分（“We go to get her together.”）需要对英文文法和词汇表有所认识。中文里一个汉字可以同时和它前面或后面的汉字组成词。正确的划分依赖于上下文。然而，分词对于中文信息检索而言不是绝对重要的。或许是因为中文中大都是二元词，2-gram 字符索引在中文中效果很好，并为评价自动单词划分提供了简单的基准。

所有的 CJK 语言都为它们的音译提供了标准转化方式，可以与罗马字母表对接。对中文来讲，现代音译标准是汉语拼音（或拼音）。在拼音中，**危机**写为“wēijī”，机场写为“jīchǎng”。重音符号代表了**音调**（tonality），是指音节发音语调的改变。

拼音为表达中文查询提供了另一种简便方式。遗憾的是，拼音的使用会导致一定程度的歧义。例如，6 个常见汉字都发音为“jī”。不仅如此，拼音查询通常还不输入音调，这就导致了更多的歧义。如果去掉音调，大概有 30 个常见汉字都发音为“yi”，更多汉字的发音为“shi”。查询词项“shishi”可以表示“时事”、“实施”和好几个其他的意思。为了处理这种歧义，搜索引擎把不同的解释展现给用户，让用户来选择正确的那个。

3.6　延伸阅读

简单说来，分词就是试图将文档划分为词。Trask（2004）从语言学的角度讨论了“词”的概念。此外，从语言学角度对分词和词形归并的讨论在第 4 章中由 Manning 和 Schütze（1999）给出。

大部分编程语言和操作系统都提供对 Unicode 和 UTF-8 的支持。Unicode 标准的细节由其官网给出[㊀]。对 Unicode 码点用 UTF-8 编码来实现，是由 Rob Pike 和 Ken Thompson 开发的，作为他们的 Plan-9 操作系统（Plan-9 Operating System）的一部分（Pike 和 Thompson，1993）。在 Pike 和 Thompson 开发出 UTF-8 之后，为了兼容 Unicode 的变化及扩展它，UTF-8 也一直被不断修改。它在 RFC 3639[㊁]中被定义为互联网标准。

除 Porter 词干提取器外（Porter，1980），早期的英文词干提取器包括了 Lovins（1980）、Frakes（1984）和 Paice（1990）的工作。Harman（1991）给出了一个简单的**S 词干提取器**（S stemmer），用于将复数形式的单词转为单数形式。她将这个 S 词干提取器与 Porter 和 Lovins 词干提取器进行了性能比较，但是发现它们与不提取词干相比，并没有在检索性能上有显著提高。通过使用更大的测试集来进行更全面的实验，Hull（1996）证明了词干提取器能显著提高检索的平均有效性。然而，他警告说，过度的词干提取在显著提高平均有效性并大大提高少数查询的性能的同时，会降低其他多数查询的性能。

Snowball[㊂]词干提取器为创建词干提取器提供了一个算法框架。即便如此，为一种新语言开发和验证一个词干提取器是一个很耗费人力的过程。因此，已有尝试通过使用大量语料库来提高语言样本，从而希望可以自动地创建词干提取器。Creutz 和 Lagus（2002）介绍了一种通过最小化代价函数来进行单词划分的方法，并将他们的方法与已有的英文和芬兰语词干提取器进行了对比。Majumder 等人（2007）基于词形分析提出了一个将单词聚集到等价类的方法。他们在几种语言上评价了这个方法，包括孟加拉语。

McNamee 和 Mayfield（2004）概述了用于信息检索的 n-gram 分词方法。McNamee 等人（2008）和 McNamee（2008）将 n-gram 索引方法与 Snowball 词干提取器和 Creutz 和

㊀ www.unicode.org
㊁ tools.ietf.org/html/rfc3629
㊂ snowball.tartarus.org

Lagus（2002）的方法进行比较。此外，字符 n-gram 也可以用于处理由**光学字符识别**（Optical Character Recognition，OCR）导致的错误。Beitzel 等人（2002）概述了这一领域的发展情况。

从 1994～1996 年，西班牙语的信息检索系统是 TREC 中一个实验任务的主题。1996～1997 年，中文信息检索系统是一个任务的主题（Voorhees 和 Harman，2005）。包括阿拉伯语和其他语言在内的多语言任务则从 1997 年一直持续到 2002 年（Gey 和 Oard，2001）。从 2000 年开始，跨语言评价讨论组（CLEF[㊀]）在欧洲语言上进行了很多多语言和跨语言的实验。表 3-3 的结果就是基于 CLEF 测试集得出的。从 2001 年起，日本国立信息学研究所信息检索系统测试集项目（NTCIR[㊁]）就提供了一个类似的亚洲语言上的实验讨论组。印度信息检索评价讨论组（FIRE[㊂]）在印度次大陆语言也进行了实验。

用计算机处理亚洲语言存在困难和复杂的问题，至今仍是许多研究的主题。一个非常重要的期刊，《ACM Transactions on Asian Language Information Processing》（ACM 亚洲语言信息处理学报），完全致力于该主题。Luk 和 Kwok（2002）全面介绍了中文信息检索中的单词划分和标记化技术。Peng 等人（2002）探索了中文信息检索中分词的重要性，评价和对比了多个有竞争力的技术。Braschler 和 Ripplinger（2004）研究了对德语的分词（也叫做**分解**（decompounding））。Kraaij 和 Pohlmann（1996）研究了对荷兰语的分解。《Information Retrieval Journal》（信息检索期刊）的一个特刊关注非英文网站检索（Lazarinis 等人，2009）。

对一种新语言分词需要仔细考虑它的正字法和词形。Fujii 和 Croft（1993）讲述了日文分词的基本知识。Asian 等人（2005）讨论了印度尼西亚语的分词。Larkey 等人（2002）讨论了阿拉伯语的分词。Nwesri 等人（2005）介绍并评价了一种移除阿拉伯语中表示连词和介词词缀的算法。

拼写的正确性与分词紧密联系。因为词条把查询和文档联系起来，错误的拼写会使匹配变得更困难。Kukich（1992）概述了识别和纠正拼写错误的基础工作。Brill 和 Moore（2000）为拼写纠正提供了一个错误模型。Ruch（2002）在一个基础的特定任务中评测了拼写错误对查询的影响。Cucerzan 和 Brill（2004）以及 Li 等人（2006）从一个商业 Web 搜索引擎查询日志里确定拼写错误及对应的纠正，从而创建了拼写检查器。Jain 等人（2007）介绍了用查询日志来扩展缩略语的方法。

3.7 练习

练习 3.1 在至少两个商业搜索引擎中输入下面的查询：

(a) to be or not to be

(b) U.S.

(c) US

(d) THE

(e) The The

(f) I'll be back

(g) R-E-S-P-E-C-T

每一个查询都有特定的含义，分词之后这些含义都可能会丢失。判断返回的前 5 条结果，看看哪些是相关的？

㊀ www.clef-campaign.org

㊁ research.nii.ac.jp/ntcir

㊂ www.isical.ac.in/~clia

练习 3.2　在 Web 上，某些词项出现得过于频繁以至于被视为是停用词。在至少两个商业搜索引擎上输入如下查询：

(a) www

(b) com

(c) http

判断返回的前 5 条结果，看看哪些是相关的？

练习 3.3　Unicode 赋予字符 β 的码点为 U+03B2，则相对应的二进制 UTF-8 的表示形式是什么？

练习 3.4（项目练习）　对你在练习 1.9 中创建的文档集进行分词。在这个练习中，只保留那些由英文字母表中的字符构成的词条。忽略标签和由非 ASCII 字符构成的词条。根据 3.3 节介绍的方法，从这些词条中生成 5-gram。绘制一个关于词频和排名数的对数图，与图 1-5 或者是练习 1.12 生成的图类似。5-gram 遵循**齐夫定律**（Zipf's law）吗？如果遵循，α 的近似值为多少？

练习 3.5（项目练习）　用维基百科的中文文档来重复练习 3.4，使用字符 2-gram 来分词。

练习 3.6（项目练习）　获取一份 Porter 的英文词干提取器或者其他英文词干提取器。对练习 1.9 中创建的文档集进行分词。和练习 3.4 一样，只保留那些由英文字母表中的字符构成的词条。删除重复的词条，为这个文档集构建一个词汇表。对这个词汇表中的词项运行词干提取器，创建等价词项集，每个集合中所有词项被提取到同一词根形式。哪一个（哪一些）集合最大？找出至少三个包含了被词干提取器提取出不恰当词根的词项。

练习 3.7（项目练习）　如果你熟悉除英文外的一门语言并存在对应的词干提取器，在这个语言上重复练习 3.6。你可以用维基百科作为文本源。

3.8　参考文献

Asian, J., Williams, H. E., and Tahaghoghi, S. M. M. (2005). Stemming Indonesian. In *Proceedings of the 28th Australasian Computer Science Conference*, pages 307–314. Newcastle, Australia.

Beitzel, S., Jensen, E., and Grossman, D. (2002). Retrieving OCR text: A survey of current approaches. In *Proceedings of the SIGIR 2002 Workshop on Information Retrieval and OCR: From Converting Content to Grasping Meaning*. Tampere, Finland.

Braschler, M., and Ripplinger, B. (2004). How effective is stemming and decompounding for German text retrieval? *Information Retrieval*, 7(3-4):291–316.

Brill, E., and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293. Hong Kong, China.

Creutz, M., and Lagus, K. (2002). Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pages 21–30.

Cucerzan, S., and Brill, E. (2004). Spelling correction as an iterative process that exploits the collective knowledge of Web users. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 293–300.

Frakes, W. B. (1984). Term conflation for information retrieval. In *Proceedings of the 7th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 383–389. Cambridge, England.

Fujii, H., and Croft, W. B. (1993). A comparison of indexing techniques for Japanese text retrieval. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 237–246. Pittsburgh, Pennsylvania.

Gey, F. C., and Oard, D. W. (2001). The TREC-2001 cross-language information retrieval track: Searching Arabic using English, French or Arabic queries. In *Proceedings of the 10th Text REtrieval Conference*, pages 16–25. Gaithersburg, Maryland.

Gore, A. (2006). *An Inconvenient Truth*. Emmaus, Pennsylvania: Rodale.

Harman, D. (1991). How effective is suffixing? *Journal of the American Society for Information Science*, 42(1):7–15.

Hull, D. A. (1996). Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society for Information Science*, 47(1):70–84.

Jain, A., Cucerzan, S., and Azzam, S. (2007). Acronym-expansion recognition and ranking on the Web. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*, pages 209–214. Las Vegas, Nevada.

Kraaij, W., and Pohlmann, R. (1996). *Using Linguistic Knowledge in Information Retrieval.* Technical Report OTS-WP-CL-96-001. Research Institute for Language and Speech, Utrecht University.

Kukich, K. (1992). Technique for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439.

Larkey, L. S., Ballesteros, L., and Connell, M. E. (2002). Improving stemming for Arabic information retrieval: Light stemming and co-occurrence analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–282. Tampere, Finland.

Lazarinis, F., Vilares, J., Tait, J., and Efthimiadis, E. N. (2009). Introduction to the special issue on non-English Web retrival. *Information Retrieval*, 12(3).

Li, M., Zhu, M., Zhang, Y., and Zhou, M. (2006). Exploring distributional similarity based models for query spelling correction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 1025–1032. Sydney, Australia.

Lovins, J. B. (1968). Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1–2):22–31.

Luk, R. W. P., and Kwok, K. L. (2002). A comparison of Chinese document indexing strategies and retrieval models. *ACM Transactions on Asian Language Information Processing*, 1(3):225–268.

Majumder, P., Mitra, M., Parui, S. K., Kole, G., Mitra, P., and Datta, K. (2007). YASS: Yet another suffix stripper. *ACM Transactions on Information Systems*, 25(4):article 18.

Manning, C. D., and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* Cambridge, Massachusetts: MIT Press.

McNamee, P. (2008). Retrieval experiments at Morpho Challenge 2008. In *Cross-Language Evaluation Forum.* Aarhus, Denmark.

McNamee, P., and Mayfield, J. (2004). Character n-gram tokenization for European language text retrieval. *Information Retrieval*, 7(1-2):73–97.

McNamee, P., Nicholas, C., and Mayfield, J. (2008). Don't have a stemmer?: Be un+concern+ed. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 813–814. Singapore.

Nwesri, A. F. A., Tahaghoghi, S. M. M., and Scholer, F. (2005). Stemming Arabic conjunctions and prepositions. In *Proceedings of the 12th International Conference on String Processing and Information Retrieval*, pages 206–217. Buenos Aires, Agentina.

Paice, C. D. (1990). Another stemmer. *ACM SIGIR Forum*, 24(3):56–61.

Peng, F., Huang, X., Schuurmans, D., and Cercone, N. (2002). Investigating the relationship between word segmentation performance and retrieval performance in Chinese IR. In *Proceedings of the 19th International Conference on Computational Linguistics.* Taipei, Taiwan.

Pike, R., and Thompson, K. (1993). Hello world. In *Proceedings of the Winter 1993 USENIX Conference*, pages 43–50. San Diego, California.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

Ruch, P. (2002). Information retrieval and spelling correction: An inquiry into lexical disambiguation. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 699–703. Madrid, Spain.

Trask, L. (2004). *What is a Word?* Technical Report LxWP11/04. Department of Linguistics and English Language, University of Sussex, United Kingdom.

Voorhees, E. M., and Harman, D. K. (2005). The Text REtrieval Conference. In Voorhees, E. M., and Harman, D. K., editors, *TREC — Experiment and Evaluation in Information Retrieval*, chapter 1, pages 3–20. Cambridge, Massachusetts: MIT Press.

第二部分 索 引

第4章

Information Retrieval: Implementing and Evaluating Search Engines

静态倒排索引

本章将介绍一些可用于实现第2章中提到的检索查询的索引结构。我们将讨论的范围暂时限定在静态文档集，即假设我们在为一个不变的文档集建立索引，而动态文档集，即可以加入或删除文档的文档集，它的索引更新策略将是第7章讨论的主题。

为了提高性能，我们希望能把整个文档集的索引都保存在内存中。然而，在很多应用中这是不可行的。例如在文件系统检索中，存储在文件系统中的全部数据的一个全文索引很容易就到达几千兆字节。由于用户并不希望把大部分的可用内存资源都贡献给检索系统，因此把整个索引保存在内存（RAM）中是不太可行的，即便对于Web搜索引擎中专用的索引服务器来说，更经济的做法也是将大部分索引存储在磁盘而非内存中，因为磁盘空间比内存便宜得多。例如，我们写这本书时，1 GB的内存空间大概需要20美元，而1 GB的磁盘空间只需大约0.2美元。它们之间存在了200倍的价格差，但是，这无法反映这两种存储介质的相对性能。对于典型的索引操作，内存索引通常比磁盘索引快10～20倍。因此，当建立两个价格相同的检索系统时，一个把索引数据存储在磁盘上，另外一个把索引数据存储在内存中，基于磁盘索引的系统可能要比基于内存索引的系统慢（该问题及相关问题更深入的讨论参见Bender等人（2007）的工作）。

因为搜索引擎要与运行在同一系统内的其他进程共享内存，或因为将数据存储在磁盘上比存放在内存中更经济，所以我们在本章中都基于一个假设，就是内存是稀缺资源。在讨论倒排索引这一数据结构时，将重点讨论混合组织结构，即把小部分的索引存放在内存中，同时把大部分的索引数据存放在磁盘上。我们将研究对搜索引擎的不同部分使用不同的数据结构，并通过大量的实验去评价它们的性能。附录中给出了做这些实验的计算机系统的性能概要。

4.1 索引的组成部分和索引的生命周期

本章在讨论倒排索引的各个组成部分时，将从两个方面去探讨：结构方面，将系统分解成各个部分，并分别研究索引在每个单独部分的各方面内容（如单独的位置信息列表）；操作方面，将研究倒排索引生命周期中的各个阶段，同时讨论各个阶段中必须实现的关键索引操作（如执行检索查询操作）。

正如第2章提到的，倒排索引的两个主要组成部分是词典（dictionary）和位置信息列表（posting list）。对于文档集中的每个词项，都有一个位置信息列表记录该词项在文档集中出现的位置。系统将根据这个位置信息列表中的信息处理检索查询。词典是建立在位置信息列表上用于索引的数据结构。对于当前查询中的每个词项，搜索引擎在开始处理查询之

前，首先需要找到每个词项对应的位置信息列表。词典就提供了查询词项到索引中的对应位置信息列表的映射。

除了词典和位置信息列表，搜索引擎通常还用到其他的数据结构。例如，很多的搜索引擎会为索引中的每个文档都维护一个**文档结构图**（document map），存储与该文档相关的特定信息，如文档的 URL、文档长度、PageRank 值（参见 15.3.1 节）等。这些数据结构的实现都非常简单，不需要特别关注。

静态文档集的静态倒排索引的生命周期可以分为以下两个不同的阶段（动态索引的两个阶段也类似）：

1）**索引建立**（index construction）：顺序处理文档集，每次读入一个词条，同时以增量方式为文档集中的每个词项建立一个位置信息列表。

2）**查询处理**（query processing）：利用第一阶段建立的索引信息实现查询处理。

第一阶段一般称为**索引阶段**（indexing time），第二阶段则称为**查询阶段**（query time）。在很多时候，这两个阶段是互为补充的；通过在索引阶段增加一些额外的处理（如预计算文档贡献得分——参见 5.1.3 节），查询阶段就能减少相应的处理工作。然而，一般来说，这两个阶段间又存在着较大的差异，通常使用不同的算法和数据结构。即使是它们中共有的索引的子部分，如搜索引擎的词典数据结构，其实现方法在两个阶段中也不尽相同。

本章的主要内容就是以上提到的建立倒排索引过程的两个主要方面。本章的第一部分（4.2～4.4 节）主要关注查询阶段的词典和位置信息列表，寻找能实现高效索引访问和查询处理的数据结构。第二部分（4.5 节）将重点关注索引建立过程并讨论如何有效实现第一部分中提出的数据结构。同时还讨论如何使用与第一部分不一样的组织方式构建词典和位置信息列表，以使索引阶段的性能达到最佳。

为简单起见，假设本章仅使用**模式独立**（schema-independent）索引。其他类型的倒排索引均与模式独立索引类似，本章介绍的方法均能应用于其上（参见 2.1.3 节中列出的各种不同类型的倒排索引）。

4.2 词典

词典是用于管理文档集中词项集的核心数据结构，它提供了一个从索引词项到其对应位置信息列表地址的映射。在查询阶段，当处理当前关键词查询的时候，定位查询词项在索引中的位置信息列表是其中一个首要步骤。在索引阶段，词典的查找功能使搜索引擎很快就能获得每个当前词项的倒排列表在内存中的地址，并在该列表后添加一个新的位置信息。

搜索引擎中的词典通常支持以下操作：

1）插入一个词项 T 的新记录。

2）查找并返回词项 T 的记录（如果存在）。

3）查找并返回所有前缀是 P 的词项的记录。

当为文档集建立索引时，搜索引擎执行第 1 和第 2 种操作在词典中查找当前词项，并在索引中添加这些词项的位置。索引建立之后，搜索引擎执行第 2 和第 3 种操作为所有的查询词项寻找位置信息列表，从而实现检索查询的处理。尽管第 3 种词典操作并不是必须的，但是该操作非常有用，因为能让搜索引擎支持形式为“inform＊”的**前缀查询**（prefix query），并匹配出所有包含前缀为“inform”的词项的文档。

对于一个典型的自然语言文档集，词典相对于整个索引所占的空间要小得多。表 4-1 中给出了本书中使用的 3 个样例文档集上的数据就很好地说明了这一点。对于相同的文档集，

未经压缩的词典所占的空间仅是未经压缩的模式独立索引所占空间的 0.6%（GOV2）至 7%（莎士比亚文集）（事实上，根据齐夫定律（Zipf's law），文档集越大，词典相对于索引所占的空间越小　参见公式（1-2））。我们因此目前可以假设，词典所占空间足够小，可以完全把其存储在内存中。

表 4-1　3 个样例文档集上不同索引类型压缩前与压缩后的大小。每一种情况里，第一个数值表示每个索引项都采用 32 位整数形式存储时的大小，第二个数值表示每个索引项都采用了字节对齐的压缩编码方法压缩后的大小

	莎士比亚文集	TREC45	GOV2
词条数	1.3×10^6	3.0×10^6	4.4×10^7
词项数	2.3×10^4	1.2×10^6	4.9×10^6
词典(未压缩)	0.4 MB	24 MB	1046 MB
文档至词索引	n/a	578 MB/200 MB	37751 MB/12412 MB
词频索引	n/a	1110 MB/333 MB	73593 MB/21406 MB
位置索引	n/a	2253 MB/739 MB	245536 MB/78619 MB
模式独立索引	5.7 MB/2.7 MB	1180 MB/532 MB	173854 MB/63670 MB

实现常驻内存词典的两种最常见的方法是：

- **基于排序的词典**（sort based dictionary），文档集中所有的词项都以字母序（即按字母顺序）存放在一个有序数组或一棵搜索树中，如图 4-1 所示。查找操作通过遍历树（当使用搜索树时）或者二分查找（当使用有序表时）实现。

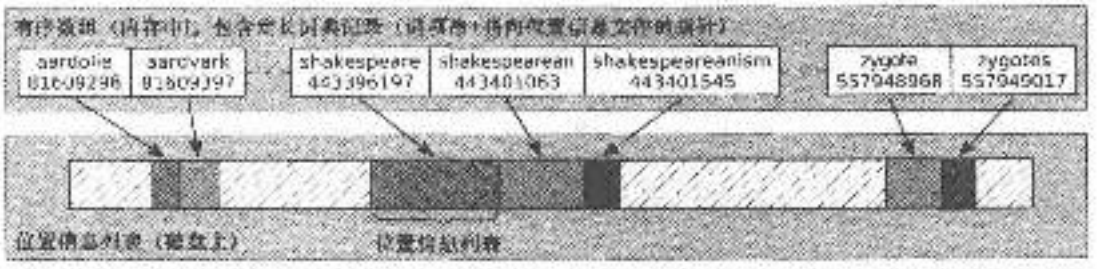


图 4-1　基于有序数组的词典数据结构（数据来源于 TREC45 的模式独立索引）。数组包含定长词典记录，由一个以 0 作为结束符的字符串和一个指向该词项位置信息列表地址的指针组成

- **基于哈希的词典**（hash-based dictionary），所有索引词项在哈希表中都有一个对应的记录。哈希表中的冲突（即两个词项有相同的哈希值）可以通过链表（chaining）方式解决——有相同哈希值的词项都存放到一个链表中，如图 4-2 所示。

1. 存储词典的记录

当用有序数组实现词典时，必须保证每个数组记录的大小都一致，否则难以实现二分查找。然而，这样会导致一些问题，例如，GOV2 中的最长字符串（即该文档集中的最长词项）有 74 147 字节，很明显，给词典中的每个词项都分配 74 KB 的内存空间是不可行的，但是即使我们不考虑这种极端情况，每个词项只分配 20 字节，仍然浪费了很多宝贵的内存资源，采用 1.3.2 节中的简单的分词过程，可以得到 GOV2 的平均词项长度是 9.2 字节，如果使用 20 字节的定长内存空间来存放每个词项，平均每个词项就浪费了 10.8 字节的空间（**内部碎片**（internal fragmentation））。

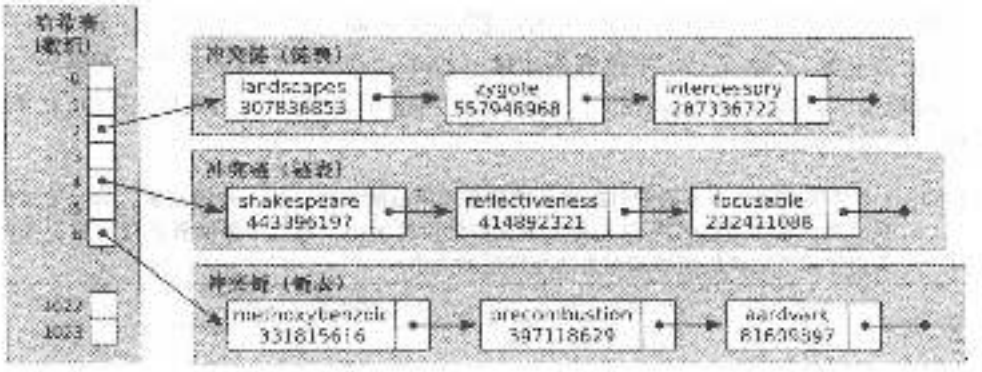


图 4-2 基于哈希表（$2^{10}=1024$ 个记录）的词典数据结构（数据来源于 TREC-6 的拟式组文索引）。有相同哈希值的词项都存放到同一链表中。每个词项信息描述块包含该词项本身、它的位置信息列表就绪和一个指向链表中下一个记录的指针

一种消除内部碎片的方法是不要在数组内存放索引词项本身，而是存放指向它们的指针。例如，搜索引擎可以维护一个主（primary）词典数组，包含指向辅（secondary）数组的 32 位指针。辅数组中保存着实际的词典记录，包括词项本身和对应的指向位置信息列表的指针。图 4-3 展示了以这种方式组织的搜索引擎的词典数据。有时把这称为字符串词典方法（dictionary-as-a-string approach），因为两个连续词典记录之间并没有明确的界限，也可将辅数组想象为一个无间断的长字符串。

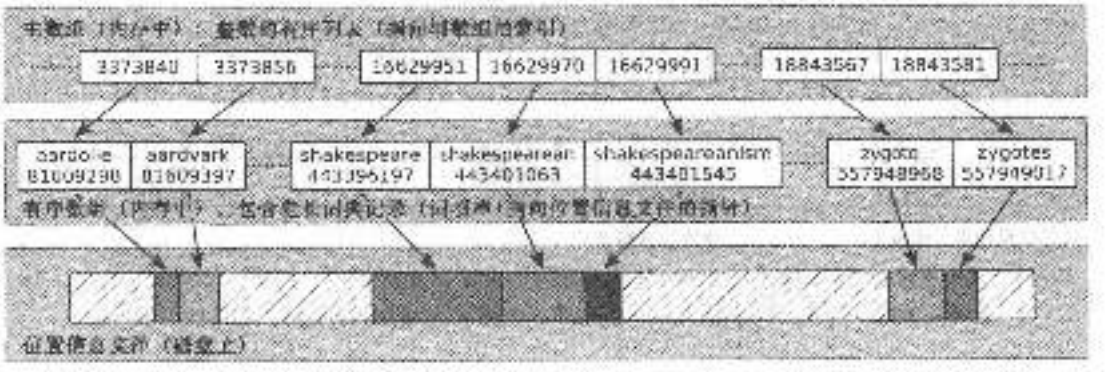


图 4-3 有额外间接层的基于排序的词典数据结构（所谓的字符串词典方法）

对于 GOV2 文档集，字符串词典方法比图 4-1 所示的词典每个记录少 6.8（10.8 − 4 − 6.8）字节的存储空间。这里 4 字节是主数组的指针造成的额外开销；10.8 字节是内部碎片所占的空间。

需要指出的是辅数组中存储的词项中不需要显式的结束符（如“\0”字符），因为主数组中的指针就隐式地给出了词典中每个词项的长度。例如，图 4-3 中，观察“shakespeare”和“shakespearean”的指针时，我们就知道“shakespeare”在词典中的记录需要 16629970−16629951−19 字节；11 字节用于存储词项，另加 8 字节用于存储指向位置信息列表的 64 位指针。

2. 基于排序的词典与基于哈希的词典

对于大部分的应用，基于哈希的词典比基于排序的词典更快，因为它不需要进行耗时的二分查找或树遍历来获得给定词项对应的词典记录。基于哈希的词典比基于排序的词典快多

少取决于哈希表的大小。表越小，导致的词项冲突越多，就可能大大降低词典的性能。根据概测法，为了使哈希表中冲突链表的长度较短，哈希表的大小应随词典中的词项数量的增长而线性增加。

表 4-2 给出了在三个样例文档集上，为每个随机索引词项查找词典记录的平均时间。一般来说，哈希表越大，查询时间越短，但莎士比亚文集不是这样，因为它太小了（仅有 23 000个不同的词项），导致本来用于减少词项冲突的缓冲效应反而降低了 CPU 缓存的利用率。这种情况下，大的哈希表反而增加了查找时间。尽管如此，只要哈希表大小合适，基于哈希的词典通常至少比基于排序的词典快上两倍。

表 4-2　查询阶段的查找性能，即基于排序的词典（如图 4-3 所示）和基于哈希的词典（如图 4-2 所示）查找一个词项的平均延时。对于后者来说，哈希表的大小（数组记录数）在 2^{18}（≈262 000）到 2^{24}（≈1680 万）之间变化

	排序	哈希(2^{18})	哈希(2^{20})	哈希(2^{22})	哈希(2^{24})
莎士比亚文集	0.32 μs	0.11 μs	0.13 μs	0.14 μs	0.16 μs
TERC45	1.20 μs	0.53 μs	0.34 μs	0.27 μs	0.25 μs
GOV2	2.79 μs	19.8 μs	5.80 μs	2.23 μs	0.84 μs

遗憾的是，虽然对单个词项进行词典查找有速度上的优势，但基于哈希的词典也有一个缺点：基于排序的词典能有效地支持前缀查询（如，“inform＊”）（而基于哈希的词典不行）。假如词典基于有序数组，那么前缀查询可以通过两次二分查找实现：首先找出与给定前缀查询匹配的第一个词项 T_j 和最后一个词项 T_k，然后对 T_j 到 T_k 之间的 $k-j+1$ 个词典记录进行线性扫描。过程的时间复杂度为：

$$\Theta(\log(|\mathcal{V}|)) + \Theta(m) \tag{4-1}$$

其中，$m=k-j+1$ 指与前缀查询匹配的词项数量，$\mathcal{V}$ 指搜索引擎的词汇表。

如要使基于哈希的词典支持前缀查询，那只能通过线性扫描哈希表中的所有词项来实现，需要$\Theta(|\mathcal{V}|)$次字符串比较。因此搜索引擎通常使用两种不同的词典数据结构：在索引建立阶段使用基于哈希的词典，对操作 1（插入）和操作 2（单个词项查找）提供有效支持；在索引建立后创建基于排序的词典，对操作 2（单个词项查找）和操作 3（前缀查询）提供有效支持。

由于在索引建立阶段比查询阶段更需要对单个词项进行词典查找提供高性能的支持，这促使了索引阶段的词典与查询阶段的词典在结构上有很多的不同点。在查询阶段，为所有查询词项查找相应的词典记录的开销可以忽略（几微秒），因为处理查询时一些必须执行的其他计算花费的时间开销比它大得多。然而，在索引阶段，需要为文档集中的每个符号都执行词典查找——GOV2 中共执行了 440 亿次查找操作。因此，在索引建立过程中，词典是主要的瓶颈，所以查找操作需要尽可能快。

4.3　位置信息列表

查询处理中使用到的实际索引数据，其实是存放在索引的位置信息列表中，通过搜索引擎的词典来访问。每个词项的位置信息列表中存储着该词项在文档集中出现的位置信息。根据索引的类型（文档编号、词频、位置或模式独立——参见 2.1.3 节），词项的位置信息列表中可以包括更多或更少的内容，相应的所需的存储空间也将增加或减少。即使不考虑索引的具体类型，位置信息数据仍然是索引数据的主要部分。总体来说，它们非常大，以至于无

法存放在内存中，只能存储到磁盘上。只有在查询处理过程中，在需要的基础上，作为查询处理例程，才把查询词项的位置信息列表（或其中的一小部分）加载到内存里。

为了使位置信息列表从磁盘加载到内存中的传输尽量高效，每个词项的位置信息列表都应该存储在磁盘的连续空间中。这样才能使访问列表时，所需的磁盘寻道操作次数最少。实验中使用的计算机磁盘驱动器（参见附录）一次磁盘寻道就读入 0.5 MB 数据，不连续的位置信息列表会大大降低系统的查询性能。

1. 随机列表访问：单项索引

搜索引擎的位置信息列表在查询阶段的访问模式由处理查询的方式决定。对于某些查询，以几乎连续的方式访问位置信息列表。对于其他查询，则要求搜索引擎对位置信息列表能执行高效的随机访问。词组查询就是随机访问的一个例子，等价于布尔与搜索（从本质上讲，在模式独立索引中处理词组查询与在文档编号索引中进行布尔与搜索的过程是等价的）。

回顾第 2 章，其中提到倒排索引提供的两种主要访问方式：next 和 prev，为给定词项返回在某一索引地址后（前）的第一次（最后一次）出现的位置。假设现在需要找出词组“iterative binary search”在 GOV2 中的所有出现位置。当发现词组“iterative binary”在该文档集中仅出现了一次，在位置［33 399 564 886，33 399 564 887］，调用函数

next(“search”,33 339 564 887)

就可以知道词组“iterative binary search”是否出现在文档集中。如果返回值是33 399 564 888，则表示文档集包含该词组，否则表示不包含。

如果位置信息列表以整数数组的形式存储在内存中，则通过二分查找（或**跳跃式搜索**（galloping search）——参见 2.1.2 节）该操作就能变得十分高效。因为词项“search”在 GOV2 中大概出现了 5000 万次，所以二分查找过程中需要

$$\lceil \log_2(5\times10^7) \rceil = 26$$

次随机列表访问。如果位置信息列表存储在磁盘上，检索过程的实现方式从理论上来讲可与上述过程一样。然而，磁盘并不是一种专门用于随机访问的设备，磁盘寻道是非常耗时的操作，这种方法代价就非常高了。考虑到盘片旋转延时和寻道开销，使用二分查找进行 26 次随机磁盘访问所需要的时间很容易就超过了 200 ms。

另外一种替代的方案就是只使用一个顺序读入操作将整个位置信息列表逐步加载到内存中，这样就可以避免昂贵的磁盘寻道操作。但这并不是一种好的解决方案。假设每个位置信息需要 8 字节的磁盘空间，那么计算机将需要超过 4 秒的时间把 5000 万个位置信息从磁盘加载到内存中。

为了对存储在磁盘上的位置信息列表提供高效的随机访问，每个列表都使用了一种辅助数据结构，称为**单项索引**（per-term index）。该数据结构存储于磁盘中，并位于对应的每个位置信息列表的头部。该数据结构包含了列表中位置信息子集的一个副本，例如，每 5000 个位置信息的一个副本。当需要访问给定词项 T 在磁盘上的位置信息列表时，在对该列表进行任何实际的索引操作之前，搜索引擎首先将词项 T 的单项索引加载到内存中。next 方法中所需的随机访问操作可以通过在内存里词项 T 的单项索引上进行二分查找来实现（确定大约是在磁盘上的位置信息列表中的哪 5000 个位置信息中），然后把对应的 5000 个位置信息加载到内存中，最后在这些位置信息上进行随机访问操作。

这种随机访问列表的方法有时也被称做**自索引**（self-indexing）（Moffat 和 Zobel，1996）。单项索引中的记录称为**同步指针**（synchronization points）。图 4-4 给出了词项“denmark”的位置信息列表，取自莎士比亚文集，该列表采用了单项索引，粒度为 6（即一个同步指针

指向表中的 6 个位置信息）。在图中，如果调用 next（250 000），该方法会首先确定从位置 248 080 开始的位置信息块是可能包含候选位置的数据块。把该数据块加载到内存中，并在该数据块上进行二分查找，最后返回 254 313。类似地，在前面对“iterative binary search”进行词组查找的例子中，对词项“search”的列表进行随机访问仅需 2 次磁盘寻道以及将约 15 000个位置信息（单项索引的 10 000 个位置信息和候选数据块包含的 5000 条记录）加载到内存中就能实现——共需约 30 ms 的执行时间。



图 4.4　一个关于“dearsirk”（取自莎士比亚文集）的模式独立位置信息列表，该列表采用单项索引：一个同步指针指向 6 个位置信息，同步指针的数据自列表长度模式地决定。[276]—5

需要合理选择单项索引的粒度，即两个同步指针间的记录数目。太大的粒度会增大两个同步指针间的数据量，也就是增大了每次随机访问时所需加载到内存的数据量；相反，太小的粒度会增大单项索引的体积，同时也就增大了初始化位置信息列表时需要从磁盘读入的数据量（练习 4.1 中要求为给定的列表计算其最优的粒度）。

可以想象，从理论上讲，对于一个包含上亿条记录的长位置信息列表，即使选用已经优化的单项索引（该索引的粒度能让总的磁盘搜索次数达到最少），该列表也会非常庞大以至于不能完全放入内存。在这种情况下，可以为单项索引再建立一个索引，甚至递归地产生多个这样的索引，最后就衍化成多层静态 B 树，它可为位置信息列表提供高效的随机访问。然而在实际应用中，这样复杂的结构很少能用上。一个简单的二级结构就足够了，此结构的每个磁盘上的位置信息列表上有一个单项索引结构。例如，文档集 GOV2 中出现最频繁的词项“the”，该词项在文档集中出现了约 10 亿次，在未经压缩前，使用模式独立索引的位置信息列表大约需要占据 80 亿字节的空间（每个位置信息占用 8 字节）。假设词项“the”上的单项索引为每 20 000 个位置信息设置一个同步指针，将整个包含 50 000 个记录的单项索引加载到内存中仅需要一次磁盘寻道，顺序传输 40 000 字节（≈4.4 ms）。对该词项位置信息列表的每次随机访问操作都需要一次额外的磁盘寻道，同时把 160 000 字节数据加载到内存中（≈1.7 ms）。因此，对该词项位置信息列表的一次随机访问，共需约 30 ms（两次随机磁盘访问，每次访问需要 12 ms，加上从磁盘读入 560 000 字节所需的时间）。作为比较，如果我们在单项索引上再做一级索引，那么一次随机访问过程将需要额外增加至少 3 次的磁盘寻道操作，这将会大大降低索引随机访问的性能。

与直接在磁盘上的位置信息列表进行二分查找相比，引入单项索引大大提高了随机访问操作的性能。然而，该方法真正强大之处在于允许存储可变长的位置信息，如这样的形式（文档编号，词频，（文档中出现位置）），特别是支持压缩的位置信息。如果位置信息不是以固定大小（如 8 字节）的整数形式存储的，而是以压缩形式存储的，那么即使是一个简单的二分查找也无法实现。然而，通过把位置信息都压缩成一些小的数据块，每个数据块头都对

应单项索引中的一个同步指针，搜索引擎对压缩的位置信息列表也能提供有效的随机访问。上述应用也很好地解释了“同步指针”的来源：同步指针能帮助在解码与编码之间建立同步，也就是说，允许在已经压缩的位置信息列表的任意位置开始解压数据。详细请参见第 6 章讲述的压缩倒排索引。

2. 前缀查询

如果搜索引擎必须支持类似于“inform＊”这样的前缀查询，那么位置信息列表必须按词项的字母顺序进行存储。在 GOV2 中，共有 4365 个不同词项的 6700 万次出现匹配“inform＊”这一前缀查询。通过将列表按字母顺序进行存储，就可以保证这 4365 个词项的倒排列表在倒排文档中彼此接近，因此在磁盘上也比较接近。这就减少了单个列表之间的寻道距离，因此可以获得更好的查询性能。如果所有列表都以某种随机形式存储在磁盘上，即使不考虑任何处理查询中所需的其他操作，光磁盘寻道和旋转延时就要将近 1 分钟（4365×12ms）。通过以对应词项的字母顺序来组织倒排列表，一个在所有文档中查询“inform＊”匹配的过程，若用词频索引，耗时不到 2 s；若用模式独立索引，耗时也才 6 s 左右。将倒排文档中的列表以某些预定义的顺序（如字母顺序）进行存储，对高效的索引更新也是很重要的，我们将会在第 7 章对其进行讨论。

3. 独立的位置索引

如果搜索引擎是基于以文档为中心的位置索引（包含文档编号、词频、词项在每个文档内出现的位置），那么将索引分成两个独立的倒排索引文件也是一种常见的做法：其中一个索引文件保存每个位置的文档编号和词频，另外一个索引文件保存着词项在每个文档中出现的具体位置。将索引文件分成两个的根本原因是，很多查询和很多评分函数都不需要访问到词项的位置信息。通过把这些信息从主索引中剔除出来，查询处理性能就可以得到提高。

4.4 交错词典和位置信息列表

对于很多文档集来说，词典所占的空间都比较少，可以完全存储在一台机器的内存中。然而大的文档集通常包含成千上万个不同的词项，所有的词典记录的合集也相当大，要将其全部放在内存中不是一件容易的事情。例如，文档集 GOV2 中约有 4900 万个不同的词项。这些词项（以“\0”作为结束符存储）共占用 482 MB 的空间。假设现在使用图 4-3所画的有序数组作为搜索引擎中词典的数据结构。在主数组中使用额外的 32 位指针，以及在辅数组中使用 64 位的文件指针，这样来维护词典中的每个词项就使得内存开销增加了 12×49＝588 MB（大约），即总共需要内存 1070 MB。因此，虽然文档集 GOV2 比较小，可以用一台机器就能对其进行管理，但是它的词典所占的空间太大导致无法将其整部存储到内存中。

在某种程度上，这个问题可以通过采用词典压缩技术（将会在 6.4 节讨论）解决。但是词典压缩技术并不是万能的。总是有这样的情况，由于文档集中不同词项的数量非常庞大，即使进行了压缩，也不能把整部词典存储到内存中。例如，考虑一种索引，每个位置信息列表不是代表一个词项，而是一个二元词组，如“information retrieval”。这样的索引对于处理词组查询来说是非常有用的。然而，文档集中不同二元词组的数量远比一元词项要多得多。从表 4-3 可见，GOV2 仅包含了 4900 万个词项，却有 5.2 亿个不同的二元词组。如果是对三元词组而不是二元词组进行索引，那么一点儿也不奇怪情况只会更糟糕——GOV2 中共有 23 亿个不同的三元词组，这样，把整部词典存储到内存更是不可能实现的了。

把整部词典存储于磁盘上的做法满足了空间上的要求，但却降低了查询的速度。不做任何更改，对于每个查询词项磁盘上的词典都至少多一次额外的磁盘寻道，这是因为搜索引擎

在处理给定查询之前，都需要先从磁盘上取出每个词项的词典记录。因此，单纯使用磁盘存储也不能得到令人满意的结果。

表 4-3　3 个文档集中一元词项、二元词组、三元词组的数量。二元词组的数量要比一元词项的数量约大一个数量级

	词条	一元词项	二元词组	三元词组
莎士比亚文集	1.8×10^5	3.5×10^4	2.9×10^5	8.5×10^5
[illegible]	3.3×10^8	1.2×10^6	4.5×10^7	9.4×10^7
GOV2	4.4×10^9	4.9×10^7	5.2×10^8	2.3×10^9

这个问题一个可能的解决方案是采用交错词典（interleaving dictionary），如图 4-5 所示。在交错词典中，所有的记录都存储于磁盘上，每个记录入口恰好位于对应位置信息列表之前，这样搜索引擎就可以通过一次顺序读操作将词典记录和位置信息列表取出。除了将数据存储于磁盘上，同时，保留某些（some）词典记录（不是所有）在内存中。当搜索引擎需要找到某个词项的位置信息列表时，它首先在内存词典记录的有序列表上进行一次二分查找，然后在得到的两个记录之间进行一次顺序扫描。如图 4-5 的例子所示，若要检索“shakespeareanism”，首先可以确定该词项的位置信息列表（如果在索引中）肯定位于“shakespeare”和“shaking”这两个位置信息列表之间，然后把这个区间内的索引项都加载到内存中，并按线性的方式扫描到“shakespeareanism”的词典记录（同时也就找到了它的位置信息列表）。

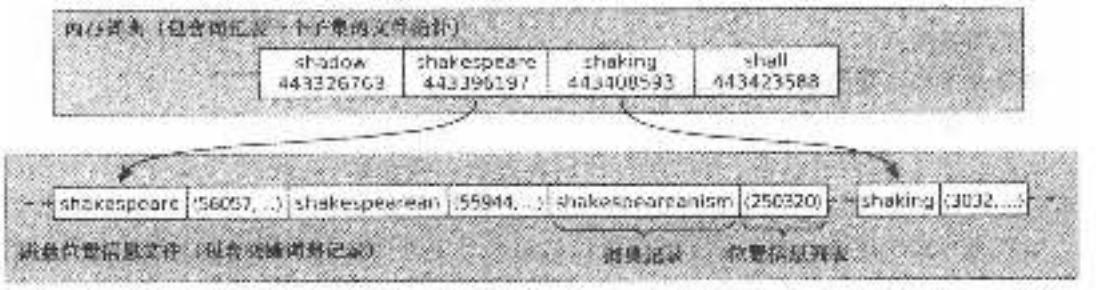


图 4-5　交错词典和位置信息列表：每个磁盘上的倒排列表都以对应词项的词典记录开头。内存词典仅包含某些（some）词项的记录。为了找寻“shakespeareanism”的位置信息列表，需要在磁盘数据“shakespeare”和“shaking”之间进行一次顺序扫描

交错词典与 4.3 节中的自索引技术非常相似，它们都是以顺序读入少量额外数据的方式避免了随机访问磁盘操作。因为对磁盘进行顺序访问比随机访问要快得多，所以只要从磁盘加载到内存的额外数据量不大，这种折中还是值得的。为了保证额外加载的数据量不大，我们需要对磁盘词典记录和须加载到内存中最接近该记录的词典记录之间的数据量设定一个上限。我们称这个上限为索引块大小（index block size）。例如，对于索引中的每个词项 T，如果在 T 的磁盘词典记录前，可以保证搜索引擎需要在磁盘上读取的数据量不超过 1024 字节，那么就可以说该索引的索引块大小为 1024 字节。

表 4-4 列出了交错词典对内存消耗和搜索引擎的列表访问性能的影响（使用 GOV2 文档集）。如果没有交错词典，搜索引擎需要维护内存中的 4950 万条词典记录，同时访问一个随机位置信息列表的第一个位置信息平均需要 11.3 ms（磁盘寻道十旋转延迟）。若把数据块的大小设定为 $B=1024$ 字节，则内存词典记录就可减少至 300 万条。同时，搜索引擎的列

表访问延时（访问某个随机选择的列表的第一个位置信息）仅增加了 0.1 ms——完全可以忽略的一个开销。随着数据块增大，内存词典记录数将减少，但是索引访问延时会增加。与完整的内存词典相比，即使是 $B=256$ KB 这样大的数据块，其额外的开销也不过是每查询几毫秒而已。

表 4-4 GOV2 的（4950 万个不同词项）模式独立索引在使用交错词典后的效果。当索引块的大小 $B=16\ 384$ 字节时，内存词典记录数减少了 99% 以上，而只是稍微减慢了查询速度：平均每个查询慢了 1 ms

索引块大小(字节)	1024	4096	16 384	65 536	262 144
内存词典记录数($\times 10^6$)	3.01	0.91	0.29	0.10	0.04
平均索引访问延迟(ms)	11.4	11.6	12.3	13.6	14.9

需要强调的是，数据块大小为 B 的交错词典，与为每 B 字节的索引数据维护一个内存词典记录相比，所耗费的内存空间是不同的。例如，文档集 GOV2 的模式独立索引（已压缩）的大小是 62 GB。可是，设置索引块大小为 $B=64$ KB，不等于词典记录是 100 万（62 GB/64 KB≈100 万）条，而是不到 1/10。这是因为像“the”和“of”这类高频词，尽管它们的位置信息列表占用的磁盘空间要远远大于 64 KB（“the”的压缩位置信息列表大小约为 1 GB），但是每个词项仅需要一个内存词典记录。

实际应用中，大小介于 4～16 KB 的数据块就能有效地将内存词典缩减至一个可接受的大小，特别是使用词典压缩技术（参见 6.4 节）将少数剩下的内存词典记录进一步压缩后，空间耗费更少了。对于每个查询，磁盘传输这样大小的数据块只需不到 1 ms，是不会引起任何性能问题的。

消除词项和位置信息间的差异性

通过消除词项和位置信息间的结构差异，同时将索引数据视为一系列这样的形式对（词项，位置信息），我们可以进一步改进交错词典方法。将磁盘上的索引分成一系列大小固定的索引块，每个块可包含 64 KB 数据。所有位置信息都按对应词项的字母顺序存储在磁盘上。与之前的做法相同，同一词项的位置信息也按升序排列。每个词项的词典记录有可能在磁盘上保存了多次，因此每个索引块中的词典记录就至少包含了词项的一个位置信息。内存中用于访问磁盘上的索引的数据结构是一个简单的数组，包含每个索引块的一个形式对（词项，位置信息），分别对应该索引块的第一个词项及其第一条位置信息。

图 4-6 给出了这种新的索引结构的一个例子（数据源于莎士比亚文集一个模式独立的索引）。在这个例子中，调用函数

next(“hurried”, 1 000 000)

时，首先会把第二个数据块（以“hurricano”开始）从磁盘加载到内存中，然后会搜索该数据块中与查询匹配的记录位置，最后会返回第一条匹配的记录位置（1 085 752）。调用函数

next(“hurricano”, 1 000 000)

时，首先会把第一个数据块（以“hurling”开始）从磁盘加载到内存中，但是没找到匹配的记录，接着会访问第二个数据块，并返回一条位置信息 1 203 814。

词典和位置信息列表的结合形式，将上面介绍的交错词典和 4.3 节介绍的自索引技术巧妙结合起来。通过这种索引结构，一次对任意词项的位置信息列表的随机访问只需要一次磁盘寻道（这里忽略了将每个词项的索引项加载到内存中这一初始化步骤）。该索引结构不好

的地方是，相比于分别采用自索引和变错词典这两项技术，它需要消耗更大的内存空间。若设数据块大小 B=64 KB，则一个 62 GB 的索引实际上需要将近 100 万个内存记录。



图 4-6　将词典和位置信息列表合并，索引被分成一个个大小均为 72 字节的数据块。内存中词典的每条记录形式为（词项，位置信息），对应给定索引块中第一个词项和第一条位置信息。为了便于阅读，在索引数据中加入了记录分隔符“#”

4.5　索引的构建

在本章前面的章节，你已经学习了有关倒排索引的各个组成部分。即使所有的位置信息列表都存储于磁盘上，同时在没有足够的内存空间为索引存放整部词典的情况下，通过将它们结合起来，依然可以对索引内容实现有效的访问。现在我们讨论如何为一个给定文档集高效地建立一个倒排索引。

从一个抽象的观点来看，一个文档集可视为是一个词项出现的序列——元组的形式为（词项，出现位置）对，其中出现位置（position）是从文档集开头作为开始位置的一个计数，词项（term）是出现在该位置上的词项。图 4-7 展示了上述理解方式下的莎士比亚文集的某个片段。当以顺序方式读入文集时，这些元组很自然地就会按其第二项进行排序，即词项在文档集的位置信息。索引构建的任务就是改变这些元组的顺序，使其按它们的第一项（即词项）来进行排序（原来第二项的关系被打破了）。一旦建立了新的顺序，创建合适的索引及其他的辅助数据结构就是一个相对容易的任务了。

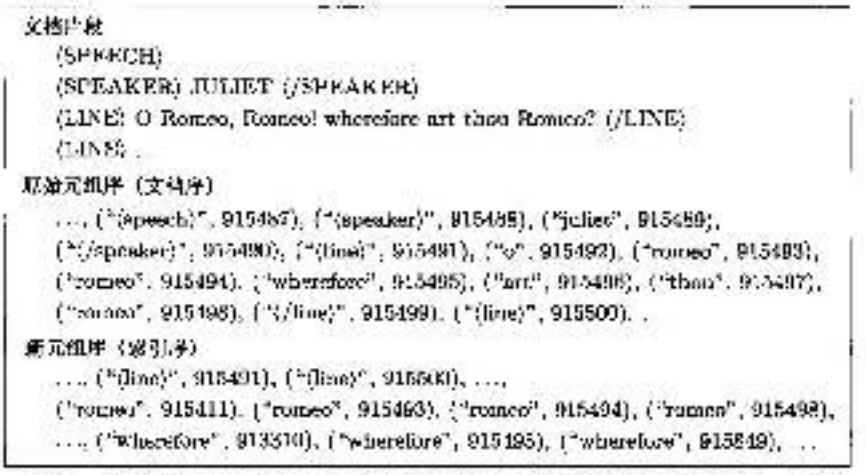


图 4-7　索引构建过程可视为是对文档集中的元组进行重排序的过程。元组由原先的文档序（collection order）（根据位置信息排序）重排为新的索引序（index order）（根据词项排序）

一般来说，建立索引的方法可以分为两类：基于内存的索引构建和基于磁盘的索引构建。基于内存的索引构建在主存里建立给定文档集的索引，因此只可用于比可用内存空间相对要小的文档集中。然而，它们是更复杂的基于磁盘的索引构建方法的基础。基于磁盘的方法用于为那些很大文档集，即比可用内存空间要大得多的文档集建立索引。

跟之前的做法一样，我们的讨论仅限于模式独立索引，因为以后在讨论更复杂的索引类型时会讨论到更多需要考虑的细节，如文档层次的位置索引，而现在我们可忽略这些细节，并将注意力放在算法的本质上。这里介绍的技术同样适用在其他类型的索引上。

4.5.1 基于内存的索引构建法

先考虑构建索引最简单的形式，即需要索引的文档集足够小，能整个放到内存中。为了为这样的文档集构建一个索引，索引过程需要维护以下数据结构：

- 一部词典，支持高效的单词项查找和插入操作；
- 一个可扩展（即动态）的链表数据结构，用于存放每个词项的位置信息。

假如这两个数据结构都已经构建好，那么索引构建过程就非常直观了，如图 4-8 所示。如果用合适的数据结构来实现词典和可扩展位置信息列表，那么这种方法就非常高效，搜索引擎可在 1 秒内对莎士比亚文档集建立索引。因此，需要讨论两个问题：1）应该为词典选用怎样的数据结构？2）应该为可扩展位置信息列表选用怎样的数据结构？

```
   buildIndex (inputTokenizer) ≡
1    position ← 0
2    while inputTokenizer.hasNext() do
3       T ← inputTokenizer.getNext()
4       从T中取出词典记录；如有必要，可创建新记录
5       在T的位置信息列表后扩展新的位置信息position
6       position ← position + 1
7    按字典序对所有词典记录排序
8    for each 词典中的词项 T do
9       将T的位置信息列表写入磁盘
10   将词典写入磁盘
11   return
```

图 4-8 基于内存的索引构建算法，使用两种抽象数据类型：基于内存的字典和扩展位置信息列表

1. 建立索引阶段的词典

索引构建过程中的词典实现方法需要对单词项查找和词项插入提供有效的支持。支持这类操作的数据结构是很常见的，很多公共程序库中都有。例如，SGI[⊖] 提供的 C++ 标准模板库（STL）中提供的 map 数据结构（二分查找树）和 hash_map 数据结构（可变长哈希表），都可执行索引构建阶段中需要的查询和插入操作。当你在实现自己的索引算法时，你可以尝试使用这些已实现的数据结构中的某一个。但是，并不建议在所有的情况下都这样做。

我们在 GOV2 的一个子集上度量 STL 数据结构的性能，首先为文档集中的前 10 000 个文本建立索引。结果如表 4-5 所示。乍看之下，STL 的 map 和 hash_map 的查找性能似乎能有效满足索引构建的需求。平均情况下，每个查找操作 map 需 630 ns；hash_map 要快一点，只需要 240 ns。

⊖ www.sgi.com/tech/stl/

表 4-5　索引文档集 GOV2 的前 10 000 个文档（约 1400 万个词条；181 334 个不同的词项）。每个词条的平均词典查找耗时都是微秒级。“哈希表”那些行表示一个自定义的基于带链表结构的定长哈希表的手工词典的实现

词典实现	查找时间	字符串比较
二分查找树（STL map）	0.63 μs 每符号单元	18.1 每符号单元
变长哈希表（STL hash_map）	0.24 μs 每符号单元	2.2 每符号单元
哈希表（2^{10}记录，向前插入）	6.11 μs 每符号单元	140 每符号单元
哈希表（2^{10}记录，向后插入）	0.37 μs 每符号单元	8.2 每符号单元
哈希表（2^{10}记录，前移）	0.31 μs 每符号单元	4.8 每符号单元
哈希表（2^{14}记录，向前插入）	0.32 μs 每符号单元	10.1 每符号单元
哈希表（2^{14}记录，向后插入）	0.09 μs 每符号单元	1.5 每符号单元
哈希表（2^{14}记录，前移）	0.09 μs 每符号单元	1.3 每符号单元

假设现在需要索引整个 GOV2 文档集，而不仅仅是前 10 000 个文本。假设查找操作性能维持大致一样的水平（一个理想化的假设，因为词典中的词汇数量将持续增长），那么用 hash_map 对 GOV 中的 440 亿个词条均执行一次查找操作，所需要的全部时间是：

$$44\times10^9\times240\text{ ns}=10\ 560\text{ s}\approx3\text{ h}$$

已知最快的公共检索系统为同样地文档集建立索引需时为 4 小时，包括读入文件、语法分析，并把（压缩后）索引写入磁盘，由此可见，我们还有提升的空间。这就需要自定义一个词典的实现。

在设计我们自己的词典实现时，我们要尝试优化要处理的数据类型的性能。由第 1 章可知，自然语言文档集中词频的出现大致符合齐夫分布。该分布的一个主要性质是文档集中大量的词条仅对应着少量的词项。例如，尽管文档集 GOV2 中大约有 5000 万个不同的词项，但超过 90% 的词项出现都是那 10 000 个最常见的词项。因此，大部分的词典查找操作都是为了加载那一小部分非常高频的词项。如果词典基于哈希表实现，且冲突通过链表（如图 4-2 所示）解决，则使这些频繁词项的词典记录都靠近每个链表的开头部分就非常重要了。这可以通过两种不同的方式来实现：

1）**向后插入启发式方法**（insert-at-back heuristic）。如果一个词项是比较频繁的，那么它在输入的词条流里的出现会较靠前。相反，如果一个词项的第一次出现位置相对靠后，那么这个词项就很有可能不是高频词。因此，当向已有哈希表中的某个链表中加入一个新词时，需要把该词项插到链表的末尾。这样做，高频词插入较早，就能留在链表的前部，而那些非高频词就会留在链表的后部。

2）**前移启发式方法**（move-to-front heuristic）。如果词项是文档集中的高频词，那么应该出现在哈希表的链表的前部。类似于向后插入启发式方法，前移启发式方法把新词项插入到对应链表的末尾。并且，当执行一次词项查找操作时，若在链表的前部没有找到对应的词项信息，那么就把该词项移到链表的头部。这样，当下一次查找该词项时，该词项的词典记录就会在哈希表的前部（或接近）。

我们评价这两种方法的查找性能（两种方法均使用哈希槽数目固定的自定义的哈希结构），同时将它们与第三种将新词项插入到对应链表头部的方法（称为**向前插入启发式方法**（insert-at-front heuristic））进行比较。表 4-5 列出了性能指标的结果。

向后插入和前移启发式方法的性能基本一样（哈希槽为 2^{14}个，每次查找需时 90 ns）。仅当哈希表较小时，前移启发式方法才比向后插入启发式方法略占优势（当哈希槽为 2^{10}个

时，要快 20%）。可是，向前插入启发式方法的性能表现就比较差，大约比前移启发式方法慢 3～20 倍。为了理解这个巨大性能差异的原因，请看表 4-5 的“字符串比较”那一列。当将 GOV2 的前 10 000 个文档的 181 344 个词项存储到一个有 16 384（2^{14}）个哈希槽的哈希表中时，我们期望平均每个链表的长度为 11。如果使用向前插入启发式方法，词典在找到查询词项前平均需要 10.1 次字符串比较操作。这是因为常见词项通常很早就出现在文档集中，而向前插入方法将它们放到了相应链表的后部。这就导致该方法查找性能较差。顺便提一句，STL 实现的 hash_map 也是将新的数据项插入到相应链表的头部，这也正是它为何在基准测试中性能不佳的原因。

采用前移启发式方法实现的基于哈希的词典对哈希表的大小非常敏感。实际上，即使在给一个包含成千上万个不同词项的文档集做索引时，一个较小的哈希表，也许只有 2^{16} 个哈希槽，就能有效地实现词典查找操作。

2. 可扩展常驻内存位置信息列表

除了词典实现，简单的内存索引构建方法还有一个有趣的事情，就是可扩展内存位置信息列表的实现。就像前面提到的，每个位置信息列表都以独立链表的形式实现，这样可以高效地支持将新的位置信息加入到已有列表中。这种方法的缺点是需要消耗大量的内存空间：对于每个 32 位（或 64 位）的位置信息，索引过程需要额外存储一个 32 位（或 64 位）的指针，因此导致存储空间增加 50%～200%。

已经提出了各种减少可扩展位置信息列表存储空间的方法。例如，可以使用定长数组代替链表。这样就可以减少链表中的 next 指针造成的额外开销。遗憾的是，每个词项的位置信息列表的长度并不能预先知道，因此这种方法需要对输入数据进行额外的扫描：第一遍扫描，收集词项的统计信息，并为每个词项的位置信息列表分配空间；第二遍扫描，将位置信息存储到预分配的数组中。当然，两次读入输入数据将会降低索引性能。

另外一个解决方法是为词典中的每个词项预先分配一个位置信息数组，但允许索引过程中动态改变数组的大小，例如，调用函数 realloc（假设编程语言和运行时环境支持这类操作）。当发现一个新词项时，就给该词项的位置信息赋予一个**初始**（init）空间（例如，$init=16$ 字节）。每当已有数组空间已满，就调用 realloc 操作去增大该数组。通过采用按比例预分配的策略，也就是，在重分配操作中总共分配以下数量的字节。

$$s_{new}=\max\{s_{old}+init,k\times s_{old}\} \tag{4-2}$$

其中，s_{old}是该数组原本的大小，k 是一个常数，称为**预分配因子**（pre-allocation factor），这样调用函数 realloc 的次数就会比较少（每个位置信息列表所占空间大小的对数）。尽管如此，该方法还是存在一些问题的。如果预分配因子设置太小，那么在索引构建过程中将会引发大量的重分配操作，有可能会影响搜索引擎的索引性能。如果太大，由于分配的空间没用上，就会浪费掉大量的内存空间（内部碎片）例如，如果 $k=2$，那么平均 25%的分配内存将空置。

第三种以消耗较少内存空间实现可扩展内存位置信息列表的方法是采用一种特别的链表结构，多个位置信息共用一个 next 指针，而指针指向**信息组**（group），而不是单个的位置信息。当向词典中加入一个新词项时，为它的位置信息分配一个小空间，比如 16 字节。当一个词项的位置信息列表的预留空间用完时，就新建一个位置信息组。需要分配给这个组的空间大小为

$$s_{new}=\min\{limit,\max\{16,(k-1)\times s_{total}\}\} \tag{4-3}$$

其中，$limit$ 是一个位置信息组的大小上界，例如 256 个位置信息；s_{total}是指当前已分配给位置信息的内存大小；k 是指预分配因子，与 realloc 中的一样。

将多个位置信息合并为组，以及若干个位置信息共用一个 next 指针的情况如图 4-9 所示。每个列表节点保留多个值的链表数据结构有时候被称做松散链表（unrolled linked list），与编译器优化技术中使用的循环展开技术类似。在索引构建过程中，我们称这种方法为**分组**（grouping）。

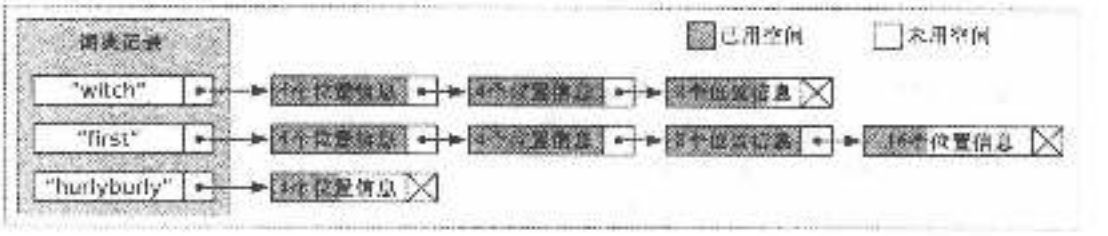


图 4-9　单项可扩展位置信息列表，链表的每一项是一个位置信息组（松散链表（unrolled linked list））。按比例预分配资源（这里预分配因子 $k=2$）为两节两类空间开销所占的比例提供了一个有用的方法。这两类空间开销是：由 next 指针导致的额外空间开销和由内部碎片导致的额外空间开销

分组技术中对预分配空间设置一个上界可以控制内部碎片。同时，又不会构成性能上的问题；并不像 realloc 中的那样，为已有列表预留更多的空间是一个非常轻量级的操作，不需要索引过程为任何信息数据重新分配。

4 种链表空间分配策略上的性能评价——简单链表、两遍扫描、realloc 和分组链表——如表 4-6 所示。两遍扫描法消耗最少的内存空间，但耗时几乎是分组链表的两倍。使用预分配因子 $k=1.2$，realloc 比空间最优的两遍扫描法要多用约 10% 的内存空间，这与约一半的预分配内存空间未使用的假设是一致的。另外，分组链表法仅比两遍扫描法多用 3% 的内存空间。除此之外，分组链表要比 realloc 约快 15%（CPU 时间分别为 61 秒和 71 秒）。

表 4-6　索引文档集 TREC45。使用不同内存分配策略去管理可扩展内存位置信息列表（32 位位置信息；32 位指针）时，索引构建的性能。以小组的形式组织同一词项的位置信息，并且使用链表将它们链接起来的策略比其他策略都要快。realloc 和分组（grouping）中使用的预分配因子 $k=1.2$

分配策略	内存消耗	时间（总计）	时间（CPU）
链表（简单）	2312 MB	88 s	77 s
两遍索引	1166 MB	123 s	104 s
realloc	1282 MB	92 s	71 s
链表（分组）	1208 MB	71 s	61 s

也许你会很惊讶地发现，位置信息分组之间的链接也比单个位置之间的链接要快（CPU 时间分别为 61 秒和 77 秒）。原因是分组技术使用了松散链表结构，这不但减少了内部碎片，而且通过将同一个词项的位置信息都存储在一起，提高了 CPU 缓存的利用率。在简单链表法的实现中，某个词项的位置信息都是随机分布在计算机内存的各个位置中，导致将该词项的位置信息写回磁盘前执行的记录收集操作引起了大量的 CPU 缓存丢失。

4.5.2　基于排序的索引构建法

前面的章节已经证明了基于哈希的内存索引的构建算法可以非常高效。但是，如果我们要索引比可用内存容量大得多的文档集，就不能使用这种将整个索引放在主存中的方法了，

而是要寻找基于磁盘的方法了。本节介绍的基于排序的索引法就是本章要介绍的两种基于磁盘的索引构建方法中的一种。该方法能用于索引比可用内存容量大得多的文档集。

回顾本节开始部分，我们将倒排索引的建立过程视为是将代表文档集的词项位置序列元组重排的过程——将它从文档序转换为索引序。这是一个排序过程，而基于排序的索引构建正是用最简单的方式实现了这个转换。从输入文件读入词条后，基于排序的索引以（termID，position）的格式对其进行记录，并立即写入磁盘。最终将得到一系列按第二项（出现位置）进行排序的记录。在所有输入数据都被处理完毕后，索引器按第一项对写入磁盘的记录进行排序，而不是按第二项。最终得到按第一项（词项编号）排序的新记录序列。这样，用内存词典中记录的信息，就很容易将这个新序列转换为一个适当的倒排索引了。

图 4-10 的伪码所示的方法易于实现，适用于构建比可用内存容量大得多的索引。它的主要局限是可用磁盘空间的大小。

```
    buildIndex_sortBased (inputTokenizer) ≡
1   position ← 0
2   while inputTokenizer.hasNext() do
3       T ← inputTokenizer.getNext()
4       从T中取出词典记录；如有必要，可创建新记录
5       termID ← unique term ID of T
6       将记录 R_position ≡ (termID, position)写入磁盘
7       position ← position + 1
8   tokenCount ← position
9   按第一项排序R_0 ... R_{tokenCount-1}；打乱按第二项排的序
10  在R_0 ... R_{tokenCount-1}上执行一次顺序扫描，创建最终的索引
11  return
```

图 4-10 基于排序的索引的建立算法，该算法为文档集构建一个模式独立的索引。算法主要的难点在于如何高效的排序磁盘上的记录

排序磁盘上的记录可能有点棘手。在很多实现方法中，都是一次将某一数量 n 的记录加载到内存中，其中 n 由记录大小和可用内存空间容量决定。这 n 个记录在内存中排序后写回磁盘。重复这一过程直到我们有$\lceil \frac{tokenCount}{n} \rceil$个有序记录块。这些块可通过**多路合并操作**（multiway merge operation）（同时处理所有的数据块）或**级联合并操作**（cascaded merge operation）（如一次合并两个数据块）进行合并，最后得到一个包含 *tokenCount* 条记录的有序序列。图 4-11 展示了一个建立基于排序的索引的过程，该过程使用级联合并操作，一次合并两个元组序列，得到一个最终元组序列。用一些辅助的数据结构，可将位置信息中的词项编号（termID）删除，这个最终的序列可视为文档集的索引。

尽管可以索引比可用内存容量大得多的文档集，但是基于排序的索引方法有两个明显的局限性：

- 它需要大量的磁盘空间，通常每个输入词条需要至少 8 字节的磁盘空间（4 字节词项编号+4 字节出现位置），对于更大的文档集或达到 12 字节（4+8）。例如当索引 GOV2 文档集时，用于存储临时文件的磁盘空间至少为 $12 \times 44 \times 10^9$ 字节（= 492 GB）——比未经压缩的文档集本身（426 GB）还要大。
- 为了能正确排序第一步中生成的（termID，docID）对，索引过程需要维护全局的词项编号（termID），这只能通过一部完整的内存词典来实现。我们前面讨论过，GOV2 的完整内存词典需要耗费超过 1 GB 的内存，因此在配置较低的计算机上就难以实现文档集索引了。

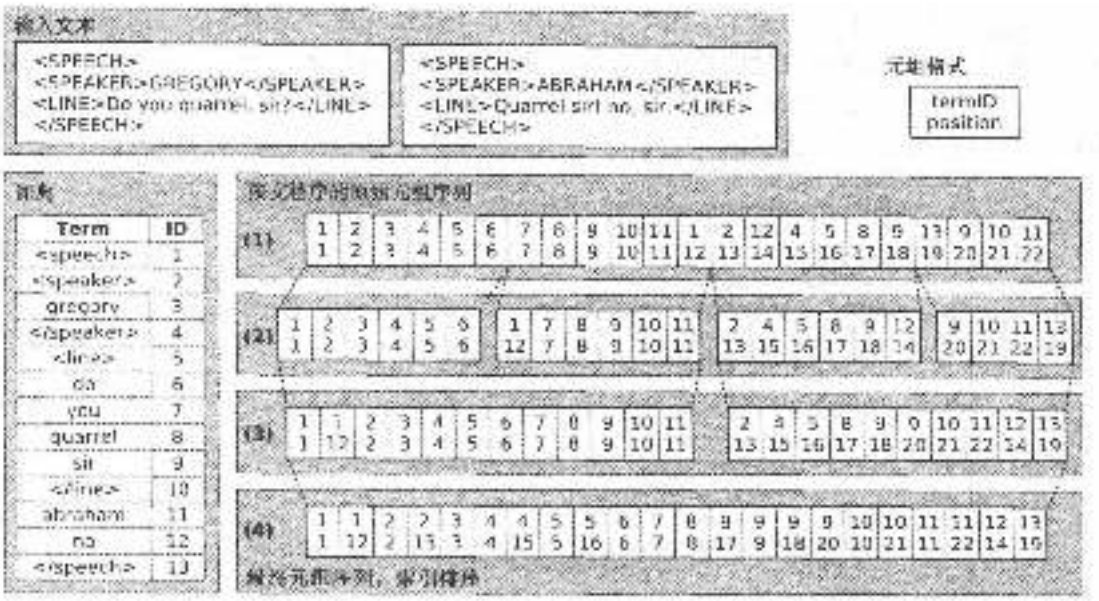


图 4-11　使用全局词项编号建立基于排序的索引。内存足够大，可以一次处理 6 个（词项编号，出现位置）元组。(1)→(2)：在内存中排序大小小于等于 6 的数据块，每次处理一个数据块。(2)+(3)和(3)+(4)：将有序数据块合并成更大的数据块

有很多方法可以解决上述问题，但是最终它们都是一致的，即将基于排序的索引方法转化为某种与基于合并的索引构建法（merge based index construction）类似的方法。

4.5.3　基于合并的索引构建法

对比起基于排序的索引构建法，本节介绍的基于合并的方法不需要维护任何全局数据结构。特别是不需要维护全局唯一的词项编号。因此需要索引的文档集的大小，仅受限于存储临时数据和最终索引的可用磁盘空间容量，而不受索引过程中可用内存容量的限制。

基于合并的索引法是 4.5.1 节中介绍的常驻内存索引构建法的一个一般化形式，利用哈希表查找建立倒排列表。实际上，如果文档集比较小，为其建立的索引足以放入内存，那么基于合并的索引法和常驻内存索引构建法是非常相似的。如果文档集太大使得对应的索引无法完全存入内存中，那么索引过程将为文档集进行动态划分（dynamic partitioning）。即一开始先建立一个常驻内存索引，一旦内存不足（或达到了预定义的内存使用阈值），就将常驻内存索引数据传输到磁盘上，建立一个磁盘的倒排文档，并删去内存中的索引，然后继续建立索引。该过程重复执行直到索引构建完毕，图 4-12 展示了该算法。

上述过程得到的结果是一个倒排文档集，每一个都代表了整个文档集的某一部分，每个这样的子索引都被称为是一个索引块（index partition）。最后一步，将所有这些索引块合并成为最终索引，即整个文档集的索引。索引块（和最终索引）中的位置信息列表通常以压缩形式存储（参见第 6 章），这样就可以使磁盘 I/O 的开销较少。

写到磁盘上的索引块作为索引过程中的中间输出彼此是完全独立的。例如，这里不需要全局唯一的词项编号；甚至不需要数字形式的词项编号。每个词项就是它的编号；每个索引块的位置信息列表按对应词项的字母顺序进行存储，访问某个词项的位置信息列表可以通过 4.3 节和 4.4 节介绍的数据结构实现。由于使用了字母顺序并且没有词项编号，将各个索引

块合并成最终索引就比较容易了。图 4-13 给出了一个简单实现方式的伪码，即重复线性探测所有的索引块。如果索引块的数量很大（超过 10），那么算法可使用一个优先队列（如堆结构）重组索引块来提高效率，即根据对应的索引块中下一词项来进行排序。这样就不必进行第 7～10 行中的线性扫描了。

```
   buildIndex_mergeBased (inputTokenizer, memoryLimit) ≡
1    n ← 0   // 初始化索引块数量
2    position ← 0
3    memoryConsumption ← 0
4    while inputTokenizer.hasNext() do
5       T ← inputTokenizer.getNext()
6       从T中取出词典记录；如有必要可创建新记录
7       在T的位置信息列表后扩展新的位置信息position
8       position ← position + 1
9       memoryConsumption ← memoryConsumption + 1
10      if memoryConsumption ≥ memoryLimit then
11         createIndexPartition()
12   if memoryConsumption > 0 then
13      createIndexPartition()
14   合并索引块I0 ... In−1，得到最终的磁盘索引Ifinal
15   return

   createIndexPartition () ≡
16   创建空磁盘倒排文件In
17   按字典序排序内存中的词典记录
18   for each词典中的词项T do
19      将T的位置信息列表添加到In中
20   删除所有内存位置信息列表
21   重置内存词典
22   memoryConsumption ← 0
23   n ← n + 1
24   return
```

图 4-12 基于合并的索引算法，生成一系列独立的子索引（索引块）。通过多路合并操作将索引块合并起来得到最终索引

```
   mergeIndexPartitions (⟨I0, ..., In−1⟩) ≡
1    创建空倒排文件Ifinal
2    for k ← 0 to n − 1 do
3    打开索引块Ik，进行顺序处理
4    currentIndex ← 0
5    while currentIndex ≠ nil do
6       currentIndex ← nil
7       for k ← 0 to n − 1 do
8          if Ik中仍有词项 then
9             if (currentIndex = nil) ∨ (Ik.currentTerm < currentTerm) then
10               currentIndex ← Ik
11               currentTerm ← Ik.currentTerm
12      if currentIndex ≠ nil then
13         Ifinal.addPostings(currentTerm, currentIndex.getPostings(currentTerm))
14         currentIndex.advanceToNextTerm()
15   delete I0 ... In−1
16   return
```

图 4-13 将编号为 $I_0 \cdots I_{n-1}$ 的 n 个索引块合并成最终索引 I_{final}，这是建立基于合并的索引构建方法的最后一个步骤

表 4-7 给出了基于合并的索引构建方法的总体性能，包括了各部分的性能。为 GOV2 建立一个模式独立的索引总共需要大约 4 小时。最后的合并操作，即将 n 个索引块合并为一

个最终索引的时间耗费，大约是生成这些索引块的时间的 30%。

表 4-7　用基于合并的索引构建方法对不同的文档集建立模式独立索引，内存索引使用 512 MB 的内存。索引阶段的词典通过有 2^{16} 个哈希槽的哈希表和前移启发式方法实现。可扩展内存位置信息列表采用松散链表的形式，即链接的是位置信息组，预分配因子 $k=1.2$

	读入、解析 & 索引	合并	总时间
莎士比亚文集	1 s	0 s	1 s
TREC45	71 s	11 s	82 s
GOV2（10%）	20 min	4 min	24 min
GOV2（25%）	51 min	11 min	62 min
GOV2（50%）	102 min	25 min	127 min
GOV2（100%）	205 min	58 min	263 min

该算法是可扩展的：在我们的计算机中，索引整个 GOV2 文档集（426 GB）的时间大约是索引一个 10%的子文档集（43 GB）的 11 倍。

但是，这个算法的可扩展性还是存在一些局限。在建立索引的最后合并索引块时，每个索引块至少有一个大小合适的预读缓冲区是非常重要的，大约几百 KB。这有助于将磁盘寻道（在不同的索引块间跳转）的次数保持在一个较低的水平。自然的，每个索引块预读缓冲区的大小不能超过 M/n，其中 M 是可用内存容量，n 是索引块的数量。因此，如果 n 变得太大，那么合并操作将变得很慢。

减少可用内存容量会对索引过程造成两方面的影响：第一，这样就减少了预读缓冲区的可用内存容量。第二，增加了索引块的数量。因此，减少 50%的内存容量将导致每个索引块的预读缓冲区的大小减少了 75%。例如，把内存限制为 $M=128$ MB，就导致需要合并 3032个索引块，每个仅有 43 KB 的预读缓冲区。图 4-14 画出了这个影响的总体趋势。由图可知，最后合并操作的性能高度依赖于索引过程中可用的内存容量。当可用内存容量分别为 128 MB 和 1024 MB 时，前者最后合并操作的时间大约是后者的 6 倍。

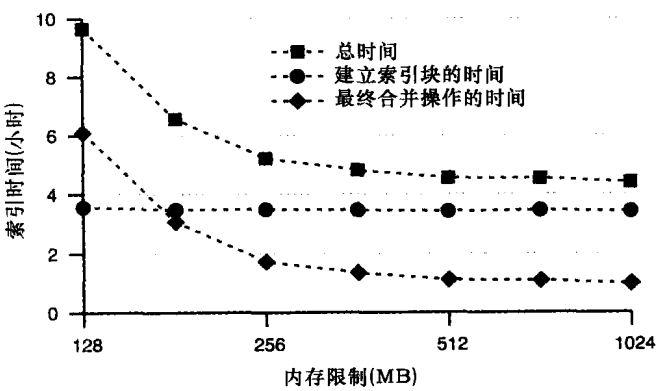


图 4-14　可用内存大小对基于合并的索引的性能造成的影响（数据集：GOV2）。第一阶段（生成索引块）的性能在很大程度上独立于可用内存容量。如果可用内存容量很少，那么第二阶段（合并索引块）的性能就会受到严重的影响

有两种对策可以克服这个问题。第一种是用级联合并法代替多路合并法。例如，需要合并 1024 个索引块，首先执行 32 路合并操作，每次合并 32 个索引块，然后再将得到的 32 个新的索引块合并为最终索引。图 4-11 给出了一般化过程，每次级联合并 2 个索引块。第二

种方法是通过压缩减少位置信息列表所需的内存空间。当需要往内存索引加入一条位置信息时，即时进行压缩，使得在超过内存容量之前索引过程可以尽可能地处理更多的位置信息，从而减少需要在磁盘上创建的索引块的数量。

总的来说，尽管最后的合并操作存在一些问题，基于合并的索引构建法还是让我们得以为大文档集建立倒排文档，即使在一台 PC 上也能实现这个过程。它优于基于排序的方法的地方是不需要全局唯一的词项编号。因此当词典词项数量非常多的时候，这种方法就特别有吸引力了。比起基于排序的方法，这种索引构建算法的另一个优点是它产生的内存索引能立即用于查询。这个性质是搜索引擎处理动态文档集时所必备的（参见第 7 章）。

4.6 其他索引

在我们讨论索引的数据结构时，把范围都限制在倒排索引。然而，倒排索引只是搜索引擎所用索引结构的其中一种。

前向索引（forward index）（或**直接索引**（direct index））将文档编号映射到文档中词项的列表中去。前向索引补充了倒排索引。它们通常不用于实际的搜索过程，而是用于在查询阶段获得文档词项分布的信息，这也是查询扩展技术如伪相关反馈（参见第 8 章）和产生结果**片段**（snippet）所需要的。对比起直接从原始文本中提取出这些信息，前向索引的好处是文本已经被分好词了，并且相关数据的提取会更有效。

签名档（signature file）（Faloutsos 和 Christodoulakis，1984）是另一种文档编号索引。与**布隆过滤器**（Bloom filter）（Bloom，1970）相似，签名档可用于获取一个**可能**（may）包含给定词项的文档列表。为了找出词项是否真的出现在一个文档中，文档本身（或前向索引）需要被考虑进去。通过改变签名档的一些参数，就有可能实现以时间换速度：使用小索引会使误检出现的可能性更大，反之亦然。

后缀树（suffix tree）（Weiner，1973）和**后缀数组**（suffix array）（Manber 和 Myers，1990）可用于找出给定 n-gram 序列在指定文档集中所有出现的位置。它既可以用于索引字符 n-gram（还没对输入文本进行分词处理），也可以用于单词 n-gram（经过分词处理后）。后缀树对词组搜索和正则表达式搜索是一个很有吸引力的数据结构，但通常比倒排索引要大，并且存储在磁盘而不是内存的时候搜索操作的性能比较低。

4.7 总结

本章介绍了建立和访问倒排索引的主要算法和数据结构。本章的要点有：

- 倒排索引通常所需的空间较大，而不能全部放入内存中。因此，通常的做法是只把相对于整个索引而言较小的词典加载到内存中，而把位置信息列表存储在磁盘上（4.2 节）。
- 对于较大的文档集，词典也可能因过大而不能全部放入内存。通过把部分词典保存于内存中，同时将词典记录和保存在磁盘上的位置信息列表交错处理，并将索引中的所有位置信息列表都按字母顺序存储，就可以大大减少词典对内存空间的需求（4.4 节）。
- 如果搜索引擎要支持前缀查询，那应该使用基于排序的词典，同时位置信息列表应该以字母顺序存储到磁盘上（4.2 节和 4.3 节）。
- 对于磁盘上的位置信息列表，使用含有部分词项位置信息的单项索引可以有效地实现对位置信息列表的半随机访问（4.3 节）。

- 采用带前移启发式方法的基于哈希的内存词典和分组链表技术实现的可扩展内存位置信息列表，可实现高效的内存索引构建方法（4.5.1 节）。
- 如果索引过程中的可用内存容量不足以允许完全在内存中建立索引，那么内存索引构建方法可扩展为基于合并的方法，即基于可用内存容量动态地将文档集划分为多个子文档集。每个子文档集的索引使用内存索引方法构建。整个文档集的索引建立好之后，采用多路合并或级联合并技术，就能把各个子文档集的索引合并起来（4.5.3 节）。
- 基于合并的索引构建法的性能与文档集的大小线性相关。然而，如果分配给子索引预读缓冲区的内存空间太少，那该方法最后的合并操作的性能将受到严重的影响（4.5.3 节）。

4.8　延伸阅读

Witten 等人（1999，第 3 章和第 5 章）以及 Zobel 和 Moffat（2006）为我们进一步理解倒排索引的结构和性能提供了一个很好的切入点。如果要更深入地了解索引的数据结构，可以参见 Google 的两位工程师 Brin 和 Page 在 1998 年发表的一些论文。

Moffat 和 Zobel（1996）讨论了倒排文档在查询阶段的性能问题，包括 4.3 节中介绍的用于随机列表访问的单项索引结构（“自索引”）。Rao 和 Ross（1999，2000）证明了随机访问所引起的问题不仅出现在磁盘索引中也会出现在内存倒排文档中。并且他们指出二分查找并**不是**（not）实现随机访问内存位置信息列表的最好方法。

Heinz 和 Zobel（2003）讨论了单遍合并的索引构建方法及其对比起基于排序的方法的优势。他们还评价了各种内存词典实现方法的效率，包括 4.5.1 节中描述的前移启发式方法（Zobel 等人，2001），并提出了一个新的词典数据结构，**爆炸树**（burst tree）（Heinz 等人，2002），可获得接近哈希表的单词项查找性能——但不像哈希表——它能实现前缀查询。

Büttcher 和 Clarke（2005）研究了可扩展内存位置信息列表（如松散链表）的内存管理策略，最近的研究成果可以参见 Luk 和 Lam（2007）。

如果以简单的方法实现基于合并的索引构建法（和基于排序的索引构建法的变种）的最后一个步骤，总的空间需求是最终索引的两倍。Moffat 和 Bell（1995）提出了一个聪明的方法，在原位上实现合并，即通过重用输入索引块中已处理部分的磁盘空间来存储最终索引。

Faloutsos 和 Christodoulakis（1984）对签名档给出了一个很好的概述，其中包括一些理论性质。Zobel 等人（1998）讨论了在进行文本搜索时，倒排文档和签名档的相对性能。他们的结论是，对于很多应用，倒排文档是个更好的选择。但 Carterette 和 Can（2005）指出，在某些情况下，签名档几乎与倒排文档一样快。

后缀树第一次出现在 Weiner（1973）的一篇论文中，名字为**位置树**（position tree）。Ukkonen（1995）为后缀树提出了线性时间的构建算法。Clark 和 Munro（1996）讨论了一个变种的后缀树结构。当数据存储于磁盘上而不是内存中时，该结构能提供高效的搜索操作。

4.9　练习

练习 4.1　在 4.3 节中我们介绍了**单项索引**（per-term index）的概念，即提高索引随机访问性能的一种方法。假设某些词项的位置信息列表由 6400 万个位置信息组成，每条位置信息占 4 字节。为了能在这个词项位置信息列表上执行随机访问，搜索引擎需要两个读磁盘操作：

1）将单项索引（同步点列表）加载到内存中。

2）将位置信息块 B 加载到内存中，其中 B 由同步指针列表上的二分查找确定。

我们将每个同步指针中包含的位置信息个数称为单项索引的**粒度**（granularity）。对于以上访问模式，最理想的粒度是多大（即能将磁盘 I/O 减到最小的那个粒度）？从磁盘中读出的总字节数为多少？

练习 4.2 在 4.3 节中介绍的单项索引是出于典型磁盘驱动的操作特性（尤其是高代价的磁盘寻道）。然而，也可以用同样地方法提高内存索引的随机访问性能。为了证实这一说法，请为内存位置信息列表设计两个不同的数据结构，并分别实现 next 访问函数（参见第 2 章）。第一个数据结构将位置信息存储在一个简单的 32 位整型数组中。它的 next 函数通过在数组中进行二分查找实现。在第二个数据结构中，使用一个辅助数组为位置信息列表中的每 64 个位置信息保存一个副本。next 函数首先在辅助数组中进行二分查找，然后**顺序扫描**（sequential scan）位置信息列表中的 64 条候选记录。分别度量两种实现的单词项查找平均延时，其中位置信息列表中有 n（$n=2^{12}$，2^{16}，2^{20}，2^{24}）个位置信息。描述并分析你的发现。

练习 4.3 建立倒排索引本质上是一个排序过程。一般排序算法的最低复杂度是 $\Omega(n\log(n))$。然而，4.5.3 节中基于合并的索引构建法所需的时间与文档集的大小线性相关（参见表 4-7）。请至少找出两处隐含对数因子的地方。

练习 4.4 图 4-12 所示的算法中，内存上限就是存储在内存中的位置信息数目。这个内存上限的定义是基于什么样的假设给出的？请给出一个文档集或一个实际应用，使得该假设不成立。

练习 4.5 在 4.5.1 节中，我们讨论了各种词典数据结构的性能特性，指出基于哈希表的实现（在索引构建阶段）可对单词项查询提供更高的性能，而基于排序的实现则更适用于多词项查询（前缀查询中需要用到）。设计并实现一种数据结构，使得该数据结构在单词项查询中优于基于排序的词典结构，并且在前缀查询中优于基于哈希表的实现方法。

练习 4.6（项目练习） 设计并实现一种索引构建方法，为给定的文档集建立一个模式独立索引。所建立的索引是一个驻留磁盘的索引。该索引不需要用到 4.3 节及 4.4 节中讨论到的任何优化处理。

- 实现 4.5.1 节中描述的内存索引构建法。当为一般英文文本建立索引时，你可以为包含词条数约为 $\frac{M}{8}$ 的文档集建立索引，其中 M 为可用内存的大小，以字节为单位。
- 扩展你的实现，使得文档集大小不再受索引过程中可用内存容量的限制。这就可能需要你建立一个模型，将两个或多个驻留磁盘索引合并成一个索引。

4.10 参考文献

Bender, M., Michel, S., Triantafillou, P., and Weikum, G. (2007). Design alternatives for large-scale Web search: Alexander was great, Aeneas a pioneer, and Anakin has the force. In *Proceedings of the 1st Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR)*, pages 16–22. Amsterdam, The Netherlands.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Brin, S., and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117.

Büttcher, S., and Clarke, C. L. A. (2005). *Memory Management Strategies for Single-Pass Index Construction in Text Retrieval Systems*. Technical Report CS-2005-32. University of Waterloo, Waterloo, Canada.

Carterette, B., and Can, F. (2005). Comparing inverted files and signature files for searching a large lexicon. *Information Processing & Management*, 41(3):613–633.

Clark, D. R., and Munro, J. I. (1996). Efficient suffix trees on secondary storage. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 383–391. Atlanta, Georgia.

Faloutsos, C., and Christodoulakis, S. (1984). Signature files: An access method for documents and its analytical performance evaluation. *ACM Transactions on Information Systems*, 2(4):267–288.

Heinz, S., and Zobel, J. (2003). Efficient single-pass index construction for text databases. *Journal of the American Society for Information Science and Technology*, 54(8):713–729.

Heinz, S., Zobel, J., and Williams, H. E. (2002). Burst tries: A fast, efficient data structure for string keys. *ACM Transactions on Information Systems*, 20(2):192–223.

Luk, R. W. P., and Lam, W. (2007). Efficient in-memory extensible inverted file. *Information Systems*, 32(5):733–754.

Manber, U., and Myers, G. (1990). Suffix arrays: A new method for on-line string searches. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 319–327. San Francisco, California.

Moffat, A., and Bell, T. A. H. (1995). In-situ generation of compressed inverted files. *Journal of the American Society for Information Science*, 46(7):537–550.

Moffat, A., and Zobel, J. (1996). Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379.

Rao, J., and Ross, K. A. (1999). Cache conscious indexing for decision-support in main memory. In *Proceedings of 25th International Conference on Very Large Data Bases*, pages 78–89. Edinburgh, Scotland.

Rao, J., and Ross, K. A. (2000). Making B^+-trees cache conscious in main memory. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 475–486. Dallas, Texas.

Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica*, 14(3):249–260.

Weiner, P. (1973). Linear pattern matching algorithm. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11. Iowa City, Iowa.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images* (2nd ed.). San Francisco, California: Morgan Kaufmann.

Zobel, J., Heinz, S., and Williams, H. E. (2001). In-memory hash tables for accumulating text vocabularies. *Information Processing Letters*, 80(6):271–277.

Zobel, J., and Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys*, 38(2):1–56.

Zobel, J., Moffat, A., and Ramamohanarao, K. (1998). Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490.

第5章

Information Retrieval: Implementing and Evaluating Search Engines

查询处理

在第4章中，我们阐述了构成倒排索引的基本数据结构。现在我们讨论如何在这些数据结构上实现高效的搜索操作。对于不同的搜索引擎，搜索过程的细节与实现都不同。然而，无论我们将要讨论的是单用户桌面搜索系统还是大规模的Web搜索引擎，基本思想和算法通常是相同的。

最基本的检索模型与我们在2.2节讨论过的布尔模型非常相似。在布尔模型中，每一个词项代表了包含它的文档集。文档集可用标准的操作符AND（集合交）、OR（并）以及NOT（非）进行合并。本章中探讨布尔模型最常见的两个替代模型。第一个是排名检索（5.1节），搜索引擎可以根据与查询的相关性对搜索结果进行排名。第二个是轻量级结构（5.2节），它是在子文档层次上对布尔模型的一种自然扩展。它不将搜索过程局限在整个文档中，而是允许用户搜索满足类布尔约束的任何文本段落（如，请给出在10个单词范围内包含“apothecary”和“drugs”的所有段落）。

尽管偶尔会提及有效性指标，如平均查准率均值（MAP），但本章主要关注搜索过程的效率而不是搜索结果的质量。各种检索方法的质量问题会在第8章和第9章中涉及。

5.1 排名检索的查询处理

正如2.2节指出的一样，布尔检索和排名检索不是互斥的，而是互补的：搜索引擎根据对用户查询的布尔解析来确定匹配的文档集。例如，给定查询（TREC主题433）：

$$Q = \langle \text{“greek”}, \text{“philosophy”}, \text{“stoicism”} \rangle \tag{5-1}$$

搜索引擎将检索出所有匹配**合取**（conjunctive）布尔查询

$$\text{“greek” AND “philosophy” AND “stoicism”} \tag{5-2}$$

或**析取**（disjunctive）布尔查询

$$\text{“greek” OR “philosophy” OR “stoicism”} \tag{5-3}$$

的所有文档，然后将这些文档根据与Q的相似性进行排名。例如，余弦相似性指标（公式(2-12)）。

传统的信息检索系统通常采用析取的方法，然而Web搜索引擎偏向采用合取查询方式。合取检索模型比析取模型的查询过程更快，因为需要评分和排名的文档更少。然而，这个性能优势是以低查全率为代价的：如果一个相关文档仅包含3个查询词项中的两个，它不会被作为结果返回给用户。这对于上述查询Q来说尤其明显。在TREC过50万的文档集中，有7834个文档匹配查询的析取式，但只有1个文档满足合取式。碰巧，这个文档甚至还是不相关的（它的内容是关于一个墨西哥演员和他最新的电影）。

合取方法对于上述查询失败的原因解释如下：对于熟悉Stoicism哲学概念的作者感觉没必要说明它是一个哲学流派或它起源于希腊；对他们来说，这是显而易见的常识。用户也许想通过提供更多的关键词“philosophy”和“greek”来帮助搜索引擎，但事实上这些词项反而使搜索结果变糟。对于长查询来说，这种效果更明显，因为任何一个不恰当（或拼写错误）的词项都会导致搜索结果变差。

我们认为为查询增加相关词项不会使搜索结果变坏。在接下来的内容中，都假设搜索引

擎使用析取方法，检索至少包含一个查询词项的文档，并用排名函数来决定哪些文档更匹配用户的查询。当然，合取模型通常还是会比析取模型快，这是毫无疑问的。因此本节中涉及的很多优化都有一个共同的目的：尽量缩小给定查询的 AND 解析式和 OR 解析式之间的性能差距。

Okapi BM25

为了便于讨论，假定搜索引擎基于文档与查询的相关性使用 Okapi BM25 评分函数（公式（8-48））来对文档进行排名。为方便起见，我们还是将这个排名公式重复一下：

$$\text{Score}_{\text{BM25}}(q,d)=\sum_{t\in a}\log\left(\frac{N}{N_t}\right)\cdot\text{TF}_{\text{BM25}}(t,d) \tag{5-4}$$

$$\text{TF}_{\text{BM25}}(t,d)=\frac{f_{t,d}\cdot(k_1+1)}{f_{t,d}+k_1\cdot((1-b)+b\cdot(l_d/l_{\text{avg}}))} \tag{5-5}$$

公式中有两个自由参数：k_1（默认值：$k_1=1.2$），用于调节 TF 组件的饱和速度；b（默认值：$b=0.75$），用于控制文档长度归一化的程度。所有其他变量都使用标准语义，如本书开头给出的“符号”表。

如果对 BM25 公式的内在理论感兴趣，可在第 8 章中找到这个公式的推导以及对它的深入讨论。本节的内容不需要对 BM25 有很深入的理解。然而，我们重点关注参数 k_1 的作用，因为它与排名检索中各种查询优化紧密相关。k_1 限制了单个查询词项的得分贡献：

$$\lim_{f_{t,d}\to\infty}\text{TF}_{\text{BM25}}(t,d)=k_1+1 \tag{5-6}$$

因为 k_1 的默认值为 1.2，因此任何给定查询词项的 TF 分数值不会超过 2.2。由于这个非常严格的上界，使得包含两个不同查询词项的文档会比仅包含一个查询词项的文档排名靠前，就算后面这个文档包含同一个词项很多次也是如此。

给定查询 $q=\langle t_1, t_2\rangle$，$N_{t_1}\approx N_{t_2}$，我们为两个与平均长度等长的文档 d_1 和 d_2 评分（即 $l_{d_1}=l_{d_2}=l_{\text{avg}}$）。进一步假设 d_1 包含了 t_1 和 t_2 各一次，而 d_2 包含 10 次 t_1 但不包含 t_2。我们有（$k_1=1.2$）：

$$\text{Score}_{\text{BM25}}(q,d_1)\approx\log\left(\frac{N}{N_{t_1}}\right)\cdot\left(2\cdot\frac{1\cdot(k_1+1)}{1+k_1}\right)\approx 2\cdot\log\left(\frac{N}{N_{t_1}}\right) \tag{5-7}$$

$$\text{Score}_{\text{BM25}}(q,d_2)\approx\log\left(\frac{N}{N_{t_1}}\right)\cdot\frac{10\cdot(k_1+1)}{10+k_1}\approx 1.95\cdot\log\left(\frac{N}{N_{t_1}}\right) \tag{5-8}$$

本节的后面部分，k_1 所提供的上界可用于裁剪掉一些位置信息，我们事先就可以知道这些位置信息所对应的文档不可能出现在搜索结果中靠前的位置。

5.1.1 document-at-a-time 查询处理

排名检索中查询处理最常见的形式被称为 document-at-a-time 方法。这种方法枚举所有匹配的文档，逐个为它们计算得分。最后所有文档都根据它们的得分进行排序，前 k 个结果（k 由用户或应用指定）将被返回给用户。

```
rankBM25_DocumentAtATime (⟨t1, ..., tn⟩, k) ≡
1   m ← 0   // m 是匹配文档总数
2   d ← min1≤i≤n{nextDoc(ti, −∞)}
3   while d < ∞ do
4       results[m].docid ← d
5       results[m].score ← Σ_{i=1}^{n} log(N/Nti) · TFBM25(ti, d)
6       m ← m + 1
7       d ← min1≤i≤n{nextDoc(ti, d)}
8   按score降序排列results[0..(m − 1)]
9   return results[0..(k − 1)]
```

图 5-1 BM25 的 document-at-a-time 查询处理

图 5-1 给出了 BM25 的 docu-

ment-at-a-time 算法。这个算法——除得分计算外——与图 2-9 中的 **rankCosine** 算法是一样的。算法的整体时间复杂度为：

$$\Theta(m \cdot n + m \cdot \log(m)) \tag{5-9}$$

其中，n 是查询词项的个数，m 是匹配的文档个数（至少包含一个查询词项）。$m \cdot n$ 对应算法第 3 行开始的循环。$m \cdot \log(m)$对应第 8 行对搜索结果进行排序。

有时候很难直接运用公式（5-9），因为在运行算法和枚举所有的搜索结果之前 m 的值是不知道的。m 取决于查询词项在文档中共同出现的频率，其取值在 $N_q/n \sim N_q$ 之间，其中，$N_q = N_{t_1} + N_{t_2} + \cdots + N_{t_n}$ 是所有查询词项出现位置的和。最坏情况下，每个匹配的文档仅包含一个查询词项。则图 5-1 算法的时间复杂度为：

$$\Theta(N_q \cdot n + N_q \cdot \log(N_q)) \tag{5-10}$$

实际上，大部分匹配文档确实只包含一个查询词项。例如，在之前的查询例子〈“greek”，“philosophy”，“stoicism”〉中，我们有 $m = 7835$，$N_q = 7921$。因此，公式（5-10）可作为公式（5-9）很好的一个近似。

图 5-1 中基本的 document-at-a-time 算法的低效率主要有两个原因：

- 不管词项是否出现在当前文档中，第 5 行和第 7 行的计算需要为所有 n 个查询词项都迭代一遍。如果 n 值很大，这就成为一个问题。考虑这样的极端情况，有 10 个查询词项，每个匹配文档只包含了一个查询词项。如果算法逐个处理每个词项，那么就得执行 10 次。
- 第 8 行最后的排序步骤在**结果集**（results）数组中对所有文档进行排序。当然，如果仅对前 k 个结果感兴趣，对整个数组进行排序是一种资源的浪费。$\Theta(m \cdot \log(m))$的时间复杂度看起来并没有那么糟糕。但记住我们处理的可能是好几百万的匹配文档，所以公式（5-9）中的 $m \cdot \log(m)$实际上会超过对应的得分计算的 $m \cdot n$。

我们用同样地方法来解决这两个问题：使用一个叫做**堆**（heap）的数据结构。如果你已经知道堆是什么，那么请直接跳到“使用堆进行高效的查询处理”这一节。

1. 二叉堆

堆（更确切地说，**最小二叉堆**（binary min-heap））是一个满足如下定义的二叉树：

1）空的二叉树是一个堆。

2）满足下面条件的非空二叉树是 $\mathcal{T}$ 一个堆：

(a) 除了最底层，$\mathcal{T}$ 中的每一层都是完全填充的（满的）。

(b) $\mathcal{T}$中最底层从左往右顺序进行填充。

(c) 对每一个节点 v，存储在 v 中的值要比存储在它的任何孩子节点中的值小。

条件（a）和（b）组合在一起的性质，有时称为**形态特性**（shape property）。由于具有这一形态特性，堆不仅可表示为树还可表示为数组。树的根节点保存在数组位置 0，节点 i 的孩子节点分别保存在位置 $2i+1$ 和 $2i+2$ 中。图 5-2a 给出了集合{1，2，3，4，5，6}对应的堆用树来表示的结果。图 5-2b 给出了等价的数组形式。

在实际实现中，一般会优选数组形式而不是树形式，因为数组形式空间效率更高（不需要孩子指针），也会更快（更好的数据局部性）。在数组形式中，条件 2（c）可以转化为：

$$\forall i \in \{0, 1, 2, \ldots, len - 1\}: \\ (2i + 1 \geqslant len \vee A[i] \leqslant A[2i + 1]) \wedge (2i + 2 \geqslant len \vee A[i] \leqslant A[2i + 2]) \tag{5-11}$$

其中，len 是数组 A 的长度。特别地，每一个有序数组都是一个堆（但不是每一个堆都是一

个有序数组)。

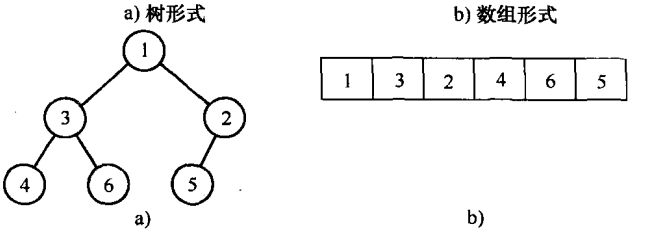


图 5-2 集合 {1，2，3，4，5，6} 对应的堆的树形式和等价的数组形式

堆支持很多有趣的操作。在这里我们感兴趣的操作叫做 REHEAP。考虑图 5-2 中的堆，假如现在我们想要用一个新值 9 来代替根节点的值 1，并且仍保持堆的特性。REHEAP 算法将 1 替换为 9，然后将 9 和 2 交换，最后将 9 和 5 交互，得到一个新数组〈2，3，4，5，6，9〉，仍然保持堆的特性。更一般的，REHEAP 用一棵二叉树来保持除 2（c）外的所有其他堆的性质，并将根的值不断下移直到满足堆的所有性质。

这个算法的时间复杂度是多少？考虑包含 n 个节点的堆$\mathcal{T}$。因为每个堆都是一棵平衡二叉树，$\mathcal{T}$ 的每个叶子高度都为$\lceil\log_2(n)\rceil$或$\lceil\log_2(n)-1\rceil$。因此 REHEAP 将在 $O(\log(n))$步内结束。

2. 使用堆进行高效的查询处理

使用 REHEAP 算法可克服图 5-1 算法的限制。在这个算法的改进版中，我们使用到两个堆：一个用来管理查询词项，对于每一个词项 t，跟踪记录下一个包含 t 的文档；另一个用于维护目前找到的前 k 个搜索结果。

改进后的算法如图 5-3 所示。**词项**（term）堆包含查询词项集合，并根据对应词项出现的**下一文档**（nextDoc）进行了排序。这使得可以在 n 个位置信息列表上进行一个高效的多路合并操作。**结果**（result）堆包含目前找到的前 k 个搜索结果，并按它们的得分排序。特别需要注意到结果的根节点（也就是算法中采用的数组形式的 *results*[0]）并不包含目前找到的最好文档，而是目前第 k 个最好的文档。这样，当我们找到一个新文档的得分比旧文档高时，就可以通过替换前 k 个文档中得分最低的那个文档（并保持满足堆的性质）来实现维护和持续更新前 k 个结果。

```
   rankBM25_DocumentAtATime_WithHeaps (⟨t_1, ..., t_n⟩, k) ≡
1    for i ← 1 to k do  // 为前k个搜索结果创建最小堆
2       results[i].score ← 0
3    for i ← 1 to n do  // 为n个查询词项创建最小堆
4       terms[i].term ← t_i
5       terms[i].nextDoc ← nextDoc(t_i, −∞)
6    按next Doc 升序排列terms // 为terms建立堆属性
7    while terms[0].nextDoc < ∞ do
8       d ← terms[0].nextDoc
9       score ← 0
10      while terms[0].nextDoc = d do
11         t ← terms[0].term
12         score ← score + log(N/N_t) · TF_BM25(t, d)
13         terms[0].nextDoc ← nextDoc(t, d)
14         reheap(terms)  // 为terms重建堆属性
15      if score > results[0].score then
16         results[0].docid ← d
17         results[0].score ← score
18         reheap(results)  // 为results重建堆属性
19   从results中，删除所有score=0的词项
20   按score的降序排列results
21   return results
```

图 5-3 BM25 的 document-at-a-time 查询处理算法，使用二叉堆来管理词项集和前 k 个文档集

最坏情况下，document-at-a-time 算法的改进版本的时间复杂度为：

$$\Theta(N_q \cdot \log(n) + N_q \cdot \log(k)) \tag{5-12}$$

其中，$N_q = N_{t_1} + N_{t_2} + \cdots + N_{t_n}$ 是所有词项位置信息的个数。第一项（$N_q \cdot \log(n)$）对应在**词项**（term）堆上进行的 REHEAP 操作，每次插入新的位置信息时重新调整堆结构使之保持堆的性质。第二项（$N_q \cdot \log(k)$）对应在**结果**（result）堆上进行的 REHEAP 操作，每次在前 k 个结果集中插入新文档时重新调整堆结构使之保持堆的性质。

对比公式（5-10），改进版本的算法有显著的改善，主要是因为将最后的排序过程限制在前 k 个文档而不是所有匹配文档上，因此获得了速度上的提高。并且，尽管维护前 k 个结果集在最坏情况下的复杂度是 $\Theta(N_q \cdot \log(k))$，但在实际中最坏情况下的复杂度没什么用；该算法的平均情况下的复杂度甚至比公式（5-12）中的还要好（见练习 5.1）。

3. 最大得分

尽管图 5-3 的算法已经可以获得比较合理的查询处理性能，但仍然存在改进的空间。回想之前讨论 BM25 的 TF 得分贡献不会超过 $k_1 + 1 = 2.2$。一个词项 t 的总得分不会超过 $2.2\log(N/N_t)$。这个上限称为该词项的**最大得分**（MaxScore）。重新考虑查询

$$Q = \langle \text{“greek”}, \text{“philosophy”}, \text{“stoicism”} \rangle \tag{5-13}$$

查询词项的文档频率、IDF 权重和对应的最大得分为：

词项	N_t	$\log_2(N/N_t)$	最大得分
“greek”	4504	6.874	15.123
“philosophy”	3359	7.297	16.053
“stoicism”	58	13.153	28.936

假设输入查询的用户对前 $k = 10$ 个搜索结果感兴趣。对几百个文档进行评分后，也许会碰到这种情况，目前找到的前 10 个最好的结果已经超过了词项“greek”的最大得分贡献的限制，即

$$results[0].score > \text{MaxScore}(\text{“greek”}) = 15.123 \tag{5-14}$$

当这种情况发生时，我们知道仅包含“greek”但不包含“philosophy”和“stoicism”的文档是不可能进入最好的前 10 个搜索结果中的。因此就不需要为任何仅包含“greek”的文档进行评分了。当找到包含任一其他两个词项的文档时，可以将这个词项从**词项**（term）堆中移除并查看它的位置信息。

对越来越多的文档评分时，可能有时会碰到以下情况：

$$results[0].score > \text{MaxScore}(\text{“greek”}) + \text{MaxScore}(\text{“philosophy”}) = 31.176 \tag{5-15}$$

这时，我们知道一个仅包含词项“stoicism”的文档是有可能进入到前 10 个结果中的，我们将“philosophy”从**词项**堆中移除，正如我们处理“greek”一样。

上述策略称为 MAXSCORE，由 Turtle 和 Flood（1995）提出。MAXSCORE 能保证产生的前 k 个结果与图 5-3 的算法产生的结果一样，但速度会快很多，因为它一旦事先发现文档不可能成为前 k 个结果就忽略它们。

注意到，尽管 MAXSCORE 从堆里移除了一些词项，但它仍用它们来进行评分。这通过维护两个数据结构来实现，一个用来保存仍在堆中的词项，另一个用来保存从堆中移除的词项。当我们找到一个文档 d 包含仍在堆中的词项时，我们迭代地从堆中移除这些词项，并为每一个词项 t 都调用 nextDoc(t, $d-1$)来确定 t 是否出现在 d 中。如果是，我们就计

算 t 的得分贡献并加到 d 的得分中去。

表 5-1 列出了 TREC TB 2006 任务中的 10 000 个查询上的每查询平均时间和每查询 CPU 时间，可与 GOV2 文档集上的磁盘频率索引效果相比（包含（d，$f_{t,d}$）的位置信息），每查询的平均时间和 CPU 时间。与图 5-3 中的算法相比，$k=10$ 时（这是许多搜索引擎的默认设置），MAXSCORE 将每查询的总时间减少了 53%，CPU 时间减少了 69%。因为平均每次查询中搜索引擎需要评分的文档总数从 440 万减少到 28 万。注意到，就算使用了 MAXSCORE，搜索引擎也比使用合取检索模型（布尔 AND）多评分了一个数量级的文档。然而，从总体性能来看，使用 MAXSCORE 的布尔 OR 与布尔 AND 也差不了多少。例如，每次查询的平均 CPU 时间只提高了 50%：从 93 ms 提高到了 62 ms。

表 5-1 在有和没有最大得分的情况下，每查询的总时间和 CPU 时间。数据集：TREC TB 2006 上的 10 000 个查询。可与 GOV2 上的频率索引效果相比

	没有最大得分			有最大得分		
	实际时间	CPU	得分文档	实际时间	CPU	得分文档
OR，$k=10$	400 ms	304 ms	$4.4 \cdot 10^6$	188 ms	93 ms	$2.8 \cdot 10^5$
OR，$k=100$	402 ms	306 ms	$4.4 \cdot 10^6$	206 ms	110 ms	$3.9 \cdot 10^5$
OR，$k=1000$	426 ms	329 ms	$4.4 \cdot 10^6$	249 ms	152 ms	$6.2 \cdot 10^5$
AND，$k=10$	160 ms	62 ms	$2.8 \cdot 10^6$	*n/a*	*n/a*	*n/a*

5.1.2 term-at-a-time 查询处理

作为 document-at-a-time 方法的一个替代，也有一些搜索引擎用 term-at-a-time 方法来处理查询。搜索引擎不使用堆来合并查询词项的位置信息列表，而是顺序地检查每个查询词项的所有（或部分）位置信息。它维护一组文档得分**累加器**（accumulator）。对于每个检查到的位置信息，它确定对应的累加器并根据该位置信息对文档的得分贡献来更新累加器的值。所有的查询词项处理完之后，累加器就包含了所有匹配文档的最后得分，这时就使用一个堆来获得前 k 个搜索结果。

term-at-a-time 方法背后的一个动机是：索引存储在磁盘上，查询词项的位置信息列表也太大以至于不能全部放入内存。在这样的情况下，document-at-a-time 的实现需要在查询词项的位置信息列表之间跳转，每次跳转后将一小部分的位置信息读入内存，因此造成了由不连续的磁盘访问（磁盘寻道）而形成的代价。对于短查询，包含 2～3 个词项，这可能不是一个问题，可以通过为每个位置信息列表分配一个大小合适的预读缓冲区来降低磁盘寻道的次数。然而，当查询包含十多个词项时（例如使用了伪相关性反馈后——参见 8.6 节），磁盘寻道就是一个问题了。term-at-a-time 实现不需要任何不连续的磁盘访问模式。搜索引擎用线性方式处理每个词项的位置信息列表，做完词项 t_i 后就移到词项 t_{i+1}。

因为 term-at-a-time 方法分开处理每一个位置信息列表，通常只适用于以下形式的评分函数：

$$\text{score}(q,d) = \text{quality}(d) + \sum_{t \in q} \text{score}(t,d) \tag{5-16}$$

在这个公式中，quality(d)是可选的，代表一个独立于查询的评分组件，例如，PageRank（公式（15-8））。很多传统的评分函数，包括 VSM（公式（2-12））、BM25（公式（8-48））以及 LMD（公式（9-32）），都属于（或等价于）这种形式。它们有时也被称为**词袋**（bag-of-words）方法。那些考虑文档中查询词项邻近度的评分函数，包括词组查询，不适用公

式（5-16）。理论上，它们仍然可以在 term-at-a-time 查询处理框架中实现。然而，除了文档得分，搜索引擎维护的单个文档累加器还需要保存一些额外信息（例如，词项位置），这会显著增加它们的大小。

图 5-4 给出了 BM25 一个可能的 term-at-a-time 查询处理算法。图中，累加器保存在数组 acc 中。对于每一个词项 t_i，按照 t_i 的位置信息列表来顺序遍历这个数组，创建一个新的数组 acc'保存更新之后的累加器。该算法在最坏的情况下的时间复杂度为：

$$\Theta\left(\sum_{i=1}^{n}(N_q/n \cdot i) + N_q \cdot \log(k)\right) = \Theta(N_q \cdot n + N_q \cdot \log(k)) \tag{5-17}$$

其中，$N_q = N_{t_1} + N_{t_2} + \cdots + N_{t_n}$，$N_q$ 是所有查询词项位置信息的总数。当 $N_{t_i} = N_q/n$ $(1 \leqslant i \leqslant n)$时是最坏情况。

```
rankBM25_TermAtATime (⟨t1, ..., tn⟩, k) ≡
1   按Nti的升序排列⟨t1, ..., tn⟩
2   acc ← {}, acc′ ← {}   // 初始化两个空的累加器集合
3   acc[0].docid ← ∞   // 列表尾标记
4   for i ← 1 to n do
5     inPos ← 0   // acc的当前位置
6     outPos ← 0   // acc′的当前位置
7       for each ti位置信息列表中的文档d do
8       while acc[inPos] < d do   // 从acc复制累加器到acc′
9         acc′[outPos++] ← acc[inPos++]
10      acc′[outPos].docid ← d
11      acc′[outPos].score ← log(N/Nti) · TFBM25(ti, d)
12      if acc[inPos].docid = d then   // 词项与累加器吻合
13        acc′[outPos].score ← acc′[outPos].score + acc[inPos++].score
14      d ← nextDoc(ti, acc′[outPos])
15      outPos ← outPos + 1
16    while acc[inPos] < ∞ do   // 从acc复制剩下的累加器到acc′
17      acc′[outPos++] ← acc[inPos++]
18    acc′[outPos].docid ← ∞   // 列表尾标记
19    swap acc and acc′
20  return acc 中的前k个项 // 用堆来选出前k个结果
```

图 5-4 BM25 的 term-at-a-time 查询处理。文档得分存储在累加器中。根据当前词项的位置信息列表来顺序遍历累加器数组

通过比较公式（5-17）和公式（5-12），我们可以看到 term-at-a-time 算法，至少是图 5-4 的这种形式，实际上会比 document-at-a-time 算法稍慢一些（$N_q \cdot n$ 代替 $N_q \cdot \log(n)$），因为它需要为每个查询词项 t_i 遍历整个累加器。理论上，用一个哈希表替换掉数组 acc，就能使每个累加器的更新在 $O(1)$时间内完成，这样就能解决上述瓶颈。这样整体的复杂度就为$\Theta(N_q + N_q \cdot \log(k)) = \Theta(N_q \cdot \log(k))$。然而实际上，出于两个原因数组实现比哈希实现要好。第一，数组的缓存效率是很高的；而哈希表由于是非顺序访问模式，有可能会导致大量的 CPU 缓存丢失。第二，也是更重要的一点，现实中实现 term-at-a-time 算法通常不需要保存所有的累加器，而是采用一个称为**累加器裁剪**（accumulator pruning）的策略。

累加器裁剪

如前所述，term-at-a-time 查询处理背后的一个动机是查询词项的位置信息列表太大而不能全部放入内存。当然，这意味着累加器集合 acc 也会太大而放不进内存。因此，图 5-4 的算法就不适用了。为了克服这种前后矛盾，我们改写这个算法，使用一个固定的内存容量来保存累加器集合。也就是，为可能创建的累加器个数设置一个上限 $a_{\max}$。

两种典型的累加器裁剪策略是 QUIT 和 CONTINUE，由 Moffat 和 Zobel（1996）提

出。在 QUIT 策略中，搜索引擎一旦处理完当前的查询词项立即检验 $|acc| \geqslant a_{max}$ 是否成立。如果成立，查询处理过程立刻停止，当前的累加器集就表示最后的搜索结果。如果使用 CONTINUE 策略，搜索引擎继续处理位置信息列表并更新已有的累加器，但不再创建新的累加器。

遗憾的是，因为一个词项的位置信息列表有可能包含超过 a_{max} 条记录，无论是 QUIT 还是 CONTITUE 实际上都无法执行一个硬限制，对于较小的 a_{max}，实际上执行的限制要高得多。这里讨论的累加器裁剪基于 Lester 等人最近的工作（2005）。Lester 的算法能保证创建的累加器数目在 a_{max} 的一个常数倍范围内。在这里讨论的变形算法中，保证累加器限制不被超过。

与 document-at-a-time 中的 MAXSCORE 策略类似，累加器裁剪的基本思想是：我们不关心匹配文档全集，而只关心前 k 个，且 k 较小。然而，与 MAXSCORE 不同的是，累加器裁剪策略不保证返回与穷举算法一样的结果集。只产生一个近似；这个近似的质量取决于使用的裁剪策略及 a_{max} 的值。

在图 5-4 的算法中，查询词项按出现频率多少进行处理，从最不频繁的到最频繁的。这对于不裁剪的 term-at-a-time 查询处理来说通常是有利的，因为需要从 acc 中复制到 acc' 的累加器更少，如果使用了累加器裁剪这就更关键了。主要基于以下观察：如果只使用有限个累加器，应尽量将其用在最有可能是前 k 个的文档中。同等条件下，如果查询词项 t 比 t' 拥有更大的权重，那么包含 t 的文档很有可能就比包含 t' 的文档更像是会在前 k 个结果中。在 BM25 中（或其他基于 IDF 评分公式的文档集中），这意味着越不频繁的词项（位置信息列表越短）应该越先处理。

为了定义一个具体的裁剪策略，我们应该制定一条规则来确定是否应为一个给定的位置信息分配一个累加器。假设处理完前 $i-1$ 个查询词项后，搜索引擎共有 $a_{current}$ 个累加器，然后继续处理 t_i。那么，有三种可能的情况：

1）$a_{current}+N_{t_i} \leqslant a_{max}$。这种情况中，仍有足够未用的累加器可分配给 t_i 的位置信息，不需要进行裁剪。

2）$a_{current}=a_{max}$。这种情况中，已经达到累加器的上限。不会为 t_i 的位置信息创建任何新的累加器。

3）$a_{current}<a_{max}<a_{current}+N_{t_i}$。这种情况中，没有足够的累加器配额为 t_i 的位置信息创建新的累加器了，需要裁剪。

情况 1 和 2 容易理解。对于情况 3，一个可能的（也是最简单的）裁剪策略是：只为前 $a_{max}-a_{current}$ 个位置信息创建新的累加器，这些位置对应的文档目前还没有累加器，其余的就忽略掉。这种方法的问题在于会偏向文档集中靠前的文档。这是不恰当的，因为没有任何证据表明这些文档就与给定查询更相关（除非文档已按某种独立于查询的质量指标排好序了，如 PageRank）。

理想情况下，我们定义一个阈值 ϑ 使得 t_i 的位置信息列表中恰好 $a_{max}-a_{current}$ 个位置信息上的得分贡献超过 ϑ。可以为得分超过 ϑ 的位置信息创建累加器，但得分低的位置信息就不创建累加器了。从字面上理解，这个方法是一个两遍查询处理策略：第一遍，为所有已评分的 t_i 的位置信息计算得分贡献，然后排序并确定阈值 ϑ。第二遍，重新计算 t_i 的位置信息的得分，并与阈值比较。如果 t_i 的位置信息很长，这个方法的计算代价还是很高的。我们使用下面两步来解决这个问题：

1）采用一个稍微不同的阈值 ϑ_{TF} 来代替原始阈值 ϑ，可以省掉为所有 t_i 的位置信息评分的必要。新的阈值将直接与每个位置的 TF 值进行比较，而不是与得分贡献比较，从而减少整体的计算代价。

2）用 ϑ_{TF} 的一个近似值来代替实际值可以避免对 t_i 的位置信息的第二遍扫描。这个近似值在线计算并根据已经扫描过的位置信息的 TF 分布定期更新。

改后的算法如图 5-5 所示。除了查询词项和用来指定待返回的文档个数 k 之外，这个算法还有两个参数：累加器上限 $a_{\max}$ 和 ϑ_{TF} 近似值的更新间隔 u。每当遇到具有给定 TF 值的位置信息并且还未有对应累加器的时候，算法用数组 *tfStats* 来记录这些位置信息的总个数。用这些统计信息对剩余的位置信息进行推理，可以预测对于给定的 TF 值我们还需要多少个位置信息，从而可计算 ϑ_{TF} 的一个估计。这个计算替代了算法的第 31～32 行。

```
    rankBM25_TermAtATimeWithPruning (⟨t_1, ..., t_n⟩, k, a_max, u) ≡
1     按 N_{t_i} 升序排引⟨t_1, ..., t_n⟩
2     acc ← {}, acc' ← {}   // 初始化两个空累加器集合
3     acc[0].docid ← ∞   // 列表尾标记
4     for i = 1 to n do
5        quotaLeft ← a_max − length(acc)   // 剩下的累加器位置
6        if N_{t_i} ⩽ quotaLeft then   // 情况1：不需要裁剪
7           do everything as in Figure 5.4
8        else if quotaLeft = 0 then   // 情况2：累加器用尽了
9           for j = 1 to length(acc) do
10             acc[j].score ← acc[j].score + log(N/N_{t_i}) · TF_BM25(t_i, acc[j].docid)
11       else   // 情况3：有剩余的累加器，需要裁剪
12          tfStats[j] ← 0  ∀j   // 初始化裁剪所需的TF统计值
13          ϑ_TF ← 1   // 为新的累加器初始化TF阈值
14          postingsSeen ← 0   // 从t_i的位置信息列表中得到的位置信息数
15          inPos ← 0   // acc 的当前位置
16          outPos ← 0   // acc' 的当前位置
17          for each t_i 位置信息列表中的文档d do
18             while acc[inPos] < d do   // 从 acc 复制累加器到 acc'
19                acc'[outPos++] ← acc[inPos++]
20             if acc[inPos].docid = d then   // 词项与累加器吻合
21                acc'[outPos].docid ← d
22                acc'[outPos++].score ← acc[inPos++].score + log(N/N_{t_i}) · TF_BM25(t_i, d)
23             else if quotaLeft > 0 then
24                if f_{t_i,d} ⩾ ϑ_TF then   // f_{t_i,d} 超过了阈值；创建一个新的累加器
25                   acc'[outPos].docid ← d
26                   acc'[outPos++].score ← log(N/N_{t_i}) · TF_BM25(t_i, d)
27                   quotaLeft ← quotaLeft − 1
28                tfStats[f_{t_i,d}] ← tfStats[f_{t_i,d}] + 1
29             postingsSeen ← postingsSeen + 1
30             if (postingsSeen mod u = 0) then   // 基于 tfStats 重计算 ϑ_TF
31                q ← (N_{t_i} − postingsSeen)/postingsSeen
32                ϑ_TF ← argmin_x{x ∈ ℕ | ∑_{j=1}^{x} (tfStats[j] · q) ⩾ quotaLeft}
33          如图5-4，从acc 复制剩下的累加器到 acc'
34          swap acc and acc'
35    return the top k items of acc   // 用堆来选出前 k 个结果
```

图 5-5 使用累加器裁剪策略的 BM25 上的 term-at-a-time 查询处理。输入参数：查询词项 t_1,…, t_n；返回的文档个数 k；累加器上限 $a_{\max}$；阈值更新间隔 u

更新间隔参数 u 用来限制定期重计算 ϑ_{TF} 的计算开销。通过我们的实验，$u=128$ 在近似精确度与计算开销之间达到了一个很好的平衡。Lester 等人（2005）建议用指数级增长的间隔来取代固定的更新间隔，主要是这样一个事实，随着算法的进行，对优化阈值 ϑ_{TF} 的近似会越来越准确，因此需要更正的次数会变少。

考虑到第 32 行阈值计算的效率，值得指出的是没有必要为 *tfStats* 数组中每个可能的 TF 值都保存一个记录。大多数 $f_{ti,d}$ 值都很小（小于 4 或 5），为了避免维护一个有很多 0 值

的 $tfStats$ 数组，将较大的值合在一起比较有用。例如，将算法第 28 行改为：

$$tfStats[\min\{15, f_{t_i,d}\}] \leftarrow tfStats[\min\{15, f_{t_i,d}\}] + 1$$

这样对 ϑ_{TF} 值没有什么影响，但在第 32 行重计算 ϑ_{TF} 时就非常高效了。

图 5-6 展示了使用图 5-5 累加器裁剪算法的检索有效性以及查询处理性能。有效性评价在 TREC TB 2004 和 2005 的 99 个特定主题上产生的短查询上进行，检索前 $k=10\,000$ 个文档。查询处理性能评价在 TREC TB 2006 的 10 000 个有效主题上进行，为每个主题检索前 $k=10$ 个文档。

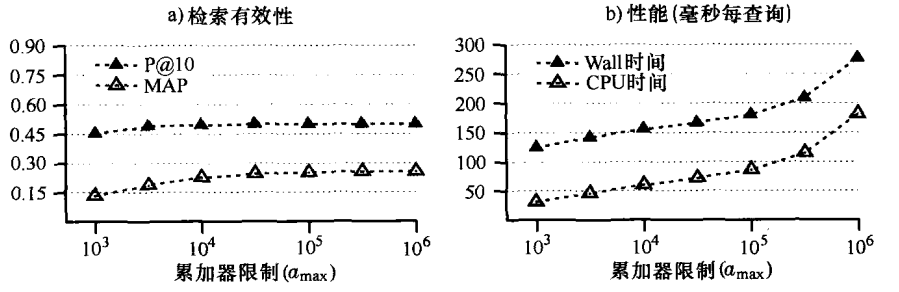


图 5-6 使用累加器裁剪的 term-at-a-time 查询处理的性能以及检索有效性。数据集：来自 TREC TB 2006 的 10 000 个查询，可与 GOV2 上的频率索引效果相比

即使对于较小的 a_{max} 值，检索有效性也相当稳定。平均查准率均值（MAP）从 $a_{max} \approx 10^5$ 才开始下跌。前 10 个文档上的查准率（P@10）只有当过度裁剪时（$a_{max} < 10^4$）才开始下降。算法的性能，用每次查询执行时间（秒）来衡量，与（使用了 MAXSCORE 的）document-at-a-time 算法性能相当。对于 $a_{max}=10^5$，两个算法的执行时间差不多，有效性水平也在同一水平。对于更大的 a_{max} 值，由于为每个查询词项遍历整个累加器数组导致的开销，term-at-a-time 方法就比 document-at-a-time 要差了。

有趣的是，对于过度裁剪（$a_{max} \leqslant 10^4$），term-at-a-time 算法实际上也比布尔 AND 方法要快（比较图 5-6 与表 5-1）。这也不完全出乎意料。根据表 5-1，如果搜索引擎采用合取查询式（Boolean AND），那么平均每次查询它得为 28 000 个匹配文档评分。因此，在某些条件下，合取方法事实上比使用了累加器裁剪的析取方法低效。当然，这种比较不完全公平，因为合取查询评价也可使用累加器裁剪（参见练习 5.2）。

5.1.3 预计算得分贡献

现在已经很显然为最终文档计算得分，例如，根据 BM25 式子（公式（5-5）），是搜索引擎查询处理过程中的主要瓶颈。无论是 MAXSCORE 还是累加器裁剪都通过将得分计算限制在最有可能是前 k 个搜索结果的文档上来获得性能上的提高。

对于采用词袋方式的评分算法（公式（5-16）），没必要在查询阶段计算查询词项的得分贡献。相反，我们可以在索引构建阶段就预计算每个位置信息的得分贡献并与文档编号一起存储在索引中，用（$docid$，$score$）形式代替（$docid$，tf）形式的位置信息。预计算得分贡献可显著减少查询处理阶段搜索引擎的 CPU 开销。当然，如果 TF 值被预计算的得分贡献代替，那么索引建立之后就不太可能对评分函数进行任何的修改。在很多应用中这不会是个问题。但很难用新的评分函数去做实验了。

一个潜在更严重的问题是预计算得分所需的空间。为了减少空间需求（和使用磁盘索引

时的磁盘 I/O 开销），倒排索引通常以压缩形式存储。至于是如何对索引进行压缩的在这里不重要（将在第 6 章中描述），但基本的想法是将小整数值用更少的位（bit）来存储。例如，倒排索引中大部分的 TF 值都是很小的（小于 4）。因此，它们很大程度可以进行压缩，索引中每个位置信息仅需 2～3 位来存储。另一方面，预计算词项得分基本上无法进行压缩。如果词项得分用 32 位浮点数表示，在索引中它们将需要 24～32 位。[⊖]

如果希望通过预计算词项贡献来加速评分过程，那么关键是将可能的评分范围离散化到一组预定义的桶中。定义 B 是为每个得分贡献预留的位数（未压缩）。一个可能的离散化过程为：

$$score' = \left\lfloor \frac{score}{score_{\max}} \cdot 2^B \right\rfloor \tag{5-18}$$

其中，$score_{\max}$是评分函数允许的最大得分贡献。在标准的 BM25 下，有 $score_{\max} = k_1 + 1 = 2.2$（不考虑评分公式中 IDF 部分）。

因为离散化之后的 $score'$值在 2^B 种可能取值上不是均匀分布的，因此可以对它们进行压缩，例如使用霍夫曼编码（见 6.2.2 节），使得每个得分贡献可以使用少于 B 位的存储空间。

图 5-7 描述了采用预计算得分贡献的 document-at-a-time 算法的实验结果（每个查询的检索有效性和处理时间）。可以看出，每个预计算得分，至少（未压缩）采用 4 位，所获得的检索有效性与在线计算所有词项得分的原算法的检索有效性是差不多的。同时，每次查询的平均时间还从 188 ms 减少到 148 ms（−21%）。

增加每个得分贡献的位数会使搜索引擎变慢，采用了预计算得分的改进算法比原算法在每次查询上需要更多的时间（例如，对于 $B=8$，从 188 ms 提高到了 207 ms）。然而注意到，变慢主要是因为磁盘 I/O 开销的增加，每次查询 CPU 平均时间还是保持近似常数。因此，如果索引保存在内存而不是磁盘中，就不会有这样的现象。

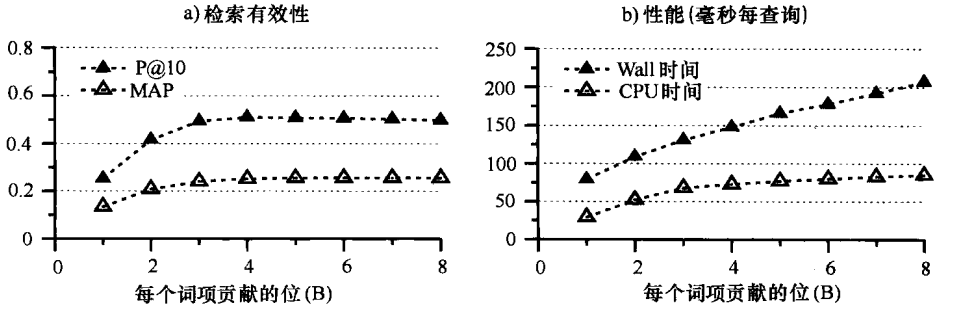


图 5-7 采用了预计算词项贡献和 MAXSCORE 策略的 document-at-a-time 查询评价：检索有效性和查询处理性能。数据集：来自 TREC TB 2006 的 10 000个查询，可与 GOV2 上的频率索引效果相比

5.1.4 影响力排序

在第 4 章描述的倒排索引数据结构中，存储在索引中的位置信息按它们出现在文档中的

⊖ 浮点数算术运算的 IEEE 标准将 32 位浮点数分成 24 位显著位和 8 位指数位。显著位几乎是不可压缩的。

先后顺序进行排序。这样组织位置信息列表的好处是索引容易建立，并且查询操作（例如合并位置信息列表）也能相对高效的执行。然而，原来的文档序不一定是对给定位置信息列表中的位置信息最好的排序方法。例如，对使用了累加器裁剪的 term-at-a-time 算法(图 5-5)，还得使用一些小技巧来高效获得 t_i 的位置信息列表中排名较前的位置信息的一个近似。如果位置信息已经按它们的得分贡献排好序了，那就简单得多了；只需要挑出列表前面的位置信息，忽略剩下的就行。

每个列表中的位置信息按它们对应的得分贡献排好序的索引称为是**基于影响力排序**（impact-ordered）的索引。基于影响力排序的索引通常包含预计算得分而不包含 TF 值，而位置信息的序就已经隐含了某个评分函数。

如果直接这样实现基于影响力排序的索引，那么会严重影响基本索引访问函数的复杂度。例如，第 2 章的 next 方法就不再是对数复杂度了，而是线性复杂度！为了避免这种情况，基于影响力排序的索引中的位置信息不是按它们的实际得分贡献排序的，而是——与之前的做法一样——根据一个量化的影响力值排序。与前面一样，我们将影响力值分组到 2^B 个离散的桶中，这里 B 通常取值为 3～5。给定词项的位置信息列表中的位置信息按它们离散化后的影响力值进行排序。但在每个桶中，还是和第 4 章标准的索引组织一样，位置信息按它们的文档编号来排序。

采用这种混合方法，与严格的文档序索引相比，随机访问操作的计算开销还是可以接受的。并且可以快速访问词项靠前的位置信息这一优势特别明显，尤其是在 term-at-a-time 查询处理框架下。

5.1.5 静态索引裁剪

5.1.2 节的累加器裁剪技术，无论是不是基于影响力排序的，都通过忽略掉大部分的查询词项位置信息而获得性能上的提高。例如，一个累加器上限 $a_{max}=1000$，搜索引擎评分的文档数就不会超过 1000 个。遗憾的是，就算大部分的位置信息没有为评分过程产生贡献，也不得不将它们从索引中读出来并进行解压缩，以使得搜索引擎可以访问到那些确实产生贡献的“有趣的”位置信息。

如果希望尽可能地提高搜索引擎的查询处理性能，我们可以预测——在索引阶段——哪些位置信息在查询处理阶段是有可能被用到的，哪些不会。基于这个预测，可将在搜索过程中不会起重要作用的位置信息从索引中移除。这个策略被称为**静态索引裁剪**（static index pruning）。由于索引中的位置信息列表变得更短了，因此性能得到显著提高。

静态索引裁剪有几个很明显的局限性。首先，既然是在一个裁剪后的索引中处理查询，对给定查询就不能保证找到的一定是得分靠前的文档。其次，如果允许用户定义结构化查询约束或词组查询，这个策略就失效了。例如，如果用户在莎士比亚文集中搜索词组“to be or not to be”，索引裁剪算法会认为莎士比亚的文集中《Hamlet》第 2 个场景第 3 幕中的“not”是不重要的，导致用户找不到她要寻找的那个场景。然而，除了不适合一些特定类型的查询，索引裁剪已被证明在传统的词袋查询处理算法中还是非常成功的。

索引裁剪算法通常是以下两种形式之一：

- **基于词项**（term-centric）的索引裁剪：它单独处理索引中的每一个词项。根据一些预定义的标准，裁剪算法从每个词项的位置信息列表中挑选出最重要的位置信息，然后剪掉剩下的。
- **基于文档**（document-centric）的裁剪算法：与上面不同，它单独检查每一个文档。

给定一个文档，算法预测文档中哪些查询词项是最重要的和最具代表性的。如果认为一个词项重要，就将它的位置信息加入到索引中；否则，就丢掉这个词项，就像它根本没在文档中出现过一样。

1. 基于词项的裁剪

Carmel 等人（2001）最先研究基于词项的索引裁剪算法。他们在论文中评价了若干种不同的裁剪算法，但这些算法都是类似的。这里只讨论他们的 top-(K, ε)基于词项的裁剪算法。

考虑一个词项 t，对应的位置信息列表 L_t，和一个包含 t 的未知查询 q。同等条件下，L_t 中给定位置信息 P 是查询 q 的前 k 个文档的概率在 P 的得分贡献中是单调的。基于词项的裁剪算法根据选好的评分函数，将 L_t 中的所有位置信息根据它们的（用选好的评分函数计算的）得分贡献进行排序。然后选出前 K_t 个位置信息保存到索引中，其他的都丢弃。

截断参数 K_t 有多种选择方法。一种是为索引中所有的词项都使用相同的 K_t 值。另一种是在索引范围内选择一个得分贡献阈值 ϑ，得分贡献超过 ϑ 的所有位置信息将被保留在索引中，其余的丢弃掉。第三种方法是保证每个词项至少有 K 个位置信息在索引中。如果词项 t 在超过 K 个文档中出现，那么它的位置信息上限 K_t 就取决于它的位置信息列表的得分贡献的分布。这就是 top-(K, ε)裁剪方法：

- 选定两个参数 $K \in \mathbb{N}$，$\varepsilon \in [0, 1]$。
- 如果词项 t 在少于 K 个文档中出现，在索引中保存 t 的所有位置信息。
- 如果词项 t 在超过 K 个文档中出现，计算出 $\vartheta_t = \text{score}(L_t | K|) \cdot \varepsilon$，其中，score $(L_t | K|)$是 t 的位置信息中得分贡献第 K 高的得分。得分贡献低于 ϑ_t 的位置信息就被丢弃掉，剩余的保留在索引中。

一般来说，如何选择 K 和 ε 不是非常的明确。然而，一旦一个参数可以确定，另一个就可以自由的变动，直至达到期望的索引大小、性能或搜索结果质量。

在我们的实验(图 5-8)中，设置 $K=1000$，通过将 ε 的值在 0.5～1 中进行变动来评价检索有效性和查询性能。从图中可以看出，当 $\varepsilon=0.5$ 时(裁剪率≈50%)，裁剪过的索引和未裁剪的索引在检索结果质量上没有显著区别(P@10 由 0.503 降为 0.500，MAP 由 0.260 降为 0.238)。然而，每次查询的平均时间由 188 ms 降到了 118 ms(−37%)。对于裁剪程度更大的裁剪参数，性能甚至会更好。但是，如果超过 70% 的位置信息从索引中被移除，搜索结果质量就开始受到严重影响。

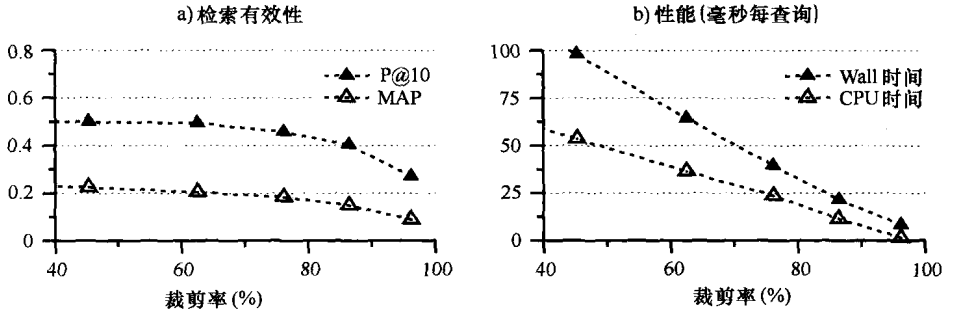


图 5-8 基于词项的索引裁剪算法，其中 $K=1000$，ε 在 0.5～1 之间。效率评价的数据集为：来自 TREC TB 2006 的 10 000 个查询。有效性评价的数据集为：TREC 编号从 701～800 的主题

2. 基于文档的裁剪

基于文档的索引裁剪算法受基于文档的伪相关反馈方法(参见8.6节)启发。伪相关反馈是一种两步查询处理策略，第一步：对于给定查询，搜索引擎确定前 k' 个得分最高的文档。它假设这 k' 个文档都是与查询相关的（**伪相关**（pseudo-relevance)），并从中选择它认为最能代表这些文档的一组词项集合。通常是基于词项分布的统计分析来进行选择。选择的词项将加到原查询中去（通常带较低的权重，因此它们不比原查询词项更重要)，第二轮查询中用这个扩展查询来评分。事实证明，由伪相关反馈选出的词项通常包含了原查询词项。因此，在索引阶段就可以为每个文档运行一个不依赖于特定查询的伪相关反馈算法——来为有可能获得高分的文档选择一组词项。

受Carpineto等人（2001）提出的伪相关反馈机制的启发，Büttcher和Clarke（2006）提出了以下基于文档的裁剪算法：

- 选择一个裁剪参数 $\lambda \in (0, 1]$。
- 对于每一个文档 d，根据以下函数对 d 中所有 n 个不同词项排序：

$$\text{score}(t, d) = p_d(t) \cdot \log\left(\frac{p_d(t)}{p_C(t)}\right) \tag{5-19}$$

其中，$p_d(t) = f_{t,d}/l_d$ 是根据文档 d 的一元语言模型计算的 t 的出现概率，$p_C(t) = l_t/l_c$ 是根据文档集 C 的一元语言模型计算的 t 的出现概率。选择前 $\lceil \lambda \cdot n \rceil$ 个词项保留在索引中，其余的丢弃掉。

如果你对信息学理论熟悉，那么公式（5-19）可能会使你想起两个概率分布之间的**Kullback-Leibler距离**（Kullback-Leibler divergence，KL divergence)。给定两个离散的概率分布 f 和 g，它们的KL距离定义为：

$$\sum_x f(x) \cdot \log\left(\frac{f(x)}{g(x)}\right) \tag{5-20}$$

KL距离常用来描述两个概率分布之间的差异。它在信息检索的很多领域都有重要作用。9.4节中用它来计算排名检索中一个语言模型框架下的文档得分。

在索引裁剪的语义中，我们用KL距离来量化给定的文档与文档集中的其他文档的差异。公式（5-19）的裁剪标准可视为是选择出对文档的KL距离贡献最大的那些词项，因此，在某种意义上，也就是最能够代表文档独特性的那些词项。请注意，这个裁剪标准独立于搜索引擎使用的评分函数。

图5-9给出了基于文档的裁剪算法的实验结果。从中可以看出适当的裁剪对检索有效性几乎没有任何影响。例如，当从索引中删除70%的位置信息后（$\lambda = 0.3$)，MAP仍然有0.238的水平（之前0.260)。P@10几乎不受影响，甚至比未裁剪的索引还要高些(0.508与0.503)。只有当过度裁剪时（$\lambda < 0.1$)，才开始发现对早期的查准率（P@10)有影响。

对比图5-8和图5-9，会发现在任何裁剪率上基于词项的裁剪方法都比基于文档的裁剪方法响应时间要短。这一现象可解释为基于词项的裁剪方法在索引中为所有词项选择位置信息。这些词项大多数是人工产生的（例如，文档时间戳)，不会在搜索查询中出现。基于文档的方法则认为这些词项对于它们代表的文档并不重要（因为它们通常在文档中只出现一次)。它从索引中去除这些词项，导致剩余词项的位置信息列表的平均长度增加。因此，同样地裁剪水平，基于文档的方法需要搜索引擎完成更多的工作。

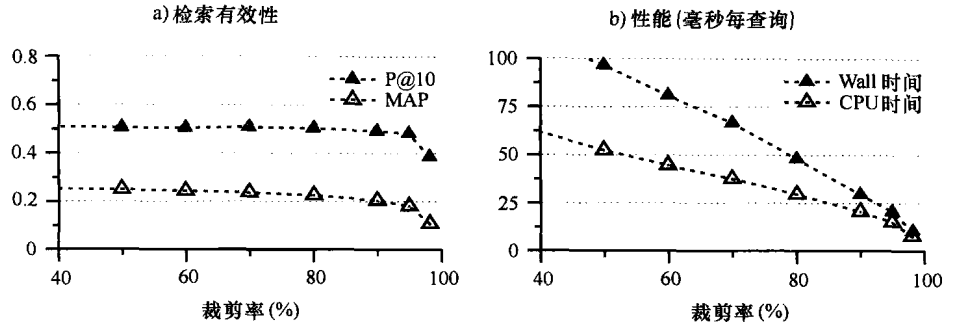


图 5-9 基于文档的索引裁剪算法，其中在 0.05～0.6 之间。效率评价的数据集为：来自 TREC TB 2006 的 10 000 个查询。有效性评价的数据集为：TREC 编号从 701～800 的主题

图 5-10 对基于词项和基于文档的裁剪方法进行了一对一的比较。在适中的性能层次上（每次查询 >60 ms），两种方法没有很大的区别，都能达到与未裁剪索引一样的查准率。然而，当追求更高的性能时，差异就开始变得明显了。例如，在固定的查准率水平 P@10＝0.45，基于文档的裁剪方法比基于词项的裁剪方法大约要快两倍。

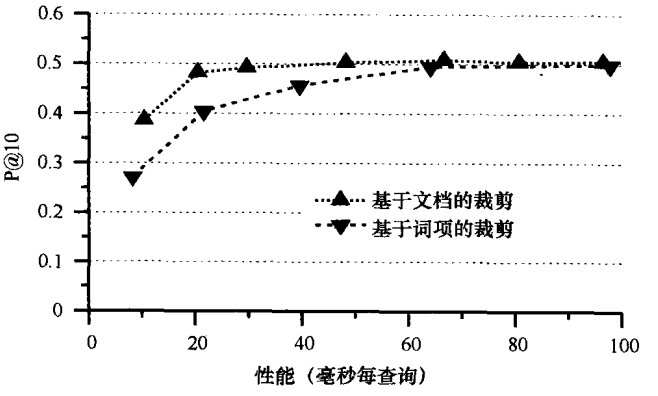


图 5-10 基于文档和基于词项的静态索引裁剪方法的效率与有效性之间的平衡

3. 正确性保证

尽管上面讨论的静态索引裁剪方法具有很有趣的特性，可以通过改变裁剪算法的参数使搜索引擎操作在效率和有效性之间达到平衡，但在现实中它们并不那么具有吸引力。例如，在 Web 搜索市场中，竞争激烈，哪怕微小的检索结果质量下降也会使竞争处于劣势，以致失去市场份额。性能改进固然是受欢迎的，但也只在不会给检索有效性带来负面影响的情况下。

Ntoulas 和 Cho（2007）提出了一种方法，可用于判断从裁剪的索引中获得的搜索结果是否与从未裁剪的索引中获得的搜索结果一致。他们的工作基于的假设是搜索引擎采用的排名函数是词袋评分模型（参见公式（5-16））。

为了简便起见，我们基于公式（5-16）的一个修改版本（无质量那个部分）来讨论 Ntoulas 方法：

$$\text{score}(q,d)=\sum_{t\in q}\text{score}(t,d) \tag{5-21}$$

现在假设搜索引擎使用上述形式的评分函数，在一个丢弃了部分位置信息的裁剪后的索引上处理查询。进一步假设使用基于词项的 top-(K,ε)裁剪机制来裁剪索引。那么对于索引中的每个词项 t，都存在一个阈值 ϑ_t，使得 t 的词项贡献都大于 ϑ_t 的所有位置信息都在索引中，同时更小贡献的位置信息已被丢弃。

可用以下事实来确定裁剪索引产生的结果是否与未裁剪索引产生的结果一样或是不同。

考虑一个三单词查询 $q=\langle t_1, t_2, t_3\rangle$ 和一个经过裁剪后只包含 t_1 和 t_2 但不包含 t_3 的文档 d。单从索引来看，因为位置信息被裁剪掉了，无法判断 d 到底是否包含 t_3。但是，可以知道就算 d 包含 t_3，score(t_3，d)也不大于 ϑ_{t_3}（因为现在索引中没有 t_3 的位置信息——译者注）。这就可以为 d 的最终得分设定一个上限：

$$\text{score}(q,d) \leqslant \text{score}(t_1,d) + \text{score}(t_2,d) + \vartheta_{t_3} \tag{5-22}$$

考虑到裁剪索引的不完整性，我们根据 d 的这个上限而不是实际得分来对 d 进行排名。修改图 5-11 中的改进版本算法，结果如图 5-11 所示。算法中，如果一个文档没有包含查询词项 t，就用裁剪因子 ϑ_t 替代实际的得分贡献。最后，如果可以得到前 k 个结果的每一个文档的完整信息（第 16 行 $results[i].numHits=n$），就知道这个排名是正确的。

```
rankBM25_DocumentAtATimeWithCorrectnessGuarantee (⟨t1, ..., tn⟩, k) ≡
1   m ← 0   // m是匹配文档的总数
2   d ← min1≤i≤n{nextDoc(ti, −∞)}
3   while d < ∞ do
4     results[m].docid ← d
5     results[m].score ← 0
6     results[m].numHits ← 0
7     for i ← 1 to n do
8       if nextDoc(ti, d − 1) = d then
9         results[m].score ← results[m].score + log(N/Nti) · TFBM25(ti, d)
10        results[m].numHits ← results[m].numHits + 1
11      else
12        results[m].score ← results[m].score + ϑti
13    m ← m + 1
14    d ← min1≤i≤n{nextDoc(ti, d)}
15  按 score的降序排列 results[0..(m − 1)]
16  if results[i].numHits = n for 0 ⩽ i < k then
17    return (results[0..(k − 1)], true)   // 结果保证是正确的
18  else
19    return (results[0..(k − 1)], false)   // 结果可能是不正确的
```

图 5-11 基于裁剪索引的 document-at-a-time 查询处理算法。根据是否能保证返回的前 k 个搜索结果的正确性，函数返回 true 或 false

图 5-11 中的算法可作为一个两层查询处理结构的一部分，其中第一层由一个裁剪索引组成，第二层由原始的未裁剪索引组成。首先将查询送到裁剪索引。如果确保裁剪索引产生的结果是正确的，工作就完成了，否则，用未裁剪索引重新处理查询。取决于裁剪索引与未裁剪索引的相对性能，以及裁剪索引能正确返回的查询比例，这个方法能在不牺牲搜索结果质量的同时降低处理每次查询平均的工作量。但是，注意到图 5-11 中的算法不能与基于文档的裁剪方法同时使用，因为后者不能为裁剪的词项提供一个得分的上限 ϑ_t。因此，尽管基于文档的裁剪方法通常不会降低搜索结果的质量，但索引也没有包含足够的信息来证明这一点。

5.2 轻量级结构

第 2 章的一开始为倒排索引提出了一个简单的抽象数据类型（ADT）。我们证明了这个 ADT 可用于词组搜索和计算涉及结构元素的简单关系，例如确定莎士比亚戏剧集中某个角色所说的所有台词。在那个例子中，我们显式地使用了 ADT 的方法找到这个角色名字出现的地方，然后确定这些地方的台词。现在使用**区域代数**（region algebra）将这个方法进行扩展和一般化。

区域代数提供支持轻量级结构的合并和处理文本区间（或**区域**（region））的操作和函

数。它们在基本的文档检索和复杂的全 XML 检索中提供一个中间点（参见第 16 章）。为支持搜索引擎和数字图书馆，它们都提供了许多统一的**高级搜索**（advanced search）功能。文献中已描述了许多区域代数的内容，可追溯到 PAT 区域代数，它在 20 世纪 80 年代早期由**新牛津英语字典**（New Oxford English Dictionary）项目（Gonnet，1987；Salminen 和 Tompa，1994）创建。本章涉及的区域代数是这组内容的代表，但相对简单，可以直接在第 2 章介绍的技术（Clarke 等人，1995a，1995b；Clarke 和 Cormack，2000）上建立。

5.2.1 广义索引表

概括地说，区域代数作用于文本区间组成的集合上。每个区间可以表示为$[u,v]$对，其中，u 代表区间的起点，v 代表区间的终点。我们介绍的区域代数对它所操作的区间集有一个简单但很重要的要求：**区间中不能嵌套另一个区间**。具有这一性质的区间集被称为**广义索引表**（generalized concordance list）或 GC-list。

GC-list 名字源于**索引表**（concordance），文档中的单词按字母顺序排列，并且它们所在的文本也一同位于索引中。早在计算机索引发明之前，纸质的索引表就已经是搜索主要作品，例如莎士比亚文集的一个工具。本质上，它们都是倒排索引的早期形式。

我们用符号$[u,v]\sqsubset[u',v']$来表示$[u,v]$嵌套在$[u',v']$中。类似的，$[u,v]\not\sqsubset[u',v']$表示$[u,v]$不嵌套在$[u',v']$中。若$[u,v]$和$[u',v']$都是我们的区域代数所操作的区间集，那么$[u,v]\not\sqsubset[u',v']$，即 $u<u'$且$v<v'$或者$u>u'$且$v>v'$。例如，下面的区间集

$$S=\{[1,10],[5,9],[8,12],[15,20]\} \tag{5-23}$$

不是一个 GC-list，因为$[5,9]\sqsubset[1,10]$。可以把这个集合约减为 GC-list，例如，通过移除$[1,10]$，得到：

$$\{[5,9],[8,12],[15,20]\} \tag{5-24}$$

请注意，$[5,9]$和$[8,12]$**交叠**（overlap）。不同于内嵌，区间交叠是允许的。实际上，交叠正是区域代数正确操作的关键所在。注意到也可通过移除区间$[5,9]$将 S 约减为 GC-list。在本节的后面会说明，优先移除大区间是首选。

可将一组区间集约减为 GC-list 的过程形式化如下：让 S 为区间集。定义函数$\mathcal{G}(S)$为：

$$\mathcal{G}(S)=\{a \mid a\in S \text{ 且 } \nexists\, b\in S \text{ 使得 } b\sqsubset a\} \tag{5-25}$$

给定一个文本区间集，这个函数去掉那些内嵌了集合中其他元素的区间，从而将集合约减为 GC-list。因此，集合S是一个 GC-list 当且仅当$\mathcal{G}(S)=S$且$\mathcal{G}(S)=\mathcal{G}(\mathcal{G}(S))$。GC-list 是从区域代数的操作中自然出现的。我们的实现不会显式地将一个区间集约减为 GC-list，这个函数为代数操作提供一个简明的解释。

GC-list 还具有许多重要的性质。既然区间不能嵌套，GC-list 中就不会有两个区间开始于同一位置。因此，给定两个区间$[u,v]$和$[u',v']$，有 $u<u'$或者$u>u'$。同样地，也不会有两个区间结束于同一位置；如果 $u<u'$，那么 $v<v'$。因此，按开始位置或结束位置对 GC-list 中的区间进行排序得到的结果是一样的。最后，每个位置最多只能是一个区间的开始位置，因此 GC-list 的大小受文档总长度的限制。

模式独立的位置信息列表可视为是一个 GC-list，其中所有区间长度都为 1，即开始和结束于同一位置。因此，图 2-1 中的“first”的位置信息列表可视为如下 GC-list

$$\text{“first”}=\{[2205,2205],[2268,2268],\cdots,[1271487,1271487]\} \tag{5-26}$$

词组搜索的结果也可看做是一个 GC-list，词组的开始和结束的位置信息就构成了 GC-list 中的一个区间。例如，莎士比亚文集中的词组“first witch”对应的 GC-list 为：

$$\text{“first witch”} = \{[745406, 745407], [745466, 745467], [745501, 745502], \cdots\} \tag{5-27}$$

5.2.2 操作符

我们介绍的区域代数有 7 种二元操作符，如表 5-2 所示。每一个操作符都定义在 GC-list 上，并且操作的结果也为 GC-list。这些操作符可以分为三类：包含、组合和排序。

表 5-2 区域代数中二元操作的定义

包含操作
Contained In：
$A \lhd B = \{a \mid a \in A \text{ 且 } \exists b \in B \text{ 使得 } a \subset b\}$
Containing：
$A \rhd B = \{a \mid a \in A \text{ 且 } \exists b \in B \text{ 使得 } b \subset a\}$
Not Contained In：
$A \not\lhd B = \{a \mid a \in A \text{ 且 } \nexists b \in B \text{ 使得 } a \subset b\}$
Not Containing：
$A \not\rhd B = \{a \mid a \in A \text{ 且 } \nexists b \in B \text{ 使得 } b \subset a\}$
组合操作
Both Of：
$A \triangle B = \mathcal{G}(\{c \mid \exists a \in A \text{ 使得 } a \subset c \text{ 且 } \exists b \in B \text{ 使得 } b \subset c\})$
One Of：
$A \triangledown B = \mathcal{G}(\{c \mid \exists a \in A \text{ 使得 } a \subset c \text{ 或 } \exists b \in B \text{ 使得 } b \subset c\})$
排序操作
Before：
$A \cdots B = \mathcal{G}(\{c \mid \exists [u, v] \in A \text{ 且 } \exists [u', v'] \in B \text{ 其中 } v < u' \text{ 且 } [u, v'] \subset c\})$

包含操作符（containment operator）从 GC-list 中选出 Contained In、Not Contained In、Containing 或 Not Containing 另一个 GC-list 的区间的区间。包含操作符根据文档中结构化元素的层次特征来形式化查询。包含操作符右边的表达式作为一个过滤器限制了左边的表达式——操作的结果是左边 GC-list 的一个子集。

两个**组合操作符**（combination operator）与标准的布尔操作符 AND 和 OR 类似。“Both Of”操作符类似于 AND：结果的每个区间包含由两个操作对象的区间。$\mathcal{G}()$函数用来确保结果仍是一个 GC-list。“One Of”操作符合并两个 GC-list：结果的每个区间都是其中一个操作对象的区间。

排序操作符产生串联。结果的每个区间以第一个操作对象的某个区间开始，以第二个操作对象的某个区间结束。第一个操作对象的区间在第二个操作对象的区间开始之前结束。排序操作符可用于以下情形，将描述结构元素的标签进行连接，产生每个区间都对应结构元素的一次出现的 GC-list。使用这 7 种二元操作符的例子将在下一节给出。

除了二元操作符，还有两个一元**投影操作符**（projection operator），π_1 和 π_2。如果 A 是一个 GC-list，定义

$$\pi_1(A) = \{[u, u] \mid \exists v \text{ 且 } [u, v] \in A\} \tag{5-28}$$

$$\pi_2(A) = \{[v, v] \mid \exists u \text{ 且 } [u, v] \in A\} \tag{5-29}$$

例如，

$$\pi_1(\{[5, 9], [8, 12], [15, 20]\}) = \{[5, 5], [8, 8], [15, 15]\} \tag{5-30}$$

$$\pi_2(\{[5,9],[8,12],[15,20]\}) = \{[9,9],[12,12],[20,20]\} \tag{5-31}$$

定义所有区间定长的 GC-list 为

$$[i] = \{[u,v] \mid v-u+1=i\} \tag{5-32}$$

例如，

$$[10] = \{\cdots,[101,110],[102,111],[103,112],\cdots\} \tag{5-33}$$

5.2.3 例子

区域代数可以用来解决 2.1.3 节给出的查询，例如：

1. 巫婆说的每一行台词

(“⟨LINE⟩” ⋯ “⟨/LINE⟩”)
◁ ((“⟨SPEECH⟩” ⋯ “⟨/SPEECH⟩”)
▷ ((“⟨SPEAKER⟩” ⋯ “⟨/SPEAKER⟩”) ▷ “witch”))

括弧括起来的表达式指示了运用操作符的顺序。查询首先确定所有是巫婆的角色。她们所讲的台词被提取处理。所有的中间结果和最终结果都是 GC-lists。

2. 说“to be or not to be”的角色名

(“⟨SPEAKER⟩” ⋯ “⟨/SPEAKER⟩”)
◁ ((“⟨SPEECH⟩” ⋯ “⟨/SPEECH⟩”)
▷ ((“⟨LINE⟩” ⋯ “⟨/LINE⟩”) ▷ “to be or not to be”))

先找出包含这个引用的台词行，就确定了对应的台词，对应的角色就可以被抽取处理。因为这个词组在莎士比亚戏剧集中只出现了一次，因此结果 GC-list 中仅包含一个区间。

3. 所有提及巫婆和打雷的戏剧的名称

(“⟨TITLE⟩” ⋯ “⟨/TITLE⟩”)
◁ ((“⟨PLAY⟩” ⋯ “⟨/PLAY⟩”) ▷ (“witch” △ “thunder”))

查询首先确定同时包含词“witch”和“thunder”的文本片段，将结果表示为一个 GC-list。然后抽取出包含这些片段的戏剧名。

这些例子中，都做了合理性假设，即莎士比亚戏剧集是用 XML 结构编码：如戏剧名、场景、戏剧和每行台词等元素都内嵌于合适的标签中；所有的台词都包括一个角色和一到多行台词。此外，假设这些结构化元素互不嵌套（如台词中不会包含另外一段台词）。

区域代数一个独有的特性是可以为布尔查询赋予一定的含义，例如“witch” △ “thunder”，因为结果可用一个 GC-list 来表达，因此不必指出一个明确的范围，如戏剧还是台词行。在莎士比亚戏剧集中，

$$\text{“witch”} \triangle \text{“thunder”} = \{[31463,36898],[36898,119010],[125483,137236],\cdots\} \tag{5-34}$$

在这个 GC-list 中，区间之间存在着交叠，还有一些区间横跨几个戏剧。因为区间之间互不嵌套，因此每一个区间以“witch”或“thunder”开始，以其他词项结束。定位戏剧、场景、台词行，或满足布尔表达式的结构化元素，实际上是用一个 GC-list 来过滤另外一个 GC-list 的过程。

组合操作符足以处理仅包含 AND 和 OR 的布尔表达式。例如，查询

(“witch” ▽ “king”) △ (“thunder” ▽ “dagger”)

确定包含“witch”或“king”，“thunder”或“dagger”的文本片段。得到的 GC-list 区间中

不包含也满足该布尔表达式的区间。任何满足该布尔表达式的大区间总是包含一个 GC-list 中的区间。在公式（5-25）定义的$\mathcal{G}(S)$中，我们总是选择移除两个嵌套区间中较大的那个。这个选择内在的原因在这里变得比较清楚了。至少在布尔表达式中，计算这些较大的区间没有什么好处。

布尔 NOT 需要一个明确的范围。例如，查询没有提及巫婆或打雷的戏剧可表示如下：

$$(\text{“⟨PLAY⟩”} \cdots \text{“⟨/PLAY⟩”}) \not\rhd (\text{“witch”} \bigtriangledown \text{“thunder”})$$

区域代数是许多搜索引擎和数字图书馆中高级搜索功能的实现方法之一。对于这些信息检索系统来说，支持布尔查询是相当常见的，这些查询限制在特定字段上，如题目、作者或摘要。这些查询可以很自然地映射为区域代数。很多 Web 搜索引擎都支持将搜索限制在某个特定网站的功能。包含查询词项“site：uwaterloo.ca”将搜索限制在滑铁卢大学的网页上。假设适当标注和索引后，这个查询可以转换为表达式

$$(\text{“⟨PAGE⟩”} \cdots \text{“⟨/PAGE⟩”}) \rhd ((\text{“⟨SITE⟩”} \cdots \text{“⟨/SITE⟩”}) \rhd \text{“uwaterloo.ca”})$$

5.2.4 实现

区域代数的实现泛化了词组搜索算法、邻近度排名算法和第 2 章介绍的布尔查询算法。如前所述，每个区间的开始位置和结束位置对 GC-list 中的元素排序会得到同样地结果。使用这个排序可为高效实现区域代数设计一个框架。使用类似在 2.1 节定义的倒排索引的 ADT，该方法将位置信息索引为 GC-list。GC-list 中区间的排序是定义这个 ADT 的基础。给定一个 GC-list 和文档集中的一个位置信息，我们索引进 GC-list 去找到某种意义上“最接近”这个位置的区间。

考虑评价表达式 $A \cdots B$（见图 5-12）。结果 GC-list 中的一个区间开始于 A 的一个区间，结束于 B 的一个区间。假设$[u,v]$是 A 中第一个区间。如果$[u',v']$是 B 中第一个满足$u'>v$的区间，那么v'一定是$A \cdots B$ 中第一个区间的结束位置。我们索引进 B 来找出满足$u'>v$的第一个区间。A 中在u'之前结束的最后一个区间是$A \cdots B$ 第一个区间的开始。我们索引进 A 来找到满足$v''<u'$的最后一个区间 $[u'',v'']$。区间 $[u'',v'']$ 就是 $A \cdots B$ 的第一个答案。首先索引进 A，再到 B，再回到 A，使用三步可以找到 $A \cdots B$ 的第一个区间。$A \cdots B$ 的下一个答案应从u''后开始。我们索引进 A 产生u''之后的第一个区间。这种先后索引进 A 和 B 的过程可以继续找到$A \cdots B$ 中剩下的区间。

实现框架由 4 个方法组成，允许以不同的方式索引进 GC-list。每一个方法都代表了 GC-list 中与文档集中指定位置“最接近的区间”的符号。我们用操作对象上的方法实现了区域代数每一个操作的 4 个方法。

- 方法$\tau(S,k)$返回 GC-list S 中开始于 k 或位于 k 之后的第一个区间：

$$\tau(S,k)=\begin{cases}[u,v] & \text{若}\exists\,[u,v]\in S\text{ 使得 }k\leqslant u \\ & \quad\text{且 }\nexists\,[u',v']\in S\text{ 使得 }k\leqslant u'<u \\ [\infty,\infty] & \text{若}\nexists\,[u,v]\in S\text{ 使得 }k\leqslant u\end{cases} \tag{5-35}$$

- 方法 $\rho(S,k)$返回 S 中结束于 k 或位于 k 之后的第一个区间：

$$\rho(S,k)=\begin{cases}[u,v] & \text{若}\exists\,[u,v]\in S\text{ 使得 }k\leqslant v \\ & \quad\text{且 }\nexists\,[u',v']\in S\text{ 使得 }k\leqslant v'<v \\ [\infty,\infty] & \text{若 }\nexists\,[u,v]\in S\text{ 使得 }k\leqslant v\end{cases} \tag{5-36}$$

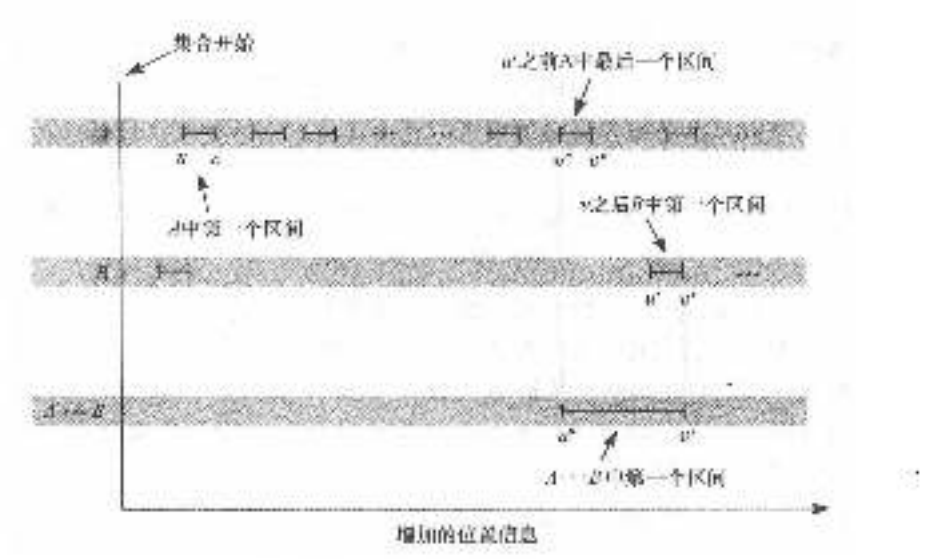


图 5-12 评价 GC1 表达式 $A \cap B$

- 方法 $\tau'(S,k)$ 是 τ 的逆方法。返回 S 中结束于 k 或位于 k 之前的最后一个区间：

$$\tau'(S,k)=\begin{cases}[u,v] & \text{若}\exists\,[u,v]\in S\text{使得}k\geqslant v\\ & \text{且}\nexists\,[u',v']\in S\text{使得}k\geqslant v'>v\\ [-\infty,-\infty] & \text{若}\nexists\,[u,v]\in S\text{使得}k\geqslant v\end{cases}\tag{5-37}$$

- 方法 $\rho'(S,k)$ 是 ρ 的逆方法。返回 S 中开始于 k 或位于 k 之前的最后一个区间：

$$\rho'(S,k)=\begin{cases}[u,v] & \text{若}\exists\,[u,v]\in S\text{使得}k\geqslant u\\ & \text{且}\nexists\,[u',v']\in S\text{使得}k\geqslant u'>u\\ [-\infty,-\infty] & \text{若}\nexists\,[u,v]\in S\text{使得}k\geqslant u\end{cases}\tag{5-38}$$

例如，当 $S=\{[5,9],[8,12],[15,20]\}$，$k=10$ 时，我们有：

$$\tau(\{[5,9],[8,12],[15,20]\},10)=[15,20]$$
$$\rho(\{[5,9],[8,12],[15,20]\},10)=[8,12]$$
$$\tau'(\{[5,9],[8,12],[15,20]\},10)=[5,9]$$
$$\rho'(\{[5,9],[8,12],[15,20]\},10)=[8,12]$$

和倒排索引的 ADT 一样，符号 ∞ 和 $-\infty$ 分别是文档的结束符和开始符。

```
τ(t,k) =
1   if k = -∞ then
2       v ← ∞
3   else if k = -∞ then
4       v ← -∞
5   else
6       v ← next(t,k-1)
7   return [v,v]

τ'(t,k) =
8   if k = ∞ then
9       v ← ∞
10  else if k = -∞ then
11      v ← -∞
12  else
13      v ← prev(t,k+1)
14  return [v,v]
```

图 5-13 $\tau(t,k)$ 和 $\tau'(t,k)$ 的伪码，其中 t 是一个词项

方法 $\tau(S,k)$、$\rho(S,k)$、$\tau'(S,k)$ 和 $\rho'(S,k)$ 与第 2 章的倒排索引 ADT 有紧密联系。对于单个查询词项，这些方法可用 ADT 来定义，如图 5-13 中的 $\tau(t,k)$ 和 $\tau'(t,k)$。因此对图 2-1 的倒排索引，我们有

$$\tau(\text{"first"},745400)=[745501,745501]\tag{5-39}$$

$\rho(t,k)$ 的定义与 $\tau(t,k)$ 的类似，$\rho'(t,k)$ 的定义与 $\tau'(t,k)$ 的类似。对于定长区间 [i]，GC list 方法的实现甚至更简单，因为可以直接从 k 中计算出结果（见练习 5.5）。

二元操作符方法建立在操作对象的方法上。图 5-14 给出了一些例子。在$\tau(A\cdots B,k)$的实现中，大多数行都用来处理∞和−∞边界。算法的核心是第 5、8 和 11 行，反映了图 5-12 中解释的思想。第 5 行计算开始于 k 或位于 k 之后的A 中第一个区间。第 8 行计算开始于或位于上述 A 的区间之后B 中第一个区间。这个区间的结束位置也正是答案$\tau(A\cdots B,k)$的结束位置。最后，第 11 行计算答案的开始位置。

```
    τ(A ... B,k) ≡
1     if k = ∞ then
2       return [∞,∞]
3     if k = −∞ then
4       return [−∞,−∞]
5     [u,v] ← τ(A,k)
6     if [u,v] = [∞,∞] then
7       return [∞,∞]
8     [u',v'] ← τ(B,v+1)
9     if [u',v'] = [∞,∞] then
10      return [∞,∞]
11    [u'',v''] ← τ'(A,u'−1)
12    return [u'',v']

    τ(A ◁ B,k) ≡
13    if k = ∞ then
14      return [∞,∞]
15    if k = −∞ then
16      return [−∞,−∞]
17    [u,v] ← τ(A,k)
18    if [u,v] = [∞,∞] then
19      return [∞,∞]
20    [u',v'] ← ρ(B,v)
21    if [u',v'] = [∞,∞] then
22      return [∞,∞]
23    if u' ≤ u then
24      return [u,v]
25    else
26      return τ(A ◁ B,u')

    ρ(A ▷ B,k) ≡
27    if k = ∞ then
28      return [∞,∞]
29    if k = −∞ then
30      return [−∞,−∞]
31    [u,v] ← ρ(A,k)
32    if [u,v] = [∞,∞] then
33      return [∞,∞]
34    [u',v'] ← τ(B,u)
35    if [u',v'] = [∞,∞] then
36      return [∞,∞]
37    if v' ≤ v then
38      return [u,v]
39    else
40      return ρ(A ▷ B,v')
```

图 5-14 $\tau(A\cdots B,k)$、$\tau(A\triangleleft B,k)$和$\rho(A\triangleright B,k)$的实现，其中 A 和B 都是 GC-list

这些方法也可以用跳跃式搜索实现（见图 2-5）。为了生成查询 Q 的所有答案，迭代的调用$\tau(Q,k)$：

```
k ← 0
while k < ∞ do
    [u,v] ← τ(Q,k)
    if k ≠ ∞ then
        output [u,v]
        k ← u+1
```

5.3 延伸阅读

5.1 节一开始就指出了，排名检索的查询处理器既可使用合取的也可使用析取的布尔模型。结合这两个模型也是可以的。例如，先用合取式来评价查询，当合取式匹配了太多文档的时候就将其转换为析取式。Broder 等人（2003）提出了这种“弱-AND”方法的更一般化的版本。

MAXSCORE 策略（5.1.1 节）由 Turtle 和 Flood 提出（1995）。Smith 的早期工作（1990）也提出过类似的算法。最近，Strohman 等人（2005）提出了 MAXSCORE 的一个改进版本，使用了预计算的最靠前文档列表（你可将这些列表视为是一个过度裁剪的索引）来获得前 k 个最好的搜索结果的得分的下界。这个下界可用于在文档被评分之前将查询词项从堆中移除。Strohman 等人（2005）报告说比原来的 MAXSCORE 算法节省了 23% 的时间。Zhu 等人（2008）提出了 MAXSCORE 的一个变形，使用了包含词项邻近度的排名函数。他们的方法与 Strohman 方法类似，但使用了一个词组索引来计算邻近度敏感的靠前文档。

Persin 等人（1996）在“基于词频的索引”的含义中首次研究了影响力排序。这个基本方法的改进随后在 Anh 等人（2001，2004，2006）一系列的文章中提出。这个系列的最后一篇文章尤其有趣，因为它说明了基于影响力排序索引的结构对于 document-at-a-time 查询处理策略的适用性——这并不是非常直观的。

PAT区域代数（Gonnet，1987；Salminen 和 Tompa，1994）最初是因为**新牛津英语字**

典项目的需要而被提出，之后被 Open Text 公司作为他们搜索引擎中的一部分而商业化了（Open Text Corporation，2001）。Tim Bray，该引擎第一版的唯一的作者，之后成为 XML 标准的创建人之一。这个引擎的最后版本在 20 世纪 90 年代中期为 Yahoo! 提供搜索服务，并一直促进该公司企业内容管理产品向前发展。

PAT的成功激发了许多对它的扩展和改进。PADRE 系统（Hawking 和 Thistlewaite，1994）实现了对 PAT的并行化。Burkowski（1992）为层次化区域代数提出了包含和集合操作。本章讨论的区域代数基于 Clarke 等人的工作（1995a，1995b），也可视为是 Burkowski 区域代数的一个扩展（和简化）。Dao 等人（1996）以及 Jaakkola 和 Kilpeläinen（1999）提出了支持对递归结构（例如，台词段中包含台词段）的进一步扩展。Consens 和 Milo（1995）探讨了区域代数的理论以及它的局限性。Navarro 和 Baeza-Yates（1997）将区域代数分成很多层，支持直接祖先/后代关系及递归结构。

读者可以在 Clarke 和 Cormack（2000）中找到更多关于高效实现组合操作的内容。Zhang 等人（2001）讨论了在关系数据库系统中高效实现包含关系的算法。Young-Lai 和 Tompa（2003）阐述了一种自底向上的，一遍扫描的实现方法。Boldi 和 Vigna（2006）探讨了一种使用懒惰评价的高效实现。

5.4 练习

练习 5.1 公式（5-12）给出了使用堆的 document-at-a-time 算法在最坏情况下的复杂度为$\Theta(N_q \cdot \log(n) + N_q \cdot \log(k))$。

(a) 描述在何种输入（即文档得分分布）下是最坏的情况。

(b) 证明这个算法的平均复杂度一定好于$\Theta(N_q \cdot \log(k))$。可以假定 $k > n$，文档得分是平均分布的，也就是，每个文档获得最高得分，第二高得分……的可能性是一样的。

练习 5.2 为合取查询（布尔 AND）设计一个支持累加器裁剪的 term-at-a-time 查询处理算法。

练习 5.3 图 5-7（b）说明了在每个预计算得分贡献中，如果 B 小于 3 位，每次查询的平均 CPU 时间就会急剧下降。这是查询处理算法采用 MAXSCORE 策略带来的副作用。解释为什么对于较小的 B 值 MAXSCORE 会更有效。

练习 5.4 基于 5.2.3 给出的假定，使用区域代数表示以下查询：

(a) 找出满足“Birnam”后紧跟“Dunsinane”的戏剧。

(b) 找出包含“Birnam”和“Dunsinane”的文本片段。

(c) 找出巫婆说了“Birnam”的戏剧。

(d) 找出第一行包含“toil”或“trouble”，第二行不包含“burn”和“bubble”的台词段。

(e) 找出幽灵所讲的一段台词，包含词“fife”且所在的场景为“Something wicked this way comes”。

练习 5.5 为定长区间写出 4 种 GC-list 方法的伪码实现：$\tau([i], k)$，$\rho([i], k)$，$\tau'([i], k)$和$\rho'([i], k)$。

练习 5.6 为两个投影操作写出 GC-list 方法的伪码实现：$\tau(\pi_1(A), k)$，$\rho(\pi_1(A), k)$，$\tau'(\pi_1(A), k)$，$\rho'(\pi_1(A), k)$，$\tau(\pi_2(A), k)$，$\rho(\pi_2(A), k)$，$\tau'(\pi_2(A), k)$，$\rho'(\pi_2(A), k)$。

练习 5.7 根据图 5-14 的算法模式，写出以下方法的伪码：

(a) $\rho(A \ldots B, k)$

(b) $\tau(A \triangle B, k)$

(c) $\tau(A \triangledown B, k)$

(d) $\tau(A \not\triangleright B, k)$

(e) $\rho(A \lhd B, k)$

(f) $\tau'(A \lhd B, k)$

练习 5.8 为词组写出 4 种 GC-list 方法的伪码实现：$\tau'(t_1 \ldots t_n, k)$，$\rho(t_1 \ldots t_n, k)$，$\tau'(t_1 \ldots t_n, k)$和

$\rho'(t_1 \ldots t_n, k)$。（**提示**：作为一个好起点，请考虑图 2-2 中 nextPhrase 函数的实现。）

练习 5.9（项目练习） 使用 document-at-a-time 评价策略，实现 BM25 排名式子（公式(8-48)）。实现应用 MAXSCORE 策略来减少需要评分的文档数。

5.5 参考文献

Anh, V.N., de Kretser, O., and Moffat, A. (2001). Vector-space ranking with effective early termination. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 35–42. New Orleans, Louisiana.

Anh, V.N., and Moffat, A. (2004). Collection-independent document-centric impacts. In *Proceedings of the 9th Australasian Document Computing Symposium*, pages 25–32. Melbourne, Australia.

Anh, V.N., and Moffat, A. (2006). Pruned query evaluation using pre-computed impacts. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 372–379. Seattle, Washington.

Boldi, P., and Vigna, S. (2006). Efficient lazy algorithms for minimal-interval semantics. In *String Processing and Information Retrieval, 13th International Conference*, pages 134–149. Glasgow, Scotland.

Broder, A.Z., Carmel, D., Herscovici, M., Soffer, A., and Zien, J. (2003). Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 12th International Conference on Information and Knowledge Management*, pages 426–434. New Orleans, Louisiana.

Burkowski, F.J. (1992). An algebra for hierarchically organized text-dominated databases. *Information Processing & Management*, 28(3):333–348.

Büttcher, S., and Clarke, C.L.A. (2006). A document-centric approach to static index pruning in text retrieval systems. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 182–189. Arlington, Virginia.

Carmel, D., Cohen, D., Fagin, R., Farchi, E., Herscovici, M., Maarek, Y., and Soffer, A. (2001). Static index pruning for information retrieval systems. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50. New Orleans, Louisiana.

Carpineto, C., de Mori, R., Romano, G., and Bigi, B. (2001). An information-theoretic approach to automatic query expansion. *ACM Transactions on Information Systems*, 19(1):1–27.

Clarke, C.L.A., and Cormack, G.V. (2000). Shortest-substring retrieval and ranking. *ACM Transactions on Information Systems*, 18(1):44–78.

Clarke, C.L.A., Cormack, G.V., and Burkowski, F.J. (1995a). An algebra for structured text search and a framework for its implementation. *Computer Journal*, 38(1):43–56.

Clarke, C.L.A., Córmack, G.V., and Burkowski, F.J. (1995b). Schema-independent retrieval from heterogeneous structured text. In *Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval*, pages 279–289. Las Vegas, Nevada.

Consens, M.P., and Milo, T. (1995). Algebras for querying text regions. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 11–22. San Jose, California.

Dao, T., Sacks-Davis, R., and Thom, J.A. (1996). Indexing structured text for queries on containment relationships. In *Proceedings of the 7th Australasian Database Conference*, pages 82–91. Melbourne, Australia.

Gonnet, G.H. (1987). PAT *3.1 — An Efficient Text Searching System — User's Manual.* University of Waterloo, Canada.

Hawking, D., and Thistlewaite, P. (1994). Searching for meaning with the help of a PADRE. In *Proceedings of the 3rd Text REtrieval Conference (TREC-3)*, pages 257–267. Gaithersburg, Maryland.

Jaakkola, J., and Kilpeläinen, P. (1999). *Nested Text-Region Algebra.* Technical Report CC-1999-2. Department of Computer Science, University of Helsinki, Finland.

Lester, N., Moffat, A., Webber, W., and Zobel, J. (2005). Space-limited ranked query evaluation using adaptive pruning. In *Proceedings of the 6th International Conference on Web Information Systems Engineering*, pages 470–477. New York.

Moffat, A., and Zobel, J. (1996). Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379.

Navarro, G., and Baeza-Yates, R. (1997). Proximal nodes: A model to query document databases by content and structure. *ACM Transactions on Information Systems*, 15(4):400–435.

Ntoulas, A., and Cho, J. (2007). Pruning policies for two-tiered inverted index with correctness guarantee. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 191–198. Amsterdam, The Netherlands.

Open Text Corporation (2001). *Ten Years of Innovation*. Waterloo, Canada: Open Text Coporation.

Persin, M., Zobel, J., and Sacks-Davis, R. (1996). Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science*, 47(10):749–764.

Salminen, A., and Tompa, F. W. (1994). PAT expressions — An algebra for text search. *Acta Linguistica Hungarica*, 41(1–4):277–306.

Smith, M. E. (1990). *Aspects of the P-Norm Model of Information Retrieval: Syntactic Query Generation, Efficiency, and Theoretical Properties*. Ph.D. thesis, Cornell University, Ithaca, New York.

Strohman, T., Turtle, H., and Croft, W. B. (2005). Optimization strategies for complex queries. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 219–225. Salvador, Brazil.

Turtle, H., and Flood, J. (1995). Query evaluation: Strategies and optimization. *Information Processing & Management*, 31(1):831–850.

Young-Lai, M., and Tompa, F. W. (2003). One-pass evaluation of region algebra expressions. *Information Systems*, 28(3):159–168.

Zhang, C., Naughton, J., DeWitt, D., Luo, Q., and Lohman, G. (2001). On supporting containment queries in relational database management systems. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 425–436. Santa Barbara, California.

Zhu, M., Shi, S., Yu, N., and Wen, J. R. (2008). Can phrase indexing help to process non-phrase queries? In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 679–688. Napa, California.

索引压缩

给定文档集上的倒排索引会非常大，特别当包含了文档集中所有词项出现的位置信息时。在典型的英文文档集中，大约每 6 字节的文本就有一个词条（包括标点和空格）。因此，如果位置信息以 64 位整数存储，我们希望文档集未压缩的位置索引耗费的空间是未压缩的原始文本数据大小的 130%～140%。表 6-1 证实了这个估计，它列出了本书中用到的 3 个文档集未压缩的大小、压缩之后的大小以及未压缩和压缩之后的模式独立索引的大小。例如，TREC45 文档集未压缩的模式独立索引大约要占 331 MB 的空间，是原始文档集大小的 122%。[⊖]

表 6-1 三个样例文档集的未压缩和压缩后的（gzip-best）文档集大小；未压缩（64 位整数）和压缩后的（vByte）模式独立索引的大小

	文档集大小		索引大小	
	未压缩	压缩后	未压缩	压缩后
莎士比亚文集	7.5 MB	2.0 MB	10.5 MB（139%）	2.7 MB（36%）
TREC 45	1904.5 MB	582.9 MB	2331.1 MB（122%）	533.0 MB（28%）
Gov2	425.8 GB	79.9 GB	328.3 GB（77%）	62.1 GB（15%）

减小索引大小的一个显著方法是不要用 64 位整数而用$\lceil \log(n) \rceil$位整数给每个位置信息编码，其中 n 是文档集词条的个数。对于 TREC45 来说，（$\lceil \log(n) \rceil = 29$）将把索引从 2331.1 MB 缩小到 1079.1 MB——是原始文档大小的 57%。对比起原来的 64 位编码，这是一个重大改进。然而，从表 6-1 中可以看出，这距离使用真正的索引压缩技术达到的 533 MB 还差很远。

因为倒排索引由两个主要部分组成：词典和位置信息列表。因此我们研究两种不同类型的压缩方法：词典压缩和位置信息列表压缩。因为词典比起所有位置信息列表来说通常要小得多（见表 4-1），所以研究者和实际工作者都更关注位置信息列表的压缩。然而，词典压缩有时候也是值得的，因为这可以减少搜索引擎对主存的需求，并将释放的内存用于其他目的，例如缓存位置信息列表或搜索结果。

本章剩下的内容主要由三个主要部分组成。第一部分（6.1 节和 6.2 节）简要介绍了通用符号数据压缩技术。第二部分（6.3 节）是位置信息列表的压缩。讨论几种倒排列表的压缩方法并指出倒排索引不同类型之间的差异。还介绍了如何使用文档重排技术提高这些方法的有效性。最后一部分（6.4 节）涵盖了词典数据结构的压缩算法，以及如何通过将内存词典记录以压缩形式存储来显著降低搜索引擎的内存需求。

6.1 通用数据压缩

一般来说，数据压缩算法取出数据块 A，转换成另一个数据块 B，使得（或者期望）B 比 A 要小，也就是，使得在信道上传输 B 或在介质中存储 B 需要更少的位。每个压缩

⊖ GOV2 文档集的数字要比其他两个文档集小是因为它的文档包含了很多 JavaScript 和不需要索引的其他数据。

算法都由两部分组成：**编码器**（encoder）（或**压缩器**（compressor））和**解码器** decoder（或**解压器**（decompressor））。编码器以原始数据 A 为输入，输出压缩数据 B。解码器以 B 为输入，产生某个输出 C。

某个压缩方法可以是**有损的**（lossy）或**无损的**（lossless）。在无损算法中，解码器输出的 C 是原始数据 A 的一个精确副本。在有损算法中，C 不是 A 的一个精确副本而是一个近似，即类似于原始版本。在压缩图像（如 JPEG）或音频文件（如 MP3）时，有损压缩非常有用，这时与原始版本的微小偏差不会被人所感知。但是，为了估计由这些微小偏差导致的质量损失，压缩算法必须预先了解被压缩数据的结构或含义。

本章只关注无损压缩算法，即解压器产生原始数据的精确副本。这主要是因为不清楚一个给定位置信息列表近似重构的值会是什么（除非有时候访问一个近似的词项位置已经足够）。当人们说起**有损索引压缩**（lossy index compression）时，通常不是指数据压缩方法而是指索引裁剪模式（见 5.1.5 节）。

6.2 符号数据压缩

当压缩一个给定数据块 A 时，通常不太关心 A 作为一串字符的实际形式，而是关心 A 中包含的内容（information）。这个内容也称为**消息**（message），用 M 表示。很多数据压缩技术将 M 视为是符号集 S（称为**字母表**（alphabet））中的一个**符号**（symbol）串：

$$M = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle, \quad \sigma_i \in S \tag{6-1}$$

这样的压缩方法称为是**基于符号的**（symbolwise）或者**基于统计的**（statistical）。根据特定的方法和（或）应用，符号可以是一个比特位、一个字节、一个单词（压缩文本中）、一个位置信息（压缩倒排索引中）或者其他的形式。为符号找到一个恰当的定义有时候是困难的，但是一旦确定了这个定义，为了降低总体存储需求，就可以使用常见的统计技术对 M 中的符号进行重编码。符号数据压缩背后的基本思想有两层含义：

1）M 中所有符号不都以相同的频率出现。频繁符号比不频繁符号使用更少的位编码，可用更少的总位数来表示消息 M。

2）符号串 $\langle \sigma_1, \sigma_2, \cdots, \sigma_i, \sigma_n \rangle$ 中的第 i 个符号有时依赖于前面出现的符号 $\langle \cdots, \sigma_{i-2}, \sigma_{i-1} \rangle$。考虑符号之间的这种相互依赖关系，还可以节省更多的空间。

考虑对莎士比亚文集中的文本进行压缩（带 XML 标记的英文文本），并以 ASCII 形式存储，每个字符占 8 字节。不需要特别费力，就能将存储需求从每个字符 8 字节减少到 7 字节，因为文档集中仅含 86 个不同字符。但就算是文档集中出现的字符，也能看出这些字符出现频率差距很大。例如，莎士比亚文集中 6 个最频繁和 6 个最不频繁的字符是：

1. “ ”:	742,018	4. “<”:	359,452		81. “(”:	2	84. “8”:	2
2. “E”:	518,133	5. “>”:	359,452	...	82. “)”:	2	85. “$”:	1
3. “e”:	410,622	6. “t”:	291,103		83. “5”:	2	86. “7”:	1

如果将最频繁的字符（空格）用少 1 位（由 7 位减为 6 位）重编码，最不频繁的两个字符用多 1 位（由 7 位增加为 8 位）重编码，那么可以节省 742 016 位。

另外，文档集中连续字符之间是互相依赖的。例如，字符“u”，在文档集中出现了 114 592 次，在总体频率范围内大致位于中间位置。但是，字符“q”的每一次出现后面总是紧跟“u”。因此，前面是“q”的“u”就不需要编码了，因为“q”后面不可能是其他字符。

6.2.1 建模和编码

符号压缩方法通常分为两步：建模和编码。建模阶段计算出一个概率分布$\mathcal{M}$（也被称为**模型**（model)），将符号映射为它们出现的概率。编码阶段根据码$\mathcal{C}$对消息 M 中的符号重编码。码是从每个符号σ到它对应的码字$\mathcal{C}(\sigma)$的一个映射，通常是一个位串。$\mathcal{C}(\sigma)$取决于由压缩模型$\mathcal{M}$得到的σ的概率。如果$\mathcal{M}(\sigma)$很小，那么$\mathcal{C}(\sigma)$会比较长；如果$\mathcal{M}(\sigma)$很大，那么$\mathcal{C}(\sigma)$会比较短（σ的码字的长度由它所占的位数来度量)。

根据建模阶段如何以及何时发生，符号压缩方法可能是以下三种类型中的一种：

- **静态方法**（static method)。模型$\mathcal{M}$独立于要压缩的消息 M。它假定消息中的符号满足一个事先定义的概率分布。否则，压缩结果会相当令人失望。
- **半静态方法**（Semi-static method)。在消息 M 上执行一次初始的扫描并计算出用于压缩的模型$\mathcal{M}$。与静态方法相比，半静态方法的好处是不会盲目假定一个特定分布。但是，由编码器计算出的模型$\mathcal{M}$需要传输到解码器端（否则，解码器将不知如何处理编码过的符号串)，因此需要尽可能地压缩。如果模型本身太大，半静态方法就失去了对比静态方法的优势。
- **自适应压缩方法**（adaptive compression method)。编码过程从初始静态模型开始，基于 M 中已编码符号的特征逐步调整模型。当压缩器编码 σ_i 时，使用的模型$\mathcal{M}_i$仅依赖于前面已编码的符号（和初始静态模型)：

$$\mathcal{M}_i = f(\sigma_1, \cdots, \sigma_i - 1)$$

当解压器需要解码 σ_i 时，它已经处理 σ_1、…、σ_{i-1}，因此可以使用同一个函数 f 来重构模型$\mathcal{M}_i$。因此，自适应方法的好处是不需要将模型从编码器传输到解码器。然而，由于必须持续更新压缩模型，这种方法需要更复杂的解码流程。因此，解码通常比半静态方法要慢一点。

压缩模型$\mathcal{M}$中的概率不一定是无条件的。例如，选择这样一个模型是很常见的，这个模型根据前 1、2、3…个已编码符号来决定符号σ的概率（在莎士比亚文集中已经看到这种情况，“q” 的出现已经足以知道下一个字符是 “u”)。这样的模型称为是**有限上下文**（finite-context）模型，或是**一阶**（first-order）模型、**二阶**（second-order）模型、**三阶**（third-order）模型、…。符号的概率独立于前面出现字符的模型$\mathcal{M}$称为**零阶**（zero-order）模型。

压缩模型和码之间是紧密联系的。每个压缩模型$\mathcal{M}$都有一个与之对应的码（或一组码)：码最小化了由$\mathcal{M}$产生的符号串的平均码字长度。相反，每个码也都有对应的概率分布：使码最优的分布。例如，考虑零阶压缩模型$\mathcal{M}_0$：

$$\mathcal{M}_0(\text{“a”}) = 0.5,\ \mathcal{M}_0(\text{“b”}) = 0.25,\ \mathcal{M}_0(\text{“c”}) = 0.125,\ \mathcal{M}_0(\text{“d”}) = 0.125 \tag{6-2}$$

一个对于模型$\mathcal{M}_0$是最优的码字$\mathcal{C}_0$有以下性质：

$$|\mathcal{C}_0(\text{“a”})| = 1,\ |\mathcal{C}_0(\text{“b”})| = 2,\ |\mathcal{C}_0(\text{“c”})| = 3,\ |\mathcal{C}_0(\text{“d”})| = 3 \tag{6-3}$$

（其中$|\mathcal{C}_0(X)|$代表$\mathcal{C}_0(X)$的位长度)。下面这个码满足这个要求[⊖]：

$$\mathcal{C}_0(\text{“a”}) = \overline{0},\ \mathcal{C}_0(\text{“b”}) = \overline{11},\ \mathcal{C}_0(\text{“c”}) = \overline{100},\ \mathcal{C}_0(\text{“d”}) = \overline{101} \tag{6-4}$$

它将符号串 “aababacd” 编码为

⊖ 为了避免混淆数字 “0” 和 “1” 与对应的比特值，除非文中明确指出是后一种意思，我们定义比特值为$\overline{0}$和$\overline{1}$。

$$\mathcal{C}_0(\text{“aababacd”}) = \overline{00110110100101} \tag{6-5}$$

码$\mathcal{C}_0$一个很重要的特性是它**无前缀**（prefix-free），也就是，不存在某个码字$\mathcal{C}_0(X)$是另外一个码字$\mathcal{C}_0(y)$的前缀。一个无前缀码也被称为**前缀码**（prefix code）。不是无前缀的码通常不能用于压缩（也有例外；见练习 6.3）。例如，考虑另外一个码$\mathcal{C}_1$：

$$\mathcal{C}_1(\text{“a”}) = \overline{1},\ \mathcal{C}_1(\text{“b”}) = \overline{01},\ \mathcal{C}_1(\text{“c”}) = \overline{101},\ \mathcal{C}_1(\text{“d”}) = \overline{010} \tag{6-6}$$

根据这些码字的长度，这个码看上去也是$\mathcal{M}_0$的一个最优码。但是，符号串“aababacd”现在被编码为：

$$\mathcal{C}_1(\text{“aababacd”}) = \overline{11011011101010} \tag{6-7}$$

当解码器看到这个符号串时，它不知道原始符号串是“aababacd”还是“accaabd”。因此这个码含义不清晰，不能用于压缩。

前缀码$\mathcal{C}_0$可以看做是一棵二叉树，其中每一个叶子节点对应一个符号σ。从树根到叶子的路径上与相关的标签就定义了符号的码字：$\mathcal{C}(\sigma)$。叶子的深度等于对应符号码字的长度。

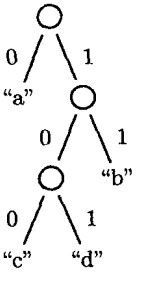


图 6-1 前缀码$\mathcal{C}_0$对应的二叉树。码字分别为：$\mathcal{C}_0(\text{“a”}) = \overline{0}$, $\mathcal{C}_0(\text{“b”}) = \overline{11}$, $\mathcal{C}_0(\text{“c”}) = \overline{100}$, $\mathcal{C}_0(\text{“d”}) = \overline{101}$

图 6-1 是码$\mathcal{C}_0$对应的码树。压缩算法的解码过程按照树边将位串翻译回原始符号串，一旦到达叶子——输出一个符号——跳回根节点，实质上就是用树作为二元决策图。二元码的树形式也说明了前缀性质为何这么重要：不是无前缀码的码树中，某些码字会被赋予中间节点；当解码器到达这个节点时，它不知道是输出对应的符号并跳回树根还是继续按树边遍历直至叶子节点。

现在考虑符号集$\{\sigma_1, \cdots, \sigma_n\}$上的压缩模型$\mathcal{M}$，每一个符号的概率都是 2 的幂的倒数：

$$\mathcal{M}(\sigma_i) = 2^{-\lambda_i};\ \ \lambda_i \in \mathbb{N}\ \text{ for } 1 \leqslant i \leqslant n \tag{6-8}$$

由于$\mathcal{M}$是一个概率分布，因此有

$$\sum_{i=1}^{n} \mathcal{M}(\sigma_i) = \sum_{i=1}^{n} 2^{-\lambda_i} = 1 \tag{6-9}$$

让我们为这个分布找到一个最优的码树。树中的每个节点必须或者是一个叶子节点（带一个码字）或者是一个只有两个孩子的中间节点。这样的树称为是一棵**满二叉树**（proper binary tree）。如果树中某个中间节点只有一个孩子，那么就可以通过移除中间节点来改进码，其后代的深度就可以减 1。于是这个码也不是最优的了。

对于满二叉树的叶子节点的集合$\mathcal{L} = \{L_1, \cdots, L_n\}$，以下公式成立：

$$\sum_{i=1}^{n} 2^{-d(L_i)} = 1 \tag{6-10}$$

其中$d(L_i)$是节点L_i的深度，也是分配给该节点对应的符号的码字长度。由于公式（6-9）和公式（6-10）的相似性，很自然地会以如下方式将码字分配给符号：

$$|\mathcal{C}(\sigma_i)| = d(L_i) = \lambda_i = -\log_2(\mathcal{M}(\sigma_i))\ \ 1 \leqslant i \leqslant n \tag{6-11}$$

得到的树代表了给定概率分布$\mathcal{M}$上的一个最优码。为什么呢？如果根据$\mathcal{M}$对符号串编码，考虑用$\mathcal{C}$之后每个符号的平均位长为：

$$\sum_{i=1}^{n} \Pr[\sigma_i] \cdot |\mathcal{C}(\sigma_i)| = -\sum_{i=1}^{n} \mathcal{M}(\sigma_i) \cdot \log_2(\mathcal{M}(\sigma_i)) \tag{6-12}$$

因为$|\mathcal{C}(\sigma_i)| = -\log_2(\mathcal{M}(S_i))$。根据以下定理，这是可达到的最优长度了。

信源编码定理（香农，1948）

给定一个符号源S，根据概率分布p_s输出字母集$\mathcal{S}$中的符号，得到的符号串压缩后平均每个字符不会少于

$$\mathcal{H}(S) = -\sum_{\sigma \in \mathcal{S}} p_S(\sigma) \cdot \log_2(p_S(\sigma))$$

位。$\mathcal{H}(\mathcal{S})$称为这个符号源$\mathcal{S}$的**熵**（entropy）。

将香农定理运用到模型$\mathcal{M}$定义的概率分布上，我们可以看到选定的码实际上对于给定模型而言是最优的。因此，如果最初假设所有概率都是2的幂的倒数（见公式（6-8））是成立的，我们就能知道如何快速找到根据模型$\mathcal{M}$产生的符号串的最优编码。当然实际上这很少见。概率可以是区间[0，1]之间的任意值。为给定的概率分布$\mathcal{M}$找到一个最优前缀码$\mathcal{C}$要比当$\mathcal{M}(\sigma_i) = 2^{-\lambda_i}$时困难一些。

6.2.2 哈夫曼编码

最广泛使用的按位编码技术之一由哈夫曼（1952）提出。对于一个有限符号集$\{\sigma_1, \cdots, \sigma_n\}$上给定的概率分布$\mathcal{M}$，哈夫曼方法产生一个前缀码$\mathcal{C}$，使下式最小化

$$\sum_{i=1}^{n} \mathcal{M}(\sigma_i) \cdot |\mathcal{C}(\sigma_i)| \tag{6-13}$$

在每个符号使用整数个比特位来表示的码中，哈夫曼编码被证明是最优的。如果放宽限制，允许码字用分数个比特位来表示，就有其他的方法，如算术编码（6.2.3节），能达到更好的压缩效果。

假定有一个压缩模型$\mathcal{M}$，满足$\mathcal{M}(\sigma_i) = \Pr[\sigma_i] (1 \leqslant i \leqslant n)$。哈夫曼方法从两个具有最小概率的符号开始，以自底向上的方式为这个模型构建一个最优码树。算法可以被视为在一组树上进行操作，每一棵树都被赋予了一个概率质量。刚开始每个符号σ_i只有一棵树T_i，满足$\Pr[T_i] = \Pr[\sigma_i]$。算法每执行一步，具有最小概率质量的两棵树T_j和T_k就被合并为一棵新树T_l。这棵新树的概率质量为$\Pr[T_l] = \Pr[T_j] + \Pr[T_k]$。重复这个步骤直到只剩一棵树$T_{\text{Huff}}$，满足$\Pr[T_{\text{Huff}}] = 1$。

图6-2展示了在符号集$\mathcal{S} = \{\sigma_1, \sigma_2, \sigma, \sigma_4, \sigma_5\}$上算法执行的每一步，这个符号集具有如下概率分布：

$$\Pr[\sigma_1] = 0.18,\ \Pr[\sigma_2] = 0.11,\ \Pr[\sigma_3] = 0.31,\ \Pr[\sigma_4] = 0.34,\ \Pr[\sigma_5] = 0.06 \tag{6-14}$$

这棵树生成之后，按照自顶向下的方式遍历这棵码树为符号分配码字。例如，σ_3的码字为$\overline{01}$，σ_2的码字为$\overline{110}$。

最优性

为什么这种方法产生的码是最优的？首先，注意到一个最优前缀码$\mathcal{C}_{opt}$必须满足下面的条件：

$$\Pr[x] < \Pr[y] \Rightarrow |\mathcal{C}_{opt}(x)| \geqslant |\mathcal{C}_{opt}(y)| \quad \text{对于每一对符号}(x, y) \tag{6-15}$$

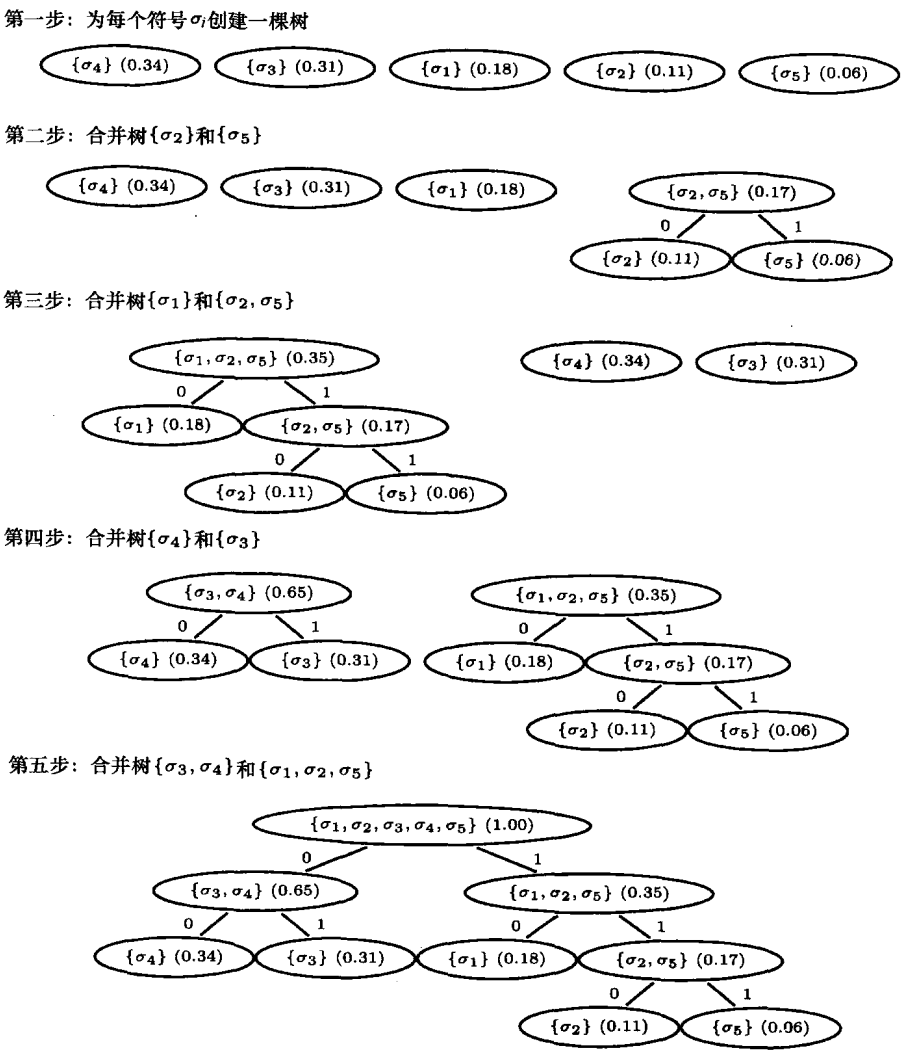


图 6-2 符号集$\{\sigma_1,\sigma_2,\sigma_3,\sigma_4,\sigma_5\}$对应的概率分布为：$\Pr[\sigma_1]=0.18$，$\Pr[\sigma_2]=0.11$，$\Pr[\sigma_3]=0.31$，$\Pr[\sigma_4]=0.34$，$\Pr[\sigma_5]=0.06$ 为这个符号集建立哈夫曼编码树

（否则，我们可以简单交换 x 和 y 的码字，来获得更好的码）。并且，由于最优前缀码总是被表示成为一棵**满二叉树**，因此两个最不相同的码字总是具有相同的长度 d（它们的节点应该是兄弟，位于二叉树$\mathcal{C}_{\text{opt}}$的最底层；如果它们不是兄弟，我们就重排最底层的叶子使得它们可以变成兄弟）。

通过对构建前缀码的符号数目 n 做归纳，我们可以证明哈夫曼算法的最优性。当 $n=1$ 时，算法产生一个高度为 0 的编码树，显然是最优的。对于一般情况，考虑符号集$\mathcal{S}=\{\sigma_1,\cdots,\sigma_n\}$。让 σ_j 和σ_k 分别代表具有最小概率的两个符号（$j<k$ w.l.o.g.）。它们的码字长度都为 d。因此，对于由$\mathcal{S}$中符号组成的符号串，码字的期望长度（比特位/符号）为：

$$\mathrm{E}[\mathrm{Huff}(\mathcal{S})]=d\cdot(\Pr[\sigma_j]+\Pr[\sigma_k])+\sum_{x\in(\mathcal{S}\setminus\{\sigma_j,\sigma_k\})}\Pr[x]\cdot|\mathcal{C}(x)| \tag{6-16}$$

考虑符号集：

$$\mathcal{S}'=\{\sigma_1,\cdots,\sigma_{j-1},\sigma_{j+1},\cdots,\sigma_{k-1},\sigma_{k+1},\cdots,\sigma_n,\sigma'\} \tag{6-17}$$

删除符号 σ_j 和 σ_k，并用新符号 σ'来代替它们，可得下面的符号集：

$$\Pr[\sigma'] = \Pr[\sigma_j] + \Pr[\sigma_k] \tag{6-18}$$

因为在$\mathcal{S}$的哈夫曼编码树中，σ'是σ_j 和σ_k 的父节点，所以在$\mathcal{S}'$树中，σ'的深度为$d-1$。对于由$\mathcal{S}'$中符号组成的符号串，码字的期望长度为：

$$\mathrm{E}[\mathrm{Huff}(\mathcal{S}')] = (d-1)\cdot(\Pr[\sigma_j]+\Pr[\sigma_k]) + \sum_{x\in(\mathcal{S}'\setminus\{\sigma'\})} \Pr[x]\cdot|\mathcal{C}(x)| \tag{6-19}$$

也就是，E[Huff($\mathcal{S}'$)]=E[Huff($\mathcal{S}$)]−Pr[σ_j]−Pr[σ_k]。

通过归纳我们知道，$\mathcal{S}'$的哈夫曼树是一个最优前缀码（因为$\mathcal{S}'$中包含 $n-1$ 个元素）。现在假设$\mathcal{S}$的哈夫曼树不是最优的。那么一定存在着另外一个最优编码树，使得期望代价是：

$$\mathrm{E}[\mathrm{Huff}(\mathcal{S})] - \varepsilon\ (\text{部分}\ \varepsilon > 0) \tag{6-20}$$

如前所述，对这棵树中的节点 σ_j 和 σ_k 做分裂，我们可以得到$\mathcal{S}'$的一个新码树，期望代价为：

$$\mathrm{E}[\mathrm{Huff}(\mathcal{S})] - \varepsilon - \Pr[\sigma_j] - \Pr[\sigma_k] \;=\; \mathrm{E}[\mathrm{Huff}(\mathcal{S}')] - \varepsilon \tag{6-21}$$

（这总是成立的，因为 σ_j 和σ_k 在最优编码树中是兄弟，前面也解释过）。然而，这与$\mathcal{S}'$的哈夫曼树是最优的这一前提矛盾。因此代价小于 E[Huff($\mathcal{S}$)]的$\mathcal{S}$的前缀码不存在。$\mathcal{S}$的哈夫曼编码树必须是最优的。

1. 复杂度

哈夫曼算法的第二个阶段通过遍历第一个阶段建立的编码树为每一个符号分配码字。这个过程能够在 $\Theta(n)$时间内完成，因为树中只有 $2n-1$ 个节点。第一个阶段建树的过程稍微复杂些。它需要我们不断跟踪具有最小概率质量的两棵树。这可以通过维持一个优先队列（如最小堆）来实现，使得队头总是保存那棵具有最小概率的树。这样的数据结构支持在 $\Theta(\log(n))$时间内的 INSERT 和 EXTRACT-MIN 操作。因为哈夫曼算法总共需要执行 $2(n-1)$次 EXTRACT-MIN 操作和 $n-1$ 次 INSERT 操作，所以第一阶段的时间复杂度为 $\Theta(n\log(n))$。因此，这个算法的总的时间复杂度也为 $\Theta(n\log(n))$。

2. 范式哈夫曼码

当使用哈夫曼编码进行数据压缩时，需要在压缩后的符号串前面添加实际码的描述信息，使得解码器能够将编码后的位串还原成原始的符号串。这个描述信息被称为**前言**（preamble）。它通常是所有符号及其码字组成的列表。对于图 6-2 的哈夫曼码，前言如下：

$$\langle(\sigma_1,\overline{10}),(\sigma_2,\overline{110}),(\sigma_3,\overline{01}),(\sigma_4,\overline{00}),(\sigma_5,\overline{111})\rangle \tag{6-22}$$

用这样的方式描述哈夫曼码是非常耗费存储空间的，特别是当实际被压缩的消息相对较短时。

幸运的是，实际上不需要包含对实际码的详尽描述，只描述某些方面就够了。再考虑图 6-2 构建的哈夫曼树。图 6-3 给出了一棵与之等价的哈夫曼树。新树中每个符号 σ_i 对应的码字长度与旧树的一样。因此，没有理由能够说明图 6-2 的码比图 6-3 的码更优越。

从左到右读图 6-3 中的符号 σ_i，并按照它们对应的码字长度排序（码字短的排在前面）；符号的自然序（如 σ_1 排在 σ_3 前面）就被打乱了。具有这种特性的码叫做**范式哈夫曼码**（canonical Huffman code）。对于每一个可能的哈夫曼码$\mathcal{C}$，都有一个与之等价的范式哈夫曼码$\mathcal{C}_{can}$。

当描述一个范式哈夫曼码时，列出所有符号码字的位长就够了，不需要描述实际的码字。例如，图 6-3 中的树可以描述为

$$\langle(\sigma_1,2),(\sigma_2,3),(\sigma_3,2),(\sigma_4,2),(\sigma_5,3)\rangle \tag{6-23}$$

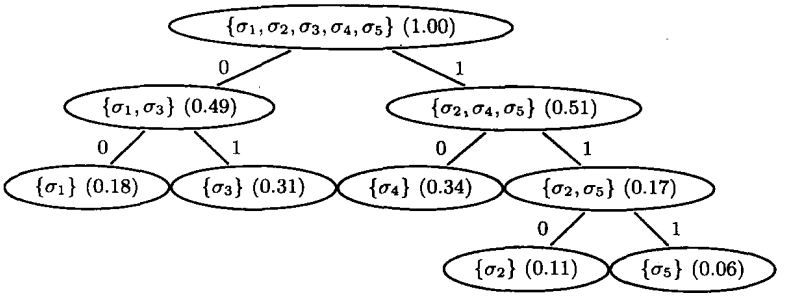


图 6-3 图 6-2 中的哈夫曼码的范式版本。树中同一层的符号按照字母顺序排序

同样地，图 6-1 对应的范式哈夫曼码为

$$\langle(\text{“a”},\overline{0}),(\text{“b”},\overline{10}),(\text{“c”},\overline{110}),(\text{“d”},\overline{111})\rangle \tag{6-24}$$

它可以描述为

$$\langle(\text{“a”},1),(\text{“b”},2),(\text{“c”},3),(\text{“d”},3)\rangle \tag{6-25}$$

如果解码器事先知道符号集S，那么这个范式哈夫曼码可以表示为⟨1，2，3，3⟩，这种方式会比任意哈夫曼码的描述形式要紧凑两倍。

3. 限长的哈夫曼码

有的时候对给定的哈夫曼码中的码字长度限定一个上界是很有用的。这主要是由于性能方面的原因，当码字不会太长的时候，解码操作才会更有效（我们将在 6.3.6 节中详细介绍）。

我们知道，对于具有 n 个符号的字母表，我们总是能找到一个前缀码，它的每个码字不超过$\lceil\log_2(n)\rceil$位。给定码字长度的一个上界 $L(L\geqslant\lceil\log_2(n)\rceil)$，存在有这样的算法能够产生所有码字长度不超过 L 的前缀码中的最优前缀码C_L。大多数这样的算法首先构造一个普通的哈夫曼码C，然后在表示C的二叉树中做一些变换来计算C_L。从技术上来说，这样的码不再是哈夫曼码，因为它已不是全局最优（只在所有限长的码中是最优的）。但是实际上，结果码中额外的冗余是可以忽略的。

构造限长前缀码最著名的方法之一是 Larmore 和 Hirschberg（1990）提出的 PACKAGE-MERGE 算法。它能够在 $O(nL)$时间内产生一个最优的限长前缀码，其中 n 是字母表中的符号数。因为 L 通常是比较小的（毕竟，这也是整个操作的目的），所以，PACKAGE-MERGE 算法并没有给基于哈夫曼的压缩算法的编码部分增加不合理的复杂度。并且，正如普通的哈夫曼码一样，我们总是能为任何给定限长码找到一个等价的范式码，从而可以进行如前一样的优化。

6.2.3 算术编码

哈夫曼编码的主要限制在于它不能正确处理符号分布中概率接近 1 的那些符号。例如，考虑以下两个符号的字母表$S=\{\text{“a”},\text{“b”}\}$的概率分布：

$$\Pr[\text{“a”}]=0.8,\ \Pr[\text{“b”}]=0.2 \tag{6-26}$$

由香农定理可知，根据这个概率分布产生的符号串编码之后每个符号平均所需的位数不会少于：

$$-\Pr[\text{“a”}]\cdot\log_2(\Pr[\text{“a”}])-\Pr[\text{“b”}]\cdot\log_2(\Pr[\text{“b”}])\approx 0.2575+0.4644=0.7219 \tag{6-27}$$

然而，使用哈夫曼编码，最好的情况是每一个符号占用1位，因为每一个码字所用的位数必须为整数。因此，与香农定理可获得的下界相比，哈夫曼的方法需要的存储容量增加了39%。

为了改进哈夫曼方法的性能，我们必须摒弃为每个符号单独分配一个码字的想法。例如，我们可以将多个符号组合成 m-元组，然后给每个唯一的 m-元组分配一个码字（这种技术也被称为**分块**（blocking））。对于上面的例子，选择 $m=2$，将得到下面的概率分布：

$$\Pr[\text{“aa”}]=0.64,\ \Pr[\text{“ab”}]=\Pr[\text{“ba”}]=0.16,\ \Pr[\text{“bb”}]=0.04 \tag{6-28}$$

对于这个分布，在哈夫曼码中，每一个2-元组将需要1.56位，或者说每一个符号平均需要0.78位（见练习6.1）。然而，有的时候将符号分块有些麻烦并且会增加前言（哈夫曼码的描述信息）的大小，前言也还是需要从编码器传输到解码器的。

算术编码（Arithmetic coding）用一种更简洁的方式解决了这个问题。考虑符号集$\mathcal{S}=\{\sigma_1,\cdots,\sigma_n\}$中的 k 个符号组成的串：

$$\langle s_1,s_2,\cdots,s_k\rangle \in \mathcal{S}^k \tag{6-29}$$

每一个这样的串都对应一个概率。例如，如果固定 k 值大小，那么上述符号串出现的概率为：

$$\Pr[\langle s_1,s_2,\cdots,s_k\rangle]=\prod_{i=1}^{k}\Pr[s_i] \tag{6-30}$$

显然地，具有相同长度 k 的所有符号串出现的概率总和为1。因此，我们可以把每一个这样的符号串 x 视为一个区间$[x_1,\ x_2)$，且$0\leqslant x_1\leqslant x_2\leqslant 1$，$x_2-x_1\Pr[x]$。这样的区间可以看做是[0，1)的子区间，按照对应符号串的字母顺序，从而可以得到[0，1)的一个划分。

重新考虑公式（6-26）中的分布。使用区间表示方法，符号串“aa”对应[0，0.64)；符号串“ab”对应[0.64，0.80)；符号串“ba”对应[0.80，0.96)；最后“bb”对应[0.96，1.0)。这个方法可以一般化到任意长度的符号串上，如图6-4所示。

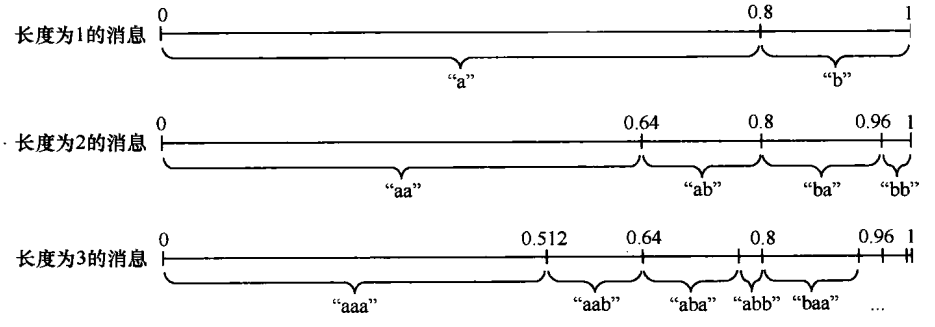


图6-4 算术编码：将符号串转换成在[0，1)区间上的子区间。消息 M（符号串）被编码为一个与 M 对应的二元子区间。本例中，概率分布为：Pr[“a”]=0.8，Pr[“b”]=0.2

有了消息和区间之间的映射，给定消息就可编码为相应区间$\mathcal{I}$，而不再是消息本身。可能直接对$\mathcal{I}$进行编码有些困难，因此，算术编码对区间$\mathcal{I}$中一个更小的区间$\mathcal{I}'$进行编码（即$\mathcal{I}'\subseteq\mathcal{I}$），具有如下特殊形式：

$$\mathcal{I}'=[x,x+2^{-q}),\ 且\ x=\sum_{i=1}^{q}a_i\cdot 2^{-i}\ (a_i\in\{0,1\}) \tag{6-31}$$

我们将其称为**二元区间**（binary interval）。二元区间可以用很直接的方法编码成位串$\langle a_1, a_2, \cdots, a_q\rangle$。例如，用位串$\overline{0}$来表示区间[0，0.5)；用位串$\overline{01}$来表示区间[0.25，5)；用位串$\overline{010}$来表示区间[0.25，0.375)。

以下两个步骤的结合就称为**算术编码**（arithmetic coding）：1）将一个消息转换成等价区间$\mathcal{I}$；2）将$\mathcal{I}$中的二元区间编码为简单位串。当收到消息“aab”时，算术编码器将找到对应的区间$\mathcal{I}=[0.512, 0.64)$（见图 6-4），接着在这个区间中确定二元子区间：

$$\mathcal{I}'=[0.5625,0.625)=[x,x+2^{-4}) \tag{6-32}$$

其中$x=1.2^{-1}+0.2^{-2}+0.2^{-3}+1.2^{-4}$。因此，消息“aab”被编码为$\overline{1001}$。

对于同样地概率分布，看起来占用 4 位的$\overline{1001}$反而比 3 位的哈夫曼码还要多出 1 位。但对于消息“aaa”，哈夫曼码同样需要 3 位，但算术编码仅需要 1 位（$\overline{0}$）。一般来说，发生概率为 p 的消息对应区间$\mathcal{I}=[y, y+p)$，我们需要找到一个二元区间$[x, x+2^{-q})$，满足：

$$y\leqslant x<x+2^{-q}\leqslant y+p \tag{6-33}$$

能够证明，对于所有满足$2^{-q}\leqslant p/2$的 q，总能找到一个这样的二元区间。满足这个条件的最小的 q 为：

$$q=\lceil-\log_2(p)\rceil+1 \tag{6-34}$$

所以，为了编码具有概率$p=\prod_{i=1}^{k}\Pr[s_i]$的消息 M：

$$M=\langle s_1,s_2,\cdots,s_k\rangle \tag{6-35}$$

我们需要的位数不超过$[-\log_2(p)]+1$。对于仅包含少数几个符号的消息，“+1”这一项可能会使算术编码的空间效率不如哈夫曼编码。但是，当$k\to\infty$时，根据香农信源编码定理（见 6.2.2 节），算术编码的每个符号的平均位数接近理论最优。因此，如果压缩模型$\mathcal{M}$能够准确地描述待压缩符号串的统计特性，算术编码近似最优。

算术编码有两个有利的性质：

- 在输入符号串的不同位置可以使用不同的模型（取决于解码器已访问过的信息的限制），从而使得算术编码是可选的自适应压缩方法之一。
- 解码器事先不需要知道消息的长度，而是将一个消息结束的符号加入到字母表中，并与字母表中的其他符号一样等同对待（特别的，在压缩模型中不断更新它的概率，使得当消息越来越长的时候它所需的概率质量越来越小）。

本章中我们不会详细地介绍算术编码细节的内容。如果对细节感兴趣，你可以在任何一本数据压缩的教科书中（见 6.6 节）找到这些细节。

1. 实现

从我们对算术编码的描述看来，编码和解码的过程需要浮点数操作，并且在实现中容易受到四舍五入错误和其他一些问题的影响。但是不会有这样的情况。例如，Witten 等人（1987）介绍了如何使用整数算术来实现算术编码，代价是冗余稍微会增加（每符号比特位增加不超过10^{-4}位）。

2. 哈夫曼编码与算术编码

从每个符号所需的位数来看，算术编码与哈夫曼编码相比，所具有的优势取决于概率分

布$\mathcal{M}$。明显地，当所有符号概率都是2的幂的倒数时，那么哈夫曼编码就是最优的，算术编码没有任何优势可言。然而，对应任意一个模型$\mathcal{M}$，与最优的算术编码相比，很容易就能看出哈夫曼编码具有冗余（即每一个符号编码浪费的位数），它比最优的算术编码多出1位。这是符合事实的，因为我们能够容易地构造一个满足码字长度$|\mathcal{C}(\sigma_i)| = \lceil -\log_2(p_i) \rceil$的前缀码。这个码的每个符号的冗余度不超过1位，又因为它是一个前缀码，它不可能比$\mathcal{M}$的哈夫曼码要好。

一般地，哈夫曼码的冗余度依赖于符号集的概率分布的特征。例如，Gallager（1978）证明了对于一个给定的压缩模型$\mathcal{M}$，哈夫曼码的冗余度总是小于$\mathcal{M}(\sigma_{\max}) + 0.0861$，其中$\sigma_{\max}$是最可能出现的符号。Horibe（1977）证明了冗余度至多为$1 - 2 \cdot \mathcal{M}(\sigma_{\min})$，其中$\sigma_{\min}$指的是最不可能出现的符号。

由于哈夫曼编码达到的压缩率通常已经非常接近理论最优值，又因为哈夫曼码的解码操作实现起来比算术编码更高效，所以哈夫曼编码通常被认为比算术编码更优。在索引压缩的应用中更是如此，因为开发一个搜索引擎时，通常主要考虑的指标是查询处理的性能而不是倒排索引的大小。限长的范式哈夫曼码解码时效率非常高，比算术码要快很多。

6.2.4 基于符号的文本压缩

刚才介绍的通用的基于符号的数据压缩可直接用于文本压缩问题。例如，如果想减少一个给定文档集的存储容量，你可以写一个半静态的基于哈夫曼压缩算法的算法，使用两个步骤来压缩文档。第一步：收集文档集中所有字符出现的频率，得到一个零阶压缩模型。然后根据这个模型构造一个哈夫曼码$\mathcal{C}$。第二步：将每一个字符σ用它对应的码字$\mathcal{C}(\sigma)$代替。

如果你实现了这个算法并在任意英文文本上使用，你将发现这个文本的压缩版本每个字符都需要超过5位来表示。这与英文文档的零阶熵大概是每字符5位的观点是一致的。表6-2列出了在三个样例数据集上的具体实验结果。

表6-2 三个文档集上的文本压缩率（每字符所占位数）。其中哈夫曼编码和算术编码中没有计入压缩模型（例如，哈夫曼树）的大小，而压缩模型也是需要发送到解码器端的

文档集	哈夫曼编码			算术编码			其他	
	0阶	1阶	2阶	0阶	1阶	2阶	gaip	bzip2
莎士比亚文集	5.220	2.852	2.270	5.190	2.686	1.937	2.115	1.493
TREC45	5.138	3.725	2.933	5.105	3.676	2.813	2.448	1.812
GOV2	5.413	3.948	2.832	5.381	3.901	2.681	1.502	1.107

你可用一阶压缩模型代替零阶压缩模型来优化算法。这需要构建和保存（发送）多达256棵哈夫曼树，每一棵对应一个可能的上下文内容（即未压缩文本中前一个字节的内容），但却能提高压缩效果，使得每个字符所占的空间减少到不足4位。这个方法还能够使用二阶模型、三阶模型……来扩展，得到越来越好的压缩率。这表明英文文本中的相邻字符存在很强的相依性（例如，莎士比亚文集中每个“q”后都跟着“u”），通过利用这种依赖关系，可以得到比每字符5位好得多的压缩率。

考虑到哈夫曼编码和算术编码的相对性能，这个表说明算术编码并不比哈夫曼编码好多少。对于零阶模型，三个文档集的差异每字符不到0.04位。对于更高阶的模型，差距就变大了（二阶模型为每字符0.12～0.33位），因为在高阶模型中字符的概率分布更具有倾斜

性，因此更有利于算术编码。

但是，请注意表 6-2 中的数据有轻微的误导性，因为没有考虑压缩模型（哈夫曼树）所占的空间。例如，对于三阶压缩模型，在对实际文本数据进行编码之前，需要传输 $256^3=1680$ 万棵哈夫曼树。对于像 GOV2 这样的大文档集来说，这么做还值得。但是，对于像莎士比亚文集这样的小文档集来说就不是这样了。因此，实际的压缩算法在处理有限个上下文模型和大的字母表时，都使用自适应的方法。这样的例子有 PPM（部分匹配预测模型；Cleary 和 Witten，1984）以及 DMC（动态马尔可夫压缩方法；Cormack 和 Horspool，1987）。

自适应方法最开始使用一个初始模型（可能是零阶模型），然后基于目前已有数据的统计信息逐步修正压缩模型 $\mathcal{M}$。实际上，这样的方法会得到相当不错的结果，接近于半静态方法，但压缩模型不需要传送给解码器（因此比半静态方法更高效）。

最后，表中将相对简单的基于哈夫曼编码或算术编码的压缩技术与更加复杂的技术如 Ziv-Lempel（Ziv 和 Lempel，1977；也称为 gzip[⊖]）以及 Burrows-Wheeler（Burrows 和 Wheeler，1994；**bzip2**[⊜]）作比较。从本质上来说，这些方法仍然依赖于哈夫曼或算术编码，但是在进入编码阶段之前会做一些额外的工作。从表中可以看出，这些额外工作都带来了显著的压缩效果。例如，bzip2（有参数的—best）能够将这三个文档集中的文本压缩到每字符 1.1～1.8 位，与最原始的每字符使用 8 位编码方法相比，提升了 80%。这个结果与一般的假设——英文文档的熵在每字符 1～1.5 位之间（这个范围是在半个多世纪之前确定的（Shannon，1951））——是一致的。

6.3 压缩位置信息列表

已经讨论了一些通用数据压缩的原理后，我们可用这些原理解决倒排索引压缩的问题，使得能够大大降低存储的需求。正如第 4 章提到的（表 4.1），倒排索引中的数据绝大部分都是位置信息数据。因此，如果想减少索引整体的大小，在处理索引其他组成部分（如词典）之前，应主要考虑位置信息列表的压缩。

具体使用什么样的方法来压缩给定索引中的位置信息取决于索引的类型（文档编号索引、频率索引、位置索引，还是模式独立索引），但是不同类型的索引之间也有共同点，这使得我们可以使用一些通用的方法来处理它们。

考虑下面这个整数序列，它可能是文档编号索引中某个词项的位置信息列表的开头部分：

$$L = \langle 3, 7, 11, 23, 29, 37, 41, \cdots \rangle$$

使用标准的压缩方法，如哈夫曼编码来直接压缩是行不通的；列表中元素非常多，并且每个元素仅出现一次。然而，由于 L 中的元素形成了一个单调递增的序列，这个列表可以转换成一个等价的序列：相邻两个元素的差序列，称为 Δ-value：

$$\Delta(L) = \langle 3, 4, 4, 12, 6, 8, 4, \cdots \rangle$$

这个新列表 $\Delta(L)$ 与原始列表 L 相比，有两大优势。首先，$\Delta(L)$ 中的元素比 L 中的要小，这意味着可以使用更少的位来编码。其次，$\Delta(L)$ 中有些元素多次出现，意味着可以根据 Δ-value 在列表中出现的频率来给它们分配码字，从而进一步节约存储空间。

⊖ gzip（www.gzip.org）在初期发展的时候，由于 Ziv_Lempel 的使用受专利的限制，所以 gzip 实际上不是基于 Ziv_Lempel 压缩方法，而是基于另一个与 Ziv_Lempel 略有不同的 DEFLATE 算法。

⊜ bzip2（www.bzip.org）是可免费获得，基于 Burrows-Wheeler 转换的无专利限制的数据压缩软件。

以上转换显然适用于文档编号索引和模式独立索引，但它同样适用于基于文档的位置索引中的位置信息，形式为$(d, f_{t,d}\langle p_1, \cdots, p_{f_{t,d}}\rangle)$。例如，下面的列表：

$$L = \langle (3,2,\langle 157,311\rangle),(7,1,\langle 212\rangle),(11,3,\langle 17,38,133\rangle),\cdots\rangle$$

可以转换成等价的 Δ-list

$$\Delta(L) = \langle (3,2,\langle 157,154\rangle),(4,1,\langle 212\rangle),(4,3,\langle 17,21,95\rangle),\cdots\rangle$$

也就是，每一个文档编号用它与前一个文档编号的间距来表示；每一个文档内偏移位置用它与前一个文档内偏移位置的间距来表示；频率值不变。

因为 3 个结果列表中值的概率分布相差很大（比如，频率值通常要比文档内偏移位置小很多），所以通常会使用 3 个不同的压缩方法，对应转换之后的 Δ-list 的 3 个子列表。

倒排索引中的位置信息的压缩方法一般可分为两类：**参数**（parametric）码和**无参数**（nonparametric）码。当对列表中的位置信息进行编码时，无参数码通常不考虑给定位置信息列表实际的间距分布情况。相反，它假定所有的位置信息列表看上去都差不多一样，并且具有一些共同的性质——例如，小的间距比大的间距更常见。反过来说，在压缩之前，参数码会分析待压缩位置信息列表的统计性质。基于分析的结果，选择一个参数值，编码位置信息串的码字由这个参数决定。

根据 6.2.1 节介绍的一些术语，我们将无参数码称为静态压缩方法，参数码为半静态方法。由于更新压缩模型所带来的复杂性，自适应方法一般不用于压缩索引。

6.3.1 无参数间距压缩

最简单的正整数无参数码是**一元码**（unary code）。在这种码中，正整数 k 用 $k-1$ 个 $\bar{0}$ 后面再跟一个 $\bar{1}$ 的串来表示。当位置信息列表中 Δ-value 服从以下的几何分布时，一元码是最优的：

$$\Pr[\Delta = k] = \left(\frac{1}{2}\right)^k \tag{6-36}$$

也就是，长度为 $k+1$ 的间距出现的概率是长度为 k 的间距的一半（见 6.2.1 节，该节介绍了码和概率分布之间的关系）。对于倒排索引中的位置信息，这个概率分布很少出现，除最频繁词项外，如“the”和“and”，这些频繁词项很可能出现在每一个文档中。尽管如此，一元编码方法在索引压缩中起着重要作用，因为其他的一些压缩技术（如下面要介绍的 γ 编码）都依赖于它。

1. Elias 的 γ 编码

γ 编码是最早介绍对正整数进行编码且是比较重要的无参数编码方法之一，由 Elias (1975) 首先提出。在 γ 编码中，正整数 k 的码字包含两个组成部分。第二部分：**正文**（body），包含了 k 的二进制表示。第一部分：**选择器**（selector），包含了正文长度的一元表示。例如，整数 1、5、7 和 16 的码字如下：

k	selector(k)	body(k)
1	1	1
5	001	101
7	001	111
16	00001	10000

你可能注意到，在上面这个表格中每一个码字的正文都是以 $\bar{1}$ 开头。这不是巧合。如果整数 k 的选择器的值 selector(k)$=j$，那么可得 $2^{j-1} \leqslant k < 2^j$。因此，在这个数的二进制表示中，

第 j 个不显著位恰好是码字正文的第一位，一定是 $\overline{1}$。当然，这也意味着正文中的第一位实际上是多余的，可以删除它为每个位置信息节省一位。这样，上面 4 个整数的 γ 码字分别为：$\overline{1}(1)$，$\overline{001\ 01}(5)$，$\overline{001\ 11}(7)$，$\overline{00001\ 0000}(16)$。

正整数的二进制表示包含 $\lfloor \log_2(k) \rfloor + 1$ 位。因此，在 γ 编码中 k 的码字长度为：

$$|\gamma(k)| = 2 \cdot \lfloor \log_2(k) \rfloor + 1(\text{位}) \tag{6-37}$$

用间距分布的术语就是，这意味着 γ 编码对于符合以下概率分布的整数串来说是最优的：

$$\Pr[\Delta = k] \approx 2^{-2 \cdot \log_2(k)-1} = \frac{1}{2 \cdot k^2} \tag{6-38}$$

2. δ 编码和 ω 编码

当压缩列表中小间距（比如小于 32）占大多数时，γ 编码是比较合适的，但如果间距较大，就比较浪费了。对于这样的列表，γ 编码的一种变形——δ 编码会是一个更好的选择。δ 编码与 γ 编码非常类似。然而，δ 编码并不将给定整数的选择器部分保存为一元形式，而是用 γ 对选择器进行编码。因此，整数 1、5、7 和 16 的 δ 码字如下：

k	selector(k)	body(k)
1	1	
5	01 1	01
7	01 1	11
16	001 01	0000

（所有情况已忽略多余的 $\overline{1}$）。整数 16 的选择器部分为 $\overline{001\ 01}$，因为这个数的二进制表示需要 5 位，5 的 γ 码字是 $\overline{001\ 01}$。

编码中整数的码字长度近似为 $2 \cdot \log_2(k)$，与之相比，对于同一个整数，δ 编码仅需：

$$|\delta(k)| = \lfloor \log_2(k) \rfloor + 2 \cdot \lfloor \log_2(\lfloor \log_2(k) \rfloor + 1) \rfloor + 1\ (\text{位}) \tag{6-39}$$

其中第一部分来自于码字的正文，第二部分是选择器 γ 编码的长度。如果位置信息列表中的间距满足以下概率分布，则 δ 编码是最优的：

$$\Pr[\Delta = k] \approx 2^{-\log_2(k)-2 \cdot \log_2(\log_2(k))-1} = \frac{1}{2k \cdot (\log_2(k))^2} \tag{6-40}$$

对于间距很大的列表，δ 编码的效率可以是 γ 编码的两倍。然而，实际的索引中这样大的间距是很少出现的。相反，与 γ 相比，δ 编码通常能节省 15%～35% 的空间。例如，整数 2^{10}、2^{20}、2^{30} 的 γ 码字长度分别为 21、41 和 61 位。而对应的 δ 码字长度分别为 17、29 和 39 位（−19%，−29%，−36%）。

δ 编码对给定码字的选择器使用 γ 编码的思想，也可以递归应用，产生了一种叫做 ω 编码的技术。正整数 k 的 ω 码构造步骤如下：

1）输出 $\overline{0}$。

2）如果 $k=1$，停止。

3）否则，将 k 编码为二进制形式（包括首位 $\overline{1}$），并将其扩展到已写入位的前面。

4）$k \leftarrow \lfloor \log_2(k) \rfloor$。

5）跳回到第 2）步。

例如，$k=16$ 的 ω 码为

$$\overline{10\ 100\ 10000\ 0} \tag{6-41}$$

因为 16 的二进制表示为 $\overline{10\ 000}$，4($=\lfloor \log_2(16) \rfloor$)的二进制表示为 $\overline{100}$，2($=\lfloor \log_2(\lfloor \log_2$

(16)」)」的二进制表示为$\overline{10}$。整数 k 的 ω 码字长度近似为：

$$|\omega(k)| = 2 + \log_2(k) + \log_2(\log_2(k)) + \log_2(\log_2(\log_2(k))) + \cdots \tag{6-42}$$

表 6-3 列出了几个整数的 γ 码字、δ 码字和 ω 码字。当整数 $n \geqslant 32$ 时，δ 编码比 γ 编码的空间效率更高。当整数 $n \geqslant 128$ 时，ω 编码比 γ 编码效率更高。

表 6-3 使用多种无参数码来编码正整数

整数	γ 编码	δ 编码	ω 编码
1	1	1	0
2	01 0	01 0 0	10 0
3	01 1	01 0 1	11 0
4	001 00	01 1 00	10 100 0
5	001 01	01 1 01	10 101 0
6	001 10	01 1 10	10 110 0
7	001 11	01 1 11	10 111 0
8	0001 000	001 00 000	11 1000 0
16	00001 0000	001 01 0000	10 100 10000 0
32	000001 00000	001 10 00000	10 101 100000 0
64	0000001 000000	001 11 000000	10 110 1000000 0
127	0000001 111111	001 11 111111	10 110 1111111 0
128	00000001 0000000	0001 000 0000000	10 111 10000000 0

6.3.2 参数间距压缩

无参数码的缺点在于没有将待压缩列表的特殊特征考虑进去。如果给定列表的间距满足码中暗含的概率分布，那么一切就很好办了。然而，如果实际间距的分布与暗含的那个是不一样的，那么无参数码就很浪费了，这时应用参数方法来代替。

参数压缩方法可以分为两类：**全局**（global）方法和**局部**（local）方法。全局方法对索引中的所有倒排列表都使用同一个参数值。局部方法为索引中的每一个列表都选择一个不同的参数值，或者甚至为给定列表中的每一个位置信息块选择不同的参数值，其中每一个块一般由几百或几千个位置信息组成。你可以回忆一下 4.3 节，每一个位置信息列表中包含一系列的同步点帮助我们能够随机访问这个位置信息列表。同步点很自然地可用于基于块的列表压缩（实际上，这也是同步点命名的来由）；每个同步点对应着一个新块的开始。对于一个异构的位置信息列表，列表中的不同部分具有不同的统计特征，基于块的压缩方法能够显著地提升整体压缩效果。

在大多数场合中，局部方法能够比全局方法获得更好的结果。然而，对于非常短的列表，选择局部方法有可能会使存储参数的开销比方法本身节省的空间还大，特别是如果参数好像一棵哈夫曼树一样复杂的时候。这种情况下，选择一个全局方法或对拥有同一性质（例如相似的平均间距大小）的所有列表都使用同样参数值的**批**（batched）方法，可能会更好。

因为选择的参数被认为是压缩模型的一个简要描述，所以局部和全局方法之间的关系是我们在 6.1 节中讨论的通用数据压缩技术中建模和编码之间关系的一个特例：何时停止建模以及何时开始编码？选择局部方法会得到一个更精确的模型，准确描述了给定列表中的间距描述。选择全局方法（或批方法）则得到一个没有那么精确的模型，但减小了模型（分摊的）规模，因为大量的列表使用同样的模型。

1. Golomb/Rice 编码

假设我们想压缩一个列表，它的 Δ-value 满足几何分布（geometric distribution），也就是，间距大小为 k 的概率为：

$$\Pr[\Delta = k] = (1-p)^{k-1} \cdot p \tag{6-43}$$

其中 p 是 0~1 之间的常数。在倒排索引中，这是一个实际的假设。考虑一个包含 N 个文档的文档集，词项 T 出现在其中 N_T 个文档中。随机地从文档集中抽选一个文档，找到词项 T 的概率为 N_T/N。因此，在这个假设下，所有文档都是彼此独立的，词项两次相邻出现的位置间距为 k 的概率为：

$$\Pr[\Delta = k] = \left(1 - \frac{N_T}{N}\right)^{k-1} \cdot \frac{N_T}{N} \tag{6-44}$$

也就是，当 T 出现一次后，接下来是 $k-1$ 个不包含 T 的文档（每一个的概率是 $1-\frac{N_T}{N}$），然后再是一个包含 T 的文档（概率是 $\frac{N_T}{N}$）。这样可以得到一个 $p=\frac{N_T}{N}$ 的几何分布。

图 6-5 展示了某一满足 $p=0.01$ 的词项的（几何）间距分布。图中也给出了当我们根据 Δ value 的位长 $\text{len}(k)=\lfloor \log_2(k) \rfloor + 1$ 把它们分到不同的桶中然后计算每个桶的概率，得到的概率分布情况。图中，有 65% 的间距值满足 $6 \le \text{len}(k) \le 8$。由此得到以下编码过程：

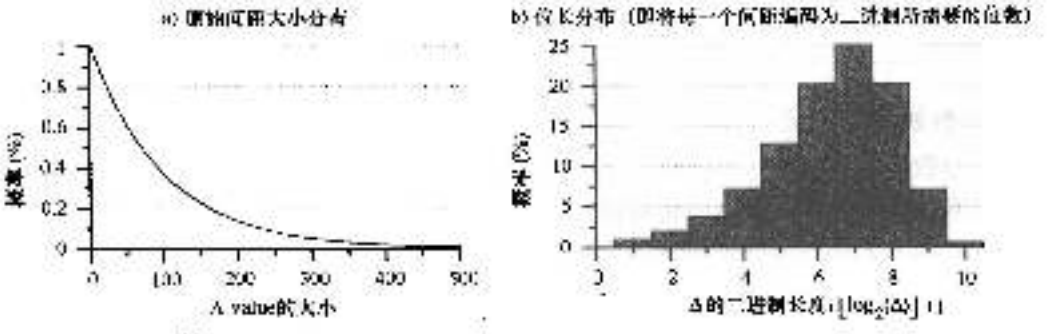


图 6-5 满足 $\frac{N_T}{N}=0.01$ 的位置信息列表的分布情况（即词项 T 出现在 1% 的文档中）。

1）选择一个整数 M 作为模（modulus）。

2）将每个 Δ value k 分成两部分，商 $q(k)$ 和余数 $r(k)$：

$$q(k) = \lfloor (k-1)/M \rfloor, \quad r(k) = (k-1) \bmod M$$

3）将 $q(k)+1$ 写成一元表示，后面再跟上 $r(k)$ 的 $\lceil \log_2(M) \rceil$ 或 $\log_2(M)$ 位的二进制表示，从而将 k 编码。

对于图 6-5 中显示的概率分布，我们选择 $M=2^7$。只有少数的 Δ-value 比 2^7 大，所以大部分位置信息所对应的商 $q(k)$ 仅需不到 3 位。相反地，只有少数的 Δ value 比 2^7 小，这意味着分配给每个余数 $r(k)$ 7 位只有一小部分是浪费的。

对于任意模 M，上述的一般化的编码过程，被称为 Golomb 编码（Golomb coding），以它的提出者 Solomon Golomb（1966）的名字命名。如果 M 取值为 2 的幂，那么这种编码方法被称为 Rice 编码（Rice coding），也是以它的提出者 Robert Rice（1971）的名字命名的。[3]

---

[3] 严格地说，Rice 并没有发明 Rice 编码——它只是 Golomb 编码的一个子集，并且 Golomb 的研究成果比 Rice 要早 5 年发表。但是，Rice 的工作使得 Golomb 编码得以结合实际应用，这也是 Rice 通常也被认为是这种编码方法的发明者的原因。

让我们看一个实例。假定现在选择模 $M=2^7$，要对间距值 $k=345$ 编码。得到的 Rice 码字为

$$\text{Rice}_{M=2^7}(345) = \overline{001\ 1011000}$$

因为，$q(345)=\lfloor(345-1)/2^7\rfloor=2$，$r(345)=(345-1)\bmod 2^7=88$。

可以很容易地对一个给定的码字进行解码。对于原始位置信息列表中的每个位置信息，首先找到码串中的第一个 $\overline{1}$ 位，从而获得 $q(k)$。然后从位串中移除前 $q(k)+1$ 位，抽取出接下来的 λ（$\lambda=\lceil\log_2(M)\rceil$）位，然后获得 $r(k)$ 并计算 k：

$$k = q(k)\cdot 2^{\lambda} + r(k) + 1 \tag{6-45}$$

从效率来讲，与 2^{λ} 相乘通常是通过移位操作来实现。

遗憾的是，上面介绍的简单的编码/解码过程只适用于 Rice 编码，却不适用于 M 不是 2 的幂的 Golomb 编码。我们给每个余数 $r(k)$ 分配 $\lceil\log_2(M)\rceil$ 位，但仅有 $M<2^{\lceil\log_2(M)\rceil}$ 个可能的余数，码空间的一部分未被使用，因此不是一个最优码。实际上，如果不管 M 的值，给每个余数都分配 $\lceil\log_2(M)\rceil$ 位，得到的 Golomb 码总是比参数为 $M'=2^{\lceil\log_2(M)\rceil}$ 的 Rice 码要差。

若要利用码空间的未被使用部分，可以将一部分余数用 $\lfloor\log_2(M)\rfloor$ 位编码，剩下的用 $\lceil\log_2(M)\rceil$ 位编码。哪些余数的码字应该短一些，哪些余数的码字应该长一些？因为 Golomb 编码方法基于的假设是 Δ-value 满足几何分布（公式 6-44），我们希望小间距比大间距出现的概率要高。因此，余数 $r(k)$ 的码字可以如下分配：

- 在区间 $[0,2^{\lceil\log_2(M)\rceil}-M-1]$ 内的长度为 $\lfloor\log_2(M)\rfloor$ 位。
- 在区间 $[2^{\lceil\log_2(M)\rceil}-M,M-1]$ 内的长度为 $\lceil\log_2(M)\rceil$ 位。

图 6-6 可视化了 $M=6$ 的 Golomb 编码机制。满足 $0\leqslant r(k)<2$ 的余数得到的码字长度为 $\lfloor\log_2(6)\rfloor=2$；其他余数的码字长度为 $\lceil\log_2(6)\rceil=3$。从稍微不同的角度观察这个方法，我们可以说 M 个不同余数的码字是根据范式哈夫曼码（见图 6-3）分配的。表 6-4 展示了在不同的 M 值下多个正整数 k 的 Golomb/Rice 码字。

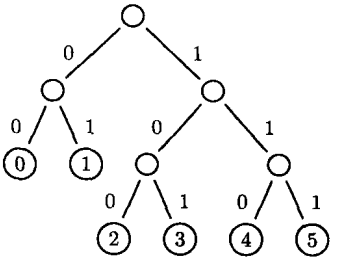


图 6-6 参数 M=6 的 Golomb 编码中，余数 $0\leqslant r(k)<6$ 的码字

表 6-4 用带参数的 Golomb/Rice 编码方法对正整数（如 Δ-gap）编码。每个码字的第一部分对应商 $q(k)$，第二部分对应余数 $r(k)$

整数	Golomb 编码			Rice 编码	
	$M=3$	$M=6$	$M=7$	$M=4$	$M=8$
1	1 0	1 00	1 00	1 00	1 000
2	1 10	1 01	1 010	1 01	1 001
3	1 11	1 100	1 011	1 10	1 010
4	01 0	1 101	1 100	1 11	1 011
5	01 10	1 110	1 101	01 00	1 100
6	01 11	1 111	1 110	01 01	1 101
7	001 0	01 00	1 111	01 10	1 110
8	001 10	01 01	01 00	01 11	1 111
9	001 11	01 100	01 010	001 00	01 000
31	00000000001 0	000001 00	00001 011	00000001 10	0001 110

与相对简单的 Rice 编码相比，Golomb 编码的压缩率稍微好一点，但比较复杂。Rice 解码器处理的 $r(k)$码字都是一样长的，只需要简单位移就可以解码，而 Golomb 解码器需要区别不同长度的 $r(k)$码字（导致分支预测错误），由于 M 不是 2 的幂，还需要执行相对耗时的整数乘法操作。因此，Rice 解码器通常比 Golomb 解码器快 20%～40%。

2. 寻找最优的 Golomb/Rice 参数值

一个我们尚未讨论的问题就是：如何选择模 M 使得平均码字长度最小。回想一下 6.2.1 节介绍的内容，给定概率分布$\mathcal{M}$，如果对于两个符号 σ_1 和 σ_2，满足以下关系的码$\mathcal{C}$是最优的：

$$|\mathcal{C}(\sigma_1)| = |\mathcal{C}(\sigma_2)| + 1 \tag{6-46}$$

则

$$\mathcal{M}(\sigma_1) = \frac{1}{2} \cdot \mathcal{M}(\sigma_2) \tag{6-47}$$

我们知道整数 $k+M$ 的 Golomb 码字比整数 k 的要多出 1 位（因为 $q(k+M)=q(k)+1$）。因此，最优参数值 M^* 应该满足如下的公式：

$$\begin{aligned} \Pr[\Delta = k + M^*] = \frac{1}{2} \cdot \Pr[\Delta = k] &\Leftrightarrow (1 - N_T/N)^{k+M^*-1} = \frac{1}{2} \cdot (1 - N_T/N)^{k-1} \\ &\Leftrightarrow M^* = \frac{-\log(2)}{\log(1 - N_T/N)} \end{aligned} \tag{6-48}$$

遗憾的是，M^* 通常都不是整数。对于 Rice 编码，我们在 $M=2^{\lceil \log_2(M^*) \rceil}$和 $M=2^{\lfloor \log_2(M^*) \rfloor}$之间进行选择。对于 Golomb 编码，我们在 $M=\lfloor M^* \rfloor$和 $M=\lceil M^* \rceil$之间进行选择。一般来说，不能确定哪个值更好。有时某个值可以得到更好的码，但有时使用其他的值可以得到更好的码。Gallager 和 van Voorhis（1975）证明了

$$M_{\text{opt}} = \left\lceil \frac{\log(2 - N_T/N)}{-\log(1 - N_T/N)} \right\rceil \tag{6-49}$$

总是可以产生最优码。

作为一个例子，考虑一个出现在 50% 文档中的词项。我们有 $N_T/N=0.5$，因此：

$$M_{\text{opt}} = \lceil -\log(1.5)/\log(0.5) \rceil \approx \lceil 0.585/1.0 \rceil = 1 \tag{6-50}$$

这个例子中的最优 Golomb/Rice 码，正如所预计的一样，是一元码。

3. 哈夫曼码：LLRUN

如果给定的列表中的间距不满足几何分布——例如，文档间独立性假设不成立，或列表不是简单的文档编号列表——Golomb 编码不会得到很好的结果。

我们已经知道一个对任意分布的列表都适用，且可以达到最优压缩率的压缩方法：哈夫曼编码（见 6.2.2 节）。遗憾的是，哈夫曼方法很难直接用于给定的 Δ-value 序列。难度在于一个典型的位置信息列表中不同间距的集合大小与列表本身是一样的。例如，在 TREC45 文档集中，词项“aquarium”出现在 149 个文档中。它的文档编号列表中包含 147 个不同的 Δ-value。用哈夫曼方法编码这个列表不太可能会大幅减少列表的长度，因为这 147 个不同的 Δ-value 还需存储在前言（比如哈夫曼树）中，以便让解码器知道哈夫曼码的间距串中的码字对应哪一个 Δ-value。

不直接对于间距值使用哈夫曼编码，先将相似大小的间距放进不同的桶中，同一个桶中的间距共用一个哈夫曼码字。例如，假设区间$[2^j, 2^{j+1}-1]$内的所有间距都有相同概率，可

以创建桶 B_0、B_1、……，满足 $B_j=[2^j,2^{j+1}-1]$。桶 B_j 中所有的 Δ-value 都共用同一个哈夫曼码字 ω_j。它们的编码形式就是 ω_j，后面跟对应的间距值的 j 位二进制表示（省略开头的 $\overline{1}$，因为它暗含在 ω_j 中）。

这个压缩方法被称为 LLRUN，由 Fraenkel 和 Klein（1985）提出。虽然没有使用一元编码选择器的值（整数 j 定义给定 Δ-value 所在的桶），但根据最少冗余的哈夫曼码，这种方法和 Elias 的 γ 方法非常相似。LLRUN 比最原始的哈夫曼方法好的地方在于，使用桶的技术能够显著地减少需要建立哈夫曼树的符号集的大小。甚至在模式独立索引中，Δ-value 也很少大于 2^{40}。因此，对应的哈夫曼树最多有 40 个叶子节点。用一个范式限长的哈夫曼码和每码字长度 $L=15$ 的限制（见 6.2.2 节中限长的哈夫曼码的详细介绍），这个码用 $4\times40=160$ 位就能表示。并且，除了很小的 j 值外，同一个桶 B_j 中间距值出现概率大致相等这一假设也符合索引中数据反映的事实。因此，压缩效果并没有因为将相似大小的间距放入桶中和让它们共享同一个哈夫曼码而变差。

6.3.3 上下文感知的压缩方法

目前讨论的方法都将位置信息列表中的间距视为是彼此独立的。在 6.1 节中，我们看到有时通过考虑待编码符号之间的上下文关系可以提高压缩方法的效率。在这里使用同样地想法。有时同一词项的相邻出现会形成簇（在小部分文本中大量出现同一词项）。对应的位置信息就会非常接近，Δ-value 就比较小。能够正确反映这种现象的压缩方法能比无法反映这种现象的方法达到更好的压缩率。

1. 哈夫曼码：有限上下文 LLRUN

很容易调整上述 LLRUN 方法使之能够在编码当前 Δ-value 时考虑前一个 Δ-value。这里不使用原版 LLRUN 中的零阶模型，而是采用一阶模型，当前间距选择器的码字 ω_j 依赖于前一个间距选择器的值，用大约 40 棵不同的哈夫曼树（假设没有 Δ-value 大于 2^{40}），每一个对应一个可能的前面的值。

这种方法的缺点在于需要编码器传输 40 棵不同的哈夫曼树的描述，而不仅仅是 1 棵，这会增加压缩列表的空间需求。然而实际上，使用 40 棵不同的哈夫曼树也不比仅用几棵哈夫曼树（例如 2～3 棵）更有优势。我们将这改进后的 LLRUN 方法称为 LLRUN-k，其中 k 指编码器/解码器使用的不同模型（即哈夫曼树）的个数。

图 6-7 给出了 LLRUN-2 的编码算法。图中清晰地显示了阈值参数 ϑ 可由算法自动选择，算法在 $0<\vartheta<\log_2(\max_i\{\Delta(L)[i]\})$ 之间尝试所有可能的值，并选择一个最小化列表压缩大小的值。注意到并不需要编码器真正将整个列表压缩 $\Theta(\log(\max_i\{\Delta(L)[i]\}))$ 次，只需要分析跟在 $B_{j'}$ 中的 Δ-value后的每个桶 B_j 中的 Δ-value 的频率。该分析的时间复杂度为 $\Theta(\log(\max_i\{\Delta(L)[i]\})^2)$。

2. 插值编码

另一种上下文感知压缩技术是由 Moffat 和 Stuiver（2000）提出的**插值编码**（interpolative coding）方法。与其他列表压缩方法一样，插值编码也是基于给定的倒排索引列表中的位置信息是按升序存储这一事实，但是它的做法略微不同。

考虑在 TREC45 文档集的文档编号索引中词项“example”的位置信息列表 L 的起始部分：

$$L=\langle 2,9,12,14,19,21,31,32,33\rangle$$

```
encodeLLRUN-2 (⟨L[1],...,L[n]⟩; ϑ, output) ≡
1   定义 Δ(L) 为列表 ⟨L[1], L[2] − L[1], ..., L[n] − L[n − 1]⟩
2   Δ_max ← max_i{Δ(L)[i]}
3   初始化数组 bucketFrequencies[0..1][0..⌊log2(Δ_max)⌋] 为零
4   c ← 0  // 用于有限上下文建模的上下文
5   for i ← 1 to n do  // 收集特定上下文统计值
6     b ← ⌊log2(Δ(L)[i])⌋  // 当前 Δ-value 的下角
7     bucketFrequencies[c][b] ← bucketFrequencies[c][b] + 1
8     if b < ϑ then c ← 0 else c ← 1
9   for i ← 0 to 1 do  // 建立两棵哈夫曼树，每个上下文一棵
10    T_i ← buildHuffmanTree(bucketFrequencies[i])
11  c ← 0  // 重置上下文
12  for i ← 1 to n do  // 压缩位置信息
13    b ← ⌊log2(Δ(L)[i])⌋
14    根据树 T_c, 把 b 的哈夫曼码字写到 output
15    把 Δ(L)[i] 以 b 位二进制形式写到 output (忽略第1个 1)
16    if b < ϑ then c ← 0 else c ← 1
17  return
```

图 6-7 LLRUN-2 的编码算法，使用两棵不同哈夫曼树的 LLRUN 有限上下文方法。阈值参数 ϑ 定义了可能的上下文二元分割

插值方法首先使用其他的压缩方法，如 γ 编码，对这个列表的第一个元素编码为 $L[1]=2$，最后一个元素编码为 $L[9]=33$。然后编码 $L[5]=19$。可是，当编码 $L[5]$时，解码器已经知道 $L[1]$和 $L[9]$。由于所有的位置信息都是严格递增的，所以能保证下面的式子成立：

$$2=L[1]<L[2]<\cdots<L[5]<\cdots<L[8]<L[9]=33$$

因此，根据列表中包含 9 个元素这个信息，以及 $L[1]$和 $L[9]$的值，我们能确定 $L[5]$的值位于区间[6,29]内。因为这个区间中包含的元素不超过个 $2^5=32$ 个，所以 $L[5]$可以用 5 个位来编码。接着算法递归地处理，编码 $L[4]$使用 4 位（因为根据 $L[1]$和 $L[5]$的值可知，$L[3]$一定位于区间[4,17]内），编码 $L[2]$使用 4 位（因为它一定位于[3,11]内），以此类推。

图 6-8 给出了插值方法编码步骤的更形式化的定义。表 6-5 中给出了使用插值编码方法压缩列表得到的位串结果。有必要指出的是，$L[8]=32$ 不用编码，因为 $L[7]=31$，$L[9]=33$ 使得 $L[8]$只有一个可能的值。

```
encodeInterpolative (⟨L[1],...,L[n]⟩, output) ≡
1   encodeGamma (n)
2   encodeGamma (L[1], output)
3   encodeGamma (L[n] − L[1], output)
4   encodeInterpolativeRecursively (⟨L[1],...,L[n]⟩, output)

encodeInterpolativeRecursively (⟨L[1],...,L[n]⟩, output) ≡
5   if n < 3 then
6     return
7   middle ← ⌈n/2⌉
8   firstPossible ← L[1] + (middle − 1)
9   lastPossible ← L[n] + (middle − n)
10  k ← ⌈log2(lastPossible − firstPossible + 1)⌉
11  把 (L[middle] − firstPossible) 以 k 位二进制形式写到 output
12  encodeInterpolativeRecursively (⟨L[1],...,L[middle]⟩, output)
13  encodeInterpolativeRecursively (⟨L[middle],...,L[n]⟩, output)
```

图 6-8 使用插值编码方法压缩位置信息列表$\langle L[1],\cdots,L[n]\rangle$。产生的位串写入到 *output* 变量中

在使用 Golomb 编码的时候，图 6-8 中由 *firstPossible* 和 *lastPossible* 定义的间距很少

是2的幂。因此，将区间中每一个可能的值都编码为 k 位，会浪费一些码空间。和前面一样，这个缺点可以使用这种方法来解决：$2^k = -$（$lastPossible - firstPossible + 1$）个可能值使用 $k-1$ 位编码，其余的值用 k 位来编码。由于这个机制比较简单，图6-8就没有显示出来。相关的细节可以在6.3.2节中关于Golomb编码的介绍中找到。

表6-5 在词项“example”的文档编号列表的前9个元素上使用插值编码方法得到的结果（数据来源于TREC45）

位置信息（原始顺序）	位置信息（访问顺序）	压缩后位串	
$(n=9)$	$(n=9)$	0001001	(γ 码字 $n=9$)
2	2	010	(γ 码字2)
9	33	000011111	(γ 码字 $31=33-2$)
12	19	01101	($13=19-6$ 是5位二进制数)
14	12	1000	($8=12-4$ 是4位二进制数)
19	9	0110	($6=9-3$ 是4位二进制数)
21	14	001	($1=14-13$ 是3位二进制数)
31	31	1010	($10=31-21$ 是4位二进制数)
32	21	0001	($1=21-20$ 是4位二进制数)
33	32		($0=32-32$ 是0位二进制数)

不像Golomb编码，我们无法确定（$lastPossible - firstPossible + 1$）个值中哪些出现的可能性更高。因此，不知道哪些值应该分配 k 位的码字，哪些值应该分配 $k-1$ 位的码字。Moffat和Stuiver（2000）建议给位于区间[$firstPossible$,$lastPossible$]中间位置的值分配更短的码字，除非这个区间仅包含一个位置信息（对应函数encodeInterpolativeRecursively中的 $n=3$），在这种情况下，他们建议给区间的两个端点分配较短的码字，从而可利用潜在的聚簇效果。实验表明这种策略能够使每个位置信息平均节约0.5位。

6.3.4 高查询性能的索引压缩

将位置信息列表以压缩形式存储有两个理由。首先，它能够减少搜索引擎索引所占的空间。其次，作为一个附加效应，它能够减少查询阶段的磁盘I/O开销，从而能够提升查询性能。前面章节讨论的压缩方法都是为了前者，而忽略了查询阶段要执行的解码操作的复杂性。

当在两种不同的压缩方法 A 和 B 之间做选择时，为了获得最好的查询性能，一个比较好的经验法则是：考虑存放索引的存储介质（如计算机硬盘）的读取性能，比较每个方法的解码开销。例如，硬盘可以以每秒5000万字节（=4亿位）的速率传输位置信息。如果方法 A 与方法 B 相比，平均每个位置信息少用1位，并且每个位置信息的解码开销比 B 少2.5 ns（即从硬盘中读取一位的时间），那么 A 比 B 优。反过来，如果相应的开销多2.5 ns，那么 B 更优。

2.5 ns算不上很长的时间，特别是对现代的微处理器来说。根据CPU的时钟频率，它可以等于2～10个CPU周期。因此，即使方法 A 的每个位置信息比方法 B 少占好几位，它的解码流程也必须非常的高效以获得查询性能上的绝对优势。

虽然上面的经验法则没有考虑可以使用并行的I/O操作对压缩列表解码或在内存中缓存位置信息列表的可能性，但是，当需要对比两个压缩方法对搜索引擎查询性能的影响时，

这个法则可以作为一个很好的着手点。正如在 6.3.6 节中我们将看到的，目前讨论的压缩方法不一定会得到最优的查询性能，因为它们的解码流程相当复杂和耗时。接下来介绍两个专门设计的用于高解码吞吐量的方法。

1. 字节对齐编码

字节对齐间距压缩是可用的最简单的压缩方法之一。它被广泛使用部分是由于它简单，但大部分是由于它具有高效的解码性能。考虑词项“aligned”的位置信息列表的起始部分，它是从 GOV2 文档集的文档编号索引中抽取出来的。

$$L = \langle 1624, 1650, 1876, 1972, \cdots \rangle$$
$$\Delta(L) = \langle 1624, 26, 226, 96, 384, \cdots \rangle \tag{6-51}$$

为了避免耗时的位提取操作，我们使用整数个字节来编码每一个 Δ-value。最简单的做法就是将每一个 Δ-value 的二进制表示分割成 7 位的块，并且在每一个块前面加一个**延续标志**（continuation flag）——一个用来指示当前块是否是最后一个块的位。这种方法称为 **vByte**（**可变字节**（variable-byte）编码）。不仅是搜索引擎，很多应用中都使用这种方法。以上文档编号列表的 vByte 编码形式（为了方便阅读插入了空格）为：

1 1011000 0 0001100 0 0011010 1 1100010 0 0000001 0 1100000 1 0000000 0 0000011 ...

例如，第一块$\overline{1011000}$的正文是整数 88 的 7 位二进制表示。第二块$\overline{0001100}$的正文是整数 12 的 7 位二进制表示。第二块起始位指示它是当前码字的结束块。当解码器看到这个标志时，它把两个数合在一起得到 $88+12\times 2^7=1624$，即列表的第一个元素。

图 6-9 显示了 vByte 方法的编码和解码流程。当阅读图中伪码时，你将会发现 vByte 显然不是为了最大压缩而优化的。例如，它为大小为 0 的间距分配码字$\overline{00000000}$。很显然，这样的间距是不存在的，因为位置信息构成的是严格递增的序列。其他的压缩方法（如 γ、LLRUN、…）都不会为大小为 0 的间距分配码字。然而，对于 vByte，情形则不同，它留下这个不用的码字做其他用途。vByte 的解码流程被高度优化，使得解码每个位置信息只需要几个 CPU 周期。增加更多的操作（即便是简单的操作，如 +1 或 −1）会增加它的复杂度，这样将会损害 vByte 方法较之其他压缩方法的速度优势。

```
encodeVByte (⟨L[1], ..., L[n]⟩, outputBuffer) ≡
1   previous ← 0
2   for i ← 1 to n do
3       delta ← L[i] − previous
4       while delta ⩾ 128 do
5           outputBuffer.writeByte(128 + (delta & 127))
6           delta ← delta ≫ 7
7       outputBuffer.writeByte(delta)
8       previous ← L[i]
9   return

decodeVByte (inputBuffer, ⟨L[1], ..., L[n]⟩) ≡
10  current ← 0
11  for i ← 1 to n do
12      shift ← 0
13      b ← inputBuffer.readByte()
14      while b ⩾ 128 do
15          current ← current + ((b & 127) ≪ shift)
16          shift ← shift + 7
17          b ← inputBuffer.readByte()
18      current ← current + (b ≪ shift)
19      L[i] ← current
20  return
```

图 6-9 vByte 的编码和解码流程。通过位移操作（“≪”：左移；“≫”：右移；“&”：按位与）实现了高效的乘和除操作

2. 字对齐编码

与处理整个字节比处理单个位更有效的思路一样，在解码一个压缩的位置信息列表时，访问整个机器字一般会比逐个获取所有字节要更有效。因此，如果我们强制用 16 位、32 位或 64 位的整数来存储给定位置信息列表中的每个位置信息，那么我们期望可以获得更高的解码速度。遗憾的是，这样做与压缩索引的目的背道而驰。如果我们用 32 位整数编码每个 Δ-value，那还不如使用更简单的编码器，直接将每

个位置信息存储为未压缩的32位整数。

下面的这个想法解决以上问题：不要将每个Δ-value使用一个单独的32位机器字来存储，而是将连续几个值，比如 n 个，用一个字来存储。例如，只要编码器看见三个连续的间距 $k_1 \cdots k_3$，满足 $k_i \leqslant 2^9 (1 \leqslant i \leqslant 3)$，就用一个字来存储它们，每个 k_i 分配9位，共占27位，剩下的5个未用位可以用作其他用途。

Anh和Moffat（2005）讨论了多种字对齐的编码方法，它们都基于上述类似的想法。最简单的方法叫做**Simple-9**。这种方法探测位置信息序列中后几个Δ-value，试图将尽量多的位置信息挤进一个32位的机器字中。当然，当解码器看到一个32位的字，推测这个字中可能包含多个Δ-value，但不知道这个机器字中给每个Δ-value预留了几位。所以，Simple-9将每个字的前4位留给选择器：这个4位的整数就通知解码器当前这个字的划分情况。字内剩下的28位，共有9种不同的方法将其划分为等大的块（如表6-6）。这也是这种方法名字的来由。

表6-6 Simple-9的字对齐位置信息压缩。从32位中预留4位用来保存选择器的值，共有9种可能的方法将剩下的28位划分成等大的块

选择器	0	1	2	3	4	5	6	7	9
Δ数	1	2	3	4	5	7	9	14	28
每Δ位数	28	14	9	7	5	4	3	2	1
每字未用位数	0	0	1	0	3	0	1	0	0

对于前面那个相同的Δ-value序列（词项“aligned”在GOV2文档集中的文档编号列表），对应的码串为：

$$0001\ 00011001011000\ 00000000011001\ 0010\ 011100001\ 001011111\ 101111111\ \mathrm{U} \tag{6-52}$$

其中，$\overline{0010}$（=2）是第二个机器字的选择器，$\overline{011100001}$是整数225(=1876−1650−1)的9位二进制表示。“U”表示一个未使用的位。

这个样例序列中，Simple-9并没有比vByte得到更紧凑的表示。这是因为词项“aligned”出现相对较少，并且位置信息列表的间距都很大。然而，对于具有较小间距的频繁词项，Simple-9方法明显比vByte要好，因为它能够用每个间距1位来编码一个位置信息列表（加上选择器的一些开销）。另外，Simple-9的解码性能几乎和vByte一样好，因为用于从给定的机器字中抽取出 $n=\left\lceil \frac{28}{n} \right\rceil$ 位整数而需要的移动掩码操作可以非常高效的执行。

3. 非对齐码的高效解码

即使前面章节介绍的非对齐方法与vByte和Simple-9不同——前者并不是为达到高效解码的目的而专门设计的，但是大多数仍然允许高效解码。然而，为了达到这个目标，应尽可能地避免耗时的按位解码操作。

我们通过一个例子来说明。考虑6.3.1节中的 γ 编码。将以下位置信息列表

$$L = \langle 7, 11, 24, 26, 33, 47 \rangle \tag{6-53}$$

转换成一个等价的Δ-value序列

$$\Delta(L) = \langle 7, 4, 13, 2, 7, 14 \rangle \tag{6-54}$$

用 γ 编码器得到下面的位串

$$\gamma(L) = \overline{001\ 11\ 001\ 00\ 0001\ 101\ 01\ 0\ 001\ 11\ 0001\ 110} \tag{6-55}$$

为了确定这个串中每个码字的位长，解码器需要重复地查找第一个 $\overline{1}$ 的位置，因为这个

位指示了码字选择器部分的结束。这可以通过按位方式处理 $\gamma(L)$，探测每一个位是 $\overline{0}$ 还是 $\overline{1}$。然而，这样的解码过程非常低效。不仅因为每一个码位需要执行至少一次 CPU 操作，而且，判断“当前位是 $\overline{0}$ 吗?”所需要的条件跳转就有可能会导致大量的分支误判，这将清空 CPU 执行管道，显著减慢解码过程（见附录中高性能计算简介部分；或者查看 Patterson 和 Hennessy（2009）获取这方面更详细的介绍）。

假定我们知道 $\Delta(L)$ 中的元素都不大于 2^4-1（正如上面的例子）。这意味着码串中所有的选择器都不超过 4 位。因此，我们可以构建一个包含 $2^4=16$ 个元素的表 T，指示位串中第一个 $\overline{1}$ 的位置，给定后面 4 位：

$$\begin{aligned} &T[\overline{0000}]=5, \quad T[\overline{0001}]=4, \quad T[\overline{0010}]=3, \quad T[\overline{0011}]=3,\\ &T[\overline{0100}]=2, \quad T[\overline{0101}]=2, \quad T[\overline{0110}]=2, \quad T[\overline{0111}]=2,\\ &T[\overline{1000}]=1, \quad T[\overline{1001}]=1, \quad T[\overline{1010}]=1, \quad T[\overline{1011}]=1,\\ &T[\overline{1100}]=1, \quad T[\overline{1101}]=1, \quad T[\overline{1110}]=1, \quad T[\overline{1111}]=1 \end{aligned} \tag{6-56}$$

（其中“5”表示不在前面 4 位中）。

我们可以使用这个表来实现码的高效解码，如图 6-10 所示。算法不是单独测试每个位，而是将码串每次 8 位的加载进 64 位的**位缓冲区**（bit buffer），然后用存储在查找表 T 中的信息，每次处理前面的 4 位。像这个算法这样的方法被称为**表驱动解码**（table-driven decoding）。适用于 γ、δ、Golomb、Rice 码和基于哈夫曼方法的 LLRUN 方法。然而，在 LLRUN中表 T 的内容取决于编码步骤所选择的码。因此，解码器在开始解码前，需要处理哈夫曼树来初始化查找表。

```
    decodeGamma (inputBuffer, T[0..2^k − 1], ⟨L[1], ..., L[n]⟩) ≡
1     current ← 0
2     bitBuffer ← 0
3     bitsInBuffer ← 0
4     for i ← 1 to n do
5       while bitsInBuffer + 8 ≤ 64 do
6         bitBuffer ← bitBuffer + (inputBuffer.readByte() ≪ bitsInBuffer)
7         bitsInBuffer ← bitsInBuffer + 8
8       codeLength ← T[bitBuffer & (2^k − 1)]
9       bitBuffer ← bitBuffer ≫ (codeLength − 1)
10      mask ← (1 ≪ codeLength) − 1
11      current ← current + (bitBuffer & mask)
12      bitBuffer ← bitBuffer ≫ codeLength
13      bitsInBuffer ← bitsInBuffer − 2 × codeLength − 1
14      L[i] ← current
15    return
```

图 6-10 表驱动的 γ 解码方法。使用一个位缓存和大小为 2^k 的查找表 T 避免了按位解码操作，表 T 用于决定当前码字的长度。通过位移操作（“≪”：左移；“≫”：右移；“&”：按位与）实现了高效的乘和除操作

表驱动的解码方法解释了为什么限长的哈夫曼码（见 6.2.2 节）对高性能查询处理是如此重要：如果我们知道每个码字都不超过 k 位，那么解码流程仅需要一个表查找操作（在大小为 2^k 的表中进行）就能找出给定位串的下一个码字。这样比显式地以按位方式遍历哈夫曼树要快很多。

6.3.5 压缩效果

让我们来看一下目前讨论的各种方法能够达到的压缩率。表 6-7 给出了三个样例文档集

上的不同类型（文档编号，TF值，文档内位置，模式独立）的位置信息列表上，通过每个位置信息所占的位数来衡量的压缩率一览表。由于莎士比亚文集不含任何实际的**文档**（document），为了能够进行压缩实验，我们将每一个〈SPEECH〉…〈/SPEECH〉这样的XML元素间的内容看做是一个文档。

表6-7 不同压缩方法对位置信息列表的压缩效果，评价在三个不同的文档集上进行。所有的数字表示压缩效果，用每个位置信息所占的位数来衡量。每一行中最好的结果用黑体标识

	列表类型	γ	δ	Golomb	Rice	LLRUN	Interp.	vByte	S-9
Shakesp.	文档编号	8.02	7.44	6.48	6.50	6.18	**6.18**	9.96	7.58
	TF值	1.95	2.08	2.14	2.14	1.98	**1.70**	8.40	3.09
	文档内位置	8.71	8.68	6.53	6.53	**6.29**	6.77	8.75	7.52
	模式独立	15.13	13.16	10.54	10.54	**10.17**	10.49	12.51	12.75
TREC45	文档编号	7.23	6.78	5.97	6.04	5.65	**5.52**	9.45	7.09
	TF值	2.07	2.27	1.99	2.00	1.95	**1.71**	8.14	2.77
	文档内位置	12.69	11.51	8.84	8.89	**8.60**	8.83	11.42	10.89
	模式独立	17.03	14.19	12.24	12.38	**11.31**	11.54	13.71	15.37
GOV2	文档编号	8.02	7.47	5.98	6.07	5.98	**5.97**	9.54	7.46
	TF值	2.74	2.99	2.78	2.81	2.56	**2.53**	8.12	3.67
	文档内位置	11.47	10.45	9.00	9.13	**8.02**	8.34	10.66	10.10
	模式独立	13.69	11.81	11.73	11.96	**9.45**	9.98	11.91	n/a

表中提到的方法有：γ和δ编码（6.3.1节）；Golomb/Rice以及基于哈夫曼的LLRUN（6.3.2节）；插值编码（6.3.3节）；以及两个面向性能的方法——vByte和Simple-9（6.3.4节）。在所有情况下，位置信息列表在压缩前被分割成多个包含大约16 000个位置信息的块。对于有参数的方法（Golomb、Rice、LLRUN），单独为每个块选择压缩参数（**局部参数化**（local parameterization）），这使得这三种方法都考虑了同一个位置信息列表中不同部分小的分布差别——这种能力将在6.3.7节派上用场。

正如从表6-7中看到的那样，插值编码在三个文档集上对于文档编号和TF值都是最好的，紧接着是LLRUN和Colomb/Rice。比起其他方法，插值编码最大的优势在于当大部分间距值为1时，它可以用平均不到1位对位置信息进行编码。这正是文档编号列表中多数频繁词项的情况（例如“the”在GOV2 80%的文档和TREC45 99%的文档中出现），在TF值列表中也一样：当随机选择一个文档D并随机从D中挑选一个词项T时，T在D中可能只出现1次。

对于其他类型的列表（**文档内位置**（within-document positions）和**模式独立**（schema-independent）），这些方法的排名颠倒了，LLRUN是最好的，后面跟着的是插值编码和Golomb/Rice编码。Golomb/Rice编码在这两种类型的列表上表现不好，原因是Golomb编码的基本假设（即间距满足几何分布）对于这两种列表都不成立。

假定文档之间相互独立，我们知道在一个给定的文档编号列表中的间距大致服从下面形式的分布（见公式（6-43））：

$$\Pr[\Delta = k] = (1-p)^{k-1} \cdot p \tag{6-57}$$

对于文档内位置和模式独立列表这个假定不再成立。如果一个随机词项T出现在靠近文档开始的部分，与它出现在靠近文档结束部分相比，这个词项更有可能再一次出现在同一个文档中。因此，这些列表中的元素不是相互独立的，间距值也不服从几何分布。

GOV2文档集中的词项“huffman”的出现模式是这一现象的好例子。图6-11展示了这

个词项的文档编号列表、基于文档的位置信息列表以及模式独立列表的间距值分布情况。做了对数转换后，将具有相同长度 $\text{len}(\Delta)=\lfloor\log_2(\Delta)\rfloor+1$ 的间距放到同一个桶中，图 6-11a 显示的曲线具有几何分布的特征，在 $\text{len}(\Delta)\approx10$ 的周围有一个明显的峰值。图 6-11b 是文档内位置的列表，也有一个峰值，但是它不如 6-11a 中的那么清晰。最后，图 6-11c 描绘的是模式独立列表，它根本就不符合几何分布。相反，我们能看到两个峰值：一个对应同一文档中同项“huffman”两次以上的出现（$\text{len}(\Delta)\approx6$），另一个对应不同文档中的连续出现（$\text{len}(\Delta)\approx22$）。显然，如果一个方法是面向几何分布设计的，它就不可能在明显不服从几何分布的位置信息上得到很好的结果，正如图 6-11c 所示。这也正是 LLRUN 在位置型的位置信息列表（基于文档或模式独立）上的性能比 Golomb/Rice 要好很多的原因 多达每条位置信息 2.5 位。

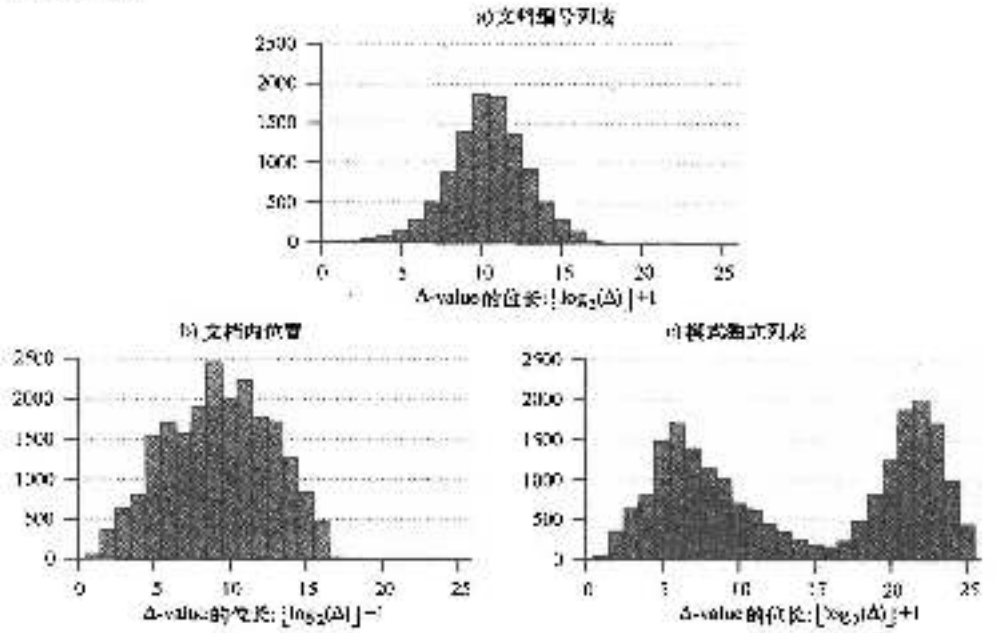


图 6-11 在 GOV2 文档集中词项“huffman”的几个不同类型位置信息列表的间距值分布情况。纵轴是具有特定大小的间距值个数。只有文档编号列表中的间距服从几何分布

LLRUN-k 方法是一种根据连续出现的位置信息之间的依赖关系专门设计的方法（6.3.3 节）。LLRUN k 与 LLRUN 非常类似，除了它是使用一阶压缩模型，并由 k 棵不同的哈夫曼树而不是 1 棵哈夫曼树编码得到的。表 6-8 对比了 LLRUN k 方法与原始的零阶 LLRUN 方法的压缩率。

表 6-8 零阶 LLRUN 与一阶 LLRUN 的对比。值得注意的是，只在 GOV2 的两种位置型的索引上（文档内位置索引和模式独立索引）有所提高。所有其他的索引要么太小，抵消了存储开销（因为更复杂的模型），要么列表元素之间不存在很强的依赖性

列表类型	THREC45			GOV2		
	LLRUN	LLRUN-2	LLRUN-4	LLRUN	LLRUN-2	LLRUN-4
文档内位置	6.60	6.57	6.57	8.02	7.85	7.82
模式独立	11.21	11.28	11.28	9.45	9.29	9.23

对于两个小的文档集，这个方法就没有显著的效果，因为编码上获得的一点效率却被前面的更大的实际码字串抵消了。然而，对于 GOV2 的两个位置型索引，我们确实看到了倒排索引列表空间需求有小幅减少。对于文档内位置信息，LLRUN-2/LLRUN-4 能够使得每个位置信息所占空间减少 0.14/0.20 位（−1.7%/−2.5%）；对于模式独立索引，它们能够使每个位置信息所占空间减少 0.16/0.22 位（−1.7%/−2.3%）。这种提高是否有价值取决于具体的应用。对于搜索引擎来说，它们通常是没有什么价值的，因为更加复杂的解码过程所带来的代价超过了减少 2% 的空间所带来的好处。

6.3.6 解码性能

正如 6.3.4 节在引入字节对齐和字对齐压缩方法时所指出的那样，减少倒排索引的存储空间只是应用压缩技术的一个原因。另外一个更加重要的原因是，更小的索引能够带来更好的查询性能，至少当位置信息列表存储在磁盘上时是这样的。作为一个经验法则，如果解码开销（用纳秒/每个位置信息来衡量）低于以压缩形式存储的位置信息的磁盘 I/O 时间，那么这个索引压缩方法应该能够提高查询性能。

表 6-9 列出了对文档编号列表的多种压缩方法的压缩效果、解码效率以及开销之和。所有的值是通过运行来自 2006 年 TREC TB 专题有效性任务中的 10 000 个查询和解压所有查询词项（忽略了停用词）的位置信息列表得到的。请注意，表（"bits/docid"）中列出的压缩效果与前面的表有很大的差异，这是因为这些结果不是根据整个索引而是根据出现在查询中的词项得到的。这些差异是可预期的：文档集中的频繁词项往往也是查询流中的频繁词项。由于频繁词项比不频繁词项的位置信息列表的间距更小，因此表 6-9 中的压缩率会比表 6-7 中的好。

表 6-9 GOV2 的文档编号索引的磁盘 I/O 和解压缩的总开销列表。总开销是基于顺序磁盘吞吐量 87 MB/s（≙1.37 ns/bit）来计算的

压缩方法	压缩（位/文档编号）	解码（纳秒/文档编号）	总开销（解码+磁盘 I/O）
Gamma	4.56	7.73	13.98 ns
Golomb	3.78	10.85	16.03 ns
Rice	3.81	6.46	11.68 ns
LLRUN	3.89	7.04	12.37 ns
Interpolative	3.87	26.50	31.80 ns
vByte	8.11	1.39	12.50 ns
Simple-9	4.91	2.76	9.49 ns
未压缩（32 位）	32.00	0.00	43.84 ns
未压缩（32 位）	32.00	0.00	87.68 ns

为了计算搜索引擎解码和磁盘 I/O 的总开销，我们假定硬盘以 87 MB/s（≙1.37 ns/bit；见附录）的速率传输位置信息数据。例如，对于 vByte 方法，每一个压缩后的文档编号值需要 8.11 位（平均），换算为磁盘 I/O 开销就是每个位置信息 8.11×1.37=11.11 ns。再加上解码每个位置信息的开销 1.39 ns，因此每个位置信息的总开销为 12.50 ns。

大多数的方法表现的性能处在一个水平，每个位置信息的总开销为 10～15 ns。由于插值编码具有相当复杂的递归解码过程，因此它是一个例外，每个位置信息的总开销超过 30 ns。与其他技术相比，字节对齐的 vByte 方法的表现相当平庸，被其他三个方法远远超出，包括相对复杂的 LLRUN 算法。simple-9 由于 vByte 具有相当不错的压缩率和很低的解

码开销，因此在实验中的性能表现最好。

当然，表中的数字仅仅代表了在文档编号索引上得到的实验结果。比如，在位置型位置信息的索引或者是模式独立索引中，结果明显更有利于 vByte。不仅如此，如果使用缓存或是**背景 I/O**（Background I/O），那么这个简单的性能评价模型（计算磁盘 I/O 和解码操作的时间和作为总开销）的有效性将会遭到质疑。但是不管如何，可以清楚地得到一个结论：与未经压缩的索引（每个位置信息占 32 位或 64 位）相比，每种压缩方法都能提高查询性能。

6.3.7 文档重排

目前为止，当讨论各种方法来压缩存储在索引位置信息列表中的信息时，我们总是假定存储在位置信息列表中的实际信息是固定的、不会被修改的。显然这是一个过度简单化的假设。事实上，文档不是由一个数字标识符而是由文档名字、URL 或其他描述它来源的文字信息所表示。在将检索结果展现给用户之前，为了将搜索结果转换为它们本来的内容，每一个数字文档标识符都要转换成文档位置或内容的文本描述。这种转换通常借助于一个称为**文档地图**（document map）的数据结构来实现（见 4.1 节）。

这意味着文档编号可以是任意的，而且不必具有任何内在的语义信息。因此我们可以按想要的方式来重新分配数字标识符，只要保证所有的改变都能被文档地图正确反映。例如，我们可以尝试以索引压缩率最大这种方法重分配文档编号。这个过程通常被称为**文档重排**（document reordering）。为了看到这个过程的作用，考虑以下文档编号列表。

$$L = \langle 4, 21, 33, 40, 66, 90 \rangle \tag{6-58}$$

它的编码表示为（46 位）：

$$\overline{001\ 00\ 00001\ 0001\ 0001\ 100\ 001\ 11\ 00001\ 1010\ 000001\ 00010} \tag{6-59}$$

如果我们按照如下的方式重新分配文档标识符，则列表变为

$$L = \langle 4, 6, 7, 45, 51, 120 \rangle \tag{6-60}$$

因此得到的编码表示为（36 位）

$$\overline{001\ 00\ 01\ 0\ 1\ 000001\ 00110\ 001\ 10\ 0000001\ 000101} \tag{6-61}$$

比原来减少了 22% 的存储空间。

从这个例子中可以清楚地看到，我们不太关注位置信息列表间距值的平均大小（公式（6-58）中为 17.2，而公式（6-60）中为 23.2），而更加关注如何最小化平均码字长度，这个值与每个间距值的对数值紧密相关。在上面的例子中，和的值

$$\sum_{i=1}^{5} \lceil \log_2(L[i+1] - L[i]) \rceil \tag{6-62}$$

从 22 减少到 18（−18%）。

当然，现实中事情往往没有那么简单。通过重新分配文档标识符来减少一个位置信息列表所占的存储大小的同时，又可能导致另外一个位置信息列表所占存储大小的增加。找到最优的分配方案，最小化索引的整体压缩大小，被认为是一个难以计算的问题（即 NP 完全问题）。幸运的是，还是有许多文档重排的启发式规则，尽管找到的是一个次优的文档编号重分配方案，但通常都会显著地提升压缩效果。大部分这些启发式规则都是基于聚类算法，这类算法需要复杂的实现过程和大量的计算资源。这里，介绍两种特殊的方法，实现相当简单，需要很少的计算资源，但仍然可以获得相当不错的效果：

- 第一种方法根据每个文档中所含的不同词项的个数来对文档集中的文档重新排序。

这个方法的思想是：同时包含大量不同词项的两个文档更有可能具有相同词项，而一个文档包含大量不同词项，另一个文档只包含少量不同词项，则不太可能具有相同词项。所以，使同时包含大量不同词项的文档分配到的文档编号比较接近，就可以比随机分配文档编号要获得更好的聚类效果。

- 第二种方法假定所有的文档都是从 Web（或者企业内部网）上抓取得到的。它根据 URL 的字母顺序重新分配文档编号。这个方法的思想是：来自相同 Web 服务器（甚至同一服务器下的相同目录）的两个文档更有可能具有相同词项，而来自 Internet 上不相关地址的两个文档则不然。

表 6-10 显示了这两个文档重排启发式规则对各种索引压缩方法的压缩效果的影响。“Original”这一行是指官方对 GOV2 文档集的排序，代表着文档从 Web 上抓取下来的顺序。其他的行分别代表了按随机排序、按包含不同词项个数排序和按 URL 排序的方法。

表 6-10 文档重排对压缩效果（单位：位）的影响。所有的数据都来自于 GOV2

	文档序	γ	Golomb	Rice	LLRUN	Interp.	vByte	S-9
文档ID	原序	8.02	6.04	6.07	5.98	5.97	9.54	7.46
	随机	8.80	6.26	6.28	6.35	6.45	9.86	8.08
	按词项数	5.81	5.08	5.15	4.60	4.26	9.18	5.63
	按URL	3.88	5.10	5.25	3.10	2.84	8.76	4.18
TF值	原序	2.74	2.78	2.81	2.56	2.53	8.12	3.67
	随机	2.74	2.86	2.90	2.60	2.61	8.12	3.87
	按词项数	2.74	2.59	2.61	2.38	2.31	8.12	3.20
	按URL	2.74	2.63	2.65	2.14	2.16	8.12	2.83

对于文档编号列表，我们可以看到文档重排可以提高表中每一种压缩方法的效果。具体的效果视不同的算法而不同，因为有些方法（如插值编码）比其他方法（如 Golomb 编码）更容易适应新的数据分布，但是总体趋势是一样的。有些情况效果相当显著。例如，用插值编码，平均空间开销由每个位置信息 5.97 位减少到 2.84 位，减少超过了 50%。这主要是因为插值编码能够用平均不到 1 位来编码非常小的间距值序列。其他方法则每个位置信息至少要 1 位（尽管对于间距值非常小的序列，通过分块技术（见 6.2.3 节），LLRUN 可以很容易被调整，使得用不到 1 位来编码每个位置信息）。

令人惊讶的是，文档重排不仅提高了文档编号的压缩效果，而且还提高了 TF 值的压缩效果。实验中这种出乎意料的现象出现的原因是，每个倒排列表被分割成多个包含 16 000 个位置信息的块，压缩在每个块上而不是整个列表上进行。将倒排列表分割成多个小块的初衷是为了能够对每个位置信息列表进行高效地随机访问，因为如果将整个列表都压缩成一个原子单元是实现不了这样的随机访问的（见 4.3 节）。这样做的好处是，压缩方法可以对同一个列表的不同部分（除了无参数的 γ 和 vByte 方法）选择不同的参数值（即压缩模型）。因为文档重排的一个基本思想是：将数值接近的文档编号分配给内容相似的文档，同一个倒排列表的不同部分就可以具有不同的性质，因此可以得到更好的整体压缩率。例如，当使用 LLRUN 编码后，每个 TF 值平均所占位数由 2.56 减少到 2.14（−16%）。

6.4 压缩词典

在第 4 章中，我们已经指出文档集的词典可能会相当大。虽然没有位置信息列表那么大，但仍很有可能因为太大而无法载入内存。此外，即便词典小到可以装入内存，但减少它

的大小仍是一个好想法，因为这样就可以使搜索引擎更好地利用它的内存资源——比如，缓存需频繁访问的位置信息列表或者缓存频繁查询的检索结果。这里的目标是尽可能地减少词典的大小，同时仍要保证访问词典记录和位置信息列表的低延迟。因为词典查找是索引构建中的主要瓶颈之一，又因为4.5.3节提出的基于合并的索引算法很大程度上独立于索引过程中可用的内存容量，所以没有任何理由需要在索引阶段实施词典压缩。我们就把讨论的范围限制在索引建立后查询阶段的词典上的可用的压缩技术。

回顾图4-1，基于排序的内存词典实质上是一个整数数组（主（primary）数组），其中每个整数是一个指向变长词项描述块（即词典记录）的指针，而词项描述块是辅（secondary）数组中的元素。每个词项描述块包含词项本身以及一个文件指针，该指针指示词项对应的倒排列表在位置信息文件中的位置。在这个数据结构中，可通过主数组上的二分查找来找到词典记录，跟随指针再到辅数组中去执行串比较操作。主数组上的指针通常每个占32位（假定辅数组小于2^{32}字节）。

基于排序的词典包括三个内存消耗的地方：主数组中的指针，词项描述块中的文件指针以及词项本身。考虑到辅数组中所有的数据都有序这一事实，我们同时考虑这三个组成部分：将多个连续的词项描述块合并为组，并对每个组的内容进行压缩。

1. 词典分组

词典分组的基本出发点就是没有必要（而且事实上是不利的）在主数组中为词典中的每个词项都单独分配一个指针。假设我们将连续的g个词项分为一组（g称为组大小（group size）），并只为每个组中的第一个词项（称为组头（group leader））维护一个指针，如果$g=1$，查找与之前的一样——通过在包含所有$|V|$个词项描述块的列表上执行二分查找实现。如果$g>1$，则首先在$|V|/g$个组头上执行二分查找，然后在找到的组内顺序扫描剩下的$g-1$个元素。

修改后的基于排序的词典数据结构如图6-12所示（组大小为3），当使用图中的词典查找词项“shakespearean”时，首先确定组头为“shakespeare”可能包含该词项。然后为了找到“shakespearean”的词典记录，用线性方式处理该组中所有的词项。从概念上来说，这个查找过程与4.4节中讨论的词典纠错技术非常类似。

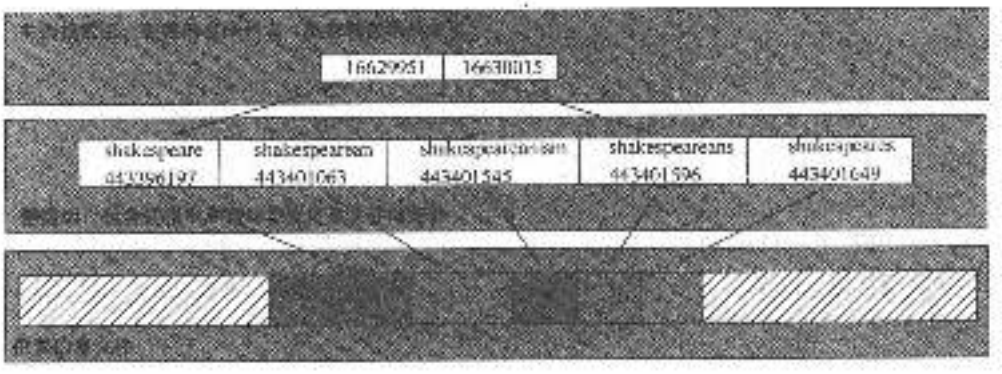


图6-12 分组的基于排序的词典：通过将多个词典记录合并成组减少主数组中指针的开销

来计算一下查找一个词项T时，需要进行字符串匹配的次数。为简单起见，我们假定词项T确实出现在词典中，词典共包含$|V|=2^n$个词项，组大小为2^m，其中m和n都是正整数（$m<n$且$n\geq 1$）。

- 当 $g=1$ 时，一次查找平均需要大概 $n-1$ 次字符串比较。“−1”是因为一旦查找到就可终止二分查找。n 次比较之后找到 T 的概率为 50%，$n-1$ 次比较后为 25%，$n-2$次比较后为 12.5%，以此类推。因此，我们平均需要进行 $n-1$ 次字符串比较。
- 当 $g>1$ 时，可以分为两种情况：

1）词项 T 是$|\mathcal{V}|/g$ 个组头中的一个。这种情况发生的概率为 $1/g$，平均需要 $n-m-1$ 次字符串比较。

2）词项 T 不是组头。这种情况发生的概率为$(g-1)/g$。平均需要$(n-m)+g/2$ 次字符串比较，其中第一个被加数对应在$|\mathcal{V}|/g$ 个组头上做完全二分查找时的比较次数（因为词项 T 不是组头，二分查找不会提前停止），第二个被加数对应在最后确定的组中线性扫描 $g-1$个非组头成员的比较次数。当找到匹配的词项时扫描终止。

结合 1 和 2，比较操作的期望次数为：

$$\frac{1}{g}(n-m-1)+\frac{g-1}{g}\left(n-m+\frac{g}{2}\right)=\log_2(|\mathcal{V}|)-\log_2(g)-\frac{1}{g}+\frac{g-1}{2}$$

表 6-11 列出了在大小为$|\mathcal{V}|=2^{20}$的词典上，用不同的组大小得到的平均比较次数。表中显示出选择 $g=2$ 时，每次查找的比较次数并没有增加。选择 $g=8$ 时增加了 7%。但与此同时，它大大减少了主数组中指针的开销，也提高了词典的缓存效率，因为需要访问的内存页更少了。

表 6-11 在一个包含$|\mathcal{V}|=2^{20}$个词项的基于排序的词典上查找一个词项平均所需的字符串比较次数

组大小	1	2	4	8	16	32	64
字符串比较	19.0	19.0	19.3	20.4	23.4	30.5	45.5

2. 前端编码

一旦将词典记录合并为组后，就可以将组头作为同步点，从而压缩词项描述块。因为无论是以压缩还是非压缩形式存储给定组内的词项描述块，它们总是要被顺序遍历的，所以我们不妨对其进行压缩，以减少存储需求和查找时所需访问的数据量。

通常使用一种称为**前端编码**（front coding）的技术来压缩基于排序的词典中的词项字符串。因为词项是按照字母顺序排序的，因此连续的词项几乎都具有相同的前缀。这个前缀会很长。例如，图 6-12 中的 5 个词项具有相同的前缀“shakespeare”。

在前端编码中，位于给定词项前面的词项已经暗含了前缀，因此我们删除给定词项的前缀，仅存储前缀的长度。这样，一个词项的前端编码可表示为如下的三元组形式：

$$(prefixLength,\ suffixLength,\ suffix) \tag{6-63}$$

为了便于实现，通常限定前缀和后缀的长度最多为 15 个字符，实现词典的时候就可以用一个字节（4 位）来存储 $prefixLength$ 和 $suffixLength$。如果后缀长度超过 15 个字符，就存储一个转义字符$(prefix,suffix)=(*,0)$，再以其他方式编码这个字符串。

图 6-12 中的词项采用前端编码可表示为

$$\langle \text{“shakespeare”},(11,2,\text{“an”}),(13,3,\text{“ism”})\rangle,\ \langle \text{“shakespeareans”},(11,1,\text{“s”}),\cdots\rangle$$

之前也提过，每组的第一个元素存储为未压缩形式，作为一个二分查找的同步点。

最后一步，当压缩完词项字符串之后，我们仍然可以减少词典中的文件指针的开销。因为位置信息文件中的列表是按照字母顺序排序的，与词典中词项的顺序一致，文件指针组成

了一个单调递增的整数序列。因此，我们可以使用 6.3 节中介绍的任何一种基于间距值的列表压缩技术来减少存储需求。例如，使用字节对齐的 vByte 方法，以“shakespeare”开头的组压缩形式如下：

$$\begin{array}{l}\langle(\langle 101,96,54,83,1\rangle,\text{“shakespeare”}),(\langle 2,38\rangle,11,2,\text{“an”}),(\langle 98,3\rangle,13,3,\text{“ism”})\rangle \\ \langle(443396197,\text{“shakespeare”}),(4866,11,2,\text{“an”}),(482,13,3,\text{“ism”})\rangle\end{array} \tag{6-64}$$

其中，每个词典记录的第一个元素是 Δ-encoded 文件指针（第一行中是 vByte，第二行是 Δ-value）。在前端编码中，每个组头的文件指针都是以未压缩的形式存储的。

表 6-12 列出了上述方法在词典大小和平均词项查找时间上的效果。通过将连续的词典记录合并为组，但不进行压缩，词典大小减少了 18%（858 MB 与 1046 MB）。结合分组技术和前端编码，词典存储需求又将近减少一半。如果再使用 vByte 编码文件指针，词典就仅为原来的 25%。

表 6-12 在 GOV2 的基于排序的词典上进行词典压缩的效果，这个词典包含 4950 万个词典记录。表中列出了不同组大小和不同压缩方法下词典大小和平均词项查找时间

组大小	未压缩	前端编码	前端编码 +vByte	前端编码 +vByte+LZ
1 个词项	1046 MB/2.8 μs	*n/a*	*n/a*	*n/a*
2 个词项	952 MB/2.7 μs	807 MB/2.6 μs	643 MB/2.6 μs	831 MB/ 3.1 μs
4 个词项	904 MB/2.6 μs	688 MB/2.5 μs	441 MB/2.4 μs	533 MB/ 2.9 μs
16 个词项	869 MB/2.7 μs	589 MB/2.2 μs	290 MB/2.3 μs	293 MB/ 4.4 μs
64 个词项	860 MB/3.9 μs	576 MB/2.4 μs	252 MB/3.2 μs	195 MB/ 8.0 μs
256 个词项	858 MB/9.3 μs	570 MB/4.8 μs	243 MB/7.8 μs	157 MB/29.2 μs

对于查找性能，我们得到两个有趣的观察结果。首先，尽管将词项描述块分组会稍微增加字符串比较的次数，但仍然提高了词典查找性能，因为减少主数组中的指针数使得二分查找的缓存效率更高。其次，无论组大小 g 为何值，前端编码词项使得词典查找更快了，因为查找中需要排序和比较的词项更少了。

表 6-12 最后一列的压缩方法我们现在还没介绍：结合了分组、前端编码和 vByte，在应用了前端编码和 vByte 后再使用了通用的 Ziv-Lempel 压缩算法（在 zlib[⊖] 压缩方法库中实现了）。这么做是因为，即便是前端编码后的词典仍然具有相当程度的冗余。例如，在一个 TREC 文档集（包含 120 万个不同词项）上的前端编码词典中，后缀“ation”出现了 171 次，后缀“ing”出现了 7726 次，后缀“ville”出现了 722 次。前端编码具有这样的冗余是很显然的，因为它仅关注于词项的前缀。对于转换过后的 Δ-transformed 文件指针也存在同样地问题。因为很多位置信息列表（特别是仅包含一个位置信息的列表）具有相同的大小，我们会发现很多文件指针具有相同的 Δ-value。我们可以在使用了前端编码和 vByte 之后，再使用标准的 Ziv-Lempel 数据压缩方法来减少这种冗余。表 6-12 表明这种方法尽管对于小的组效果不是很好，但 $g \geqslant 64$ 时，与原来未压缩的索引相比，能够将词典的规模减少最高可达 85%。

3. 结合词典压缩和词典交错

在 4.4 节中，我们讨论了另外一种减少词典内存需求的方法。通过交错词典记录和保存

⊖ www.zlib.net

在磁盘中的位置信息列表，仅在内存中为大概每 64 KB 的索引数据维持一个词典记录，我们可以大幅减少词典的内存需求：将千兆字节降为几百兆字节。很自然地想到，结合使用词典压缩和词典交错，使搜索引擎的内存词典可以获得更好的压缩效果。

表 6-13 总结了使用这种方法能够获得的效果。取决于索引块的大小 B（两个连续的内存词典记录之间最大的磁盘索引数据量）和内存词典组大小 g，可能将总存储量减少到小于 1 MB（B=32 768，g=256）。这种方法对查询阶段不利的地方在于增加了 I/O 开销，因为每个查询词项均需额外读取 32 768 字节；用我们实验中的硬件驱动器只需不到 0.5 ms。

更为显著地是，当 B=512，g=256 时，词典的大小可以减少为不到 30 MB。选择索引块大小 B=512 不会明显降低查询性能，因为 512 字节（=1 个扇区）的数据是很多硬盘的最小传输单元。无论如何，与不使用词典交错或词典压缩的数据结构相比，词典所需的内存还是能减少 97% 以上。

表 6-13 结合使用词典压缩和 4.4 节介绍的交错词典组织形式得到的内存词典大小。压缩方法：FC+vByte+LZ。索引数据结构：GOV2 上建立的词频索引（文档编号+TF 值）

		索引块大小（字节）						
		512	1024	2048	4096	8192	16 384	32 768
组大小	1 个词项	117.0 MB	60.7 MB	32.0 MB	17.1 MB	9.3 MB	5.1 MB	9.9 MB
	4 个词项	68.1 MB	35.9 MB	19.2 MB	10.4 MB	5.7 MB	3.2 MB	1.8 MB
	16 个词项	43.8 MB	23.4 MB	12.7 MB	7.0 MB	3.9 MB	2.3 MB	1.3 MB
	64 个词项	32.9 MB	17.8 MB	9.8 MB	5.4 MB	3.1 MB	1.8 MB	1.0 MB
	256 个词项	28.6 MB	15.6 MB	8.6 MB	4.8 MB	2.7 MB	1.6 MB	0.9 MB

6.5 总结

本章中主要的观点有：

- 很多压缩方法将待压缩的消息看做是符号序列，通过寻找一个能够反映符号在给定的消息中的统计特性的码来完成压缩，给出现频繁的符号分配更短的码字。
- 对于有限符号集上任何一个给定的概率分布，哈夫曼编码方法总能产生最优前缀码（即最小化平均码字长度的码）。
- 算术编码能够获得比哈夫曼编码更好的压缩率，因为它不为每个符号分配整数位长的码字。然而哈夫曼编码的解码速度通常比算术编码要快，因此在许多应用中仍是首选（比如搜索引擎）。
- 倒排列表的压缩方法都是基于这样一个事实：同一个位置信息列表中的位置信息是升序排序的。通常将列表转换为等价的 Δ-value 序列之后再压缩。
- 有参数的方法通常比无参数的方法产生更小的输出。当使用有参数的方法时，可以为每个列表选择不同的参数（**局部参数化**）。
- 如果位置信息列表中的间距服从几何分布，那么使用 Golomb/Rice 码将会得到比较好的压缩结果。
- 如果间距很小，插值编码将会获得非常好的压缩效果，因为它能够使用平均不到 1 位来编码每个位置信息（算术编码同样也具有这样的能力，但是它的解码操作比插值编码的解码操作更复杂）。
- 如果间距值分布范围比较广，基于哈夫曼的 LLRUN 方法可以得到非常好的结果。
- 前端编码可以显著地减少搜索引擎词典数据结构的内存需求。与词典交错方法结合

使用，能将内存词典的大小减少 99% 以上，同时几乎不会降低查询性能。

6.6 延伸阅读

通用数据压缩（包括文本压缩）的一个很好的综述由 Bell 等人（1990）给出，更近期的 Sayood（2005）和 Salomon（2007）也做了同样地工作。Witten 等人（1999）总结了倒排文件中位置信息列表的压缩技术。Zobel 和 Moffat（2006）对倒排索引中的压缩技术的研究做了综述，包括对已有的倒排列表压缩方法的概述。

哈夫曼码由 David Huffman 在 1951 年提出，当时他还是麻省理工学院的一名学生。他的一个教授——Robert Fano 让他的学生想出一个对任何给定的（有限长度的）源消息产生最优二分码的方法。Huffman 成功了，并取得该课程的免考资格作为奖励。在过去的 60 年里，人们对哈夫曼码进行广泛的研究，并在与理论最优码（算术码）相比的冗余度方面，已经取得重要的成果（Horibe，1977；Gallager，1978；Szpankowski，2000）。

算术编码可以克服哈夫曼编码为每个符号分配整数个位所导致的缺陷。算术编码最早由 Rissanen（1976），Rissanen 和 Langdon（1979）以及 Martin（1979）提出，当时称为**区间编码**（range encoding），但没有得到广泛地采用，直到 20 世纪 80 年代中期，Witten 等人（1987）发表了一种高效的算术编码器。许多现代的文本压缩算法，如 DMC（Cormack 和 Horspool，1987）以及 PPM（Cleary 和 Witten，1984），都是基于算术编码的变形得来的。

γ、δ 以及编码由 Elias（1975）提出，他同时还给出了**普遍性**（universality）的概念（在将码字 C_i 分配给符号 σ_i 时，如果出现概率越高的符号分到的码字越短，就称这个码字集合是**普遍的**（universal），每个编码符号期望的位数是符号源的熵的常数因子倍）。Golomb 编码由 Golomb（1966）在一篇颇有娱乐性质的短文中提出的，短文是关于“特工 00111”轮盘游戏的。插值编码由 Moffat 和 Stuiver（2000）提出，除了可以用于倒排文件压缩，他们还指出了其他一些应用，如高效地表示哈夫曼编码（即编码器使用的码的描述），以及编码用基于 Burrows-Wheeler 变换的压缩算法中的前移值。

字节对齐压缩算法如 vByte（6.3.4 节）最先由 Williams 和 Zobel（1999）进行学术研究，但很久之前就开始使用了。Trotman（2003）基于磁盘索引对压缩效果和解码效率之间的平衡关系进行了研究，证明可变字节编码通常比按位编码可获得更好的查询性能。Scholer 等人（2002）得到同样地结论，但他们还证明了字节对齐压缩方案有时还能改进未压缩的**内存**（in-memory）索引。在最近的研究中，Büttcher 和 Clarke（2007）表明即便搜索引擎的索引访问模式是完全随机的，这个观点也是正确的。

Blandford 和 Blelloch（2002）最早研究了对文档重排以获得更好的压缩率，他们的方法基于聚类方法，尽量将相近的文档编号分配给内容相似的文档。与他们的研究不同，Shieh 等人（2003）提出了一种类似的方法，将文档重排约减为旅行商问题（TSP）。Silvestri（2007）提出了根据文档的 URL 对文档进行排序以提高压缩效果的想法。

6.7 练习

练习 6.1 考虑符号集 $\mathcal{S}$={“a”,“b”}的概率分布为：Pr[“a”]=0.8，Pr[“b”]=0.2。当对 $\mathcal{S}$ 上的消息编码时，假定将符号分成块，每块包含 m=2 个符号（得到新的分布 Pr[“aa”]=0.64，Pr[“ab”]=0.16 等）。为这个新分布构造一棵哈夫曼树。

练习 6.2 证明 γ 码是无前缀码（即不存在一个码字是另外一个码字的前缀）。

练习 6.3 给出不是无前缀码的无二义二元码的一个例子。在什么样的概率分布下，这个码是最优的？

为同一分布构造一个最优无前缀码。

练习 6.4 写出 ω 码的解码过程（见 6.3.1 节）。

练习 6.5 找出最小的整数 n_0，使得对于所有满足 $n \geqslant n_0$ 的整数对应的 ω 码的表示长度比对应的 δ 码要短。注意 n_0 可能非常大（$>2^{100}$）。

练习 6.6 给定词项 T 的位置信息列表服从几何分布，且 $N_T/N=0.01$，当参数时 $M=2^7$，使用 Rice 编码压缩这个位置信息列表，每个码字的期望位数是多少？

练习 6.7 考虑一个间距服从几何分布的位置信息列表，间距的平均大小为 $N/N_T=137$（即 $p=0.0073$）。如果使用 Golomb 编码，最优的参数值 M^* 为多少？如果使用 Rice 编码，最优的参数值 M^*_{Rice} 为多少？如果使用最优的 Rice 编码代替 Golomb 编码，那么平均会浪费多少位？如果使用最优的 Golomb 编码代替算术编码，那么平均又会浪费多少位？

练习 6.8 在 6.2.2 节中，我们讨论了如何在 $\Theta(n\log(n))$ 时间内构建一棵哈夫曼树，其中 $n=|\mathcal{S}|$ 为输入字母表的大小。现在假定 $\mathcal{S}$ 中的符号在到达编码器之前已经根据它们的概率排序。设计一个能够在 $\Theta(n)$ 时间内构建一棵哈夫曼树的算法。

练习 6.9 6.3.4 节介绍的 Simple-9 压缩方法将 Δ-value 序列存入 32 位的机器字中，预留 4 位保存选择器值。考虑一个类似的方法：Simple-14，使用 64 位的机器字。假定每 64 位字中预留 4 位来保存选择器值，列举对余下的 60 位进行划分的所有方法。你认为这两种方法中哪一种在典型的文档编号索引上能够产生更好的压缩效果？描述在哪种类型的文档编号列表上，Simple-9 比 Simple-14 会获得更好的/更差的结果。

练习 6.10 6.3.7 节介绍的文档重排方法中，有一种是按照文档的 URL 的字母顺序分配数字文档标识符的。是否存在另外一种也是基于文档 URL，但可以获得更好的压缩率的方法？考虑 URL 的组成部分，想想可以怎么利用它们？

练习 6.11（项目练习） 对你已实现的倒排索引数据结构使用索引压缩。实现 6.3.4 节中的字节对齐 vByte 方法以及 γ 编码（6.3.1 节）或 Simple-9（6.3.4 节）中的一种。你的新索引必须以压缩形式存储所有的位置信息列表。

6.8 参考文献

Anh, V. N., and Moffat, A. (2005). Inverted index compression using word-aligned binary codes. *Information Retrieval*, 8(1):151–166.

Bell, T. C., Cleary, J. G., and Witten, I. H. (1990). *Text Compression*. Upper Saddle River, New Jersey: Prentice-Hall.

Blandford, D. K., and Blelloch, G. E. (2002). Index compression through document reordering. In *Data Compression Conference*, pages 342–351. Snowbird, Utah.

Burrows, M., and Wheeler, D. (1994). *A Block-Sorting Lossless Data Compression Algorithm*. Technical Report SRC-RR-124. Digital Systems Research Center, Palo Alto, California.

Büttcher, S., and Clarke, C. L. A. (2007). Index compression is good, especially for random access. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management*, pages 761–770. Lisbon, Portugal.

Cleary, J. G., and Witten, I. H. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402.

Cormack, G. V., and Horspool, R. N. S. (1987). Data compression using dynamic Markov modelling. *The Computer Journal*, 30(6):541–550.

Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203.

Fraenkel, A. S., and Klein, S. T. (1985). Novel compression of sparse bit-strings. In Apostolico, A., and Galil, Z., editors, *Combinatorial Algorithms on Words*, pages 169–183. New York: Springer.

Gallager, R. G. (1978). Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, 24(6):668–674.

Gallager, R. G., and Voorhis, D. C. V. (1975). Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information Theory*, 21(2):228–230.

Golomb, S. W. (1966). Run-length encodings. *IEEE Transactions on Information Theory*, 12:399–401.

Horibe, Y. (1977). An improved bound for weight-balanced tree. *Information and Control*, 34(2):148–151.

Larmore, L. L., and Hirschberg, D. S. (1990). A fast algorithm for optimal length-limited Huffman codes. *Journal of the ACM*, 37(3):464–473.

Martin, G. N. N. (1979). Range encoding: An algorithm for removing redundancy from a digitised message. In *Proceedings of the Conference on Video and Data Recording*. Southampton, England.

Moffat, A., and Stuiver, L. (2000). Binary interpolative coding for effective index compression. *Information Retrieval*, 3(1):25–47.

Patterson, D. A., and Hennessy, J. L. (2009). *Computer Organization and Design: The Hardware/Software Interface* (4th ed.). San Francisco, California: Morgan Kaufmann.

Rice, R. F., and Plaunt, J. R. (1971). Adaptive variable-length coding for efficient compression of spacecraft television data. *IEEE Transactions on Commununication Technology*, 19(6):889–897.

Rissanen, J. (1976). Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20(3):198–203.

Rissanen, J., and Langdon, G. G. (1979). Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162.

Salomon, D. (2007). *Data Compression: The Complete Reference* (4th ed.). London, England: Springer.

Sayood, K. (2005). *Introduction to Data Compression* (3rd ed.). San Francisco, California: Morgan Kaufmann.

Scholer, F., Williams, H. E., Yiannis, J., and Zobel, J. (2002). Compression of inverted indexes for fast query evaluation. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 222–229. Tampere, Finland.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656.

Shannon, C. E. (1951). Prediction and entropy of printed English. *Bell System Technical Journal*, 30:50–64.

Shieh, W. Y., Chen, T. F., Shann, J. J. J., and Chung, C. P. (2003). Inverted file compression through document identifier reassignment. *Information Processing & Management*, 39(1):117–131.

Silvestri, F. (2007). Sorting out the document identifier assignment problem. In *Proceedings of the 29th European Conference on IR Research*, pages 101–112. Rome, Italy.

Szpankowski, W. (2000). Asymptotic average redundancy of Huffman (and other) block codes. *IEEE Transactions on Information Theory*, 46(7):2434–2443.

Trotman, A. (2003). Compressing inverted files. *Information Retrieval*, 6(1):5–19.

Williams, H. E., and Zobel, J. (1999). Compressing integers for fast file access. *The Computer Journal*, 42(3):193–201.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images* (2nd ed.). San Francisco, California: Morgan Kaufmann.

Witten, I. H., Neal, R. M., and Cleary, J. G. (1987). Arithmetic coding for data compression. *Commununications of the ACM*, 30(6):520–540.

Ziv, J., and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343.

Zobel, J., and Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys*, 38(2):1–56.

第7章

Information Retrieval: Implementing and Evaluating Search Engines

动态倒排索引

在第4章中，当我们讨论如何在给定的文档集上构建倒排索引时，许多观点都是基于这样一个假设——文档集是**静态的**（static）且在构建索引过程中以及索引构建完之后都不发生改变。显然，这个假设并不总是成立的。在很多应用中，可以预料到文档集是会随着时间而发生改变的。如文件系统、数字图书馆以及Web都是**动态**（dynamic）文档集的实例。

动态文档集一般允许有三种类型的操作：文档插入、删除以及修改。为动态文档集构建倒排索引时需考虑数据的易变性，并采取恰当的策略保持索引与文档集一致。这种**索引维护策略**（index maintenance strategy）是本章讨论的主题。

本章第一部分（7.1节），我们讨论如何处理半静态的文档集。半静态文档集允许插入和删除操作（将文档修改视为删除后再插入新文档），但索引不需要立即反映出文档集中的变化。相反，几个小时甚至几天的短暂延迟都是可以容忍的。Web搜索引擎一般将Web上的大部分的内容当做是一个半静态的文档集；一个新的Web页面可能要过几天或者几周才会被加入到搜索引擎的索引中（但受欢迎的页面，如www.whitehouse.gov，通常都会被频繁地重索引）。

第二部分（7.2节）主要关注**增量**（incremental）文档集。增量文档集允许新文档加入到文档集中。然而，现有的文档不允许修改或删除。我们评价了几种增量文档集的索引更新策略，结果表明将索引分割成多个独立的子索引（“索引分区”）能够使搜索引擎在高效更新索引结构的同时不会对查询性能造成不合理的影响。这种方法可用于将索引保持在最新的状态，即使比较频繁地加入新文档也是适用的。

在本章第三部分，我们讨论**非增量**（non-incremental）动态文档集的索引更新策略，这种文档集支持全部的更新操作：插入、删除和修改。在7.3节中，我们展示了如何实现一个懒惰的删除过程，它并不立即将位置信息从索引中删除，而只是标识这些文档已被删除；定期执行的垃圾回收操作将清理索引并删除所有的垃圾位置信息。7.4节讨论了如果搜索引擎必须支持文档修改（例如追加操作），那么搜索引擎的索引结构需要进行哪些修改。我们指出了在这样的情况下会出现的一些困难。我们的讨论解释了为何搜索引擎一般不直接支持文档修改，而将它视为是删除后插入的组合操作——就算这样对于长文档而言相当低效。

与第4章一样，贯穿本章的基本假设是：内存资源相对于索引大小来说非常稀少，因此大部分的索引数据需要保存在磁盘上。对于有大量内存资源的大规模搜索引擎，可能这里讨论的技术不那么适用，但基本问题的共性无论在哪种情况下（基于磁盘和基于内存）通常都是非常相似的。

7.1 批量更新

在一些应用中，索引不立刻对文档集的更新做出反应是可以接受的。在使用一次性**批量更新**（batch update）操作将所有新数据加入到索引之前，索引过程可能会等上一阵子，不断收集新的文档。在Web搜索中，这是一种典型的索引更新方式，新的网页可能要经过几天或几周才会加入到主流Web搜索引擎的索引中。对一个已有的索引进行批量更新有两种实现方法：

- 从头开始建立一个新的索引。当索引构建完成时，旧的索引被删除，用新的索引来取代。这个过程被称为 REBUILD 更新策略。
- 根据新插入的文档内容构建一个索引。当这个索引被构建完之后，它将与原始的索引进行合并，得到一个新的索引，并取代原来的索引。这个过程被称为 REMERGE 更新策略。REMERGE 更新策略中的合并操作实现的方式本质上与基于合并的索引构建过程的最后一步合并操作是一样的（见 4.5.3 节）。

哪种策略更好，REBUILD 还是 REMERGE？从开发者的角度来看，REBUILD 更具有吸引力，因为它更容易实现。然而，REUBILD 需要解析和重索引文档集中的所有文档，无论它们是否已经被索引过。而采用 REMERGE，我们可以重用已有的索引中的数据，因此比 REBUILD 可能更具有潜在的性能优势。

如果批量更新中只有文档插入（没有删除操作），那么很明显 REMERGE 策略比 REBUILD 具有更好的性能。然而，如果更新操作不是严格地只增添文档，而同时有插入和删除，那么情况就不那么明朗了。在这种情况下，REMERGE 需要花一定的时间来读取和解码被删除的文档的位置信息，这些位置信息不会被包括在新的索引中。而 REBUILD 永远不用处理这些位置信息，因此比 REMERGE 具有潜在的优势。

假设初始索引 I_{old} 中包含有 d_{old} 个文档的位置信息，这些文档的大小均等。进一步假设批量更新中包含有 d_{delete} 次删除和 d_{insert} 次插入。在批量更新之后，新的索引 I_{new} 将包含

$$d_{\text{new}} = d_{\text{old}} - d_{\text{delete}} + d_{\text{insert}} \tag{7-1}$$

个文档的位置信息。REBUILD 构建索引 I_{new} 所需的时间仅由 d_{new} 决定。使用第 4 章中的基于合并的索引构建方法，构建 I_{new} 所需的时间为

$$\begin{aligned} time_{\text{rebuild}}(d_{\text{old}}, d_{\text{delete}}, d_{\text{insert}}) &= c \cdot d_{\text{new}} \\ &= c \cdot (d_{\text{old}} - d_{\text{delete}} + d_{\text{insert}}) \end{aligned} \tag{7-2}$$

其中 c 为系统指定的常数。

如果使用 REMERGE 策略，构建 I_{new} 的时间又为多少呢？答案取决于搜索引擎中建立索引和合并索引操作的相对性能。根据表 4-7，我们知道在基于合并的索引构建中，为 GOV2 建立一个全文索引，最后一步合并操作占全部时间的 25%。其余 75% 的时间花在解析文档和构建子索引上。因此，我们可以认为使用 REMERGE 为文档集构建一个新的索引所需的总的时间大约为

$$time_{\text{remerge}}(d_{\text{old}}, d_{\text{delete}}, d_{\text{insert}}) = c \cdot d_{\text{insert}} + \frac{c}{4} \cdot (d_{\text{old}} + d_{\text{insert}}) \tag{7-3}$$

其中第一个被加数对应为 d_{insert} 个新到来的文档建立索引 I_{insert} 所需的时间代价，第二个被加数对应合并 I_{old} 与 I_{insert} 得到新索引 I_{new} 所需的时间代价。

公式（7-3）没有考虑到这里执行的合并操作比 4.5.3 节的合并操作稍微长一些，因为这里的合并操作不只是合并位置信息列表，还需要回收那些被删除文档的位置信息。然而，为简单起见，我们假设这部分的代价可忽略不计，因此合并操作的速度实际上是重建索引操作速度的 4 倍（7.3 节合理地分析了删除操作对更新索引性能的影响）。基于这种假设，当满足如下条件时：

$$time_{\text{rebuild}}(d_{\text{old}}, d_{\text{delete}}, d_{\text{insert}}) = time_{\text{remerge}}(d_{\text{old}}, d_{\text{delete}}, d_{\text{insert}}) \tag{7-4}$$

$$\Leftrightarrow \quad d_{\text{old}} - d_{\text{delete}} + d_{\text{insert}} = d_{\text{insert}} + \frac{1}{4} \cdot (d_{\text{old}} + d_{\text{insert}}) \tag{7-5}$$

$$\Leftrightarrow \quad d_{\text{delete}} = \frac{3}{4} \cdot d_{\text{old}} - \frac{1}{4} \cdot d_{\text{insert}} \tag{7-6}$$

REBUILD 和 REMERGE 具有相同的性能。

尽管文档集一直在改变，但是它们还是会有一个**稳定状态**（steady state），也就是说，插入的文档数等于删除的文档数（$d_{\text{insert}} = d_{\text{delete}}$）。事实上，这是很常见的，因为许多文档集尽管可能会有大量的更新操作，但是相对于文档集总的大小来说，增长速度非常慢。公式(7-6)可以化简为：

$$d_{\text{delete}} = \frac{3}{5} \cdot d_{\text{old}} \tag{7-7}$$

所以，如果至少有 60% 的文档从旧的索引中删除且再也不出现在新的索引中，那么 REBUILD 策略比 REMERGE 策略更加有效。如果删除的比例不到 60%，那么 REMERGE 将会是更好的选择。

在大多数应用中，连续的两次批处理更新所删除的相对文档数远小于 60%。这不一定是因为文档集变化得很慢，而是因为如果搜索引擎检索结果集指向的文档总是不存在，那么这样的搜索引擎对用户而言就没什么用了。为确保用户满意，快速变化的文档集需要更频繁的索引更新。因此实际上，REMERGE 通常比 REBUILD 的结果要好。

鉴于每个具有基于合并的索引构建过程的搜索引擎（见 4.5.3 节）都自动地具有合并两个倒排文件的能力，所以，没有理由选择 REBUILD 而不选择 REMERGE。因为 REMERGE 比 REBUILD 要快很多，所以它使得索引更新周期更短，因此索引比较新。不可否认的是，由于需要支持文档删除和合并操作，因此 REMERGE 更新策略比在基于合并的索引构建过程中的最后一步合并操作要稍微复杂些。然而这个调整也不会太复杂。7.3 节讨论从已有的索引中删除垃圾位置信息的方法。

7.2　增量式索引更新

上一节描述的批量更新策略可能适用于某些特定的应用，在多数搜索环境中，搜索引擎的索引还是需要及时反映文档集当前的内容。例如网络新闻搜索、文件系统搜索以及互联网运营商（如亚马逊[⊖]和 ebay[⊜]）所使用的专用搜索系统都是这样。所有这些例子中，一旦检测到文档集的变化，索引必须立即更新。这时批量更新策略就不再适用了。本节我们将讨论如果文档集是严格增量的，那么这个目标如何实现。严格增量是指只允许新文档的加入，而不允许已有文档的修改和删除。

回顾 4.5.1 节介绍的由基于哈希的内存索引构建方法构建的索引结构。这个索引的骨干结构是一个基于内存的哈希表，将词项字符串映射为对应的位置信息列表的内存地址。尽管这个数据结构最初是为满足索引过程的特殊需要而提出的，但它同样可以用于查询处理。

当搜索引擎在构建索引时，假设在创建完 n 个磁盘索引分区后，我们想让搜索引擎处理一条包含 m 个查询词项的关键字查询。这可通过对这 m 个词项重复以下过程来实现：

1）从这个 n 磁盘索引分区的每一块中取出这个词项的位置信息列表片段。

2）使用内存哈希表取出词项在内存中的位置信息列表片段。

3）连接这 $n+1$ 个片段得到这个词项的位置信息列表。

⊖ www.amazon.com

⊜ www.ebay.com

图 7-1 描述了这一过程。这个过程总共需要 $m+n$ 次随机磁盘读取。我们将这一过程称为 NO MERGE 索引更新策略，接下来会介绍原因。

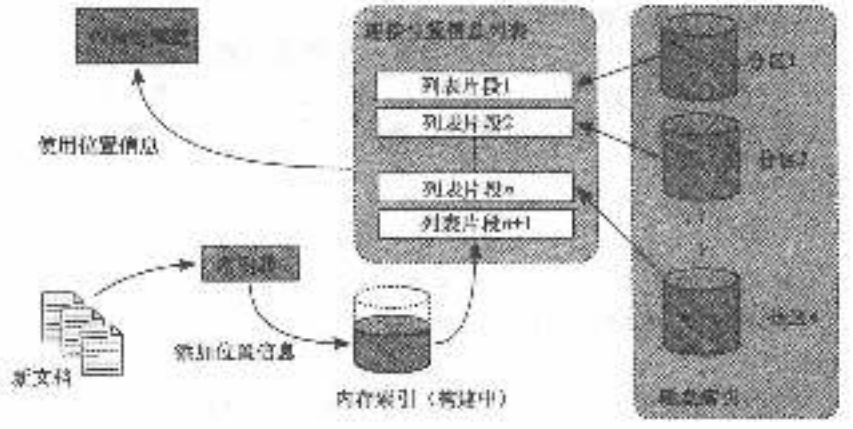


图 7-1 将离线的基于合并的索引构建转换成为一种在线的索引维护策略（NO MERGE）。从内存中正在构建的索引以及之前构建的磁盘索引中取出每个查询词项的位置信息列表，通过连接多个列表片段得到最终的倒排列表

尽管 NO MERGE 可以用于将静态文档集上的离线索引构建方法转换为同时允许文档插入和搜索查询的在线方法，但它并不是一个好的策略，因为处理搜索查询时需要大量的磁盘寻道（每个查询词项和索引分区都要一次寻道）。如果索引分区的数量很大，那么 NO MERGE 的查询性能就非常差。尽管如此，这个方法为其他更复杂的策略提供了一个基准。

本节余下部分，我们将讨论几种增量式文档集的索引更新策略，这些策略都是基于同一个想法，也是 NO MERGE 策略的基础：对新文档建立索引，就像它们是静态文档集的一部分一样，然后通过连接查询词项在磁盘索引和内存索引中的位置信息列表来处理查询。然而与 NO MERGE 不同，这些策略的目标是通过将小的列表片段组合成大的列表，使磁盘索引中的片段数量保持在比较低的水平。

7.2.1 连续倒排列表

保证磁盘位置信息列表片段比较少的最有效方法是，不允许为一个列表片段已在索引中的词项创建新的磁盘列表片段。这可通过两个完全不同的方法来实现：合并更新和就地更新。

1. 合并更新

回顾 7.1 节介绍的 REMERGE 策略，REMERGE 不仅可用于半静态文档集中的批量更新操作，而且还可用于增量式文档集的索引更新策略：搜索引擎处索引模块就像在静态文档集中那样处理新的文档。与静态文档集不同的是，每当索引器耗尽内存时，就合并内存中累积的数据和磁盘上已有的索引，从而创建

```
buildIndex_ImmediateMerge (M) =
1   I_mem ← ∅   // 初始化内存索引
2   currentPosting ← 1
3   while 还有待处理文档 do
4       T ← next token
5       I_mem.addPosting(T, currentPosting)
6       currentPosting ← currentPosting + 1
7       if I_mem 包含多于 M−1 个符号单元 then
8           if I_disk 存在 then
9               I_disk ← mergeIndices({I_mem, I_disk})
10          else
11              I_disk ← I_mem
12          I_mem ← ∅   // 重置内存索引
13  return
```

图 7-2 带有 IMMEDIATE MERGE 的在线索引构建方法。mergeIndices 过程如图 4-13 所示

一个新的磁盘索引。一般过程如图 7-2 所示。

将旧的索引数据与内存中的数据合并需要搜索读取引擎同时读出已有的磁盘索引和写入新的索引。以大块的方式读/写所有的数据，一次可能若干兆字节，非顺序磁盘操作的数量（在新旧索引中切换）可以维持在比较低的水平，整个更新操作得以高效的执行。

在实时（非批量）更新的环境下，REMERGE 有时也被称为 IMMEDIATE MERGE，以区别于其他基于合并的更新策略，这些策略在索引过程耗尽内存时不总是执行合并操作。

为了量化比较 IMMEDIATE MERGE 和 NO MERGE 性能上的不同，以及说明连续的方式对于存储磁盘位置信息列表的重要性，我们让搜索引擎索引 GOV2 文档集中的 2520 万个文档，使用大约 800 MB 的内存预算（足以建立一个 250 000 个文档的内存索引）。只要系统耗尽内存，就立即执行一个磁盘索引更新，合并内存索引和已有的磁盘索引，然后再处理 1000 个搜索查询。

图 7-3 给出了这两种索引更新策略中每个查询的平均时间。不出所料，由于 IMMEDIATE MERGE 总是以连续的方式处理磁盘位置信息列表，因此比 NO MERGE 展现了更好的查询性能。但是有点出乎意料的是，它比 NO MERGE 好出一大截。在整个文档集都被索引后，NO MERGE 比 IMMEDIATE MERGE 慢了整整 7 倍（IMMEDIATE MERGE 需 690 ms，NO MERGE 需5100 ms）。

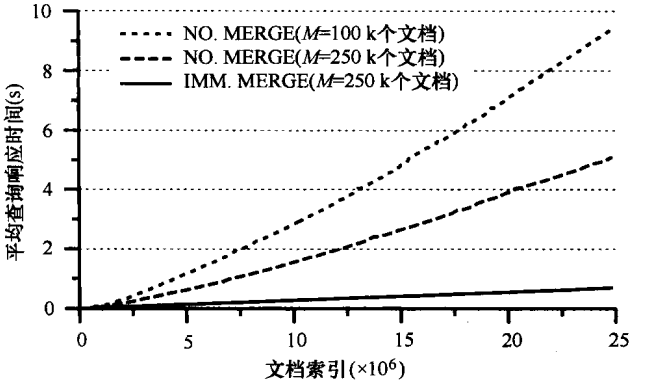


图 7-3　IMMEDIATE MERGE 和 NO MERGE 的平均查询性能，这个性能是索引大小的函数。文档集：GOV2（2520 万个文档）。查询集：来自 TREC TB 2006 的随机查询，使用邻近度排名方法进行评价（2.2.2 节）

是什么造成了如此巨大的差距？内存预算足够索引 250 000 个文档，NO MERGE 在 GOV2 累积了 100 个磁盘索引分区，使得每个查询词项都几乎需要 100 次随机磁盘寻道。每次随机访问的代价大约是 12 ms（磁盘寻道加上旋转延迟），每个查询平均包含 3.5 个词项，我们估计每次查询的总开销为：

$$12\ \text{ms} \cdot 100 \cdot 3.5 = 4200\ \text{ms} \tag{7-8}$$

比 IMMEDIATE MERGE 每个查询仅需 690 ms 慢了 7 倍。这个计算结果表明 NO MERGE 的实用性存在一些基本的限制。即使我们将内存预算增加到原来的 5 倍，仍然会得到 20 个索引分区，每个查询随机访问的开销大概也有 800 ms。相反，如果索引过程的内存预算减少，查询性能将会变得无法接受，正如图 7-3 所示，文档数 $M=100\ 000$。

遗憾的是，尽管与 NO MERGE 策略相比具有明显的优势，但 IMMEDIATE MERGE 并不是解决索引更新问题的最终办法。每当索引过程耗尽内存时，它都要从磁盘中读出整个索引，将其与内存索引进行合并，然后再写回磁盘。久而久之，这个策略所需要的读/写操作次数将是文档集大小的平方。

假设索引过程将 M 个位置信息缓存在内存中，为一个包含 N 个词条的文档集建立索引。当第一次内存耗尽时，M 个位置信息将被传送到磁盘。当第二次内存耗尽时，这 M 个位置信息就被加载入内存，与当前内存中的索引进行合并，然后 $2M$ 个位置信息又写回磁盘。因此，对于具有 N 个词条的文档集，传出/传入磁盘的位置信息总数为：

$$\sum_{k=1}^{\lfloor N/M \rfloor} (k-1)M + kM = M \cdot \sum_{k=1}^{\lfloor N/M \rfloor} (2k-1) \tag{7.9}$$

$$= M \cdot \left(\left\lfloor \frac{N}{M} \right\rfloor\right)^2 < \Theta\left(\frac{N^2}{M}\right) \tag{7.10}$$

如果可用内存容量相对于文档集整体大小来说较小，那么这个平方复杂度会使 IMMEDIATE MERGE 变得不适用，正如图 7-4 所示，图中给出了 NO MERGE 和 IMMEDIATE MERGE 随着索引大小的增加，对应的累加索引更新代价。就算一个大小适中的内存预算为 800 MB（≈250 000 个文档），IMMEDIATE MERGE 也比 NO MERGE 要慢大约 6 倍。

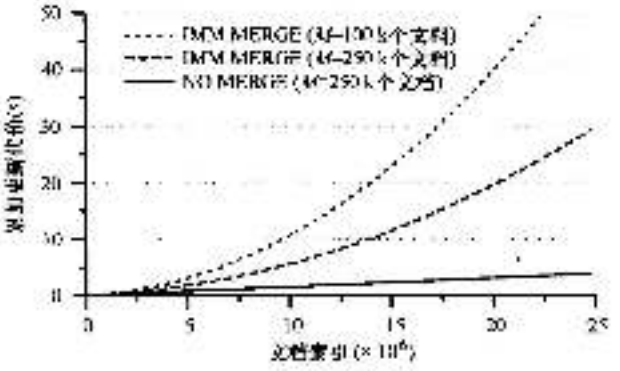


图 7-4 IMMEDIATE MERGE 和 NO MERGE 策略的累加索引更新代价，它是索引大小的函数，数据集：GOV2（2520 万个文档）

2. 就地更新

就地索引更新策略试图突破 IMMEDIATE MERGE 的限制。当内存耗尽时它不需要搜索引擎从磁盘中读出整个索引。相反，磁盘索引更新的时候只要有位置信息列表写到磁盘，就在这个位置信息列表的末端预留一些空间。随后只要同一个词项的位置信息需要传输到磁盘，就在已有列表的末端进行扩展。如果这个列表末端的空间不够存储到来的新位置信息了，那么整个列表就被重新装载到磁盘索引的另一个新位置，这个位置足够放下旧索引和新索引。图 7-5 给出了这个一般过程的一个实例。

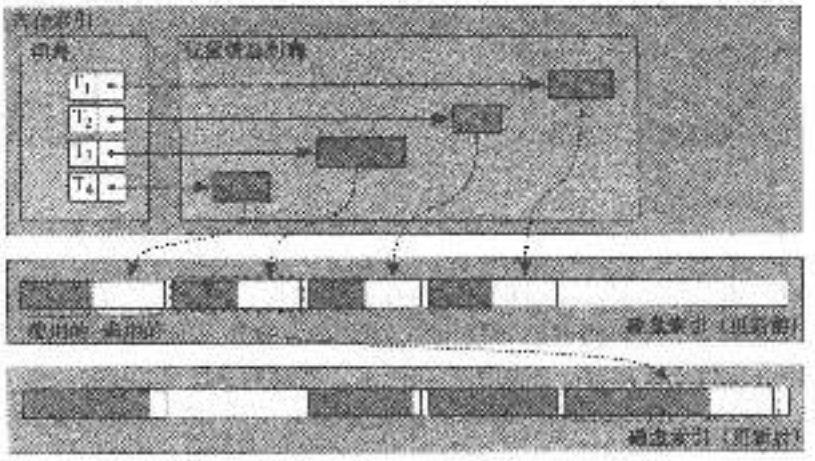


图 7-5 部分配的就地索引更新策略。左词项 T_3 的磁盘位置信息列表末端没有足够的空间来存储新的位置信息，这个列表必须重新放在另外一个位置

不同的就地更新策略的区别主要在于它们为给定列表末端预留空间的方式不同。最常见的方式（从 4.5.1 节中你已经了解它了）是按比例预分配（proportional pre allocation）。假

设一个大小为 b 字节的位置信息列表 L 被传送到磁盘（或重装载到一个新位置）。搜索引擎为 L 预留 $k \cdot b$ 字节的总空间。也就是说，这个列表末端的 $(k-1) \cdot b$ 字节可用于未来的位置信息。一般预分配因子 $k=1.5$ 或 $k=2$。与其他预分配策略相比，如在列表末端预留常数个字节，按比例预分配的优势在于重装载操作中的磁盘读/写字节数与列表最终的大小是线性关系（见练习 7.2）。在本节的后面部分，我们把按比例预分配的就地更新策略简单地称为 INPLACE。

因为对于给定列表，INPLACE 传入/传出磁盘的总字节数与这个列表的大小成线性关系，所以我们可以知道为整个倒排索引传入/传出磁盘的总字节数与索引的大小成线性关系。这似乎意味着 INPLACE 可以比平方磁盘复杂度的 IMMEDIATE MERGE 获得更好的更新性能。但事实并非如此。就地更新策略需要大量的非顺序磁盘访问模式，即耗时的随机访问磁盘操作。因此，粗略估计传入/传出磁盘的数据量并不能体现整体的更新性能。

回顾图 7-5 的例子。因为内存索引中没有足够的空间存储新的位置信息，词项 T_3 的位置信息列表必须重装载。重装载这个列表需要两个随机磁盘访问：一个用于从旧位置读出原始列表，另一个用于将扩展之后的列表（包含了内存索引中的位置信息）写到新的位置。我们实验中用到的硬盘（见附录 A）执行一次随机磁盘访问的代价与顺序读或写 1 MB 数据的代价相当。

因为大部分位置信息列表都非常短（齐夫定律），INPLACE 的非顺序访问索引方式对它的更新性能有灾难性的影响。例如，就地更新 100 000 个非常短的位置信息列表需要大约 20 分钟（假设每个列表需一次磁盘寻道）。如果按照顺序的读/写方式，更新这些位置信息列表（作为合并更新的一部分）只需几秒。

有很多关于索引维护方面的文献优化了基本的 INPLACE 策略，例如，Shieh 与 Chung (2005) 以及 Lester 等人 (2006)。提出的方法效果在文献里已有讨论，我们把关注点放在一种更基本的减少整体更新开销的方法：将 INPLACE 和 IMMEDIATE MERGE 结合在一起，得到**混合**（hybrid）更新策略。

3. 混合索引维护

INPLACE 和 IMMEDIATE MERGE 相比，我们可以说 INPLACE 的主要缺陷在于需要相对多次的磁盘寻道，而 IMMEDIATE MERGE 的主要缺陷在于搜索引擎的内存耗尽时需要将整个磁盘索引读出/写入。有可能设计这样一种更新策略：通过为每个位置信息列表而不是整个索引选择使用 INPLACE 或 IMMEDIATE MERGE，将两种方法的优势结合起来。

考虑磁盘上的单个位置信息列表 L。假设在第 i 个索引更新周期（也就是搜索引擎第 i 次耗尽内存）之后，L 总的大小为 s_i 字节。在第 $i+1$ 个更新周期后，L 的大小为 s_{i+1} 字节。现在，假设搜索引擎的索引所在的磁盘顺序读/写的吞吐量为每秒 b 字节，并且随机磁盘访问平均需要 r 秒。如果按照 IMMEDIATE MERGE 策略更新，那么在第 $i+1$ 个更新周期过程中，更新代价为：

$$D_{i+1} = \frac{s_i + s_{i+1}}{b} \tag{7-11}$$

其中，两个被加数分别是读取旧列表和写新列表的开销。如果使用就地更新策略，那么更新代价为：

$$D_{i+1} = 2r + \frac{s_{i+1} - s_i}{b} \tag{7-12}$$

假设更新这个列表需要两次磁盘寻道（在混合索引维护策略中，每一次就地更新需要两次磁盘寻道而不是一次，因为它中断了顺序合并操作）。令公式（7-11）和公式（7-12）相等，得到：

$$s_i = r \cdot b \tag{7-13}$$

也就是，只要磁盘中的位置信息列表大于 $r \cdot b$ 字节，就地更新就比 REMERGE 更有效。对于普通用户的硬盘来说，大概 $s_i = 0.5$ MB。列表越短，IMMEDIATE MERGE 效率越高。对于较长的列表，INPALCE 更加适用。

图 7-6 中的算法形式化了这种想法。这个算法在磁盘上维护两个索引：I_{merge} 与 I_{inplace}。最开始所有的位置信息列表都存放在 I_{merge} 中。每当列表达到了预定义的阈值 ϑ（由硬件指定的 REMERGE 与 INPLACE 之间的平衡点），就从 I_{merge} 传输到 I_{inplace}。算法用变量 $\mathcal{L}$ 来跟踪保存就地索引中所有的长列表。

```
buildIndex_HybridImmediateMerge (M) ≡
1   定义I_merge为主磁盘索引（合并维护，初始化为空）
2   定义I_inplace为辅磁盘索引（就地更新，初始化为空）
3   I_mem ← ∅  // 初始化内存索引
4   currentPosting ← 1
5   L ← ∅  // 长列表集合初始化为空
6   while 还有符合单元需索引 do
7     T ← next token
8     I_mem.addPosting(T, currentPosting)
9     currentPosting ← currentPosting + 1
10    if I_mem包含超过 M − 1 个位置信息
11      //通过按字典序遍历I_mem 和I_merge，合并它们
12      I'_merge ← ∅  // 创建新的磁盘索引
13      for each term T ∈ I_mem ∪ I_merge do
14        if (T ∈ L) ∨ (I_mem.getPostings(T).size + I_merge.getPostings(T).size > ϑ) then
15          I_inplace.addPostings(T, I_merge.getPostings(T))
16          I_inplace.addPostings(T, I_mem.getPostings(T))
17          L ← L ∪ {T}
18        else
19          I'_merge.addPostings(T, I_merge.getPostings(T))
20          I'_merge.addPostings(T, I_mem.getPostings(T))
21      I_merge ← I'_merge  // 用新磁盘索引替代旧的
22      I_mem ← ∅  // 重初始化内存索引
```

图 7-6 使用连续位置信息列表上的 HYBIRD IMMEDIATE MERGE（HIMc）构建在线索引的过程。输入参数：长列表阈值 ϑ

因为这种更新策略结合了 REMERGE 和 INPLACE 的优点，它通常也被称为**混合索引维护**（hybird index maintenance）。图中所示的这种方法被称为**连续位置信息列表**（contigous posting list）下的 HYBRID IMMEDIATE MERGE（HIMc）。Büttcher 和 Clarke（2008）分析了这种方法，他们证明了这种方法理论上的复杂度依赖于文档集中词项的分布情况，复杂度为：

$$\Theta\left(\frac{N^{1+1/\alpha}}{M}\right) \tag{7-14}$$

其中，N 为文档集的大小，M 为索引过程中内存的预算，α 为正在被构建索引的文档集的齐夫参数。对于与 GOV2 大小相当的文档集来说，HIM_C 的索引更新代价比 IMMEDIATE MERGE 大概要少 50%。但另一方面，它们的查询性能几乎是一样的，因为它们都将位置信息列表维持在严格连续的状态。

7.2.2　非连续倒排列表

混合索引维护策略是基本的 IMMEDIATE MERGE 策略的一种显著改进。然而，如果内存资源很少，它的更新性能仍然有些不尽如人意，很容易成为搜索引擎的主要瓶颈。遗憾的是，如果坚持按照连续的方式来存储磁盘上的所有位置信息列表，那么最好也就达到这种程度。为了获得更好的更新性能，我们必须丢弃将所有位置信息列表按照连续的方式来存储的想法。

1. 索引分割

索引更新策略维护多个磁盘倒排文件，其中每一个倒排文件都包含任一给定倒排列表的一小部分，则称为**索引分割方案**（index-partitioning scheme）。将磁盘索引分为一组独立的索引分区可以大大提高搜索引擎的更新性能。然而，不利的一面是：因为每个位置信息列表不再存储在硬盘的连续区域，查询性能会慢一些。通过确保索引分区的数量较少，可以把这种不利保持在可接受的水平。

2. LOGARITHMIC MERGE

最常见的一种索引分割算法是 LOGARITHMIC MERGE。它是多个开源文本检索系统（包括 Lucene 和 Wumpus）的标准索引维护策略。LOGARITHMIC MERGE 维护多个索引分区。每个分区有一个标签 g，标识这个分区是第几**代**（generation）的。每当搜索引擎索引过程耗尽内存时，它就为内存索引中的数据创建一个新的磁盘索引分区。这个新的倒排文件获得一个临时标签“1”。然后搜索引擎检测目前已创建的所有索引分区，检查是否存在有两个相同标签 g'的分区。如果是，则两个分区合并为一个新的分区，标签为 $g'+1$。重复这个过程直到不存在这种标签冲突为止（也就是，直到任意给定标签 g 最多对应一个索引分区）。

上述基本策略可以通过**预测冲突**（anticipating collision）得到略微改进。例如，当合并两个第g'代的分区时，如果已经存在第 $g'+1$ 代的分区，我们可以简单地在第一次合并操作中包括这个分区，使得无需在第二次合并中对第一次操作产生的冲突又进行处理。不再是合并完两个第 g'代的分区后再合并两个第 $g'+1$ 代的分区，我们只执行一次三路合并操作，将两个第 g'代的分区和一个第$g'+1$ 代的分区合并得到一个新的第 $g'+2$代的分区。

```
buildIndex_LogarithmicMerge (M) ≡
1   I0 ← ∅   // 初始化内存索引
2   currentPosting ← 1
3   while 还有符号单元需索引 do
4      T ← next token
5      I0.addPosting(T, currentPosting)
6      currentPosting ← currentPosting + 1
7      if I0 contains more than M − 1 tokens then
8         // 构建 I，用于合并的索引分区集合
9         I ← {I0}
10        g ← 1
11        while Ig exists do   // 预测冲突
12           I ← I ∪ {Ig}
13           g ← g + 1
14        Ig ← mergeIndices(I)
15        删除每个 I ∈ I
16        I0 ← ∅   // 重置内存索引
17  return
```

图 7-7　使用 LOGARITHMIC MERGE（带预估计的版本）构建在线索引的过程。mergeIndices 过程如图 4-13

形式化过程如图 7-7 所示，给出了 LOGARITHMIC MERGE 的预测版本。在这个算法中，I_g 是第g'代的磁盘索引分区。I_0 是指内存索引。

分析 LOGARITHMIC MERGE 的更新复杂度之前，让我们来看一个具体的实例。假设搜索引擎最多可以在内存中缓存 M 个位置信息。搜索引擎开始为一个给定的文档集构建索引。当积累的位置信息到达 M 个之后，它开始根据内存索引中的数据构建倒排文件。这个

新创建的索引分区获得标签“1”。搜索引擎继续索引其他文档。当第二次耗尽内存时（此时有 $2M$ 个位置信息），它开始创建另外一个索引分区。这个新分区不能标记为“1”，因为已经有一个分区的标签为“1”。因此，它和现有分区合并，产生一个新的分区，标记为“2”。当搜索引擎第三次耗尽内存时（此时有 $3M$ 个位置信息），它又创建一个标记为“1”的新分区。由于此时不存在其他标记为“1”的分区，因此没有必要执行合并操作，系统继续索引文档。整个过程如表 7-1 所示。

表 7-1 由 LOGARITHMIC MERGE 维护的非空索引分区序列。变量 M 代表内存中缓存的位置信息数。“写入位置信息数”这一列给出了每一次合并操作之后写入磁盘的位置信息数的累计和

读入词条数	索引分区						写入位置信息数
	1	2	3	4	5	#	
$0\times M$	·	·	·	·	·	0	0
$1\times M$	*	·	·	·	·	1	$1\times M$
$2\times M$	·	*	·	·	·	1	$3\times M$
$3\times M$	*	*	·	·	·	2	$4\times M$
$4\times M$	·	·	*	·	·	1	$8\times M$
$5\times M$	*	·	*	·	·	2	$9\times M$
$6\times M$	·	*	*	·	·	2	$11\times M$
$7\times M$	*	*	*	·	·	3	$12\times M$
$8\times M$	·	·	·	*	·	1	$30\times M$

这个基本的（即不带预测）LOGARITHMIC MERGE 策略执行的索引维护操作写入磁盘的位置信息总数是多少呢？假设我们为一个包含 $N=2^kM$ 个词条的文档集构建索引（k 为正整数）。当索引过程完成后，就得到一个第 $k+1$ 代的索引分区 I_{k+1}。I_{k+1} 由合并两个第 k 代的索引分区得到。同理，每个第 k 代的分区也是由合并两个第 $k-1$ 代的分区得到，以此类推。

每个合并操作得到第 i 代的分区需要搜索引擎将 $2^{i-1}M$ 个位置信息写回磁盘。因此，LOGARITHMIC MERGE 写入磁盘的位置信息总数为：

$$2^kM+2\cdot(2^{k-1}M+2\cdot(2^{k-2}M+\cdots)) \tag{7-15}$$

$$=\sum_{i=1}^{k+1}2^{k+1-i}\,2^{i-1}\,M \tag{7-16}$$

$$=(k+1)\cdot 2^kM \tag{7-17}$$

$$=\left(\log_2\left(\frac{N}{M}\right)+1\right)\cdot N \tag{7-18}$$

因为 $k=\log_2(N/M)$。

从磁盘**读取**（read）的位置信息数的上界受限于**写入**（written）的位置信息数（不会读两次）。因此，用输入/输出磁盘的总位置信息数来衡量基本的 LOGARITHMIC MERGE 策略的总磁盘复杂度为：

$$D_{\text{LogMerge}}(N)\in\Theta\left(N\cdot\log\left(\frac{N}{M}\right)\right) \tag{7-19}$$

带预测的 LOGARITHMIC MERGE 仅仅是基本版本的常数倍改进，因此它们具有相同的渐

进复杂度（见练习 7.3）。

图 7-8 给出了 LOGARITHMIC MERGE 以及对比方法的更新和查询性能，更新的缓存大小约为 250 000 个文档。LOGARITHMIC MERGE 在两个方面（更新和查询性能）都表现很好。它的累积索引构建代价比 NO MERGE 的仅多出 40%，平均查询响应时间在整个实验中都与 IMMEDIATE MERGE 接近。

图 7-8 中两幅图中的跳跃点是由于合并操作导致的。因为实验中的缓存大小为 250 000 个文档。每当有 $2^k \cdot 250\,000$（例如 400 万、800 万、1600 万）个文档加入索引时，LOGARITHMIC MERGE 就执行一次完全合并操作。这个在磁盘索引中的重合并结果只包含一个分区。在查询性能方面，短时间内 LOGARITHMIC MERGE 和 IMMEDIATE MERGE 没什么差别，因为两种方法都是连续索引。

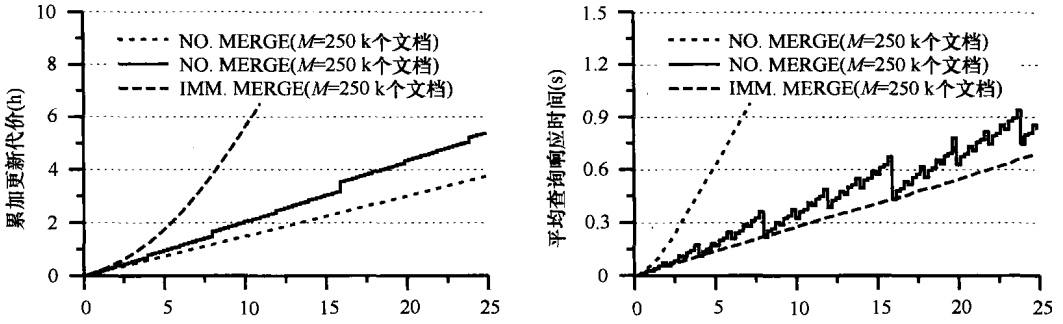


图 7-8　LOGARITHMIC MERGE 的整体性能，对比的方法有 NO MERGE 以及 IMMEDIATE MERGE。数据集：GOV2（2520 万个文档）。水平轴表示被索引的文档数（$\times 10^6$）

7.3　文档删除

尽管对增量式更新的支持已经满足一些应用的需要，但在多数现实情况下搜索引擎还是需要能够处理删除的问题。例如，在 Internet 上常常有一些网页不久就不见了。这样的文档不应再出现在搜索引擎的返回结果列表里。如果删除操作出现的频率很低，并且如果可以接受小部分的检索结果指向不存在的文档，这时 7.1 节的 REBUILD 就是一个很适用的索引维护策略。然而，我们还是建议被删除的文档不应该出现在呈现给用户的检索结果列表中。如果搜索引擎总是让用户链接到不存在的文档，用户很快就会感到失望并且不再使用这个搜索引擎。

7.3.1　无效列表

确保被删除的文档不出现在搜索结果中的一种可能的方法是从索引中移除这些文档对应的位置信息。但是，这需要读取每个位置信息列表，解压后将作废的位置信息移除，然后重新压缩这个列表并写回磁盘。在增量式更新中，每次删除文档之后都进行一次这样的操作是不可行的。另一种做法是，删除文档的信息可以在搜索引擎中累积，在搜索过程中维护一个**无效列表**（invalidation list）来过滤作废的位置信息，或在可将最终搜索结果返回给用户之前过滤掉被删除的文档。

如果搜索引擎基于面向文档的索引，比如文档编号索引或者是词频索引，那么这个无效列表只是文档编号的列表。如果使用的是模式独立索引，那么这个无效列表是形如（start，end）元组的序列，指示了被删除文档在索引地址空间中的起始和结束偏移。

假设为 5 个文档 D_1，…，D_5 建立了一个索引，每个文档分别有 100、200、150、100

和 200 个词条。现在从这个索引中删除 D_2 和 D_5。那么当采用模式独立形式的索引时，这个无效列表为：

$$I = \langle (101, 300), (551, 750) \rangle \tag{7-20}$$

将 I 应用于位置信息列表或者检索结果集不是很复杂。如果实现了 5.2 节中的结构化查询操作，那么只需简单地将查询树上的每个位置信息列表 P 用对应过滤后的结果 $P^{(I)} = (P \not\lhd I)$ 来代替，其中 $\not\lhd$ 指**不包含**（not-contained-in）关系（表 5-2 给出了具体定义）。

在实际应用中，这种方法效果相当不错，特别是当 $\not\lhd$ 操作被设计成**懒惰求值**（lazy evaluation）形式时，允许在过滤后的列表 $P^{(I)}$ 中进行高效的随机访问。然而，如果搜索引擎总是积极地计算整个列表 $P^{(I)}$，而不管查询处理是否真的用到这个列表的时候，例如图 2-5 中的 next 方法这样的高级操作就有可能会失去与简单线性访问方式相比的优势。

尽管使用无效列表的背后基础思想相当简单，但具体的实现细节对这个方法的实际性能也有很大的影响——通常都是这样的。例如，尽管我们说每个位置信息列表 P 都要使用到无效列表 I，但有时对查询树重排序，只在所有查询词项 T_i 对应的位置信息列表 P_i 合并后再应用 I 会更好。重新考虑在莎士比亚文集上执行这个查询 “to be or not to be”，这个词组在文集中只出现了一次（第 1 场，第 3 幕），但单个词项出现的频率却在 2425（“or”）和 19 898（“to”）之间。因此，不要通过对过滤后的位置信息列表求 $P_i^{(z)}$ 交集来寻找上述词组，这样得一遍又一遍地过滤 I，只需要计算未过滤列表 P_i 的交集并在每次计算的最后执行一次 “$(\cdots \not\lhd I)$” 会更有效。

如何将无效列表应用于一个给定查询与传统（即关系型）数据库系统中的优化问题紧密相关。传统数据库系统中，最优查询计划依赖于查询中每个谓词的选择（参考 García-Molina 等人（2002）中关系型数据库系统的介绍）。回到文本检索的语境，这就意味着如果一个词项在删除文档中出现了很多次，那么应直接在这个词项的位置信息列表中运用无效列表 I。

如果用布尔 AND 的方式来处理查询中的词项，那么只在词项上运用一次 I 就足够了，因为对其他词项的操作已经暗含在 AND 操作的语义中。对于布尔 OR 查询，Büttcher 和 Clarke（2005b）文献中提到一个经验法则，建议将限制应用在查询树中尽可能高的地方。然而，对于不同的应用以及不同的查询，最优的选择往往是不同的。

7.3.2 垃圾回收

上面介绍的无效列表方法仅当废弃的位置信息（即被删除的文档对应的位置信息）数比索引中的位置信息总数小时才是可行的。因为倒排列表通常都以信息块的方式来组织，并以块为单位进行压缩（见 4.3 节以及 6.3.2 节）。在查询处理过程中，因为同在一个块的其他有用位置信息需要用来处理查询，所以也不得不将很多废弃的位置信息装进内存并进行解压。如果被删除的文档数随时间不断增加，由废弃位置信息带来的影响会越来越大，最终使查询性能降低到无法接受的程度。因此，某个时候所有废弃的位置信息必须从索引中移除，得到一个新的索引，仅包含那些还存在的文档的位置信息。这个过程称为**垃圾回收**（garbage collection）。

垃圾回收通常是按照逐个列表的方式执行。从抽象的角度来看，它与查询处理并无多大区别：我们只是简单地将无效列表 I 应用于原索引中的每个位置信息列表 P，得到一个过滤之后的列表 P'，并将它加入到新的没有废弃位置信息的索引中。然而，还是有一个细微的不同。因为由垃圾回收过程创建的新索引将包含原索引中的所有位置信息（当然，废弃的位置信息除外），所以没有必要执行懒惰求值 $\not\lhd$ 操作。相反，原索引中的每个位置信息列表 P 可以积极地根据无效列表 I 进行过滤，每次处理一个位置信息。图 7-9 给出了这种逐个列表

处理的垃圾回收过程的一种可能实现，使用了第 2 章基于跳跃式搜索实现的 next 索引访问方法，实现位置信息列表 P 和无效列表 I 的快速交集操作。

```
   collectGarbage (P, I) ≡
1    (startDeleted, endDeleted) ← nextInterval(I, −∞)
2    for i ← 1 to |P| do
3      if P[i] > endDeleted then
4        //找到下一个可能包含P[i]的区间
5        (startDeleted, endDeleted) ← nextInterval(I, P[i] − 1)
6      if P[i] < startDeleted then
7        output P[i]
8      else
9        // 回收当前位置信息

   nextInterval (I, current) ≡
10   intervalEnd ← next(I.end, current)
11   // c_{I.end} 现在包含了位置信息列表 I.end 中缓存的位置信息
12   // (see Figure 2.5 for details)
13   if intervalEnd = ∞ then
14     intervalStart ← ∞
15   else
16     intervalStart ← I.start[c_{I.end}])
17   return (intervalStart, intervalEnd)
```

图 7-9　从一个模式独立的位置信息列表上回收废弃的位置信息。collectGarbage 函数有两个参数：位置信息列表 P 和无效列表 $I=(I.start[], I.end[])$，后者对应着被删除的索引范围。它输出不被 I 所包含的位置信息集合。辅助函数 nextInterval 返回 I 中第一个结束位置在当前索引位置 current 之后的区间。它使用了图 2-5 中的 next 函数

多久调用一次垃圾回收呢？这主要取决于应用环境能接受的查询变慢的程度。例如，如果查询性能降低了 10% 是可接受的，那么就无需在索引只包含 5% 的垃圾位置信息的时候进行垃圾回收。一般来说，建议不要太频繁地进行垃圾回收。垃圾回收是很耗时的，甚至比一般的合并操作代价还要大。除了将旧索引读入内存和将新索引写回磁盘，系统还要解压位置信息和计算列表交集，很容易就比一般的合并更新还要高出 30% ～40% 的代价。因此，为了降低整体的代价，所有的垃圾回收操作必须集成到日常的索引合并操作，使得合并和垃圾回收操作能够共同分担将索引装入内存和写回磁盘的代价。

具体实现的时候，通过设定一个阈值参数 ρ 来定义垃圾回收策略。每当需要执行合并操作且输入索引中的垃圾位置信息数量超过 ρ 的时候，也就是如果

$$\frac{\text{输入索引中的垃圾位置信息数}}{\text{输入索引中的位置信息数}} > \rho$$

那么应该将垃圾回收操作加入到合并操作中。

阈值 ρ 除了会影响查询性能以外，还会影响搜索引擎的索引更新操作性能。如果 ρ 太小，那么垃圾回收操作就会频繁调用，更新性能就会降低。相反，如果 ρ 太大，合并操作可能会处理太多的垃圾位置信息，同样也会影响系统的更新性能。

图 7-10 给出了实验结果，实验中我们让搜索引擎为 50% 的 GOV2 文档（=1260 万个文档）建立索引。索引建立之后，交错地让搜索引擎执行插入和删除命令，逐渐地增加被索引文档集的大小，每 10 次插入操作对应 9 次删除。系统的合并策略设为 IMMEDIATE MERGE，内存足够用来缓存 250 000 个文档的位置信息。根据不同的阈值 ρ 来评价索引更新的整体开销。

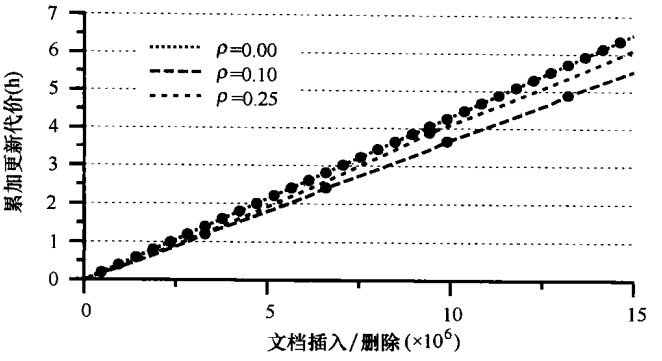


图 7-10 非增量式索引更新，删除/插入比为 0.9（即每 10 次插入对应 9 次删除）。数据集：GOV2。内存限制：$M=250\ 000$ 个文档。包含垃圾回收的合并操作用符号“·”表示

实验结果证实了我们的猜测，每次合并操作中如果主动回收所有的垃圾位置信息，那就达不到最优的更新性能。实际上，实验中设置 $\rho=0$ 得到的索引更新开销比 $\rho=0.1$ 时高出几乎 20%。而第三个测试值 $\rho=0.25$，得到的更新性能位于这两者之间，其优势是垃圾回收的频率不高，但合并操作中处理了过多的垃圾位置信息。

如何找出使整体更新代价最小的阈值？我们知道没有垃圾回收的普通合并操作的代价大致与输入和输出索引的位置信息数成线性关系：

$$C_{\text{merge}}(I_{\text{in}}, I_{\text{out}})=c\cdot(\#\text{postings}(I_{\text{in}})+\#\text{postings}(I_{\text{out}})) \tag{7-21}$$

c 是系统指定的常数。如上所述，集成垃圾回收操作使合并操作的代价增加了 30%～40%。然而，垃圾回收过程的主复杂度主要源于用无效列表过滤输入的位置信息列表，因此很大程度上独立于输出索引 I_{out} 的大小。因此我们可以假设集成有垃圾回收的合并操作的代价大致为：

$$C_{\text{merge+gc}}(I_{\text{in}}, I_{\text{out}})=c\cdot(1.6\cdot\#\text{postings}(I_{\text{in}})+\#\text{postings}(I_{\text{out}})) \tag{7-22}$$

c 与前面的一样。

让我们忽略与删除文档无关的有效位置信息所带来的更新代价，仅关注索引中垃圾位置信息的影响。对一个一般的合并操作，由垃圾位置信息所引起的开销与它们在索引中所占比例有直接关系：

$$O_{\text{merge}}(I_{\text{in}}, I_{\text{out}})=c\cdot(\#\text{garbage}(I_{\text{in}})+\#\text{garbage}(I_{\text{out}})) \tag{7-23}$$

不执行垃圾回收时，我们有 $\#garbage(I_{\text{in}})=\#garbage(I_{\text{out}})$，从而得到：

$$O_{\text{merge}}(I_{\text{in}}, I_{\text{out}})=2c\cdot\#\text{garbage}(I_{\text{in}}) \tag{7-24}$$

对于具有垃圾回收的合并操作，开销为：

$$O_{\text{merge+gc}}(I_{\text{in}}, I_{\text{out}})=c\cdot(\#\text{garbage}(I_{\text{in}})+0.6\cdot\#\text{postings}(I_{\text{in}})) \tag{7-25}$$

因为系统首先要读取垃圾位置信息（除有效位置信息之外），然后用无效列表过滤所有位置信息，不管是有效的还是废弃的。这就占用了 60% 的开销。

现在假设按照每 n 次合并操作就执行一次垃圾回收的方式来选择阈值 ρ。进一步假设最开始索引包含 P 个位置信息，没有垃圾位置信息，并假设两次连续的合并操作之后，搜索引擎会收集到 M 个位置信息，文档删除与插入的相对比率为 g（例如，$g=0$ 意味着没有删

除操作；$g=1$ 意味着系统在**稳定状态**（steady state），索引中文档数保持为一个常数）。那么参与第 k 次（$k \leqslant n$）合并操作的垃圾位置信息数为 $k \cdot g \cdot M$。因此，包含垃圾回收操作的连续 n 次合并操作（前 $n-1$ 次合并操作没有垃圾回收操作，最后一次合并操作有）的总开销为：

$$O_{\text{total}}(n) = \sum_{k=1}^{n-1}(2kgM) + ngM + 0.6 \cdot (P + nM) \tag{7-26}$$

$$= (n-1)ngM + ngM + 0.6 \cdot (P + nM) \tag{7-27}$$

$$= n^2gM + 0.6 \cdot nM + 0.6 \cdot P \tag{7-28}$$

（忽略常数 c，因为公式各项中 c 值都一样）。每 n 次合并操作的平均开销为：

$$O_{\text{avg}}(n) = \frac{O_{\text{total}}(n)}{n} = ngM + 0.6 \cdot M + 0.6 \cdot \frac{P}{n} \tag{7-29}$$

寻找使 $Q_{\text{avg}}(n)$ 最小的最优 n 值等价于寻找令导数 $\frac{\mathrm{d}}{\mathrm{d}n}\left(Q_{\text{avg}}(n)\right)$ 为 0 的 n 值：

$$\frac{\mathrm{d}}{\mathrm{d}n}(O_{\text{avg}}(n)) = 0 \quad \Leftrightarrow \quad gM - 0.6 \cdot \frac{P}{n^2} = 0 \quad \Leftrightarrow \quad n = \sqrt{\frac{0.6 \cdot P}{gM}} \tag{7-30}$$

根据这个不等式，我们可以计算最优阈值 p_{opt}：

$$\frac{(n_{\text{opt}} - 1) \cdot gM}{P + (n_{\text{opt}} - 1) \cdot M} < \rho_{\text{opt}} < \frac{n_{\text{opt}} \cdot gM}{P + n_{\text{opt}} \cdot M} \tag{7-31}$$

例如，如果没有文档被删除（$g=0$），我们可以得到 $n_{\text{opt}}=\infty$，$\rho_{\text{opt}}=$未定义，也就是，不应该进行垃圾回收操作。如果 $P=50 \cdot M$ 且 $g=0.9$，正如图 7-10 所示的实验中那样，最优值 $n_{\text{opt}}=\sqrt{33.3} \approx 6$，$0.082 < \rho_{\text{opt}} < 0.096$。实验中，$\rho=0.1$ 得到的效果最好，每 7 次合并操作之后触发一次垃圾回收操作，接近最优值。

值得一提的是，因为 n_{opt} 仅随 $\sqrt{P}$ 增长，当索引越来越大时，P 就成为公式（7-31）的主导项，而 ρ_{opt} 会变小。因此，仅当索引的大小预计几乎不随时间变化时，才建议垃圾回收阈值 ρ 取常数。

双层垃圾回收策略

到目前为止，我们只讨论了 IMMEDIATE MERGE 作为主要更新策略时的垃圾回收方法。当然，垃圾回收还可以集成到那些维护多个索引分区的更新模式中，比如 LOGARITHMIC MERGE，但是这么做会稍微复杂些。

如果索引采用 IMMEDIATE MERGE 方法来维护，那么我们可以肯定时常发生的合并操作会涉及索引中的所有位置信息。在合并操作开始前，搜索引擎根据内部删除日志计算垃圾位置信息和索引中所有位置信息的比率。如果这个比率超过垃圾回收阈值 ρ，那么合并操作就会包括垃圾回收。

假设系统处在稳定状态，删除文档的比率与插入文档的比率大致相等。那么，我们知道搜索引擎在两次连续的合并操作之间会收集到大概 M 个垃圾位置信息（M 是系统的内存容量，用位置信息数来衡量）。如果 M 比索引规模 N 小很多，如 $N>10M$，那么我们知道在两次合并之间，索引中垃圾位置信息率的增长不会超过 10%。这使我们能够很好地控制垃圾位置信息的总数。

如果使用索引分割方案，单个合并操作不再需要涉及整个索引。因此，很有可能垃圾位置信息都集中在某个合并操作不会处理到的旧分区中。现在假设使用 LOGARITHMIC

MERGE 更新策略，搜索引擎正好完成了一次合并操作，创建了一个新的第 g 代索引分区 I_g，它包含 2^{g-1} 个位置信息。在 I_g 中的位置信息参与下次合并操作之前，搜索引擎要为 $2^{g-1}M$ 个词条建立索引。因为索引处在稳定状态，此时 $2^{g-1}M$ 个位置信息将被废弃。在这个假设下，废弃的位置信息在整个索引中是均匀分布的，索引中的 $2^{g-1}M$ 个位置信息将出现在 I_g 中。因此，I_g 中的垃圾量为 50%，整个索引中的垃圾量在 33%～50% 之间，具体取决于从更早的索引分区 I_1，…，I_{g-1} 中移除垃圾位置信息的频率。对很多应用而言，这有些太频繁了。

上述问题可以通过定义两个阈值 ρ 和 ρ'（$0 \leqslant \rho < \rho'$）来解决。每当系统要合并索引分区集合 S 时，系统将会采取下面的三种方式之一：

1）如果 S 中垃圾位置信息的比例超过 ρ，那么合并操作中加入垃圾回收。

2）如果在整个索引中垃圾位置信息的比例超过 ρ'，那么系统将合并索引中的所有分区（而不只是 S），收集它能找到的所有垃圾位置信息并产生一个没有分段和垃圾的新索引。

3）如果以上条件均不满足，那么搜索引擎对 S 中的索引分区执行一个普通的合并操作，不处理任何垃圾信息。

这种双层垃圾回收机制确保了不会有大量的垃圾位置信息集中在旧的索引分区中。不利的一面是，它抵消了一些索引分割方案（如 LOGARITHMIC MERGE）带来的性能优势，因为这样执行的完全合并操作就更多了。

最后要说明的是，有必要指出 7.2.2 节描述的最基本的 LOGARITHMIC MERGE 方法即便是使用双层垃圾回收策略，在有文档删除的情况下，性能也不会很好。如果合并操作输出的索引分区比输入分区还小（如果索引中存在很多垃圾位置信息时，这可能会发生），就不存在**索引分代**（index generation）的概念了。图 7-7 中的算法需要做一定的修改，使得在决定哪些分区加入到下一次的合并操作时，考虑每个分区的大小（而不是它们的代数）。

7.4 文档修改

第三种更新操作类型就是文档修改了。很多搜索引擎都没有使用专门的策略来处理这种类型的更新，而是将每次的修改视为删除之后又插入当前文档的新版本。如果每个文档与文档集相比是很小的，那么这种方法的性能通常都是可接受的。然而，当对相对较大的文档做相对较小的修改时，这种方法就不适用了。

作为一个说明问题的例子，考虑 UNIX 风格的操作系统所维护的日志文件（比如 Linux 中的/var/log/messages）。当检索这些日志文件时，能同时匹配多个日志项的检索操作通常是很有用的——例如，确定未授权入侵之类的重复模式。尽管这些日志文件整体很大，可能有几十或几百兆字节，但每次更新只涉及一行内容，一般只有几十或几百字节。每次更新后都重新索引整个文件很显然是不可行的。将日志文件的每一行都视为一个单独的文件也不是一个好方法。这使得我们无法在连续多行上指定检索约束条件。如果有一种索引更新机制能够反映出对这种文件进行更新操作的特点，那么将会很方便。

遗憾的是，使用本章前面几节介绍过的索引维护框架将很难找到这样的更新方法。例如，假设我们处理一个位置索引包括以下形式的位置信息：

$$(d,\ f_{t,d},\ \langle p_1, \cdots, p_{f_{t,d}} \rangle) \tag{7-32}$$

搜索引擎使用 7.2.2 节中的 LOGARITHMIC MERGE 更新策略。如果搜索引擎对文件追加操作只是简单地索引新的词条，并当内存耗尽时将对应的位置信息写入磁盘，那么最后会有许多个具有（term，document）形式的位置信息存在，每一个索引分区中都会有一个。这

增加了查询处理以及索引更新的复杂度：

- 如果搜索引擎维护 n 个独立的索引分区，那么调用 next 方法一次需要 n 次随机访问操作，而不只一次，因为这 n 个索引分区中的每个都可能包含结果的一部分。
- 7.3.2 节中的垃圾回收机制不能只在一个索引分区上执行，因为被删除文档的位置信息可能分散在整个索引中。这意味着每次调用垃圾回收都会触发一次整个索引上的合并操作。

Büttcher（2007，第 6 章）给出了一种带动态地址空间转换的基于模式独立索引的更新方法，可以在一定程度上解决上面的问题。然而，代价是更多的内存消耗和查询处理性能的降低，因此并不总是可取的。

如何以满意的方式处理任意文档的修改（或上述的简单追加操作），仍然是个不明朗的问题，因此通常就只是忽略它。例如，很多桌面检索系统就只索引每个文件/文档前面的几千字节内容，而忽略后面的内容。这使得这些系统可以把文件的修改看做是删除和插入操作的组合，而又无需对好几兆内容进行重新索引。

7.5　讨论及延伸阅读

Cutting 和 Pedersen（1990）的一篇文献首次对就地更新以及合并更新的索引维护策略进行了比较性评价，他们发现合并更新性能通常更好。他们的就地更新的索引维护策略的实现基于存储于磁盘的 B-树，使用内存缓存近期访问的树节点，与今天使用的就地更新策略有很大的不同。Lester 等人（2004，2006），在十多年后对多种不同的更新策略进行了最新的实现，对比了 INPLACE、REBUILD 以及 REMERGE 方法的性能。他们的实现证实了 Cutting 和 Pedersen 的发现，仅当内存资源稀少但需要频繁更新磁盘索引时，就地更新策略才具有竞争力。

Zobel 等人（1993）讨论了一种用于变长记录的存储管理策略及其在就地更新索引维护策略中的应用。Shieh 和 Chung（2005）描述了 INPLACE 的一种变形，其中预分配因子是依赖于词项的，由带常数预分配因子的反向 INPLACE 方法得到的历史数据来计算得到。

7.2.1 节中的混合索引维护策略来源于 Büttcher 和 Clarke（2006，2008），它将索引分为就地更新部分和合并维护部分。这种方法的精髓与 Cutting 和 Pedersen（1990）一篇文章中描述的**脉冲**（pulsing）方法类似。Shoens 等人（1994）以及 Tomasic 等人（1994）对相关的双索引组织结构进行了阐述。

Büttcher 和 Clarke（2005a）研究了 logarithmic 合并方法，一般的形式称为**几何分割**（geometric partitioning），Lester 等人（2005）提出了这个概念。两项研究都发现了这种方法能够显著减少搜索引擎的索引维护开销，代价是查询性能的略微下降，一般不超过 20%。他们得出结论：折中索引时间和查询时间两个考虑因素，logarithmic 合并通常是个不错的选择。Büttcher 等人（2006）讨论了 LOGARITHMIC MERGE 的一些混合应用。

尽管索引分割策略如 LOGARITHMIC MERGE 具有很大的优势，但只有当索引是一个单纯的倒排索引时，才能完全地发挥它们的优势。在一些应用中，如 Web 搜索，索引中还包含一些用于排名的辅助信息，如锚文本、PageRank 分值等（见第 15 章）。在这种情况下，加入一个文档 D_1 可能会影响到 D_2 对应的索引数据，因为 D_1 可能会包含指向 D_2 的链接。插入文档不再是一个只影响单个文档的操作，而是必须被当做为一个具有全局影响的操作，这使得只有 INPLACE/REBUILD/REMERE 才是可行的方案。而另一方面，为了提高查询性能，系统可能会决定忽略那些跟应用相关的索引组成部分，并且只是偶尔更新锚信息和

PageRank 分值。然而，这种方法对搜索引擎的效率以及用户的满意度的影响还尚未研究。

Chiueh 和 Huang（1998）以及 Büttcher 和 Clarke（2005a）讨论了几种处理文档删除的方法。Chiueh 和 Huang 的方法特别有趣。他们提出垃圾回收不仅要集成到搜索引擎的合并操作中，而且还应合并到查询处理逻辑中：每当从索引中获取一个查询词项的位置信息列表，就用当前的无效列表来进行过滤，再将结果列表放回索引中。

Lim 等人（2003）讨论了一种方法，可用于加速文档集中内容保真变换（即文档修改）的索引维护。他们指出他们的 landmark-diff 方法与基本的 REBUILD 方法相比，能够减少大概 50% 的索引维护开销。然而，考虑到总的索引构建代价大部分源于读入和分词输入数据（见 4.5.3 节），并且他们论文中的性能没有把得到旧文档与新文档之间的差别所用的时间算入进去，因此，很难说在实际应用中这种方法将会怎样。

在对索引维护策略进行讨论时，我们忽略了块索引更新的含义。例如，图 7-2 中的 IMMEDIATE MERGE 算法，每当内存耗尽时执行一次合并操作。当在执行合并操作时，不允许新的位置信息加入到索引中。在 GOV2 的例子中，将内存索引与磁盘上已有的索引合并可能需要一个小时——很难快到可以成为**实时**（real-time）操作。Strohman（2005）描述了很多用来支持真正的实时更新和查询操作的技术（如背景磁盘 I/O）。

最后再说明一下，有必要指出本章介绍的所有索引维护机制的性能与索引所使用的压缩技术紧密相关。例如，如果合并操作的索引对象未压缩或是压缩得不好，那么很容易就需要花两倍的时间，因为磁盘 I/O 增加了。越好的压缩率可直接导致越好的更新性能。另一方面，对于垃圾回收，相当一部分的复杂度来源于对位置信息列表的解压和再压缩，因此，选择一个可以高效编码和解码的压缩方法是很重要的；这时压缩率倒没有那么重要。特别地，无参数编码，如 vByte（见 6.3.4 节），应该比带参数的编码，如 LLRUN（见 6.3.2 节）更受青睐，因为当编码一个给定列表时，前者只需要在输入数据上扫描一遍。

7.6 练习

练习 7.1 假设搜索引擎采用就地更新策略来维护磁盘上的索引。内存足以用来缓存 1000 万个位置信息，且搜索引擎正在索引的文档集的齐夫参数 $\alpha=1.33$（GOV2）。请问，当搜索引擎耗尽内存时，需要创建或者更新多少个磁盘上的倒排列表？如果每个列表更新都需要一次磁盘寻道，每次磁盘寻道需要 10 ms，那么更新所有列表需要的磁盘寻道时间为多少？

练习 7.2 假设搜索引擎采用按比例预分配（预分配因子：$k=2$）的 INPLACE 策略。证明：给定任意列表，传出/传入磁盘的总字节数小于 $5\times s$，s 是磁盘列表的字节数。你可以假设所有的位置信息的大小相等（比如 4 字节）。

练习 7.3 7.2.2 节给出了基本的 LOGARITHMIC MERGE 策略的索引构建复杂度为 $\Theta(N \cdot \log(N/M))$。证明：带预测的 LOGARITHMIC MERGE 版本执行避免标签冲突的多路合并的时间复杂度也是 $\Theta(N \cdot \log(N/M))$。

练习 7.4（项目练习） 实现 7.2.1 节中的 IMMEDIATE MERGE 索引维护策略。实现之后，你的搜索引擎就能够在文档被加入到索引中后立即在新加入的文档上作查询，而不需等下一次的合并操作。

7.7 参考文献

Büttcher, S. (2007). *Multi-User File System Search.* Ph.D. thesis, University of Waterloo, Waterloo, Canada.

Büttcher, S., and Clarke, C. L. A. (2005a). *Indexing Time vs. Query Time Trade-offs in Dynamic Information Retrieval Systems.* Technical Report CS-2005-31. University of Waterloo, Waterloo, Canada.

Büttcher, S., and Clarke, C. L. A. (2005b). A security model for full-text file system search in multi-user environments. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies*, pages 169–182. San Francisco, California.

Büttcher, S., and Clarke, C. L. A. (2006). A hybrid approach to index maintenance in dynamic text retrieval systems. In *Proceedings of the 28th European Conference on Information Retrieval*, pages 229–240. London, England.

Büttcher, S., and Clarke, C. L. A. (2008). Hybrid index maintenance for contiguous inverted lists. *Information Retrieval*, 11(3):175–207.

Büttcher, S., Clarke, C. L. A., and Lushman, B. (2006). Hybrid index maintenance for growing text collections. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 356–363. Seattle, Washington.

Chiueh, T., and Huang, L. (1998). *Efficient Real-Time Index Updates in Text Retrieval Systems.* Technical report. SUNY at Stony Brook, Stony Brook, New York.

Cutting, D. R., and Pedersen, J. O. (1990). Optimization for dynamic inverted index maintenance. In *Proceedings of the 13th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 405–411. Brussels, Belgium.

García-Molina, H., Ullman, J., and Widom, J. (2002). *Database Systems: The Complete Book.* Upper Saddle River, New Jersey: Prentice Hall.

Lester, N., Moffat, A., and Zobel, J. (2005). Fast on-line index construction by geometric partitioning. In *Proceedings of the 14th ACM Conference on Information and Knowledge Management*, pages 776–783. Bremen, Germany.

Lester, N., Zobel, J., and Williams, H. E. (2004). In-place versus re-build versus re-merge: Index maintenance strategies for text retrieval systems. In *Proceedings of the 27th Conference on Australasian Computer Science*, pages 15–22. Dunedin, New Zealand.

Lester, N., Zobel, J., and Williams, H. E. (2006). Efficient online index maintenance for contiguous inverted lists. *Information Processing & Management*, 42(4):916–933.

Lim, L., Wang, M., Padmanabhan, S., Vitter, J. S., and Agarwal, R. (2003). Dynamic maintenance of web indexes using landmarks. In *Proceedings of the 12th International Conference on World Wide Web*, pages 102–111. Budapest, Hungary.

Shieh, W. Y., and Chung, C. P. (2005). A statistics-based approach to incrementally update inverted files. *Information Processing & Management*, 41(2):275–288.

Shoens, K. A., Tomasic, A., and García-Molina, H. (1994). Synthetic workload performance analysis of incremental updates. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 329–338. Dublin, Ireland.

Strohman, T. (2005). *Dynamic Collections in Indri.* Technical Report IR-426. University of Massachusetts Amherst, Amherst, Massachusetts.

Tomasic, A., García-Molina, H., and Shoens, K. (1994). Incremental updates of inverted lists for text document retrieval. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 289–300. Minneapolis, Minnesota.

Zobel, J., Moffat, A., and Sacks-Davis, R. (1993). Storage management for files of dynamic records. In *Proceedings of the 4th Australian Database Conference*, pages 26–38. Brisbane, Australia.

第三部分　检索和排名

第8章

Information Retrieval: Implementing and Evaluating Search Engines

概率检索

在这一章和下一章，我们将检视几个对研究和实际都产生重大影响的信息检索模型。本章我们讨论**概率模型**（probabilistic model）。第9章我们将涵盖**语言模型**（language modeling）及相关方法。在信息检索中，“模型”一词至少有两个不同的重要含义（Ponte 和 Croft，1998）。其中一个含义表示“检索任务本身的一个抽象”（例如向量空间模型）。正如我们将看到的那样，当涉及相关性模型或文档内容模型时，它也用于表示“统计模型”。

尽管很多信息检索模型表达为概率的形式，但这个词往往与 Stephen Robertson，Karen Spärck Jones 以及他们在伦敦城市大学、剑桥大学、剑桥微软研究院的同事从 1970 早期开始提出的方法相关联。在那段时间里，他们通过一系列的创新性工作提出了概率检索模型，并将其发展为至今最成功的信息检索模型之一。很多其他的研究小组将这个模型与自己提出的检索公式进行结合，用于实验性的信息检索系统，并由此评价他们提出的方法。

本章的组织紧紧围绕这个模型的发展历史，依次关注每一个重要的扩展和创新。本章的大部分内容都建立在公式（8-48）上，即 BM25 公式，它代表了当前在该领域内性能最好（也是最著名）的检索公式。尽管使用 BM25 公式的时候可以不必了解它的发展过程，但出于它本身的缘故和考虑到作为多数信息检索理论的推理过程的样例，我们还是给出一些关于它的发展细节。

本章包括了相关性反馈的一个综述，它是与概率和向量空间模型紧密相关的一个查询扩展技术。我们还介绍了 BM25 公式的一个重要扩展——BM25F，它通过给出现在文档不同部分的查询词项赋予不同的权重，从而可以识别出现在文档主题中的词项比出现在文档正文中的词项要指示更高的相关性。

8.1　相关性建模

概率模型从第 1.2.3 节中介绍的概率排名原则（Probability Ranking Principle）开始：

如果一个信息检索系统对查询的响应是按文档集中文档的相关性概率递减排名的，那么对用户来说，系统的整体有效性就达到最大。

正如在第 1.2.3 节中介绍的那样，这一原则忽略了很多如新颖性、特异度和详尽性等的问题，由此在很多方面简化了检索问题。尽管如此，它还是直接为我们勾勒了一个检索算法的轮廓：给定一个查询，决定每个文档在文档集中的相关性概率，并依次对文档进行排序。这个算法的实现马上就把我们带到 Spärck Jones 等人（2000a）所说的“基本问题”上去了，这里我们稍微改变了一下陈述形式：

用户判断**这个**文档和**这个**查询相关的概率是多少?

我们现在转向如何估计这个概率的问题。

采用 Lafferty 和 Zhai (2003) 所讨论的概率模型方法，我们引入三个随机变量来将基本问题转化为统计表示：D 是文档、Q 是查询、二元随机变量 R 是用户对相关性的判断。对于随机变量 D，作为搜索引擎索引的文档集，你可以看到它的样本空间或结果集。然后你可以想象一组用户在同一个搜索引擎中输入查询，这些查询是搜索引擎允许用户输入的形式，即样本空间 Q。搜索引擎可能会要求查询的形式是词项向量，但现在我们不考虑这个限制。最后，二元随机变量 R 的值为 1 或 0，表示相关或不相关。用以上这些表述，我们就将基本问题转换为估计一个正相关性判断的概率问题，如下表示：

$$p(R=1|D=d,Q=q) \tag{8-1}$$

该公式可视为是文档和查询上的概率密度函数。为了和研究文献中的符号表示一致，我们通常用“D”代表“$D=d$”和用“Q”代表“$Q=q$”；因此

$$p(R=1|D,Q) \tag{8-2}$$

同样，我们用“r”表示“$R=1$”和“$\bar{r}$”来表示“$R=0$”。因此，我们有：

$$p(r|D,Q)=1-p(\bar{r}|D,Q) \tag{8-3}$$

贝叶斯定理（一个概率论中的基本定理）阐述如下：

$$p(A|B)=\frac{p(B|A)\,p(A)}{p(B)} \tag{8-4}$$

对公式 (8-3) 应用贝叶斯定理可以得到：

$$p(r|D,Q)=\frac{p(D,Q|r)\,p(r)}{p(D,Q)} \tag{8-5}$$

$$p(\bar{r}|D,Q)=\frac{p(D,Q|\bar{r})\,p(\bar{r})}{p(D,Q)} \tag{8-6}$$

现在，我们不继续讨论概率，我们先来看看**对数比值** (log-odds)，或 logit，它可以简化公式的表达和操作。给定一个概率 p，p 的 logit 定义为

$$\text{logit}(p)=\log\left(\frac{p}{1-p}\right) \tag{8-7}$$

其中对数的底可任意选择。本书中如果要为例子或实验定一个底，我们选择 2 为底。

log-odds 拥有很多有用的性质。当 p 在 0～1 之间变化时，logit(p)从$-\infty$～∞之间变化。如果比值均等（即 $p=0.5$），则 logit(p)=0。给定两个概率 p 和 q，logit(p)> logit(q)当且仅当 $p>q$。因此 log-odds 和概率是**等秩** (rank-equivalent) 的——对一个排名与对另一个排名得到的结果是一样的。等秩变换（也成为**保秩** (rank-preserving) 或**保序** (order-preserving) 变换）对于简化信息检索模型的发展是非常有用的工具。

取公式 (8-2) 的 log-odds，运用贝叶斯定理（公式 (8-5) 和 (8-6)），可得：

$$\log\frac{p(r|D,Q)}{1-p(r|D,Q)}=\log\frac{p(r|D,Q)}{p(\bar{r}|D,Q)} \tag{8-8}$$

$$=\log\frac{p(D,Q|r)\,p(r)}{p(D,Q|\bar{r})\,p(\bar{r})} \tag{8-9}$$

转换成 log-odds 的直接好处就是去掉了词项 $p(D,Q)$，不用估计它了。

我们将公式 (8-8) 中的联合概率用等价式 $p(D,Q|R)=p(D|Q,R)\cdot p(Q|R)$扩展

为条件概率。这样扩展条件概率和第二次运用贝叶斯定理，可得：

$$\log \frac{p(D,Q|r)\,p(r)}{p(D,Q|\bar{r})\,p(\bar{r})} = \log \frac{p(D|Q,r)\,p(Q|r)\,p(r)}{p(D|Q,\bar{r})\,p(Q|\bar{r})\,p(\bar{r})} \tag{8-10}$$

$$= \log \frac{p(D|Q,r)\,p(r|Q)}{p(D|Q,\bar{r})\,p(\bar{r}|Q)} \tag{8-11}$$

$$= \log \frac{p(D|Q,r)}{p(D|Q,\bar{r})} + \log \frac{p(r|Q)}{p(\bar{r}|Q)} \tag{8-12}$$

词项“$\log(p(r|Q)/p(\bar{r}|Q))$”独立于 D。你可以将其视为一个查询难度的指标。若为了排名，在任何情况下都可以忽略它——另一种保序转换——留给我们排名公式

$$\log \frac{p(D|Q,r)}{p(D|Q,\bar{r})} \tag{8-13}$$

这个式子是概率检索模型的核心，当我们研究估计它的几种方法时会再次提到它。

8.2　二元独立模型

在我们第一次为公式（8-13）估计值时，只考虑文档和查询里词项的存在或缺失。使用二元随机变量的向量形式重新定义代表文档的随机变量 D，即 $D=\langle D_1, D_2, \cdots\rangle$，其中每一维代表一个词汇表 $\mathcal{V}$ 中的一个词项，$D_i=1$ 表示对应词项在文档中存在，$D_i=0$ 表示对应词项在文档中缺失。类似的，我们使用二元随机变量的向量形式重新定义代表查询的随机变量 Q，即 $Q=\langle Q_1, Q_2, \cdots\rangle$，其中 $Q_i=1$ 表示对应词项在查询中存在，而 $Q_i=0$ 表示对应词项在查询中缺失。

现在我们给出两个强假设。其中第一个是独立假设：

假设 T：给定相关性，词项在统计上互相独立。

换句话说，给定正相关判断，一个词项的存在或缺失并不依赖于其他任何词项的存在或缺失。相似的，给定负相关判断，一个词项的存在或缺失也不依赖于任何其他词项的存在或缺失。自然的，如果不在相关性上给予条件，词项之间并不是相互独立的，因为往往有多个词项的出现概率依赖于相关性。

这个假设并没有准确地反映现实。例如，“shakespare”这个词项在文档中的存在将会提高词项“william”、“hamlet”及“stratford”等在文档中出现的概率。但这个假设又确实简化了为公式（8-13）估计值的问题。与这个假设相仿的独立假设在信息检索中比较常见。在这个假设之下建立的方法尽管含有上述不现实的特征，通常却提供了较好的效果。

作为上述假设的结果，我们可以重写公式（8-13）中的概率，将其转化为 D 中各维度上单个随机变量概率的积：

$$p(D|Q,r) = \prod_{i=1}^{|\mathcal{V}|} p(D_i|Q,r) \tag{8-14}$$

$$p(D|Q,\bar{r}) = \prod_{i=1}^{|\mathcal{V}|} p(D_i|Q,\bar{r}) \tag{8-15}$$

公式（8-13）变为：

$$\log \frac{p(D|Q,r)}{p(D|Q,\bar{r})} = \sum_{i=1}^{|\mathcal{V}|} \log \frac{p(D_i|Q,r)}{p(D_i|Q,\bar{r})} \tag{8-16}$$

第二个强假设将一个词项在查询中的出现与它在一个相关文档中的出现概率联系起来，从而明晰了查询应该扮演用户需求与文档相关性之间桥梁的角色。

假设 Q：只有一个词项出现于查询中时，它在文档中的存在才依赖于相关性。

为了形式化这个假设，我们需要调整查询的定义，给定 $Q=q=\langle q_1, q_2, \cdots\rangle$，其中 q_i 或为 0 或为 1。根据假设 Q，如果 $q_i=0$，则

$$p(D_i|Q,r)=p(D_i|Q,\bar{r})$$

所以

$$\log\frac{p(D_i|Q,r)}{p(D_i|Q,\bar{r})}=0$$

这个假设的效果是将公式（8-16）中对所有词汇表中词项求和转化为对所有查询中词项求和。因为对查询进行条件分析现在已是多余，我们可以将其忽略，排名公式将变为

$$\sum_{t\in q}\log\frac{p(D_t|r)}{p(D_t|\bar{r})} \tag{8-17}$$

其中 D_t 是向量$\langle d_1, d_2, \cdots\rangle$中词项 t 对应的随机变量。

同假设 T 一样，假设 Q 也没有准确地反映现实。假设我们对 William Shakespare 的婚姻有兴趣，键入查询

$$\langle \text{“william”}, \text{“shakespeare”}, \text{“marriage”}\rangle$$

一个相关文档与一个不相关文档比起来，将有更大的概率包含词项 “hathaway”，即使这个词项并没有出现在查询中。但是，从实际的角度来说，对查询中所有词项求和要比对词汇表中所有词项求和更可行，我们欢迎这个进展，它在某种程度上减轻了计算负担。

我们现在调整 $D=d=\langle d_1, d_2, \cdots\rangle$，其中 d_i 或为 0 或为 1，因此将随机变量的值显式表示出来了。扩展我们早先的符号，用 $D_t=d_t$ 表示$\langle d_1, d_2, \cdots\rangle$中对应词项 t 的随机变量值：

$$\sum_{t\in q}\log\frac{p(D_t=d_t|r)}{p(D_t=d_t|\bar{r})} \tag{8-18}$$

当所有的查询词项都不出现在文档中时所有 D_t 等于 0，从公式（8-18）中减去它本身的值：

$$\sum_{t\in q}\log\frac{p(D_t=d_t|r)}{p(D_t=d_t|\bar{r})}-\sum_{t\in q}\log\frac{p(D_t=0|r)}{p(D_t=0|\bar{r})} \tag{8-19}$$

因为公式（8-18）的值在所有查询词项都不出现的情况下是一个常数，且同时减去一个常数对排名并无影响，这个减法代表了另一个保序变换。进行一些小调整：

$$\sum_{t\in(q\cap d)}\log\frac{p(D_t=1|r)\,p(D_t=0|\bar{r})}{p(D_t=1|\bar{r})\,p(D_t=0|r)}-\sum_{t\in(q\setminus d)}\log\frac{p(D_t=0|r)\,p(D_t=0|\bar{r})}{p(D_t=0|\bar{r})\,p(D_t=0|r)} \tag{8-20}$$

其中左边的求和包含了所有出现在查询和文档中的词项，右边的求和包含了所有在查询中出现但未在文档中出现的词项。因为右边的词项为 0，公式变为：

$$\sum_{t\in(q\cap d)}\log\frac{p(D_t=1|r)\,p(D_t=0|\bar{r})}{p(D_t=1|\bar{r})\,p(D_t=0|r)} \tag{8-21}$$

这种在假设 T 与假设 Q 下，只考虑词项的存在与缺失，对公式（8-13）进行改进的方法称之为**二元独立模型**（Binary Independence Model）。

8.3 Robertson/Spärck Jones 权重公式

为简洁起见，我们现在大幅调整符号。去除公式（8-21）中的杂乱表示，我们将其重

写为

$$\sum_{t\in(q\cap d)} w_t \tag{8-22}$$

其中 w_t 是每个词项的权重。如果我们让 $p_t = p(D_t=1 \mid r)$ 和 $\bar{p}_t = p(D_t=1 \mid \bar{r})$，则 w_t 变为

$$w_t = \log \frac{p_t\ (1-\bar{p}_t)}{\bar{p}_t\ (1-p_t)} \tag{8-23}$$

因为 $p(D_t=0 \mid r)=1-p(D_t=1 \mid r)=1-p_t$ 且 $p(D_t=0 \mid \bar{r})=1-p(D_t=1 \mid \bar{r})=1-\bar{p}_t$。

如果我们能估计文档集中相关文档的数量期望与包含 t 的相关文档的数量期望，那么对 p_t 与 $\bar{p}_t$ 的估计也可以确定。这些估计，可以通过例如对文档集进行采样并对样本进行相关判断来取得。虽然处理每个查询时都使用这样的方式会变得异常地繁重，但如果我们多次处理同一个查询，这样的方式也不是不可取，因为文档集会周期性的变化。

给定一个查询，用 N_r 表示文档集中相关文档的数量期望，$N_{t,r}$ 表示包含 t 的相关文档的数量期望。我们可以估计 p_t 与 $\bar{p}_t$ 如下

$$p_t = \frac{N_{t,r}}{N_r} \quad 且 \quad \bar{p}_t = \frac{N_t - N_{t,r}}{N - N_r} \tag{8-24}$$

其中，N 是文档集中文档的数量，N_t 是包含 t 的文档数量。结合公式（8-23），权重如下

$$w_t = \log \frac{N_{t,r}\ (N - N_t - N_r + N_{t,r})}{(N_r - N_{t,r})\ (N_t - N_{t,r})} \tag{8-25}$$

当我们知道实际的相关文档数量和不相关文档数量时，这个公式通常写为

$$w_t = \log \frac{(n_{t,r}+0.5)\ (N - N_t - n_r + n_{t,r}+0.5)}{(n_r - n_{t,r}+0.5)\ (N_t - n_{t,r}+0.5)} \tag{8-26}$$

其中，n_r 表示对相关文档的数量，$n_{t,r}$ 表示包含 t 的相关文档的数量。式中出现的 0.5 使计数变得平滑，例如，计数 0 并不会导致不合理（即无限）的权重。正如第 1 章提到的，这种简单的平滑处理在信息检索中比较常见，在其他的统计自然语言处理应用中也颇为常见。

公式（8-25）与公式（8-26）也许使你联想到第 2 章介绍的逆文档频率（IDF）公式（公式（2-13）），因为文档数 N 出现在分子部分，而包含词项 t 的文档数 N_t 出现在分母上。将这个观察推进一步，我们使用两项重写公式（8-23），其中一个与相关性联系，另一个与不相关性联系：

$$w_t = \log \frac{p_t}{1-p_t} + \log \frac{1-\bar{p}_t}{\bar{p}_t} \tag{8-27}$$

$$= \log \frac{N_{t,r}}{N_r - N_{t,r}} + \log \frac{N - N_r - N_t + N_{t,r}}{N_t - N_{t,r}} \tag{8-28}$$

左边的项是包含词项 t 的相关文档的对数比值。如果我们假设 N_r 与 $N_{t,r}$ 相对 N 和 N_t 来说较小——这对于大文档集中的常见词项来说是合理的，我们可以通过设定 $N_r = N_{t,r} = 0$ 逼近右边的项

$$w_t = \mathrm{logit}(p_t) + \log \frac{N - N_t}{N_t} \tag{8-29}$$

这个形式与 IDF 更像了。提出这个公式的 Croft 与 Haper（1979）注意到，如果 $p_t = 0.5$，且 N_t 相对 N 来说较小，则这个公式近似于标准的 IDF 公式。Robertson 和 Walker（1997）注意到，如果使 $p_t = 1/(1+((N-N_t)/N))$，那么公式变为

$$w_t = \log \frac{N}{N_t} \tag{8-30}$$

即为标准的 IDF 公式。

公式（8-23）称为 Robertson/Spärck Jones 权重公式（Robertson/Spärck Jones weighting formula）。根据可用的相关性信息，我们可以使用公式（8-25）、公式（8-26）或公式（8-30）中的任意一个来估计它的值。Robertson 和 Walker（1997）建议在只有最少相关性信息可用的时候使用其他的变形公式。在本章接下来的部分中，w_t 的出现代表了任何或所有这些变形。

当没有相关性信息可用时，通常假设 w_t 就代表了标准的 IDF 公式。同样可以被接受的是通过设定 $n_r = n_{t,r} = 0$ 来使用公式（8-26），但这里值得特别注意的是权值不能为负（参见练习 8.2）。如果一个权值应该为负数，那就用 0 来替代。

当有相关性信息可用时，即使当我们并没有 $N_{t,r}$ 和 N_r 的精确估计，使用公式（8-26）也是合理的。例如，在执行完一个查询后，搜索引擎可能会要求用户指出一小部分的相关文档。如果用户指出三个相关文档，其中两个都含有词项 t，我们使 $n_r = 3$ 且 $n_{t,r} = 2$。同样地，特别注意要避免负的权重。

由此，我们转了一整圈又回到了 IDF，这个在第 2 章中我们仅凭直觉与实验来证明是正确的公式。虽然我们行走的是一条遍布不切实际的独立假设与粗糙近似的崎岖路，但现在却切实地从一条以概率排名原则为起点的理论线路又回到了 IDF。这也是典型的信息检索研究过程。成功的检索方法通常都是在理论与实验的相互支持中被创造与验证的。

遗憾的是，从设计一个有效排名公式的角度来说，我们目前还没有什么进展。在没有相关性信息的情况下直接应用公式（8-22），与第 2 章中提到的将 TF 系数设为常数 1 且不使用文档长度归一化的向量空间模型相比，并没有取得更好的效果。

作为一个简单的例子，让我们回到表 2-1 中的文档集。应用公式（8-22），并使用 IDF 估计 w_t，从而针对查询〈“quarrel”，“sir”〉为文档集排名。这个文档集包含了 $N = 5$ 个文档，其中两个文档包含词项“quarrel”，四个文档包含词项“sir”，给定 IDF 值如下：

$$w_{\text{quarrel}} = \log(5/2) \approx 1.32, \quad w_{\text{sir}} = \log(5/4) \approx 0.32 \tag{8-31}$$

使用这些权重，我们为文档 1 和文档 2 赋予 1.64 的得分，文档 3 与文档 5 赋予 0.32 的得分，文档 4 赋予 0 得分。由于这些得分只反映了词项的存在，即使文档 2 中词项“sir”出现次数比文档 1 中多一次，文档 1 与文档 2 的分数还是一样的。

8.4 词频

在第 2 章中我们列出了一些简单的文档特征，基本排名方法使用这些文档特征比较文档并确定合适的排名。这些特征包括**词频**（term frequency）、**词项邻近度**（term proximity），和**文档长度**（document length）。迄今为止，我们建立的概率模型中只考虑了词项存在，这本质上是词频的一个弱化的形式。为了将模型拓展到完全适应词频，我们必须回到公式(8-13)并重新考虑使用随机变量 D 对文档进行表示。

在 8.2 节的开头我们使用了二元向量的向量形式重新定义了随机变量 $D = \langle D_1, D_2, \cdots \rangle$，其中每个随机变量 D_i 表示一个特定词项的存在或缺失。回到这一点，我们可以重新使用修改后的 D 的定义，使 $D = \langle F_1, F_2, \cdots \rangle$，其中每个 F_i 表示文档中一个对应词项的出现频率，进而重复 8.2 节中的步骤。如我们在 8.2 节中做的一样，我们还定义这个向量中的随机变量 F_t 对应一个词项 t。

与我们在 8.2 节中做的一样，保持假设 T 与假设 Q 不变，同时进行微小的调整从而使其能够适应我们从词项存在转变为词频的变动，能够得到与公式（8-21）相似的如下公式：

$$\sum_{t\in q}\log\frac{p(F_t=f_t|r)\,p(F_i=0|\bar{r})}{p(F_t=f_t|\bar{r})\,p(F_i=0|r)} \tag{8-32}$$

其中，f_t 表示词项 t 在文档中出现的次数（参见练习 8.3）。为了使用这个公式，我们需要估计 $p(F_t=f_t|r)$和 $p(F_t=f_t|\bar{r})$，也就是相关或不相关的词频概率。为了估计这些值，我们必须反过来考虑词频与相关性之间的关系。

当撰写某个主题的文档时，作者通常会选择与该主题相关的词项。因此，如果一个词项与一个特定的主题有关联，我们可以期望它在关于这个主题的文档中出现的次数远远多于它在与此主题不相关之文档中出现的次数。但另一方面，这个词项依然会偶尔出现在与该主题不相关的文档中。

Bookstein 和 Swanson（1974）尝试用 Robertson 等人（1981）以及 Robertson 和 Walker（1994）称为**精华性**（eliteness）的概念来捕捉主题与词项之间的关系。一个文档被称为是词项 t 的**精华**（elite），是指它在某种程度上与这个词项相关。通过转换这种关系，我们可从词频中推测词频是精华的概率——词频越大，这个词项在文档中是精华的可能性就越大。因此，相关性与词频之间的关系就和精华性这个概念吻合了。粗略来说，关于特定主题的文档有更大可能在与这个主题相关的词项上是精华，因此这些词项将有更大可能在这些文档中出现。

对于每个词项 t，我们都定义一个对应 F_t 的二元随机变量 E_t 来表示它的精华性。$E_t=1$ 表示一个文档在 t 上是精华，$E_t=0$ 表示此文档在 t 上不是精华。与相关性一样，我们定义 e 作为 $E_t=1$ 的缩写，$\bar{e}$ 为 $E_t=0$ 的缩写。我们在缩写中省略了下标 t，因为所代指的词项往往是明显的。接着相关性与词频之间的关系可以规范化如下表示：

$$\begin{aligned}p(F_t=f_t|r)&=p(F_t=f_t|e)\cdot p(e|r)+p(F_t=f_t|\bar{e})\cdot p(\bar{e}|r)\\ p(F_t=f_t|\bar{r})&=p(F_t=f_t|e)\cdot p(e|\bar{r})+p(F_t=f_t|\bar{e})\cdot p(\bar{e}|\bar{r})\end{aligned} \tag{8-33}$$

将这个式子带入公式（8-32）中的项，得到

$$\sum_{t\in q}\log\frac{(p(F_t=f_t|e)p(e|r)+p(F_t=f_t|\bar{e})p(\bar{e}|r))(p(F_t=0|e)p(e|\bar{r})+p(F_t=0|\bar{e})p(\bar{e}|\bar{r}))}{(p(F_t=f_t|e)p(e|\bar{r})+p(F_t=f_t|\bar{e})p(\bar{e}|\bar{r}))(p(F_t=0|e)p(e|r)+p(F_t=0|\bar{e})p(\bar{e}|r))}$$

现在我们转向估计这个式子中的单个概率的问题。

8.4.1 Bookstein 的双泊松模型

为了将精华性的模糊定义变得更具体，我们使用一个特定的分布来表示文档中的词项——**双泊松分布**（Poisson distribution）的混合：一个泊松分布对应那些在词项上是精华的文档，另一个对应在词项上不是精华的文档。这个双泊松分布由 Bookstein 提出（参见 Harter，1975，199 页），并接下来被 Harter（1975），Bookstein 和 Swanson（1974），Bookstein 和 Kraft（1977）发展及验证。

泊松分布于 1838 年由法国数学家 Siméon Denis Poisson 发明，其后被广泛应用于对一段时间中某类预定义事件的发生情况的建模中。例如，泊松分布可以用来为一个特定路口每年发生的交通事故建模，可以为一个 Web 服务器每分钟被访问的次数建模，也可以为一克铀－238 在一秒钟释放的 α 粒子的数目建模。泊松分布中，一段时间内发生的事件被假设为与另一个不重叠时间内发生的事件是随机独立的。也就是说，一个时间段中发生的事件数目

不会影响另一个独立时间段中发生的事件数目。在我们的情况中，我们使用泊松分布来为一个给定词项在一个文档中出现的次数建模，其中一个词项的出现作为一个“事件”，而整个文档则对应“时间段”的概念。

给定一个非负整数上的随机变量 X 和一个代表给定时间段内平均事件个数的实数值参数 μ，泊松分布定义如下：

$$g(x,\mu)=\frac{e^{-\mu}\mu^{x}}{x!} \tag{8-34}$$

为了使用这个分布来为词项分布建模，我们必须假设所有文档的长度相等。这个假设并不是那么切合实际，因为一个文档集可能同时包括了书籍与电子邮件信息，它们之间的长度差异可能是 1000 倍或更多。然而，我们现在先接受这个假设，并在本章稍后时候重新讨论。

为了建立双泊松分布，我们为精华文档与不精华文档假设不同的平均数，记为 μ_e 与 $\mu_{\bar{e}}$。根据精华性的定义，有 $\mu_e>\mu_{\bar{e}}$。如果我们让 $q=p(e|r)$ 且 $\bar{q}=p(e|\bar{r})$，并带入公式（8-33），得到以下公式，组成了相关和不相关文档的两个模型：

$$p(F_t=f_t|r)=g(f_t,\mu_e)\cdot q+g(f_t,\mu_{\bar{e}})\cdot(1-q) \tag{8-35}$$

$$p(F_t=f_t|\bar{r})=g(f_t,\mu_e)\cdot \bar{q}+g(f_t,\mu_{\bar{e}})\cdot(1-\bar{q}) \tag{8-36}$$

带入公式（8-32），得到

$$\sum_{t\in q}\log\frac{(g(f_t,\mu_e)\ q+g(f_t,\mu_{\bar{e}})\ (1-q))\cdot(g(0,\mu_e)\ \bar{q}+g(0,\mu_{\bar{e}})\ (1-\bar{q}))}{(g(f_t,\mu_e)\ \bar{q}+g(f_t,\mu_{\bar{e}})\ (1-\bar{q}))\cdot(g(0,\mu_e)\ q+g(0,\mu_{\bar{e}})\ (1-q))} \tag{8-37}$$

虽然一眼看上去这个公式中的词项权重显得非常复杂，但事实上它的结构非常简单。每个出现在分子或分母中的因子都是体现精华性与非精华性的两个泊松分布的混合。给定一个词项，有四个参数需要被确定：μ_e、$\mu_{\bar{e}}$、q 和 $\bar{q}$。可以想象，这些参数的估计可以通过词项统计信息来取得。具体的，取那些使词项权重和文档集里实际词项分布达到最优拟合的参数值。

虽然众多研究者已经在这条路线上付出了诸多努力，却鲜有公开的成果。特别是由于精华性是一个不能直接观测的隐藏变量，这使估计过程变得尤其复杂。但我们依然能够从词项权重上观察到一些有用的信息。

首先，正如我们预期的那样，当一个词项缺失时（$f_t=0$），该词项的权重为 0。其次，随着 f_t 的增大，赋予的权重也随之增大。同样，这种特性与我们的预期一致。

最后，我们考虑当词项数目增长到无限（$f_t\to\infty$）时权重的特性。通过重新排列，公式（8-37）变为

$$\sum_{t\in q}\log\frac{\left(q+\frac{g(f_t,\mu_{\bar{e}})}{g(f_t,\mu_e)}(1-q)\right)\left(\frac{g(0,\mu_e)}{g(0,\mu_{\bar{e}})}\bar{q}+(1-\bar{q})\right)}{\left(\bar{q}+\frac{g(f_t,\mu_{\bar{e}})}{g(f_t,\mu_e)}(1-\bar{q})\right)\left(\frac{g(0,\mu_e)}{g(0,\mu_{\bar{e}})}q+(1-q)\right)} \tag{8-38}$$

现在

$$\frac{g(f_t,\mu_{\bar{e}})}{g(f_t,\mu_e)}=\frac{e^{-\mu_{\bar{e}}}\mu_{\bar{e}}{}_t^{f}}{e^{-\mu_e}\mu_e{}^{f_t}}=e^{\mu_e-\mu_{\bar{e}}}\cdot\left(\frac{\mu_{\bar{e}}}{\mu_e}\right)^{f_t}$$

因为 $\mu_{\bar{e}}<\mu_e$，当 f_t 趋近于无限时，此式趋近于 0，且

$$\frac{g(0,\mu_e)}{g(0,\mu_{\bar{e}})}=e^{\mu_{\bar{e}}-\mu_e}$$

所以，随着 $f_t\to\infty$，公式（8-38）中的词项权重趋近于

$$\log \frac{q\left(\bar{q}e^{\mu_{\bar{e}}-\mu_e}+(1-\bar{q})\right)}{\bar{q}\left(qe^{\mu_{\bar{e}}-\mu_e}+(1-q)\right)} \tag{8-39}$$

对任何词项来说都是一个常数。所以，随着词项数量的增加，其权重趋于饱和（saturate），达到公式（8-39）中的渐近最大值。换句话说，一个词项通过重复多次的形式来为文档贡献得分是有限度的。而且，如果我们假设$e^{\mu_{\bar{e}}-\mu_e}$较小，其对权重的贡献有限，将其近似为 0。这样权重变为：

$$\log \frac{q(1-\bar{q})}{\bar{q}(1-q)} \tag{8-40}$$

这个权重与公式（8-23）中 Robertson/Spärck Jones 权重相似，且这并不是偶然的。当式中的词项存在被精华性替代，这个公式与公式（8-23）类似。我们甚至可以将公式（8-23）看做公式（8-40）的一个近似形式。

8.4.2 双泊松模型的近似

基于我们对于公式（8-37）所列出的观察信息，Robertson 和 Walker（1994）建议对双泊松模型中的词项权重进行一个简单的近似，我们写作

$$\sum_{t\in q}\frac{f_{t,d}(k_1+1)}{k_1+f_{t,d}}\cdot w_t \tag{8-41}$$

其中，$f_{t,d}$代表词项t在文档d中的出现频率，w_t表示 Robertson/Spärck Jones 权重的任一变形，且 $k_1>0$。

与我们观察到的一致，当 $f_{t,d}=0$ 时权重为 0，且权重随着 $f_{t,d}$ 的增长而增长。当 $f_{t,d}=1$ 时，权重与 w_t 相等。随着 $f_{t,d}\to\infty$，权重趋近于$(k_1+1)w_t$。因此，权重饱和到一个 Robertson/Spärck Jones 权重的常数因子，与公式（8-38）一致。

典型的，$1\leqslant k_1<2$，且对于所有查询中的所有词项取相同的值。我们使用 $k_1=1.2$ 作为例子与实验中的默认值，因为这个值已在研究文献中被广泛接受为默认值。在实际中，因为 k_1 对所有查询中所有词项都相等，我们可以将其作为一个系统参数，进而针对文档集及应用环境来调整其取值以取得最好的效果。第 11 章给出调整此参数及其他检索公式中参数的方法。

回到表 2-1 中文档集的例子，我们应用公式（8-41）来为查询〈“quarrel”，“sir”〉对文档集进行排名。为文档 1 计算得分为

$$\frac{f_{\text{quarrel},1}(k_1+1)}{k_1+f_{\text{quarrel},1}}\cdot w_{\text{quarrel}}+\frac{f_{\text{sir},1}(k_1+1)}{k_1+f_{\text{sir},1}}\cdot w_{\text{sir}}\approx\frac{k_1+1}{k_1+1}\cdot 1.32+\frac{k_1+1}{k_1+1}\cdot 0.32\approx 1.64$$

文档 2 的得分为

$$\frac{f_{\text{quarrel},2}(k_1+1)}{k_1+f_{\text{quarrel},2}}\cdot w_{\text{quarrel}}+\frac{f_{\text{sir},2}(k_1+1)}{k_1+f_{\text{sir},2}}\cdot w_{\text{sir}}\approx\frac{k_1+1}{k_1+1}\cdot 1.32+\frac{2(k_1+1)}{k_1+2}\cdot 0.32\approx 1.76$$

词项“sir”在文档 2 中多出现一次，使得文档 2 取得较高的得分。然而需要注意的是，一个即使含有无限次词项“sir”出现，但没有“quarrel”出现的文档，其得分也低于只有一次“quarrel”出现的文档得分。即使文档长度的不同暗示了其中一个应该被排于在另一个之上，文档 3 和文档 5 还是都取得了 0.32 的得分。文档 4 得到了 0 分，因为它不含有任何查询中的词项。

注意到公式（8-41）中的词项权重属于 TF-IDF 家族。然而，其饱和性质使这个权重与例如公式（2-14）中的其他研究文献中介绍的权重又存在着显著的不同。如同这个例子所展

示的，饱和性质为一个词项影响一个文档得分的程度做出了限制，无论这个词项在文档中出现了多少次。

8.4.3 查询词频

公式（8-41）可扩展为与词项频率 q_t 相关的形式——q_t 即词项 t 在查询中出现的次数——通过类似扩展文档词频的方式来实现。在一个短查询中，查询词项的重复是相对少见的。然而，正如我们在第2章提到的，在一些环境下，整个文档可能被作为一个查询，尤其是当实现“更多相似结果”的功能时。类似的，一个用户可以很容易地通过将文档中的段落复制粘贴到搜索引擎来产生一个较长的查询。

如果我们把一个文档当做一个潜在的查询，则查询与文档间的对称关系就变得相对明显。这就暗示了一个有如下形式的查询词频因子

$$\frac{q_t(k_3+1)}{k_3+q_t} \tag{8-42}$$

其中 $k_3>0$ 是一个类似于 k_1 的系统参数。（使用 k_3 是标准的表示法，因为 k_2 已经在一些与概率模型相关的研究文献中被用作了其他用途。）将这个因子结合到公式（8-41）中，得到

$$\sum_{t\in q}\frac{q_t(k_3+1)}{k_3+q_t}\cdot\frac{f_{t,d}(k_1+1)}{k_1+f_{t,d}}\cdot w_t \tag{8-43}$$

然而，对于查询非常长的情况，查询中词项的重复比文档中词项的重复更能指示词项的重要性。因此，k_3 典型的值要比对应 k_1 的值大得多。事实上 $k_3=\infty$ 的设定也较常见，因此可将公式（8-42）简化为 q_t，极限为 $k_3\to\infty$，得到如下的排名公式

$$\sum q_t\cdot\frac{f_{t,d}(k_1+1)}{k_1+f_{t,d}}\cdot w_t \tag{8-44}$$

我们也可以将式中求和看做是在整个词汇表上进行的求和，因为没有出现在查询与文档中的词项权重为0。因此，为了与研究文献中通常使用的方式一致，我们在本章余下的部分忽略求和中的范围约束。

8.5 文档长度：BM25

双泊松模型中一个不现实的假设是所有文档的长度相同。为了考虑不同的文档长度，一个简单的方法就是将实际词频 $f_{t,d}$ 根据文档长度归一化：

$$f'_{t,d}=f_{t,d}\cdot(l_{\text{avg}}/l_d) \tag{8-45}$$

其中，l_d 是文档 d 的长度，l_{avg} 是文档集中所有文档的平均长度。这个归一化的词频接下来可以替代公式（8-44）中的实际词频：

$$\sum q_t\cdot\frac{f'_{t,d}(k_1+1)}{k_1+f'_{t,d}}\cdot w_t \tag{8-46}$$

将这个式子展开并重新排列其中的项，得到

$$\sum q_t\cdot\frac{f_{t,d}\,(l_{\text{avg}}/l_d)\,(k_1+1)}{k_1+f_{t,d}\,(l_{\text{avg}}/l_d)}\cdot w_t=\sum q_t\cdot\frac{f_{t,d}(k_1+1)}{k_1\,(l_d/l_{\text{avg}})+f_{t,d}}\cdot w_t \tag{8-47}$$

虽然这个调整与双泊松模型是一致的，但公式与现实依然未必是一致的。考虑两个文档，其中一个文档长度为另一个文档的两倍，且长文档包含两倍于短文档的词项次数。公式

(8-47) 给两个文档赋予相同的得分。虽然，当这个长文档包含的内容是由两个短文档内容粘贴得到时，这个相同得分的结果是合理的，但是在现实情况中，我们可能会期望这个长文档包含更多的信息，因而理应获得更高的得分。

Robertson 等人 (1994) 建议使用参数 b 控制文档长度的归一化程度，进而混合公式 (8-47) 和公式 (8-44)，得到

$$\sum q_t \cdot \frac{f_{t,d}(k_1+1)}{k_1((1-b)+b(l_d/l_{\text{avg}}))+f_{t,d}} \cdot w_t \tag{8-48}$$

其中 $0 \leqslant b \leqslant 1$。当 $b=0$ 时，上式等价于公式 (8-44)；当 $b=1$ 时，上式等价于公式 (8-47)。如参数 k_1 一样，参数 b 对所有查询是相同的，同样也可以根据文档集及信息检索系统的环境来进行调整。在例子与实验中，我们使用默认值 $b=0.75$。

现在概率模型的建立到达了一个关键时候。只有当第 2 章列出的基本特征是可用时，公式 (8-48) 才代表了最著名的排名检索方法之一。虽然它首次在 1993 年出现于 TREC-3，与我们在第 2 章使用的向量空间模型属于同期技术，但与下一章将介绍的更现代的语言处理模型技术相比，仍保有相当的竞争力。它通常被作为基准算法，用来评判新方法的效果。在效果上大幅超过它的方法通常需要依赖于更多信息的扩展，比如自动查询扩展、文档结构、链接结构或者在依赖大量相关信息上训练的机器学习技术。我们将在下一节中给出一种被称为伪相关反馈的自动查询扩展形式。在概率模型的上下文中考虑文档结构的内容将在 8.7 节中介绍。我们还将在后面章节讨论机器学习以及链接结构时回到概率模型。

一般将公式 (8-48) 称为“Okapi BM25”，或者简单地称为“BM25”。Okapi 是由 Robertson 与他在伦敦城市大学的同事共同创造的一个检索系统的名字。在这个系统中，第一次实现这个公式。“BM”代表“最佳匹配”(“Best Match”)。BM25 仅仅是 Okapi 系统所实现的 BM 公式家族中的一种。公式 (8-22) 本质上就是 BM1，而公式 (8-47) 本质上是 BM11，公式 (8-44) 本质上是 BM15。然而，这些方法没有一个能与 BM25 的名声相比。后者的名声，部分来自于它的简单与有效。参与 TREC 评价的研究人员从 TREC-5 开始已经通过创造和搭建自己的 BM25，而在很多个 TREC 任务中取得了可靠且可敬的成绩。

考虑将 BM25 应用在表 2-1 中的文档集在查询 〈“quarrel”，“sir”〉 上的排名。文档的平均长度 $l_{\text{avg}}=(4+4+16+2+2)/5=28/5=5.6$。文档 1 得分为：

$$\frac{f_{\text{quarrel},1}(k_1+1)}{k_1((1-b)+b(l_d/l_{avg}))+f_{\text{quarrel},1}} \cdot w_{\text{quarrel}} + \frac{f_{\text{sir},1}(k_1+1)}{k_1((1-b)+b(l_d/l_{avg}))+f_{\text{sir},1}} \cdot w_{\text{sir}}$$

$$\approx \frac{k_1+1}{k_1((1-b)+b(5/5.6))+1} \cdot 1.32 + \frac{k_1+1}{k_1((1-b)+b(5/5.6))+1} \cdot 0.32 \approx 1.72$$

类似的，文档 2 的得分为 1.98，文档 3 的得分为 0.18，文档 4 的得分为 0，文档 5 的得分为 0.44。虽然文档 3 与文档 5 中词项“sir”各出现一次且词项“quarrel”并没有出现，但是文档 5 因为文档长度的原因获得了更高的得分。

8.6 相关反馈

正如 8.3 节所述的，可用的相关性信息可以提高我们预测 w_t 取值的能力，进而提高检索的性能。一个获取这样信息的场景是，信息检索系统执行一次初步搜索，将结果展现给用户，并允许用户从结果中指出一定的相关文档。即使这种相关性的判断在数量上很少，它们也可能被用来为公式 (8-26) 中的 n_r 和 $n_{t,r}$ 设置值。最终搜索将基于这些改进的权重执行。

可能更重要的是，系统也可以使用这些相关性判断来得到关于相关文档本质的更多信息。例如，我们可以从用户指出的文档中选择合适的词项，将其加入查询中。这个扩展后的查询可以用于第二次及最后一次搜索。类似这样的**查询扩展**（query expansion）技术隐式地指出了假设 Q 的局限：一个词项在文档中的存在可以依赖于相关性，即使这个词项并不存在于查询中。

先将词项选择的问题抛开，假设我们已有一些方法能够为词项评分，基本的相关反馈（relevance feedback）过程如下所示：

1）执行用户初始查询。

2）将检索结果展示给用户，允许用户浏览结果并指出相关文档。

3）为相关文档中出现的词项评分，选择前 m 个扩展词项，选择时忽略初始查询词项。

4）将新词项加入起始查询，调整权重 w_t，并执行扩展后的查询。

5）向用户展现最后检索的结果。

第四步中词项 t 的经过调整后的权重由公式（8-26）决定，其中，将 n_r 设为用户指出的相关文档的数量，$n_{t,r}$设置为这些相关文档中包含 t 的文档数量。同时，因为初始查询中的词项是用户信息需求的最好反映，因此给予它们特殊的地位也是很常见的。这个额外的调整可以通过将扩展词项的权重乘以一个常数因子 γ 来体现，其中 $\gamma=1/3$ 是典型的取值。

相关反馈一直是研究工作中的一个重要主题，我们可以用多种方式对基本反馈过程进行扩展和修改。例如，允许用户指出不相关的文档，通过交互式的调整来选定词项列表；或者多次重复反馈过程，允许用户按意愿增加或删除词项等。并且，也有多种为词项评分的方法。我们下面介绍其中的一种。

8.6.1 词项选择

Robertson（1990）提出了一种基于将词项加入查询后相关文档和不相关文档的预期得分变化的简单词项选择方法。假设我们现在考虑向查询中加入词项 t。在 8.3 节中，我们定义 p_t 为相关文档包含 t 的概率，$\overline{p}_t$ 为不相关文档包含 t 的概率。考虑公式（8-22），如果我们将 t 加入查询，相关文档的得分将平均增加 $p_t w_t$，而不相关文档的得分将平均增加 $\overline{p}_t w_t$。

这两个预期增幅 $p_t w_t$ 及 $\overline{p}_t w_t$ 间的差越大，这个词项就能更好地区分相关与不相关文档。所以我们在相关反馈中使用**词项选择值**（term selection value）来为词项评分

$$w_t \cdot (p_t - \bar{p}_t) \tag{8-49}$$

在相关反馈的上下文中，如果用户指出了 n_r 个相关文档，其中 $n_{t,r}$个包含 t，我们可以估计 p_t 为$(n_{t,r}/n_r)$。而 $\overline{p}_t$ 的取值则难以估计。然而，我们可以假设它的值相对 p_t 来说较小，并将其当做 0。所以，词项选择值变为

$$\frac{n_{t,r}}{n_r} \cdot w_t \tag{8-50}$$

我们还可以将 n_r 从分母中删去，因为它对所有词项都是相同的，然后乘以一个常数因子，这仍然是保序的，有

$$n_{t,r} \cdot w_t \tag{8-51}$$

对于相关反馈，我们根据公式（8-51）为每一个查询词项排名，选择前 m 个，然后将其加入初始查询，必要时重新计算词项权重。

遗憾的是，这个方法在选择多少词项，即 m 的取值方面，没有给我们任何指导。已知

的是，$m=10$ 这个值在 TREC 文档集上能够提供合理的效果提升。我们在实验中使用这个值作为默认值。Robertson 和 Walker（1999）提出了一个为词项选择的边缘阈值作出估计的方法，这个方法基于接受一个噪声词项的概率。

8.6.2 伪相关反馈

伪相关反馈（PRF），也被称为**盲反馈**（blind feedback），是相关反馈的一种变形，其中交互步骤被忽略了。这里不需要用户指出相关文档，**检索系统只是简单地假设由初始查询返回的前 k 个文档都是相关文档。**可知的是，$k=20$ 的值能够在 TREC 文档集上取得合理的检索效果提升。我们将使用这个值作为实验的默认值。余下的过程流程如前：从这些文档中选择词项，将这些词项加入原始的查询，扩展后的查询被用来产生最后的检索结果。

因为 PRF 依赖于正反馈，使用它可能会严重损害检索效果，特别是前 k 个文档中只有很少甚至没有真正的相关文档时。然而，当使用多个查询的平均结果，例如平均准确度和相似有效性指标来作为衡量标准时，PRF 通常能得到显著的有效性提升。并且，它是一个完全的自动过程，不需要打扰用户，可以在每个查询中被隐式地执行。

作为一个例子，让我们回到 TREC 中关于执法狗（“law”，“enforcement”，“dogs”）的主题 426（图 1-8）。在 TREC45 文档集中的 $N=528\,155$ 个文档上采用伪相关反馈，其结果如表 8-1 所示。我们假设前 $k=n_r=20$ 个文档是相关的。

表 8-1 应用在 TREC 主题 426 上的伪相关反馈，权重调整因子 $\gamma=1/3$

查询词项 (t)	N_t	$n_{t,r}$	初始的 w_t	$n_{t,r}\cdot w_t$	调整后的 w_t
dogs	2163	20	7.931 78		13.296 48
law	49 792	19	3.406 98		6.965 09
enforcement	10 635	19	5.634 07		9.307 67
dog	3126	12	7.400 50	88.806 00	2.648 56 $= 7.953\,63\gamma$
sniffing	194	6	11.410 69	68.464 14	3.427 93 $= 10.294\,07\gamma$
canine	150	5	11.781 78	58.908 90	3.440 06 $= 10.330\,51\gamma$
pooper	20	4	14.688 67	58.754 68	4.359 52 $= 13.091\,64\gamma$
officers	15 006	11	5.137 35	56.510 85	1.789 00 $= 5.372\,39\gamma$
metro	39 887	15	3.726 97	55.904 55	1.701 28 $= 5.108\,96\gamma$
canines	34	4	13.923 14	55.692 56	4.064 35 $= 12.205\,26\gamma$
police	30 589	13	4.109 88	53.428 44	1.622 50 $= 4.872\,37\gamma$
animal	5304	8	6.637 74	53.101 92	2.020 91 $= 6.068\,79\gamma$
narcotics	3989	7	7.048 79	49.341 53	2.061 99 $= 6.192\,17\gamma$

表中的前 3 行给出了 3 个原始查询的词项统计信息。因为这些词项出现在大部分或全部的前 20 个文档中，反馈后它们对应的调整权重几乎变为了两倍。表底部的几行给出了前 10 个扩展词项，根据公式（8-51）计算出的词项选择值（$n_{t,r}\cdot w_t$）来排序。表的最后一列列出了在执行扩展查询时使用的调整后的 w_t 值，其中 $\gamma=1/3$。原始查询的 precision@10 为 0.300，MAP 值为 0.043。扩展后的查询检索效果得到了大幅提升，precision@10 为 0.500，MAP 值为 0.089。

将词项“pooper”作为一个扩展词项揭示了一个伪相关反馈可能引发的问题。查询〈“law”，“enforcement”，“dogs”〉可能与关于狗的执法有关，但同样也可能与在执法中使用狗有关。尽管后面的一种解读在对应话题的描写与叙述中已有明显的说明，但信息检索系统对这些信息并不知情。在第一种解读下，词项“pooper”是一个很好的扩展词项，因为法律常常要求宠物的主人使用长柄粪铲（pooper scooper）清理他们宠物狗的粪便。

因为伪相关反馈可以得到显著的提高，你可能会疑惑为什么我们会在一次反馈迭代后停止下来。你可能希望更多的迭代次数会带来更多的提高。然而，如果我们在这个样例中继续执行更多的迭代，将得到下面的扩展词项序列。

迭代	扩展词项
1	dog, sniffing, canine, pooper, officers, metro, canines, police, animal, narcotics
2	dog, canine, pooper, sniffing, leash, metro, canines, animal officers, narcotics
3	dog, canine, pooper, sniffing, leash, metro, canines, animal, owners, pets
4	dog, leash, animal metro, canine, pooper, sniffing, canines, owners, pets
5	dog, leash, metro, canine, pooper, sniffing, canines, owners, animal, pets
6	dog, leash, metro, pooper, canines, owners, pets, animals, canine, scooper
7	dog, leash, metro, pooper, canines, owners, pets, animals, canine, scooper

在7次迭代后，查询彻底地偏向了一种解读。再多的迭代也不会产生什么变动了。

在这个例子中出现的**查询偏移**（query drift）展现出了无差别的伪相关反馈的主要风险。即使是一次迭代有时候也会严重损害检索的效果。大多数商业化的信息检索系统可能是基于这个原因都没有加入伪相关反馈。通常情况下检索有效性的提高带来的益处并不能掩盖因一次失败而带来的负面用户体验。并且，伪相关反馈从性能上来说代价也比较高昂，因为它要求对多个文档进行分析，并需要第2次执行查询。同样地，潜在的益处并不能掩盖因其增加的响应时间。

8.7 区域权重：BM25F

在第2章中，与如词频和词项邻近度等排名的基本特征列表一起，我们也列出了一些对排名有用的其他特征，包括文档结构。例如，出现在如文档标题等特定的区域中的词项，与出现在文档正文中的词项相比，将获得更多的权重。在这一节中，我们给出一个重要的BM25扩展，它将文档结构考虑在内，称为BM25F。

在Web搜索中利用文档结构特别重要。图8-1给出了一些Web搜索引擎上可用的丰富结构信息。此图是一个大幅简化的Web页面，地址为en.wikipedia.org/wiki/Shakespeare，截自于2007年中期的英文版Wikipedia网站。

HTML页面被分为两部分：（1）**头**（header）部分，以标签〈head〉…〈/head〉划定，包括一个标题与其他描述文档的**元数据**（metadata）；（2）**正文**（body）部分，包含了供显示的文章内容。元数据包含了与Shakespeare相关的重要日期信息，也包含了Wikipedia组织相关的注解。例如，关键字“Persondata”指这个页面包含了一个传记信息的标准表格。文档的正文包括了以标签〈h1〉作标记的节标题，也包含了以标签〈b〉作标记的黑体字。

正文还包含了指向Wikipedia中“poet”与“playwright”页面的超文本链接。在这些链接的开始标签〈a〉与结束标签〈/a〉之间的**锚文字**（anchor text）可以被解读为关于被指向页面的内容信息。在这个例子中，锚文字简单地重复了这些页面的标题。锚文字是Web搜索中重要的特征，我们将在第15章继续讨论。

虽然HTML在结构方面特别丰富，但对于许多文档来说，其中的区域却是类似的。例如，电子邮件信息中的“Subject”、“To”和“From”区域可以被利用于类似的目的中。BM25F的目的是恰当地反映出这些结构暗示的重要信息。

```
<html>
  <head>
    <title>William Shakespeare - Wikipedia, the free encyclopedia</title>
    <meta
      name="keywords"
      content="William Shakespeare,Persondata,Sister projects, Earlybard,
               1582,1583,1585,1616"/>
  </head>
  <body>
    <h1>William Shakespeare</h1>
    <b>William Shakespeare</b> (baptised 26 April 1564 - died 23 April 1616)
    was an English <a href="http://en.wikipedia.org/wiki/Poet">poet</a> and
    <a href="http://en.wikipedia.org/wiki/Playwright"> playwright</a>.
    He is widely regarded as the greatest writer of the English language
    and the world's pre-eminent dramatist.  He wrote approximately 38 plays
    and 154 sonnets, as well as a variety of other poems...
  </body>
</html>
```

图 8-1 显示 Web 典型结构的简单 HTML 页面

BM25F 背后的直觉与公式（8-45）中文档长度归一化背后的直觉相似。这个公式指定了一个文档长度归一化调整后的词频，用于替代检索公式中的实际词频。我们将这个想法扩展到区域权重中，在包含了某个词项的区域中计算调整后的词频。例如，我们认为一个查询词项在标题中出现 1 次等同于它在文档里出现 10 次，从而相应地扩展词频。

为了规范化这个概念，我们将 $f_{t,d,s}$ 定义为词项 t 在文档 d 区域 s 中的出现次数。接着使用类似于 BM25 归一化的方法在区域的层次上应用面向区域的文档长度归一化，从而反映出不同区域类型之间平均长度的区别。例如，一个文档的标题与正文相比，常常要短小得多。

$$f'_{t,d,s} = \frac{f_{t,d,s}}{(1-b_s)+b_s\,(l_{d,s}/l_s)} \tag{8-52}$$

其中，$l_{d,s}$ 是文档 d 中区域 s 的长度，l_s 是所有文档中区域 s 的平均长度，b_s 是一个与 BM25 中 b 参数类似的区域参数。如同 b 参数一样，b_s 参数的值在 0～1 之间，从而产生相应的完全无归一化公式与完全归一化公式。

这些面向区域调整的词频，称为伪频率（pseudo-frequency），然后与面向文档调整的词频合并为一个：

$$f'_{t,d} = \sum_s v_s \cdot f'_{t,d,s} \tag{8-53}$$

其中，v_s 是区域 s 的权重。例如，我们可以定义 $v_{\text{title}}=10$ 且 $v_{\text{body}}=1$。

我们可以用这个调整后的词频来替代公式（8-44）中的实际词频，得到 BM25F 的公式：

$$\sum q_t \cdot \frac{f'_{t,d}(k_1+1)}{k_1+f'_{t,d}} \cdot w_t \tag{8-54}$$

因为文档长度的归一化被应用于调整后词频的计算，这个公式并不需要显式地归一化。我们继续在文档层次上计算 Robertson/Spärck Jones 权重 w_t。BM25F 要求我们对每个区域的两个参数 b_s 和 v_s 进行调整，如同调整整体参数 k_1 一样。

BM25F 的实现要求文档包含合适的标签，并要求词频计算是针对每一个区域的。第 5 章介绍的轻量级结构设计的查询过程技术可以对它的实现有所帮助。

8.8 实验对比

表 8-2 以图表的形式给出了概率模型中每一次主要创新所带来的进步，实验的数据来自 1.4 节中所述的测试文档集。这个表中的结果可以直接与表 2-5 中的结果对比。BM25 对应的行在两个表中均有出现。

表 8-2 本章讨论的几个检索方法的有效性指标

方法	TREC45				GOV2			
	1998		1999		2005		2006	
	P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
公式(8-22)	0.256	0.141	0.224	0.148	0.069	0.050	0.106	0.083
公式(8-24)	0.402	0.177	0.406	0.207	0.418	0.171	0.538	0.207
BM25	0.424	0.178	0.440	0.205	0.471	0.243	0.534	0.277
BM25 + PRF	0.452	0.239	0.454	0.249	0.567	0.277	0.588	0.314
BM25F					0.482	0.242	0.544	0.277

对 BM25，实验使用参数 $k_1=1.2$ 且 $b=0.75$。使用公式（8-30）来计算 w_t。对伪相关反馈（PRF）来说，我们取出前 $k=20$ 个文档，并将其中前 $m=10$ 个词项加入到查询中，使用 8.6 节中描述的方法在 $\gamma=1/3$ 的情况下重新调整原始查询词项与新加入查询词项的权重。对 BM25F，我们只考虑标题及正文两个区域，所有没有在标题中出现的词项都被认为出现在正文中。BM25F 的实验中，参数设置分别为 $k_1=1.2$，$v_{\text{title}=10}$，$v_{\text{body}}=1$，且 $b_{\text{title}}=b_{\text{body}}=0.75$。

正如我们从表中看到的，由于引入了词频这一信息，检索效果有大幅提升。伪相关反馈的效果则根据文档集的不同而不同，将 precision@10 提高了 3%～20%，将 MAP 提高了 13%～34%。区域权重（BM25F）对 GOV2 主题几乎没有什么影响。在第 15 章中，我们将这些检索方法应用在一个面向 Web 检索的任务上，区域权重的影响将会更加明显（参见表 15-6）。

8.9 延伸阅读

最早的针对信息检索的概率模型可能是由 Maron 和 Kuhns（1960）提出的。Fuhr（1992）提供了一个针对 Maron/Kuhns 模型与 Robertson/Spärck Jones 概率模型的比较。建立在概率模型之上的推理网络模型提供了丰富的概率操作集合（Turtle 和 Croft，1991；Greiff 等人，1999）。Lafferty 和 Zhai（2003）从本质的层次探索了概率模型与我们下一章即将介绍的语言模型之间的关系。我们在相关方法上的解释基于他们在文献中的分析。Roelleke 和 Wang（2006）拓展了这个分析，并细致地描述了两个模型中基本元素的对应关系。

Robertson/Spärck Jones 概率模型的发展可以从一系列文献中追溯。其中最值得提到的是 Spärck Jones（1972），Robertson 和 Spärck Jones（1976），Robertson（1977），Croft 和 Harper（1979），Robertson 和 Walker（1994），Robertson 等人（1994），Robertson 和 Walker（1997）。Spärck Jones 等人（2000a，b）给出了一个截至 1999 年相关发展的总结。这份总结还包含了大量关于增强与拓展相关技术的讨论。

Robertson（2004）为逆文档频率背后的理论原理以及它与概率模型之间的关系提供了更多的见解。de Vries 和 Roelleke（2005）则进一步探索了这个关系。他们将相关信息的使用视为熵损失，并且使用一个额外的“虚拟文档”的概念为公式（8-26）中使用的平滑数 0.5 给出了一个有趣的解释。Church 和 Gale（1995）将 IDF 建模为泊松分布的混合，并强调文档频率与文档集内部词频的重要区别。Roelleke 和 Wang（2008）提供了一个关于 TFIDF理论基础的细致回顾，并给出了它在不同检索模型中的不同解读。

一个最早的基于相关反馈的查询扩展算法由 Joseph J. Rocchio 在 SMART 系统（详情参见 Rocchio（1971）或 Baeza Yates 和 Ribeiro-Neto（1999））的背景下发明。Ruthven 和 Lalmas（2003）提供了关于交互相关反馈与伪相关反馈的详细综述。Wang 等人（2008）讨论了当负（不相关）例可用时的相关反馈。

TREC 中很多小组在 1992—1998 年的实验认定伪相关反馈是一个提高平均查准率的可靠技术。然而，伪相关反馈在实际环境下的益处却从未被清晰的论证。Lynam 等人（2004）给出了一些伪相关反馈实现的比较，其中的实现来自于一些领先的研究小组在 2003 年左右 TREC 风格实验中使用的版本。基于语言模型方法，Lee 等人（2008）将聚类方法应用于选择更好的反馈文档。同样基于语言模型方法，Cao 等人（2008）将分类方法应用于为反馈文档选择更佳的扩展词项。Collins-Thompson（2009）给出了一个能够最小化伪相关反馈中查询扩展技术所带来的风险的框架。

BM25F 背后的理论首见于 Robertson 等人（2004），远在 BM25 被认定为顶级的排名公式之后。本书中的 Okapi BM25F 形式，在由剑桥微软实验室开展的面向 TREC-13（Zaragoza 等人，2004）和 TREC-14（Craswell 等人，2005b）的实验中作为一部分被介绍与评价。其他 BM25 的拓展包括加入近似度的提案（Rasolofo 和 Savoy，2003；Büttcher 等人，2006）；加入查询词项在文档中位置的提案（Troy 和 Zhang，2007）；以及将锚文字和其他 Web 相关特征考虑在内的提案（Hawking 等人，2004；Craswell 等人，2005a）。Robertson 和 Zaragoza（2010）给出了一个体现在 BM25F 中的，关于概率检索最近工作的综述。

8.10 练习

练习 8.1 证明 $\text{logit}(p) = -\text{logit}(1-p)$，其中 $0 \leqslant p \leqslant 1$。

练习 8.2 在什么情况下公式（8-26）会给出一个负值？这些情况在现实中经常发生吗？

练习 8.3 请在公式（8-13）中应用假设 Q 和假设 T，推导出公式（8-32）。

练习 8.4 我们对于 BM25F 的描述隐式地假设区域 s 在一个文档中最多出现一次。然而，一个文档可能有多个章节的头部、多个黑体文字部分以及多个类似的结构元素。提出一个 MB25F 的扩展来处理同类型区域多次出现的情况。

练习 8.5（项目练习） 实现 BM25 排名公式（公式（8-48））。使用练习 2.13 中的测试文档集或如 TREC 文档集这样的文档集来测试你的实现。

练习 8.6（项目练习） 实现 8.6 节中描述的伪相关反馈方法中的词项选择步骤。给定一个查询 q，将其在任一文字文档集上执行，使用任何检索方法，返回前 20 个文档。使用公式（8-51），在这些文档中取出前 10 个词项。用以下的查询测试你的实现：

(a) 〈“law”，“enforcement”，“dogs”〉

(b) 〈“marine”，“vegetation”〉

(c) 〈“black”，“bear”，“attacks”〉

(d) 〈“journalist”，“risks”〉

(e) 〈"family", "leave", "law"〉

这些查询从 TREC 2005 Robust 专题[⊖]取得，它们旨在提高检索技术在其表现较差的主题上的一致性。伪相关反馈在这样的主题上有不利的影响。

8.11 参考文献

Baeza-Yates, R. A., and Ribeiro-Neto, B. (1999). *Modern Information Retrieval.* Reading, Massachusetts: Addison-Wesley.

Bookstein, A., and Kraft, D. (1977). Operations research applied to document indexing and retrieval decisions. *Journal of the ACM*, 24(3):418–427.

Bookstein, A., and Swanson, D. R. (1974). Probabilistic models for automatic indexing. *Journal of the American Society for Information Science*, 25(5):312–319.

Büttcher, S., Clarke, C. L. A., and Lushman, B. (2006). Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 621–622. Seattle, Washington.

Cao, G., Nie, J. Y., Gao, J., and Robertson, S. (2008). Selecting good expansion terms for pseudo-relevance feedback. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 243–250. Singapore.

Church, K. W., and Gale, W. A. (1995). Inverse document frequency (IDF): A measure of deviation from poisson. In *Proceedings of the 3rd Workshop on Very Large Corpora*, pages 121–130. Cambridge, Massachusetts.

Collins-Thompson, K. (2009). Reducing the risk of query expansion via robust constrained optimization. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 837–846. Hong Kong, China.

Craswell, N., Robertson, S., Zaragoza, H., and Taylor, M. (2005a). Relevance weighting for query independent evidence. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 416–423. Salvador, Brazil.

Craswell, N., Zaragoza, H., and Robertson, S. (2005b). Microsoft Cambridge at TREC 14: Enterprise track. In *Proceedings of the 14th Text REtrieval Conference*. Gaithersburg, Maryland.

Croft, W. B., and Harper, D. J. (1979). Using probabilistic models of document retrieval without relevance information. *Journal of Documentation*, 35:285–295.

de Vries, A. P., and Roelleke, T. (2005). Relevance information: A loss of entropy but a gain for IDF? In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 282–289. Salvador, Brazil.

Fuhr, N. (1992). Probabilistic models in information retrieval. *The Computer Journal*, 35(3):243–255.

Greiff, W. R., Croft, W. B., and Turtle, H. (1999). PIC matrices: A computationally tractable class of probabilistic query operators. *ACM Transactions on Information Systems*, 17(4):367–405.

Harter, S. P. (1975). A probabilistic approach to automatic keyword indexing: Part I. On the distribution of specialty words in a technical literature. *Journal of the American Society for Information Science*, 26:197–206.

Hawking, D., Upstill, T., and Craswell, N. (2004). Toward better weighting of anchors. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 512–513. Sheffield, England.

Lafferty, J., and Zhai, C. (2003). Probabilistic relevance models based on document and query generation. In Croft, W. B., and Lafferty, J., editors, *Language Modeling for Information Retrieval*, chapter 1, pages 1–10. Dordrecht, The Netherlands: Kluwer Academic Publishers.

⊖ trec.nist.gov/data/robust.html

Lee, K. S., Croft, W. B., and Allan, J. (2008). A cluster-based resampling method for pseudo-relevance feedback. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 235–242. Singapore.

Lynam, T. R., Buckley, C., Clarke, C. L. A., and Cormack, G. V. (2004). A multi-system analysis of document and term selection for blind feedback. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 261–269. Washington, D.C.

Maron, M. E., and Kuhns, J. L. (1960). On relevance, probabilistic indexing and information retrieval. *Journal of the ACM*, 7(3):216–244.

Ponte, J. M., and Croft, W. B. (1998). A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–281. Melbourne, Australia.

Rasolofo, Y., and Savoy, J. (2003). Term proximity scoring for keyword-based retrieval systems. In *Proceedings of the 25th European Conference on Information Retrieval Research*, pages 207–218. Pisa, Italy.

Robertson, S. (1977). The probability ranking principle in IR. *Journal of Documentation*, 33:294–304.

Robertson, S. (2004). Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520.

Robertson, S., and Spärck Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146.

Robertson, S., and Zaragoza, H. (2010). The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 4.

Robertson, S., Zaragoza, H., and Taylor, M. (2004). Simple BM25 extension to multiple weighted fields. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 42–49. Washington, D.C.

Robertson, S. E. (1990). On term selection for query expansion. *Journal of Documentation*, 46(4):359–364.

Robertson, S. E., van Rijsbergen, C. J., and Porter, M. F. (1981). Probabilistic models of indexing and searching. In Oddy, R. N., Robertson, S. E., van Rijsbergen, C. J., and Williams, P. W., editors, *Information Retrieval Research*, chapter 4, pages 35–56. London, England: Buttersworths.

Robertson, S. E., and Walker, S. (1994). Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–241. Dublin, Ireland.

Robertson, S. E., and Walker, S. (1997). On relevance weights with little relevance information. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 16–24. Philadelphia, Pennsylvania.

Robertson, S. E., and Walker, S. (1999). Okapi/keenbow at TREC-8. In *Proceedings of the 8th Text REtrieval Conference*. Gaithersburg, Maryland.

Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., and Gatford, M. (1994). Okapi at TREC-3. In *Proceedings of the 3rd Text REtrieval Conference*. Gaithersburg, Maryland.

Rocchio, J. J. (1971). Relevance feedback in information retrieval. In Salton, G., editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 14, pages 313–323: Prentice-Hall.

Roelleke, T., and Wang, J. (2006). A parallel derivation of probabilistic information retrieval models. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 107–114. Seattle, Washington.

Roelleke, T., and Wang, J. (2008). TF-IDF uncovered: A study of theories and probabilities. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 435–442. Singapore, Singapore.

Ruthven, I., and Lalmas, M. (2003). A survey on the use of relevance feedback for information access systems. *Knowledge Engineering Review*, 18(2):95–145.

Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.

Spärck Jones, K., Walker, S., and Robertson, S. E. (2000a). A probabilistic model of information retrieval: Development and comparative experiments – Part 1. *Information Processing & Management*, 36(6):779–808.

Spärck Jones, K., Walker, S., and Robertson, S. E. (2000b). A probabilistic model of information retrieval: Development and comparative experiments – Part 2. *Information Processing & Management*, 36(6):809–840.

Troy, A. D., and Zhang, G. Q. (2007). Enhancing relevance scoring with chronological term rank. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 599–606. Amsterdam, The Netherlands.

Turtle, H., and Croft, W. B. (1991). Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, 9(3):187–222.

Wang, X., Fang, H., and Zhai, C. (2008). A study of methods for negative relevance feedback. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 219–226. Singapore.

Zaragoza, H., Craswell, N., Taylor, M., Saria, S., and Robertson, S. (2004). Microsoft Cambridge at TREC 13: Web and Hard tracks. In *Proceedings of the 13th Text REtrieval Conference*. Gaithersburg, Maryland.

第9章

Information Retrieval: Implementing and Evaluating Search Engines

语言模型及其相关方法

本章从不同于第8章中的概率模型的角度来阐述排名问题。尽管有各自不同的理论基础，本章所提的检索方法还是有一些重要的共通点。所有的排名方法都可以视为是通过**语言模型**（language model）的某种形式对文档进行排名的过程，即将文档中词项的实际出现与根据文档集和文档的特征预测的期望出现进行比较的过程。这种方法与概率模型的不同之处在于它们不太关注相关性，即不会明确地考虑这个因素。

尽管在更广泛的意义上本章提出的方法可视为是语言模型方法，但9.1节和9.4节还是主要关注狭义的“语言模型方法”的基础。这种方法——基于Berger、Croft、Hiemstra、Lafferty、Ponte、Zhai等人的工作而建立起来——特点是利用文档为查询建立了一个**产生式模型**（generative model）(Ponte和Croft，1998；Hiemstra，2001；Song和Croft，1999；Miller等人，1999；Berger和Lafferty，1999；Lafferty和Zhai，2001；Zhai和Lafferty，2001，2004)。在很短的时间内，大概从1998～2001年，这种语言模型方法就从Ponte和Croft（1998）在SIGIR上发表的一篇论文开始，发展成为当前新的**信息检索**系统的主流框架（Croft和Lafferty，2003)。在研究文献中每当提及“语言模型方法”，通常就是指这种狭义的语言模型方法。

9.1节将以**概率排名原则**（Probability Ranking Principle，PRP）作为切入点，去推导和证明产生式查询模型。给定一个假设相关的文档，我们利用该文档建立一个语言模型去估计 $p(q|d)$，即为了检索出文档 d 而输入查询 q 的概率，并依此对文档进行排名。在第1章研究词频时介绍了语言模型的思想，并在第6章介绍数据压缩时进一步阐述了这种思想。本章提及的语言模型都建立在第1章介绍的最简单的模型基础上——只使用基于文档和文档集的统计信息的零阶模型。

9.2节将详细讨论从文档中构建一个语言模型的细节，主要关注平滑过程，即给未出现在文档中的词项赋予一个非零的概率。因为相关文档并非包含了每一个查询词项，所以为这些词项赋予正的概率是很重要的。9.3节将平滑语言模型应用到排名问题中，并同时结合9.1节和9.2节的结论构造出具体的排名公式。9.4节考虑另外一种基于Kullback-Leibler**距离**（Kullback-Leibler divergence，KL**距离**）理论的语言模型的理论基础，这是一种判定两个概率分布差异性的方法。基于这种理论基础，在语言模型方法下提出了一种查询扩展的方法。

后面的章节就语言模型给出更广泛的解释。9.5节介绍一种被称为**随机差异性**（divergence from randomness，DFR）的检索方法，该方法在待排名文档范围内，将随机分布词项的模型与这些词项的实际分布进行对比。9.6节介绍了一种段落检索的方法，它的理论基础与DFR和其他语言模型方法相关。9.7节给出了本章和前几章提出的几个检索方法的对比评价。9.8节列出了一些由语言模型派生或与语言模型相关的新检索方法。

9.1 从文档中产生查询

正如第8章所讲的那样，我们从讨论概率排名原则开始。由公式（8-8）有

$$\log \frac{p(r|D,Q)}{1-p(r|D,Q)}=\log \frac{p(r|D,Q)}{p(\bar{r}|D,Q)} \tag{9-1}$$

$$= \log \frac{p(D,Q|r)\ p(r)}{p(D,Q|\bar{r})\ p(\bar{r})} \tag{9-2}$$

在公式（8-10）中用公式 $p(D,Q|R)=p(D|Q,R)\cdot p(Q|R)$展开上式的联合概率。本节中则相反，用公式 $p(D,Q|R)=p(Q|D,R)$来展开联合概率，得到：

$$\log \frac{p(r|D,Q)}{p(\bar{r}|D,Q)} = \log \frac{p(D,Q|r)\ p(r)}{p(D,Q|\bar{r})\ p(\bar{r})} \tag{9-3}$$

$$= \log \frac{p(Q|D,r)\ p(D|r)\ p(r)}{p(Q|D,\bar{r})\ p(D|\bar{r})\ p(\bar{r})} \tag{9-4}$$

$$= \log \frac{p(Q|D,r)\ p(r|D)}{p(Q|D,\bar{r})\ p(\bar{r}|D)} \tag{9-5}$$

$$= \log p(Q|D,r) - \log p(Q|D,\bar{r}) + \log \frac{p(r|D)}{p(\bar{r}|D)} \tag{9-6}$$

$$= \log p(Q|D,r) - \log p(Q|D,\bar{r}) + \mathrm{logit}(p(r|D)) \tag{9-7}$$

接下来将详细研究最后一条公式中的每个概率。

对于一个给定查询 q 和一个相关文档 d，$p(Q=q|D=d,r)$的值表示用户为了检索到 d 而输入查询 q 的概率。本质上，以 d 为条件（即以 d 为条件来考察查询 q——译者注）就为我们提供了相关文档的例子。从这个例子中，我们可以估计如果希望检索到类似于 d 这样的文档，用户输入 q 的概率。特别是，如果词项出现在 d 中的频率远远大于随机情况时，我们就会认为它出现在查询中的概率也较高。

另一方面，考虑 d 在公式（9-7）中的概率 $p(Q=q|D=d,\bar{r})$所起的作用。这里以 d 为条件并不能提供给我们很多关于用户的需求方面的信息。我们只有不相关的文档例子，只能猜测什么是相关的。因此，假设这个概率独立于 d 是合理的，也就是 $p(Q=q|D=d,\bar{r})$是一个常数。如果我们接受这个假设，那就可以去掉这个常数项，因此得到一个等价的排名公式

$$\log p(Q|D,r) + \mathrm{logit}(p(r|D)) \tag{9-8}$$

第二个概率项 $p(r|D)$独立于 q 并且对所有查询都一样。这个相关性的先验概率可理解为文档质量或重要性的指标。例如，在 Web 搜索的场景中，一所大学的主页应比一个该校学生的个人主页获得更高的先验概率值（见 15.3 节）。在文件系统搜索的场景中，单独发给你的邮件也应比群发邮件获得更高的先验概率值。

尽管如此，在应用上下文信息不完整的情况下，通常假设相关性的先验概率 $p(r|D)$对所有文档都是一样的。如果接受这个假设，因为保序变换，我们可以丢掉这个常量。另外，因为幂也有保序性，所以可以去掉对数。因此得到以下的排名公式

$$p(Q|D,r) \tag{9-9}$$

上式的相关性条件可以得到简化，将上式简化为

$$p(Q=q|D=d) \tag{9-10}$$

这个非常简单的公式对于语言模型所起的作用，与第 8 章中公式（8-13）在概率检索模型中所起的作用是一样的。给定一个文档 d 和一个查询 q，为了排名，可通过估计概率 $p(q|d)$对 d 进行评分。

为了估计这个概率，文档被认为是为产生查询 q 提供了一个**模型**（model）。想象一个用户正构造一个查询去检索相关文档。她会尝试想一些经常出现在这些相关文档中，而不太出现在不相关文档中的词项。基于这种尝试，用户就会在心里想象相关文档看起来是什么样

子——哪些词项能把这个文档与文档集中的其他文档区分开来。然后她选择一些词项并输入到搜索引擎中。这些词项就成了查询 q。

为了给 d 评分，我们假设它符合用户的想象，同时估计用户选用查询 q 去检索它的概率。因为这个假设把用户的信息需求——在她心里想象的——与文档 d 联系起来了，因此可以视 d 为 q 提供了一个**产生式模型**（generative model）。

一些信息检索系统的研究人员通过一个不同的推理过程更直接地去证明公式（9-10）。他们声明只关心 $p(Q,D)$，即查询和文档的联合分布，而忽略相关性的问题。文档根据以下公式排名

$$p(Q,D)=p(Q\mid D)\cdot p(D)$$

或等价于

$$\log p(Q\mid D)+\log p(D) \tag{9-11}$$

与处理公式（9-8）中第二项的方法类似，可以把 $\log p(D)$ 理解为文档质量或其重要性的指标，并且在其他信息缺失的情况下，把它看做一个常量。不过请注意公式（9-11）的第二项与公式（9-8）的第二项之间的差别。

9.2 语言模型和平滑

第 1 章中讨论了莎士比亚文集及其他文档集中的词项分布，并引出了基本的语言模型方法。此外，在第 6 章文本压缩中，我们进一步探讨了符号集上的有限上下文模型的思想。为了预测，我们可以想象从左到右阅读一个文档中的未知文本，用语言模型去猜接下来的内容，并为每个可能出现的符号赋予一个概率。如果猜得很准，我们就能用更少的位数为文档编码了。

为了检索，我们的目标有一点点变化。我们必须想象我们已经有一个相关文档的例子并且提出问题：*用户为了检索出类似这个文档的文档，会以多大的概率输入词项 t?*

为了回答这个问题，我们用文档为将要输入用来检索该文档的查询创建一个语言模型。我们的目标是从文档 d 中构建出一个文档语言模型 $\mathcal{M}_d(t)$。最简单的文档语言模型是基于统计每个词在文档中出现次数的最大似然模型 $\mathcal{M}_d^{\text{ml}}(t)$：

$$\mathcal{M}_d^{\text{ml}}(t)=\frac{f_{t,d}}{l_d} \tag{9-12}$$

在这里，$f_{t,d}$ 仍是指词项 t 出现在 d 中的次数，l_d 是文档的长度。因此，对于没有出现在文档中的词项 $\mathcal{M}_d^{\text{ml}}(t)=0$。因为 $\mathcal{M}_d^{\text{ml}}(t)$ 是一个概率，因此有

$$\sum_{t\in\mathcal{V}}\mathcal{M}_d^{\text{ml}}(t)=\sum_{t\in d}\mathcal{M}_d^{\text{ml}}(t)=\sum_{t\in d}f_{t,d}/l_d=l_d/l_d=1 \tag{9-13}$$

举个例子，词项“lord”在长度为 43 314 的《Hamlet》中出现了 624 次，但在长度为26 807 的《Macbeth》中仅出现了 78 次。因此，将基于《Hamlet》的语言模型和基于《Macbeth》的语言模型进行比较，前者对“lord”一词的预测概率比后者要高。

$$\mathcal{M}_{\text{Hamlet}}^{\text{ml}}(\text{“lord”})=\frac{f_{\text{lord,Hamlet}}}{l_{\text{Hamlet}}}=\frac{624}{43\ 314}\approx 1.441\%$$

$$\mathcal{M}_{\text{Macbeth}}^{\text{ml}}(\text{“lord”})=\frac{f_{\text{lord,Macbeth}}}{l_{\text{Macbeth}}}=\frac{78}{26\ 807}\approx 0.291\%$$

另一个例子，因为“lady”在《Hamlet》中出现了 30 次而在《Macbeth》中出现了 196 次，所以前者对“lady”一词项的预测概率要比后者低。

$$\mathcal{M}^{\text{ml}}_{\text{Hamlet}}(\text{“lady”})=\frac{f_{\text{lady,Hamlet}}}{l_{\text{Hamlet}}}=\frac{30}{43\ 314}\approx 0.069\%$$

$$\mathcal{M}^{\text{ml}}_{\text{Macbeth}}(\text{“lady”})=\frac{f_{\text{lady,Macbeth}}}{l_{\text{Macbeth}}}=\frac{196}{26\ 807}\approx 0.731\%$$

因为$\mathcal{M}^{\text{ml}}_d(t)$仅仅是一个用文档长度度量的词频，因此它并不足以用来计算估计概率$p(q|d)$，从而提供一个满意的文档排名。尤其是当给出的文档d只是相关文档中的一个例子，而且可能仅包含数百个词项时，这个估计有可能就不是非常准确了。更糟的是，这个模型将未出现在文档中的所有词项的概率值都赋为 0，这意味着这些词项都不可能出现在查询中。

在信息检索和例如语音识别等其他领域中，为了解决在使用基于长度相近的文档或者样例的语言模型时所产生的问题，一种很常见的方法就是使用**背景语言模型**（background language model）**去平滑**（smooth）这些模型，希望由此提高准确率。在信息检索中，将文档集视为一个整体，为这种背景模型提供了一个简便的基础。

基于将文档集视为一个整体来计算的词频，定义$\mathcal{M}_C(t)$为最大似然语言模型：

$$\mathcal{M}_C(t)=l_t/l_C \tag{9-14}$$

其中，l_t是词项t在文档集C中出现的次数，l_C是文档集中词条的总数。

$$\mathcal{M}_C(\text{“lord”})=\frac{l_{\text{lord}}}{l_C}=\frac{3346}{1\ 271\ 504}\approx 0.263\%$$

$$\mathcal{M}_C(\text{“lady”})=\frac{l_{\text{lady}}}{l_C}=\frac{1031}{1\ 271\ 504}\approx 0.081\%$$

与第 1 章不同，在这个例子中l_C包括所有的词条、标签和单词。

现在看看两个著名的平滑方法。第一个是 Jelinek-Mercer **平滑**（Jelinek-Mercer smoothing）（Jelinek 和 Mercer，1980；Chen 和 Goodman，1998），它是文档语言模型和集合语言模型的一个简单的线性组合，

$$\mathcal{M}^{\lambda}_d(t)=(1-\lambda)\cdot\mathcal{M}^{\text{ml}}_d(t)+\lambda\cdot\mathcal{M}_C(t) \tag{9-15}$$

其中，λ是一个介于 0～1 间的参数，用于控制文档语言模型和集合语言模型的相对权重。例如，当$\lambda=0.5$时，

$$\begin{aligned}\mathcal{M}^{\lambda}_{\text{Hamlet}}(\text{“lord”}) &= (1-\lambda)\cdot\mathcal{M}^{\text{ml}}_{\text{Hamlet}}(\text{“lord”})+\lambda\cdot\mathcal{M}_C(\text{“lord”})\\ &= 0.5\cdot\frac{78}{26\ 807}+0.5\cdot\frac{3346}{1\ 271\ 504}\approx 0.277\%\end{aligned}$$

因为$\mathcal{M}^{\lambda}_d(t)$是一个概率，因此有

$$\sum_{t\in\mathcal{V}}(1-\lambda)\cdot\mathcal{M}^{\text{ml}}_d(t)+\lambda\cdot\mathcal{M}_C(t)=(1-\lambda)\cdot\sum_{t\in\mathcal{V}}\mathcal{M}^{\text{ml}}_d(t)+\lambda\cdot\sum_{t\in\mathcal{V}}\mathcal{M}_C(t)=1 \tag{9-16}$$

第二个平滑方法假定为文档集中的每一个文档都额外加入了$\mu>0$个词条，同时根据集合文档语言模型$\mathcal{M}_C(t)$分布这些词条。例如，当$\mu=1000$时，概念上向每个文档中添加$\mu\cdot\mathcal{M}_C(\text{“lord”})=2.6315$个词项“lord”。显然，现实中不可能向文档中添加小数个词项，但是在数学上这是可行的。基于这些新文档的最大似然语言模型为

$$\mathcal{M}^{\mu}_d(t)=\frac{f_{t,d}+\mu\mathcal{M}_C(t)}{l_d+\mu}$$

其中，$f_{t,d}$是词项t在原文档中出现的次数，l_d是原文档的长度。额外加入的词项所产生的影响依赖于文档的长度。文档越长，影响越小，当文档无限长的时候，它的值就接近于$\mathcal{M}_d(t)$。这种平滑方法称为 **Dirichlet 平滑**（Dirichlet smoothing）（Chen 和 Goodman，

1998），因为可由带适当参数的 Dirichlet 分布衍生而来的。

如果词项 t 没有出现在文档 d 中，那么 $\mathcal{M}_d^{\lambda}(t)=\lambda \cdot \mathcal{M}_C(t)$ 并且 $\mathcal{M}_d^{u}(t)=(\mu/(l_d+\mu)) \cdot \mathcal{M}_C(t)$。

这两种平滑模型中都赋予了一个文档集概率的常数因子，而赋予未出现在文档中的词项的相对概率都是一样的。

9.3 使用语言模型排名

给定 d 为相关文档的一个例子，公式（9-10）表明我们是根据用户输入 q 作为查询的概率来对文档进行排名的。现在准备应用 9.2 节的平滑语言模型估计这个概率。首先，利用类似于 8.2 节对查询 Q 所作的独立性假设，可以把问题简化为估计查询中每个词项的概率。给定一个查询向量 $q=\langle t_1, t_2, \cdots, t_n \rangle$ 和一个文档 d，可以估计概率 $p(q|d)$ 为

$$p(q|d)=p(|q|=n) \cdot \prod_{i=1}^{n} p(t_i|d) \tag{9-17}$$

其中，$p(|q|=n)$ 表示用户输入长度为 n 的查询的概率。为了估计用户输入查询 q 的概率，我们必须为查询长度估计这个概率——假设这个长度独立于文档和查询词项以及每个独立的词项的概率——假设选择一个词项独立于选择其他词项。

幸好，忽略查询长度是一种安全的做法。因为查询是固定的，只考虑用户输入 q 的长度为 n 的概率就足够了，不需要考虑输入为任意查询长度的概率。在 9.4 节将主要讨论在语言模型方法下的查询扩展，那里才需要估计查询长度。现在，估计概率 $p(q|d)$ 为

$$p(q|d)=\prod_{i=1}^{n} p(t_i|d) \tag{9-18}$$

因为词项在查询向量中出现的顺序在该公式中并没有意义，所以可以把 q 看做一个查询词项集合，并直接在公式中使用查询词项的词频：

$$p(q|d)=\prod_{t \in q} p(t|d)^{q_t} \tag{9-19}$$

可用文档语言模型为每个 $p(t|d)$ 估计一个概率值，因此，根据以下公式排名文档：

$$p(q|d)=\prod_{t \in q} \mathcal{M}_d(t)^{q_t} \tag{9-20}$$

其中，$\mathcal{M}_d(t)$ 可以是 $\mathcal{M}_d^{\lambda}(t)$、$\mathcal{M}_d^{u}(t)$ 或者其他文档语言模型。如果用 $\mathcal{M}_d^{\lambda}(t)$ 代换 $\mathcal{M}_d(t)$，则有

$$p(q|d)=\prod_{t \in q} \left((1-\lambda) \cdot \mathcal{M}_d^{\mathrm{ml}}(t)+\lambda \cdot \mathcal{M}_C(t)\right)^{q_t} \tag{9-21}$$

如果用 $\mathcal{M}_d^{u}(t)$ 代换 $\mathcal{M}_d(t)$，则有

$$p(q|d)=\prod_{t \in q} \left(\frac{f_{t,d}+\mu \mathcal{M}_C(t)}{l_d+\mu}\right)^{q_t} \tag{9-22}$$

讨论到此为止，我们可使用公式（9-21）或公式（9-22）进行文档排名。然而，为了对语言模型方法及它与其他方法的关系获得一些深入的了解，接下来几页还将继续做一些值得的扩展讨论。你也许注意到，很多在第 2 章和第 8 章看到的排名公式中的特性在这些公式中都没有出现。与前面的公式不同，这些公式并没有采用 TF-IDF 形式。并且，文档集中文档的数量（N）和包含词项的文档数量（N_t）也没有出现在这些公式中。尽管如此，在简化这些公式的过程中，很容易找出隐藏在这些公式中的类似于 TF-IDF 的形式。

为了简单，同时也为与前面的方法保持一致，从现在开始我们使用对数而不是直接使用概率。首先，分别考虑文档中包含的词项和文档中不包含的词项。

$$\log p(q|d) = \sum_{t\in q} q_t \cdot \log p(t|d) \tag{9-23}$$

$$= \sum_{t\in q\cap d} q_t \cdot \log p(t|d) + \sum_{t\in q\setminus d} q_t \cdot \log p(t|d) \tag{9-24}$$

当文档中包含词项 t 时，我们使用$\mathcal{M}_d(t)$去估计概率 $p(t|d)$，其中$\mathcal{M}_d(t)$可以是$\mathcal{M}_d^{\lambda}(t)$或者$\mathcal{M}_d^{\mu}(t)$。当文档中不包含词项 t 时，$\mathcal{M}_d(t)$可以使用 $\alpha_d\ \mathcal{M}_C(t)$这种形式，其中 α_d 是一个与 d 的特性有关的因子（而与查询 q 无关）。正如 9.2 节最后所讲的那样，对于 $\alpha_d=\lambda$有$\mathcal{M}_d^{\lambda}(t)$，对于 $\alpha_d=\mu/(l_d+\mu)$有$\mathcal{M}_d^{\mu}(t)$。

我们将$\mathcal{M}_d(t)$和 $\alpha_d \cdot \mathcal{M}_C(t)$代入公式（9-24）中，并稍作变型，消去未出现在文档中的词项的累加和：

$$\log p(q|d) = \sum_{t\in q\cap d} q_t \cdot \log \mathcal{M}_d(t) + \sum_{t\in q\setminus d} q_t \cdot \log(\alpha_d \cdot \mathcal{M}_C(t)) \tag{9-25}$$

$$= \sum_{t\in q\cap d} q_t \cdot \log \mathcal{M}_d(t) - \sum_{t\in q\cap d} q_t \cdot \log(\alpha_d \cdot \mathcal{M}_C(t)) \tag{9-26}$$

$$+ \sum_{t\in q\cap d} q_t \cdot \log(\alpha_d \cdot \mathcal{M}_C(t)) + \sum_{t\in q\setminus d} q_t \cdot \log(\alpha_d \cdot \mathcal{M}_C(t)) \tag{9-27}$$

$$= \sum_{t\in q\cap d} q_t \cdot \log \frac{\mathcal{M}_d(t)}{\alpha_d \mathcal{M}_C(t)} + \sum_{t\in q} q_t \cdot \log(\alpha_d \cdot \mathcal{M}_C(t)) \tag{9-28}$$

$$= \sum_{t\in q\cap d} q_t \cdot \log \frac{\mathcal{M}_d(t)}{\alpha_d \mathcal{M}_C(t)} + n \cdot \log \alpha_d + \sum_{t\in q} q_t \cdot \log \mathcal{M}_C(t) \tag{9-29}$$

这里，$n=\sum_{t\in q} q_t$表示查询中词条的个数。最后一项，$\sum_{t\in q} q_t \cdot \log \mathcal{M}_C(t)$，对于所有文档来说都是一个常量，可以去掉该项从而得到等价公式：

$$\sum_{t\in q\cap d} q_t \cdot \log \frac{\mathcal{M}_d(t)}{\alpha_d \mathcal{M}_C(t)} + n \cdot \log \alpha_d \tag{9-30}$$

该公式由两部分组成。与前面章节中提到的排名公式相似，左边部分是出现在文档中的查询词项的权重和。右边部分可看做是对特定文档的调整或归一化，该部分独立于特定的查询词项，但并不独立于查询或者文档的长度。

现在分别用这两种不同的平滑语言模型，$\mathcal{M}_d^{\lambda}(t)$和$\mathcal{M}_d^{\mu}(t)$，替换公式（9-30）中的$\mathcal{M}_d(t)$。代入$\mathcal{M}_d^{\lambda}(t)$得到

$$\sum_{t\in q\cap d} q_t \cdot \log \frac{\mathcal{M}_d^{\lambda}(t)}{\alpha_d \mathcal{M}_C(t)} + n \cdot \log \alpha_d = \sum_{t\in q\cap d} q_t \cdot \log \frac{(1-\lambda)\mathcal{M}_d^{\mathrm{ml}}(t) + \lambda \mathcal{M}_C(t)}{\lambda \mathcal{M}_C(t)} + n \cdot \log \lambda$$

$$= \sum_{t\in q\cap d} q_t \cdot \log \frac{(1-\lambda) f_{t,d}/l_d + \lambda l_t/l_C}{\lambda l_t/l_C} + n \cdot \log \lambda$$

其中，$n \cdot \log\lambda$是一个常数，可以去掉。做一个小小的变换后可得到最终的排名公式：

$$\sum_{t\in q} q_t \cdot \log\left(1 + \frac{1-\lambda}{\lambda} \cdot \frac{f_{t,d}}{l_d} \cdot \frac{l_C}{l_t}\right) \tag{9-31}$$

在本书的后面章节中，我们将此公式称为 Jelinek-Mercer **平滑语言模型**（language mod-

eling with Jelinek-Mercer smoothing，LMJM)。通常，λ的最优值与查询长度有关，其值越大，越适合用于长查询。在缺少训练数据或其他信息的情况下，经验值$\lambda=0.5$是比较合理的。如非特别说明，本书的例子和实验都取$\lambda=0.5$。

回到表 2-1，考虑其中的文档集。给定查询〈“quarrel”，“sir”〉，可以计算文档 1 的 LMJM 得分如下：

$$\log\left(1+\frac{1-\lambda}{\lambda}\cdot\frac{f_{\text{quarrel},1}}{l_1}\cdot\frac{l_C}{l_{\text{quarrel}}}\right)+\log\left(1+\frac{1-\lambda}{\lambda}\cdot\frac{f_{\text{sir},1}}{l_1}\cdot\frac{l_C}{l_{\text{sir}}}\right)$$
$$=\log\left(1+\frac{0.5}{0.5}\cdot\frac{1}{4}\cdot\frac{28}{2}\right)+\log\left(1+\frac{0.5}{0.5}\cdot\frac{1}{4}\cdot\frac{28}{5}\right)\approx 3.43$$

在开始前，有必要先简单研究一下公式（9-31）的结构。基于文档集的统计信息，如 N 和 N_t，并未出现在公式中。然而，出现在对数中的文档集语言模型 $l_C\,|\,l_t$，使人联想到 IDF，它与 $f_{t,d}$形式相近，也使人联想到 TF－IDF，因此第 2 章和第 8 章中的公式的一些性质也在这里体现出来了。

文档的长度扮演了一个十分有趣的角色。$f_{t,d}\,|\,l_d$ 是指每个文档词条中词项 t 出现的平均次数。当与 $l_C\,|\,l_t$ 相乘后，其结果解释为每词条中词项实际出现的次数与文档集语言模型中期望出现次数之比。

接下来，我们考虑 Dirichlet 平滑。用$\mathcal{M}_d^{\mu}(t)$代替公式（9-30）中的$\mathcal{M}_d(t)$，并在 $\mu>0$ 时，简化排名公式，有

$$\sum_{t\in q\cap d} q_t\cdot\log\frac{\mathcal{M}_d^{\mu}(t)}{\alpha_d\mathcal{M}_C(t)}+n\cdot\log\alpha_d=\sum_{t\in q} q_t\cdot\log\left(1+\frac{f_{t,d}}{\mu}\cdot\frac{l_C}{l_t}\right)-n\cdot\log\left(1+\frac{l_d}{\mu}\right)\qquad(9\text{-}32)$$

我们将其称为 Dirichlet **平滑语言模型**（language modeling with Dirichlet smoothing，LMD)。在这个公式中，文档长度规范化分解为公式的右边。与公式（9-31）一样，你会发现在词频和集合语言模型中的关系与 TF-IDF 形式相近。如非特别说明，本书的例子和实验用到该公式时，取 $\mu=1000$。但是需要强调一点，在实际应用中的值应根据相应的训练数据得出（见第 11 章)。

如果文档集中的所有文档长度相同，通过令$\mu=l_d\cdot\frac{\lambda}{1-\lambda}$可使这两种平滑方法等价。观察发现，基于一个已有的值 λ，通过用平均文档长度 l_{avg}代替 l_d，可为 μ 选择一个合适的值。如果把 0.5 作为 λ 的默认值，对应 μ 的默认值就是平均文档长度。用 l_{avg}代替公式（9-32）中的 μ，且 $l_{\text{avg}}=l_C/N$，那么 LMD 排名公式化简为：

$$\sum_{t\in q} q_t\cdot\log\left(1+f_{t,d}\cdot\frac{N}{l_t}\right)-n\cdot\log\left(1+\frac{l_d}{l_{\text{avg}}}\right)\qquad(9\text{-}33)$$

公式中出现的 N 加强了该公式与之前提到的 IDF 的关系。公式中同样也隐含着在 Okapi BM25 中出现的文档长度规范化。

对于查询〈“quarrel”，“sir”〉，使用 LMD 对表 2-1 中的文档集进行排名，可计算出文档 1 的得分为：

$$\log\left(1+f_{\text{quarrel},1}\cdot\frac{N}{l_{\text{quarrel}}}\right)+\log\left(1+f_{\text{sir},1}\cdot\frac{N}{l_{\text{sir}}}\right)-n\cdot\log\left(1+\frac{l_1}{l_{\text{avg}}}\right)$$
$$=\log\left(1+1\cdot\frac{5}{2}\right)+\log\left(1+1\cdot\frac{5}{5}\right)-2\cdot\log\left(1+\frac{4}{5.6}\right)\approx 1.25$$

9.4 Kullback-Leibler 距离

另一种用于理解和使用语言模型方法的理论框架是 Kullback-Leibler **距离**（Kullback-

Leibler Divergence，KL 距离）。KL 距离，也称**相对熵**（relative entropy），是一种比较两个概率分布的方法。

给定连续概率分布 $f(x)$和 $g(x)$，它们之间的 KL 距离定义为

$$\int_{-\infty}^{\infty} f(x) \cdot \log \frac{f(x)}{g(x)} \, \mathrm{d}x \tag{9-34}$$

信息检索中通常使用离散分布，这时 KL 距离的形式为

$$\sum_{x} f(x) \cdot \log \frac{f(x)}{g(x)} \tag{9-35}$$

其值越大表示差异越大。当 f 和 g 代表相同分布时，它们的 KL 距离为 0，因为 $\log\left(f(x)/g(x)\right)=\log 1=0$。

举个例子，抛一枚“公平”的硬币，出现正面和反面的概率是相等的。如果抛一枚“不公平”的硬币，出现正面概率为 40%、反面概率为 60%，那么两枚硬币的 KL 距离可如下计算

$$0.5 \cdot \log \frac{0.5}{0.4} + 0.5 \cdot \log \frac{0.5}{0.6} \approx 0.02945$$

因为对数的底是可以是任意的，所以在这个例子中，我们选 2 作为对数的底。

KL 距离是不对称的，互换 $f(x)$和 $g(x)$的位置将产生不同的值。例如，我们互换“公平”和“不公平”两枚硬币，KL 距离就变为

$$0.4 \cdot \log \frac{0.4}{0.5} + 0.6 \cdot \log \frac{0.6}{0.5} = 0.02905$$

由于 KL 距离的不对称性，有时用它来比较一个“真实”分布与另一个分布，例如公式（9-35）中的 $f(x)$就表示这种真实的分布。从信息理论的角度来看，假设一个消息中的符号服从 g 分布而不是真实分布 f，KL 距离就表示传输或压缩这个消息时，每个符号平均需要额外增加的位数。

为了能将 KL 距离用于排名，我们使用从文档中构造出语言模型一样的方法，从查询 $M_d(t)$中构造出语言模型。最简单的语言模型是最大似然模型：词项 t 在查询中出现的次数与查询长度的比：

$$\mathcal{M}_q^{\mathrm{ml}}(t)=\frac{q_t}{n} \tag{9-36}$$

正如为文档语言模型所做的那样，当然可以通过平滑或其他处理来构造更加复杂的查询语言模型。

通过计算查询语言模型和文档语言模型的差异，可将 KL 距离应用于文档排名：

$$\sum_{t\in\mathcal{V}} \mathcal{M}_q(t) \cdot \log \frac{\mathcal{M}_q(t)}{\mathcal{M}_d(t)} = \sum_{t\in\mathcal{V}} \mathcal{M}_q(t) \cdot \log \mathcal{M}_q(t) - \sum_{t\in\mathcal{V}} \mathcal{M}_q(t) \cdot \log \mathcal{M}_d(t) \tag{9-37}$$

左边的累加项对所有文档来说是相同的，因为保序变换，可以将其去掉。右边的累加项，如果去掉负号，那它将随着距离减少而增大。因此，它适合作为一个排名公式：

$$\sum_{t\in\mathcal{V}} \mathcal{M}_q(t) \cdot \log \mathcal{M}_d(t) \tag{9-38}$$

现在，我们用最大似然语言模型$\mathcal{M}_q^{\mathrm{ml}}(t)$代替$\mathcal{M}_q(t)$，公式变为

$$\sum_{t\in\mathcal{V}} \mathcal{M}_q^{\mathrm{ml}}(t) \cdot \log \mathcal{M}_d(t) = \frac{1}{n} \cdot \sum_{t\in q} q_t \cdot \log \mathcal{M}_d(t) \tag{9-39}$$

去掉常量$\frac{1}{n}$可以得到等价的排名公式

$$\sum_{t \in q} q_t \cdot \log \mathcal{M}_d(t) \tag{9-40}$$

这与公式（9-20）是完全等价的（在对数空间里）。

如果在公式（9-38）中不为$\mathcal{M}_q(t)$使用最大似然模型，而是把查询语言模型作为一种查询扩展，将查询语言模型扩展，用于估计未出现在原查询中的词项的非零概率。通过这种方式，可为未包含任何查询词项的文档赋予一个正的得分。例如，给定图 1-8 的查询〈“law”，“enforcement”，“dogs”〉，一个讨论使用警犬（canine）或警犬搜索队（K-9）来搜查毒品的文档肯定是相关的。因此，通过向查询中增加词项“police”、“canine”、“K-9”、“drug”和“searches”，并赋予这些扩展词项适当的权重反映它们辅助查询的作用，可以提高查询的性能。语言模型$\mathcal{M}_q(t)$架起了原始查询词项与潜在扩展词项之间的桥梁，因此为它们的识别和加权提供了一条理论上可行的途径。

Lafferty 和 Zhai（2001）提出了一种扩展查询的语言模型方法，该方法从包含一个查询词项的某个文档出发，在文档集中随机“游走”。游走的第一步随机选择一个包含一个查询词项的文档，被选中的概率可以用词频和其他的因素来衡量。然后从新文档中根据$\mathcal{M}_d(t)$随机选出一个词项。到这里，游走将以概率 p_{stop}停止，以概率 $1-p_{\text{stop}}$继续。如果继续，那又随机选择一个包含新词项的文档，从该文档中又随机选一个词项，不停地重复上述过程。基于游走过程中停在词项 t 上的概率，可构造出查询语言模型$\mathcal{M}_q(t)$。

Lafferty 和 Zhai 为这个非正式的想法介绍了一个形式化矩阵的实现方法，有效地使得可同时在所有的词项上游走。为了提高效率，游走几步后将停止，以避免给大量的词项都赋予一个很小的概率。他们的报告指出，在几个文档集上，包括 TREC45，该方法的有效性都得到了明显的提升。

9.5 随机差异性

信息检索中的**随机差异性**（divergence from randomness，DFR）方法明确假设词项在文档集中的分布是一个随机过程，然后根据文档中实际词项分布概率对文档进行排名。类似的做法也隐含在语言模型方法中，即把集合语言模型结合到平滑处理中去。例如，我们注意到在公式（9-31）中包含了一个比率，即词项在文档出现的实际次数与基于集合语言模型计算的期望出现次数的比。

DFR 的一个重要性质是它没有其他方法中的那些任意参数，这些参数需要训练集来确定。一些参数，例如 LMD 中的 μ、LMJM 中的 λ，BM25 中的 k_1，通常都很不直观，与它们最初出现的简单解释相悖。DFR 以无参数形式获取了和这些方法相媲美的检索有效性。

本节我们重温一下 8.4 节中精华性的概念。已知一个随机文档 d 中词项 t 出现次数为 $f_{t,d}$，其中 $f_{t,d}>0$，我们用精华性的概念去估计文档确实“关于”该词项含义的概率。在 8.4 节中这个概念是基于一个二值泊松分布建模的。在这一节，将用一种基于拉普拉斯**连续法则**（Laplace's Law of succession）的方法去建模。

DFR 方法最基本的核心可以归结为以下这条公式

$$(1-P_2)\cdot(-\log P_1) \tag{9-41}$$

在这个公式中，P_1 代表一个随机文档 d 中恰好包含 $f_{t,d}$ 次词项 t 的概率。对数值 $-\log P_1$ 可视为信息的位数，称作**自相关信息**（self-information），它与恰好包含 $f_{t,d}$ 次 t 的文档 d 有关（见 6.1 节）。把词项 t 分配到各个文档中的随机过程并不是把大部分的词项分配到特定文档 d 中，因此 P_1 将随着 $f_{t,d}$ 的增加而迅速减小。

P_2 提供一个反映精华性的修正，以校正这种迅速减小的现象。如果 d 在 t 中是精华，我们可以猜测 t 出现在文档中并非偶然。假设我们开始阅读 d（在 t 中是精华），从而计算出 t 在 d 中出现的次数。在读到它的结尾前，我们发现 t 出现了 $f_{t,d}-1$ 次。这个发现表明我们期望发现更多次 t，P_2 表示至少发现一次 t 的概率。P_2 随 $f_{t,d}$ 增大而增大。这样，公式（9-41）中的 $1-P_2$ 随着 $f_{t,d}$ 的增大而减小。

到目前为止，我们只讨论了单个词项的情况。为了根据多词项查询给文档排名，我们又使用了独立性假设，给出以下排名公式

$$\sum_{t\in q} q_t \cdot (1-P_2)\cdot(-\log P_1) \tag{9-42}$$

接下来主要关注估计一个给定词项 t 的 P_1 和 P_2，因为这个公式将用这些估计值来排名。

DFR 理论由 Amati 和 van Rijsbergen（2002）提出，并通过了 TREC 会议上实验的验证（Plachouras 等人，2002；Amati 等人，2003）。为了给公式（9-41）提供大量的理论支撑，Amati 和 van Rijsbergen（2002）提出并评价了七种估计 P_1 和两种估计 P_2 的方法。在这一章，我们分别只详细地讨论一种，阐述这些方法的推理过程及其检索有效性。

另外，Amati 和 van Rijsbergen 展示了两种将文档长度规范化与 DFR 结合的方法。目前，暂时忽略这一复杂的问题，而假设所有文档的长度是相同的。在本章的最后，将会用公式（8-45）的思想去处理文档长度规范化的问题，为这些文件计算一个修正词频，并以此代替实际的词频。

9.5.1　一个随机模型

假设把词项随机分布到文档中。如果词项 t 在 N 个文档出现了 l_t 次，则

$$f_{t,1}+f_{t,2}+\cdots+f_{t,N}=l_t \tag{9-43}$$

其中，$f_{t,i}$ 是词项 t 在第 i 个文档中出现的次数。假设文档是无差别的，共有多少种不同的方式可将 l_t 次 t 的出现分布到 N 个文档中呢？例如，将 t 的 4 次出现分到 3 个文档中共有 4 种不同的方式：（1）全部出现在同一个文档中；（2）一个文档中出现了 3 次，另一个文档中出现了 1 次；（3）一个文档中出现 2 次，另外两个文档中各出现 1 次；（4）两个文档中各出现 2 次。换句话说，有多少种不同的词项安排方式使之满足公式（9-43）？

为了回答这个问题，Amati 和 van Rijsbergen 发觉这是一个与 Bose-Einstein **统计**（Bose-Einstein statistics）相同的问题。Bose-Einstein 统计用于计算无区别的粒子在热平衡下可能处于的各种不同的能量状态。解决办法可用一个二项式系数表示：

$$\binom{N+l_t-1}{l_t}=\frac{(N+l_t-1)!}{(N-1)!\,l_t!} \tag{9-44}$$

另外，这个问题也等价于求将 m 个球放到 n 个不同的盒子中的所有组合数，即组合的**分装问题**（occupancy problem）。

为了计算词项 t 的估计概率 P_1，假设一个给定文档 d 中 t 出现的次数为 $f_{t,d}$。把剩下的 $l_t-f_{t,d}$ 次出现随机分配到剩余文档中，满足如下公式

$$f_{t,1}+\cdots+f_{t,d-1}+f_{t,d+1}+\cdots+f_{t,N}=l_t-f_{t,d} \tag{9-45}$$

满足这个公式的组合数可由公式（9-44）得到：

$$\binom{(N-1)+(l_t-f_{t,d})-1}{l_t-f_{t,d}}=\frac{((N-1)+(l_t-f_{t,d})-1)!}{(N-2)!\,(l_t-f_{t,d})!} \tag{9-46}$$

这个公式假设一个被选中的文档中已包含 $f_{t,d}$ 次 t 时，剩余词项的分配方式数量。公式(9-44)表示将 l_t 次 t 的出现分配到 N 个文档中的方式数量。这样，这两条公式的比值就是 P_1，即一个被选中的文档包含 $f_{t,d}$ 次 t 的概率：

$$P_1=\frac{\begin{pmatrix}(N-1)+(l_t-f_{t,d})-1\\ l_t-f_{t,d}\end{pmatrix}}{\begin{pmatrix}N+l_t-1\\ l_t\end{pmatrix}}=\frac{((N-1)+(l_t-f_{t,d})-1)!\ (N-1)!\ l_t!}{(N-2)!(l_t-f_{t,d})!\ (N+l_t-1)!} \tag{9-47}$$

遗憾的是，公式（9-47）中出现了阶乘使其无法直接使用。为此，Amati 和 van Rijsbergen 提出了两种估计该值的方法，并证明了这两种方法与原来的方法性能相近。用这些方法对 P_1 进行估计可以简单表示为：

$$P_1=\left(\frac{1}{1+l_t/N}\right)\left(\frac{l_t/N}{1+l_t/N}\right)^{f_{t,d}} \tag{9-48}$$

因此有

$$-\log P_1=\log(1+l_t/N)+f_{t,d}\cdot\log(1+N/l_t) \tag{9-49}$$

公式右半部分容易让人联想到 TF-IDF 的形式，但是这里用 l_t 代替了 N_t。

9.5.2 精华性

Amati 和 van Rijsbergen 通过拉普拉斯连续法则获得了 p_2 的一个估计，这用一个例子可以更好地说明。假设我们连续在 $m-1$ 个早晨都能看到日出。由于缺少其他信息，如地球围绕太阳运行的物理模型等，那么明早太阳会升起这一事件发生的概率会有多大呢？尽管我们相当确定这一定会发生，但非常肯定地将这个概率赋予 1 也是不恰当的。毕竟明天太阳还是有可能不会升起。因此（在没有得到有关的数学方面细节的情况下），连续法则指出这一概率值应为 $m/(m+1)$。

Amati 和 van Rijsbergen 应用连续法则进行以下估计

$$P_2=\frac{f_{t,d}}{f_{t,d}+1} \tag{9-50}$$

他们将这个公式解释为：假设相关文档的长度非常大，文档［…］中的词项中再多出现一个词条的条件概率。将这个估计公式和公式（9-49）一起代入公式（9-41）可得

$$(1-P_2)(-\log P_1)=\frac{\log(1+l_t/N)+f_{t,d}\log(1+N/l_t)}{f_{t,d}+1} \tag{9-51}$$

这个公式中的词频部分和 Okapi BM25 排名公式（公式（8-48））中的词频部分相似，并且拥有类似的饱和性。忽略 $f_{t,d}$ 的值，公式（9-51）的值上限为 $\log(1+l_t/N)+\log(1+N/l_t)$。

9.5.3 文档长度规范化

公式（9-51）假设所有文档长度相同。当文档长度改变时，Amati 和 van Rijsbergen 提出一种规范化方法，即用修正词频 $f'_{t,d}$ 替换公式中的 $f_{t,d}$。他们通过计算这个修正词频派生出两种估计方法并进行了评价。这些方法可以更好地表示为

$$f'_{t,d}=f_{t,d}\cdot\log(1+l_{\text{avg}}/l_d) \tag{9-52}$$

他们将公式（9-49）和公式（9-50）合并，并以此修正公式（9-52），可以得到 DFR 方法的 GL2 变形。我们用这种变形来描述本书中出现的实验。

9.6　段落检索及排名

本书中提及的大部分排名方法都是针对文档而言的。根据应用环境的不同，这些文档可能是网页、新闻、电子邮件或这些形式的结合以及其他文档类型。在某些环境中可能对文档内的元素进行排名会更合适，如书本中的页面或新闻中的章节等。在另一些环境中，也许将文档内的文本片段作为排名结果返回会更令人满意，这就是**任意段落检索**（arbitrary passage retrieval）。

这样的一个应用环境就是要构造一个文本片段，用于为用户呈现一个检索到的文档内容的简单概要。图 9-1 展示了搜索引擎从维基百科全书中对查询〈“shakespeare”，“marriage”〉返回的结果。搜索引擎从文档正文中抽取出一个片段，说明了搜索词项出现的内容，连同其网页标题和地址一起返回给用户。

William **Shakespeare** — Wikipedia, the free encyclopedia
At the age of 18, **Shakespeare** married the 26-year-old Anne Hathaway. The consistory court of the Diocese of Worcester issued a **marriage** licence on 27 November 1582. Two of Hathaway's neighbours posted bonds the next day ...
en.wikipedia.org/wiki/William_**Shakespeare**

图 9-1　在维基百科全书中，查询〈“shakespeare”，“marriage”〉的一个典型的搜索引擎返回的结果。结果是内容中包含了查询词项的文档中的一个简要片段

问答系统是任意段落检索的另外一个有价值的应用（Tellex 等人，2003）。给出一个查询如“What is the population of India?”，问答系统不是给出一个包含答案的文档，而是直接给出确切答案（“11.2 亿”）。段落检索通常是问答系统的前期步骤。信息检索系统会从问题中提取关键词，并构造出一个查询〈“population”，“india”〉用于处理。先从语料库中检索出包含这些词项的段落，然后对其进行分析以提取并验证出可能的答案。以相邻位置包含查询词项的文本区间（“…population of India，…”）就有可能是一个包含了答案（“…The population of India is…”）的段落，这个段落虽然看起来稍长但对完整文档而言仍然算短的。这个区间表示了文档中的“热点”，其中就有可能找到答案。

在 2.2.2 节中我们提到了一种纯粹基于词项邻近度的排名方法。该方法为查询向量 $q=\langle t_1, t_2, \cdots, t_n\rangle$ 确定一个**覆盖**（cover），这个覆盖可以定义成一个文本区间 $[u, v]$，每个查询词项在区间中至少出现一次，并且不包含一个更小区间也满足上述条件。

现在我们将覆盖的概念扩展为一个查询词项的子集。对于 $m \leqslant n$ 个词项，我们定义一个 ***m*-覆盖**（m-cover）为文档集上的一个区间 $[u, v]$，包括查询中 m 个不同词项的至少一次出现，并且不包含一个子区间也满足上述条件。为简单起见，我们假设查询 q 中没有重复的词项，因为本节中讨论的段落排名方法不考虑重复查询词项。例如，表 2-1 中查询向量〈“you”，“quarrel”，“sir”〉的 2-覆盖由以下区间组成：[2，3]、[3，4]、[4，5]、[5，6]、[8，10]、[10，12]、[12，16]、[24，28]。注意到区间 [12，24] 不在这个集合中，因为已经包含了 [12，16]。再注意到集合包含了区间 [24，28]，尽管它跨越了最后三个文档。在多数应用中，诸如此类的 m-覆盖在使用之前多数都已经被过滤掉了。然而，为简单起见，我们定义 m-覆盖时不考虑文档边界和其他结构，而应用适当的后处理步骤来处理。

m-覆盖的概念可用于支持问答系统中的片段生成和任意段落检索。假设我们在一个已索引网页中查找一个片段来显示。理想情况下，可以显示这样的片段，其中所有查询词项尽

可能地靠近，但是这个目标不总是可能的。有一些查询词项距离非常远，分别在文档开始处和结尾处。有一些查询词项只在文档标题中出现，而没有出现在正文中。有一些查询词项根本就没有在文档中出现。因此片段可能由文本碎片组合而成，这些碎片对应一个或多个 m-覆盖，覆盖中包含了尽可能多的查询词项。一个合适的 m-覆盖集一经确定，每一个覆盖都被扩展到最近的句子边界，然后裁剪为合适的尺寸。

在问答系统中，对比一个包含了所有词项但长得多的段落，一个包含了查询词项严格子集且词项出现位置邻近的段落要更接近答案。例如，假设我们要回答问题“Who starred in the film *Shakespeare in Love*?”我们可能设定查询向量为〈“starred”，“film”，“Shakespeare”，“love”〉并且分析可能的答案。相比一个包含全部词项的长段落，一个包含词项“Shakespeare”和“love”以及“starred”或“film”中的一个的 3-覆盖短段落更有可能指示了答案，因为长段落中有可能混合了其他姓名和内容。更一般的，对于片段生成和问答这样的应用，我们需要权衡区间长度和包含的词项多少来进行区间的选择。

9.6.1 段落评分

假设我们根据查询 $q=\langle t_1, t_2, \cdots, t_n\rangle$ 为区间 $[u, v]$ 评分，且该区间包含了查询词项的一个子集 $q'\subseteq q$，其中 $q'=\langle t_1', t_2', \cdots, t_m'\rangle$ 且 $m\leqslant n$。我们通过估计一个随机选择的等长区间中至少包含一个查询词项的概率，来为长度为 $l=v-u+1$ 的区间 $[u, v]$ 评分。正如 9.5 节所述，段落评分方法将词项的实际分布与一个随机分布关联起来。

为了这一目标，我们对文档集建模，将其视为一系列独立产生的词项，并假设对于 $t\in q'$，存在一个固定概率 p_t 匹配任意给定的文档位置。注意到这个假设允许多个词项匹配一个位置。尽管不现实，当 p_t 很小时，这个假设对于大多数词项还是可以接受的，这有助于简化段落评分公式的引入。

给定一个长度 $l=v-u+1$ 的区间 $[u, v]$，区间中包含一次或多次 t 的概率 $p(t,l)$ 为：

$$p(t,l)=1-(1-p_t)^l \tag{9-53}$$

$$=1-(1-lp_t+O(p_t{}^2)) \tag{9-54}$$

$$\approx l\cdot p_t \tag{9-55}$$

那么区间 $[u, v]$ 包含 q' 中所有词项的概率为：

$$p(q',l)=\prod_{t\in q'}p(t,l)\approx\prod_{t\in q'}l\cdot p_t=l^m\cdot\prod_{t\in q'}p_t \tag{9-56}$$

最后，我们估计 p_t 的值作为词项 t 的集合频率：

$$p_t=l_t/l_C \tag{9-57}$$

其中，l_t 为 t 出现在文档集中的总次数，l_C 为文档集的总长度。将其代入并取负对数（即自相关信息）可以得出：

$$\sum_{t\in q'}(\log(l_C/l_t))-m\cdot\log(l) \tag{9-58}$$

公式中长度和集合频率之间的关系与公式（9-32）中的关系类似。

9.6.2 实现

也许并不奇怪，确定 m-覆盖的算法就是图 2-10 中自适应算法的简单扩展，扩展算法的细节在图 9-2 中给出。对给定的 m 值，算法可以确定一个给定位置的下一个 m-覆盖。

算法第 1～2 行找到该给定位置后每个词项的下一次出现位置。第 m 大的词项位置就是

m-覆盖的终点（v）（第 3 行）。对于 v 之前出现的每个词项，为其找到 v 之前最后出现的位置（第 7～9 行）。这些值中最小的成为 m-覆盖的起点（u）。

```
    nextCover (⟨t1, ..., tn⟩, position, m) ≡
1     for i ← 1 to n do
2       V[i] ← next(ti, position)
3     v ← mth largest element of V
4     if v = ∞ then
5       return [∞, ∞]
6     u ← v
7     for i ← 1 to n do
8       if V[i] < v and prev(ti, v + 1) < u then
9         u ← prev(ti, v + 1)
10    return [u, v]
```

图 9-2　给定一个位置后，确定词项向量 $\langle t_1, \cdots, t_n \rangle$ m-覆盖的下一次出现的函数。整数数组 V 用于存储中间计算结果。

为获得全部 m-覆盖，对于所有的 $m > 1$，我们在文档集范围内重复调用这个 nextCover 的扩展版本。

```
for m ← n down to 2 do
  u ← −∞
  while u < ∞ do
    [u, v] ← nextCover(⟨t1, t2, ..., tn⟩, u, m)
    if u ≠ ∞ then
      report the m-cover [u, v]
```

没必要显式地生成 1-覆盖，因为可以直接从词项的位置列表信息中取得。

根据应用的不同，通常不需要在整个文档集上生成所有的 m-覆盖。对于片段生成，我们感兴趣的只是文档排名靠前部分的 m-覆盖。对于问答系统，我们感兴趣的只是在特定数量的文档中最优的那个 m-覆盖。Clarke 等人（2006）对第二种情况中快速生成 m-覆盖的算法优化进行了讨论。

9.7　实验对比

表 9-1 中显示了本章中讨论的检索方法的有效性。该表中的数据可与表 2-5 和表 8-2 中的数据进行对比。9.6 节中提到的段落检索算法没有出现在表中，因为该算法是针对段落的而不是针对文档的。尽管理论上它也可以应用于文档排名，通过给每个文档赋予其最大段落得分的方法可以达到这一目，但是这种方法不适用于单词项的查询。

表 9-1　本章讨论的检索方法的有效性

方法	TREC45				GOV2			
	1998		1999		2005		2006	
	P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
LMJM 公式(9-31)	0.390	0.179	0.432	0.209	0.416	0.211	0.494	0.257
LMD 公式(9-32)	0.450	0.193	0.428	0.226	0.484	0.244	0.580	0.293
DFR公式(9-51)/公式(9-52)	0.426	0.183	0.446	0.216	0.465	0.248	0.550	0.269

9.8　延伸阅读

本章中阐述的有关信息检索的语言模型方法的意义重大的工作，来自于 Ponte 和 Croft（1998），Berger 和 Lafferty（1999），Zhai 和 Lafferty（2004），以及 Hiemstra（2001）的博士学位论文。Croft 和 Lafferty 一起编纂的书中涵盖了直至 2003 年的大部分工作（Croft 和 Lafferty，2003）。Lavrenko 和 Croft（2001）对语言模型方法中存在的相关性进行了探索。其他的早期工作中包括 Miller 等人（1999）的工作，他们假设查询是从调用了隐式马尔可夫模型（HMM）的文档中产生出来的，其中λ是从一个文档状态和一个总体语言状态之间选出来的，并以集合形式表示，由此推导出了公式（9-21）。之后，他们将这个隐式马尔可夫模型框架扩展为伪相关反馈、邻近项和文档优先的联合。

在整个人类语言技术的发展中，平滑在语言模型化技术中扮演了重要角色。第 6 章中关于 Manning 和 Schütze（1999）的内容中包括了语言建模的概述。Chen 和 Goodman（1998）提出了一种平滑技术比较的具体方法。

语言建模方法奠定了现代信息检索研究的基础，过去几年间的多数信息检索会议文献集中都包含了大量基于语言建模的文章。尽管本书并不是要成为一个完整的参考文献，但是我们还是提供一些近期的研究成果：Metzler 和 Croft（2004）将语言建模方法和概率推论模型（Turtle 和 Croft，1991；Greiff 等人，1999）进行了结合；Cao 等人（2005）将依赖性集成于语言建模方法中。Tao 和 Zhai（2007）、Zhao 和 Yun（2009）将相似性指标集成于语言建模方法中。Cao 等人（2008）将伪相关反馈考虑进语言建模方法框架之中。Lv 和 Zhai（2009）考虑了集成词项的位置信息。Zhai（2008b）做了一个关于最近的信息检索的语言建模方法的调查研究。

9.9 练习

练习 9.1 你会发现某个文档集中的文档有一个 30 天的“半衰期”。经过 30 天的时间，文档的相关性先验概率 $p(r|D)$会减为开始时的一半。将这个信息结合进公式（9-8），简化这个公式为等价排名的形式，使得所有假设情况均成立。

练习 9.2 证明 Dirichlet 平滑的结果模型可视为概率分布，即证明$\sum_{t\in V}\mathcal{M}_d^{\mu}(t)=1$ 。

练习 9.3（项目练习） 实现 LMD 排名公式（公式（9-33））。并应用练习 2.13 中得到的文档集或其他任何文档集（如 TREC 文档集）来验证你的实现。

练习 9.4（项目练习） 实现 9.5 节中描述的 DFR 排名方法。并应用练习 2.13 中得到的文档集或其他任何文档集（如 TREC 文档集）来验证你的实现。

练习 9.5（项目练习） 实现 9.6 节中提到的段落检索和评分方法。并用你的实现为本书中描述的文档检索方法之一提供结果片段。

9.10 参考文献

Amati, G., Carpineto, C., and Romano, G. (2003). Fondazione Ugo Bordoni at TREC 2003: Robust and Web Track. In *Proceedings of the 12th Text REtrieval Conference*. Gaithersburg, Maryland.

Amati, G., and van Rijsbergen, C. J. (2002). Probabilistic models of information retrieval based on measuring the divergence from randomness. 20(4):357–389.

Berger, A., and Lafferty, J. (1999). Information retrieval as statistical translation. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 222–229. Berkeley, California.

Cao, G., Nie, J.Y., and Bai, J. (2005). Integrating word relationships into language models. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 298–305. Salvador, Brazil.

Cao, G., Nie, J.Y., Gao, J., and Robertson, S. (2008). Selecting good expansion terms for pseudo-relevance feedback. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 243–250. Singapore.

Chen, S.F., and Goodman, J. (1998). *An Empirical Study of Smoothing Techniques for Language Modeling*. Technical Report TR-10-98. Aiken Computer Laboratory, Harvard University.

Clarke, C.L.A., Cormack, G.V., Lynam, T.R., and Terra, E.L. (2006). Question answering by passage selection. In Strzalkowski, T., and Harabagiu, S., editors, *Advances in Open Domain Question Answering*. Berlin, Germany: Springer.

Croft, W.B., and Lafferty, J., editors (2003). *Language Modeling for Information Retrieval*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Greiff, W. R., Croft, W. B., and Turtle, H. (1999). PIC matrices: A computationally tractable class of probabilistic query operators. *ACM Transactions on Information Systems*, 17(4):367–405.

Hiemstra, D. (2001). *Using language models for information retrieval.* Ph.D. thesis, University of Twente, The Netherlands.

Jelinek, F., and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice.* Amsterdam, The Netherlands.

Lafferty, J., and Zhai, C. (2001). Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 111–119. New Orleans, Louisiana.

Lavrenko, V., and Croft, W. B. (2001). Relevance based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 120–127. New Orleans, Louisiana.

Lv, Y., and Zhai, C. (2009). Positional language models for information retrieval. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 299–306. Boston, Massachusetts.

Manning, C. D., and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* Cambridge, Massachusetts: MIT Press.

Metzler, D., and Croft, W. B. (2004). Combining the language model and inference network approaches to retrieval. *Information Processing & Management*, 40(5):735–750.

Miller, D. R. H., Leek, T., and Schwartz, R. M. (1999). A hidden Markov model information retrieval system. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 214–221. Berkeley, California.

Plachouras, V., Ounis, I., Amati, G., and Rijsbergen, C. V. (2002). University of Glasgow at the Web Track of TREC 2002. In *Proceedings of the 11th Text REtrieval Conference.* Gaithersburg, Maryland.

Ponte, J. M., and Croft, W. B. (1998). A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 275–281. Melbourne, Australia.

Song, F., and Croft, W. B. (1999). A general language model for information retrieval. In *Proceedings of the 8th International Conference on Information and Knowledge Management*, pages 316–321. Kansas City, Missouri.

Tao, T., and Zhai, C. (2007). An exploration of proximity measures in information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 295–302. Amsterdam, The Netherlands.

Tellex, S., Katz, B., Lin, J., Fernandes, A., and Marton, G. (2003). Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval.* Toronto, Canada.

Turtle, H., and Croft, W. B. (1991). Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, 9(3):187–222.

Zhai, C. (2008a). *Statistical Language Models for Information Retrieval.* Synthesis Lectures on Human Language Technologies: Morgan & Claypool.

Zhai, C. (2008b). Statistical language models for information retrieval: A critical review. *Foundations and Trends in Information Retrieval*, 2.

Zhai, C., and Lafferty, J. (2001). A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 334–342. New Orleans, Louisiana.

Zhai, C., and Lafferty, J. (2004). A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems*, 22(2):179–214.

Zhao, J., and Yun, Y. (2009). A proximity language model for information retrieval. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 291–298. Boston, Massachusetts.

第 10 章 |

Information Retrieval: Implementing and Evaluating Search Engines

分类和过滤

本章中，我们考虑的信息需求对于大部分用户来说是长期的、连续的或普遍的。表 10-1 给出了维基百科 60 篇文章的一些节选。你能认出每篇节选的语言吗？图 10-1 给出了一个收件箱中的 18 封邮件，你能认出其中的 10 封垃圾邮件吗？

表 10-1 维基百科中 60 个特定语言版本的文本片段。每个片段都是某篇文章的前 50 字节，这些文章都是通过“*random article*”链接得到的

18ος αιώνας | 19ος αιώνας |	8052 Novalis is a Main belt asteroid found on Sep
A távolsági jelzőmozzanatok a látás során t	Artikulu hau ez dator formatu hitzarmenekin bat.
Auzainvilliers é uma comuna francesa na região	Brgudac je selo na sjeveru Istre , u općini Lani
Burung Pacat ekor Biru adalah salah satu daripada	Byrteåi er ei elv i Tokke kommune i Telemark . H
Básendar (Bátsandar) voru fyrrum kaupstaður og	Cet article est une ébauche concernant une
Ciklobutan je bezbojni gas koji pripada grupi cik	Coordinates: 53°23′40″N 14°23′30″E
Danang (Da Nang , Đà Nẵng , fransk navn Tour	De Audi S4 is een sportieve versie van de Audi A4
Den Henri Grethen (* 16. Juli 1950 zu Esch-Uelzec	Flavius Constantius (mati 2 September 421) atau
George Thomas Moore (* 23 de febrero 1871 ,	Gijang-gun là một hạt (gun) trực thuộc
Gold Medal är den högsta utmärkelsen från Roy	Guido (-onis) [1] est nomen masculinum originis G
Her Majesty je píseň britské hudební skupiny	Hirdskjald hos Harald Hårfager , kendt fra digte
Jelizaveta Petrovna. Portree autor Charles-Amédé	Komz a raer eus filistenerezh evit ober anv en un
Koordinatės : 42°04′N 19°30′E	La strashimirita és un mineral de la classe dels
Maria Theresia Opladen (* 6. April 1948 in Engels	Ministerul Integrării Europene a fost un organ d
NGC 782 je galaxia v súhvezdí Eridanus , ktorú	Ofersæwisc rind is plante and pyrt, þe hātte
Pedr III (21 Chwefror , 1728 – 17 Gorffennaf ,	Piec martenowski – dokładnie piec Siemensa-Mar
Püstəqasım , Azərbaycan ın Quba rayonunda bi	Questa voce di asteroidi è solo
Rënia heroike e dëshmorëve Isa Kastrati, Sokol	Sun Wukong (geleneksel Çince : 孫悟空 , basi
Suster-vaihe eli Susterian oli luultavasti ylämi	USS Gato je ime več plovil Vojne mornarice ZDA :
Èdè ni ilànà kan pàtàkì tó jé gbòóg	Ĥarkiva Nacia Universitato nome de Vasil Karazin
Більче-Золотецький ландша	Война между родами Токугав
За филма от 1994 вижте Гарван	Курија е село во Општина Не
Уколико сте тражили расу п	מורוני הינה העיר הגדולה בי
مبارک آباد ، روستایی است از	هذه المقالة وسمت عن طريق بو
अनुबंधालु छगु तेल	कृपया या लेखाचा/वि
यह शब्द हिंदी में क	कानूडोस मिऊनिसिपि
जयगाँ (इंग्रेजि :Jaygaon)	१९६४ (1964) २०भी मदी दा
ஓமான் சுல்த்தானகம	หน้าปี๋เป็น ชื่อบ
პატარა გუფთა – სოფ	リューディガー・ゲルシュト (Rüdiger
袁武 , GBS , JP (1941年 11月 —), 又	아일랜드 국민당 (Irish Nationalist Party)

这两个例子中的基本信息需求——识别语言或识别垃圾邮件——都是容易理解并在很多环境中都需要用到的。搜索引擎可能需要知道查询所使用的语言，以便提供一个相关反馈；或检索的人也许希望可以明确指定检索出来的文档所用的语言（可能与查询所使用的语言不同）。一个垃圾邮件过滤器也必须识别出垃圾邮件才能阻止它进一步流传。一个客户支持的“帮助桌面”也需要路由收到的邮件，像人一样阅读和回复它。

Subject		Recipient
Washington Mutual Urgent Credit/Debit card update	service@wamu.com	enclaug@enron.com
Urgent Humanitarian Relief required for Tsunami Indonesia	SEHAT CHARITIES	sticas@enron.com
worthy Christian	Khushab	joydish@hotmail.alhamd.net
FW: Bin Laden Shoot em up	Dunnehil, Ginger	"Shapiro, Richard"
REVISION!	Fremquez, Rick	"Adams, Jacqueline P.", "Bump...
Start Date: 10/25/01; HourAhead hour: 3; <CODESITE>	Schedule Crawler	pete.davis@enron.com
Gift shipment ready KAM.KEISER@enron.com need address	DVDPlayer	KAM.KEISER@enron.com
FRV Notification: (Enron American Position Report - 11/21/2001)	Wallace, Carol	"Abel, Chris", "Allison, John", "...
Rate and Currency Counterparty Exposure as of 12/12/2001	Carrington, Clara	"Moran, Tom", "Shackleton, Sa...
The Internets #1 IRS tax solution.	[email2]	3nyjd9@msn.com	nqeb5e6@msn.com
The Internets #1 IRS tax solution.	[email2]	3nyjd9@msn.com	nqeb5e6@msn.com
The TechNews - Bulletin, April 2005	The TechNews	alcan@enron.com
Start Date: 1/31/02; HourAhead hour: 13;	Schedule Crawler	pete.davis@enron.com
Pharmacy for you	Faith Hightower	khokt
Re:	Quigley, Dutch	"Scott, Susan M."
Windows XP Pro $48.95, Office 2003 $69.95 Win XP	Lynda Hyatt	nmotley@enron.com
3 Days left to Respond	Anthony	khokt@enron.com
合化超值版12000地球物化教程仅售四位数	soulyy	kevin.hoke@enron.com

图 10-1　18 封邮件中有 10 封垃圾邮件，你可以把它们都找出来么

第一个和最后一个例子考虑面向搜索的信息需求情况，如 TREC 过滤专题中的主题 383 一样（如图 10-2 所示），这是一个关于精神病的治疗方法主题。如前面章节所述，搜索引擎可用于在语料库中找到历史文档，但是用户，也许是一个保健专家，想随时知道与这个查询相关的任何最新文章以继续了解相关的进展。这两个基本的信息需求是一样的，但在第一种情况中是短暂的，在第二种情况中是长期的。

```
<top>
<num> Number: 383
<title> mental illness drugs
<desc> Description:
Identify drugs used in the treatment of mental illness.
<narr> Narrative:
A relevant document will include the name of a specific
or generic type of drug. Generalities are not relevant.
</top>
```

图 10-2　TREC 7 特定任务与 TREC 8 过滤专题中的主题 383

分类（categorization）是为了满足某种信息需求对文档进行标记的过程；在我们的例子中文档被标记上一个标签，指示文档的语言、是垃圾信息还是有用信息（即不是垃圾信息）、相关或不相关。过滤（filtering）是一个根据某些长期信息需求而持续进行的对文档评价的过程。一般来说，过滤的结果是根据信息需求将文档放入 0 个或多个目的地。例如，垃圾邮件过滤器把垃圾邮件直接删除或放到垃圾箱里，同时将正常邮件放到收件箱里。一个新闻过滤器会把有关精神健康的文章发送给保健专家。路由（routing）本质上与过滤一样，根据文档所属的类别，把它们送到不同的地方。路由也称为信息选择性传播（Selective Dissemination of Information，SDI）。

分类和过滤的问题与检索的问题很类似。在某种意义上来说，它们只是同一个问题的两种不同描述：分类和过滤为给定文档找到所属分类；也就是，文档符合的长期信息需求。另一方面，检索是找到满足给定信息需求的所有文档；也就是，文档所属的某一类别。（与信

息需求——译者注）相关或不相关。

分类和过滤在很多方面还是不同于检索的。分类一般来说（不总是）从信息需求的描述或查询方面都与检索相差甚远。很难想象前面章节中提到的检索方法可以对诸如“找出用荷兰语写的文档”或“找出垃圾信息”这样的信息需求描述给出有意义的响应。即使给出一个可处理的信息需求，如保健专家的需求，信息需求的长期性也使得需要花费大量工作与系统沟通这个需求。一种方法是由专家人工制订一些规则：词汇表，语法和指示各种语言、垃圾信息或心理健康治疗的特征。这些规则实际上就是复杂的布尔查询。只要付出足够的努力，还是可以定义出规则来区分语言、垃圾还是非垃圾邮件、是治疗抑郁症的文章还是其他文章的。但是，正如搜索一样，自动化分类和过滤还是会更高效和更有效。我们将考虑的自动化方法就是**机器学习**（machine learning），也称为**分类器**（classifier）。分类器从样本中学习如何区分属于不同类别的文档。

从某种意义上来说，任何检索方法都是一个分类器，因为它的目标都是区分相关文档与不相关文档。同样，搜索引擎会返回一个文档集或一个有序列表，分类器也分为**硬**（hard）和**软**（soft）两种。**硬分类器**（hard classifier）会返回一个布尔结果，表明文档是否属于某个类别，而**软分类器**（soft classifier）会返回一个置信度。软分类通常也称为**排名**（ranking），而软分类器的学习称为**学习排名**（learning to rank）。

尽管第 8 章介绍的 BM25 排名方法并不是传统意义上的机器学习，但是它满足软分类器的定义，也可用于本章提出的过滤和分类问题，效果还不错。10.1 节中给出了之前所述的样例，并展示了分类和排名在解决这个问题时的作用。我们用 BM25 来解释这个作用，因为大家对它比较熟悉了，并且它也比较适合作为一个基准来与本章及其后面提出的方法进行比较。为了比较的目的，我们还提出了评估指标。

在 10.2 节中将更正式地对待分类，并在 10.3～10.7 节中介绍大量的对分类和过滤都很有用的学习方法。在 10.8 节，我们把采用这些方法的结果与 10.1 节中 BM25 的结果作了比较。

10.1 详细示例

通过一系列的示例，我们说明了典型过滤和分类问题、解决方案和评价方法。上述这些问题的变形包括：面向主题的过滤、语言分类以及垃圾信息过滤。本节将用第 8 章介绍的 BM25 排名和相关反馈方法解决这些问题。尽管 BM25 并不是专门用于过滤和分类的，但是它为我们理解这个领域提供了一个方便的切入点，并提供了一个比较的基准。在研究这些问题及其解决方案时会制定相应的评价指标。我们首先以一个类似于为 BM25 设计的排名检索任务的例子作为开始，然后说明几个变体的任务、方法和评价指标的主要区别。

10.1.1 面向主题的批过滤

考虑保健专家的信息需求。这个主题格式是非常适合作为搜索引擎的输入的，但过滤任务刚开始时，要被搜索的文档根本还不存在！因此只能等待文档的到来。如果我们能等待大批文档到来，我们可将这些文档聚合为一个语料库，建立索引，然后运用一个类似 BM25 的搜索方法去得到一个最有可能相关的文档排名列表，然后返回给用户。之后我们可以等待更多的文档，然后重复以上过程来轮流处理每一批文档。这种方法称为**批过滤**（batch filtering）。从用户的角度看，如果在一个合适的时间间隔内有足够多的文档到达，则批过滤是可行的解决方案。保健专家也会对每星期或每个月都能看到新的文章而感到满意。确实，这是文档分发的首选模式。在其他应用中，如邮件过滤，批处理带来的延迟可能就不可接受了。

为了说明批过滤方法，我们假设主题 383 反映了用户需求，并用 BM25 按时间顺序处理 TREC45 语料库。特别的：

- 我们用标题作为查询：〈“mental”，“illness”，“drugs”〉。
- 根据多语言搜索和垃圾信息过滤的经验，我们用字节 4-gram 表示词条（参见 3.3 节）。主题 383 中由于采用与不采用词干得到的结果相似，所以均不列出这些结果。
- 使用 BM25 的参数集合 $k_1=1.1$，$b=0$，这样可以有效地忽略文档长度所产生的影响。本章后续实验中均使用这个参数集。
- 本实验不使用相关反馈或仿相关反馈（8.6 节）。
- 我们首先使用 1993 年的**金融时报**（Financial Times，FT）作为文档集，然后使用 TREC45 文档集。这些文档都是按时间顺序排序的，跨度为 2 年，共有 $N=140\ 651$ 个文档，其中的 67 个与主题 383 相关。把这个文档序列用作一个评价文档集，用于评价 TREC8 过滤专题中的**路由与批过滤**（routing and batch filtering）任务的结果。
- 为了模拟文件批大小不同的批过滤过程，我们将所有文档分成多个大小相同的文档集，集合大小从 141 个（1000 批文档，每批包含一天的文档信息）连续变化到 140 651个（1 批文档，包含了两年的文档信息）。每一批都视为是一个单独的文档集进行索引。按时间顺序把每批文档送入到 BM25 过滤器中，每次一批。

评价过滤器的效果与评价排名检索的效果有些不同。我们引入一些必要概念来评价我们的例子，并把它们与更一般化的评价框架联系起来。

对于包含了 140 651 个文档的单个文档批，批过滤等价于排名检索。我们可以使用 2.3 节中介绍的任何指标，以及 12 章中介绍的扩展指标。在单个文档批上采用 BM25 产生结果为 P@10=0.1，P@1000=0.04，AP=0.053。

如果需要考虑多个文档批，一个直接但并不令人满意的方法是分别评价每批文档，然后取一个平均结果作为综合指标。这个方法不好之处在于：

- 当我们把文档分为越多批，每批包含的文档数量（n）和每批相关文档的数量（$|Rel|$）就会越少。当批的数量足够大时，很多批文档中就不包含相关文档了（即 $|Rel|=0$）。只要任何一个文档批的 $|Rel|=0$，那平均查准率（AP）就没有意义了。一般来说，任何依赖于 $|Rel|$ 的指标都不适用于小文档批。
- 如 P@k（precision at k）这样的指标受 n 的影响很大。因此，大小不同的文档批的结果就不能相互比较。

表 10-2 说明了这些问题。当文档批的大小从 $n=141$（1000 批文档）变化到 $n=28\ 140$（5 批文档）时，表中列出了前两批文档的 AP 和 P@10，以及所有$\lceil N/n \rceil$个分批上的平均值。这里，$n=1$ 表示只有一批文档。

表 10-2　面向主题的批过滤结果。短线表示结果没定义

文档批数量	n	第一批			第二批			所有批次平均		
		$\|Rel\|$	AP	P@10	$\|Rel\|$	AP	P@10	$\|Rel\|$	AP	P@10
1000	141	1	1.00	0.1	0	—	0.0	0.07	—	0.01
500	281	1	0.50	0.1	0	—	0.0	0.13	—	0.01
100	1407	3	0.33	0.2	0	—	0.0	0.67	—	0.04
50	2814	3	0.15	0.1	1	0.33	0.1	1.34	—	0.05
10	14 070	6	0.06	0.0	8	0.17	0.1	6.70	0.10	0.07
5	28 140	14	0.08	0.1	14	0.09	0.1	13.00	0.07	0.10
1	140 651	67	0.05	0.1				67.00	0.05	0.10

大多数文档批里 AP 没有定义是因为很多批文档中都不包含相关文档。P@10 和 AP（已经定义）均随文档批的大小变化而变化，并且利用这些指标评价这些方法只能从中得到很少的有用信息，达不到预期目的。

当使用大小不同的文档批时，P@k 不是一个合适的指标，因为对于任意给定的 k，检索出来的文档的数量与文档批的数量成正比。考虑两种极端情况：使用 1000 批文档和使用 1 批文档。如果使用 1000 批文档，那么 P@10 就是从每批文档中抽出 10 个，总共向用户展示 10 000 个文档时的查准率。如果只有 1 批文档，P@10 就是展示给用户 10 个文档时的查准率。一种更合理的做法是保持从所有文档批中取出并呈现给用户的文档总数是一个常数。这可以通过计算 P@$\lfloor \rho n \rfloor$实现，其中$\frac{1}{n} \leqslant \rho \ll 1$；也就是，假设呈现给用户的都是每批文档中排名靠前的（是一个得分）文档，因此，在整个语料库的文档集中也就返回了同样文档。为了实现目的，选用$\rho = \frac{k}{N}$，其中，N 是语料库的大小，k 是呈现给用户的文档总数。因此，无论文档批的大小 n 是多少，所有文档批上的平均 P@$\lfloor \frac{kn}{N} \rfloor$就可以与单批文档的 P@$k$ 进行比较了。表 10-3 列出了 P@$\lfloor \rho n \rfloor$，其中 k 的变化范围为 10～2000。每一列的结果都基于同样数量的返回文档。由这张表可以看出，在这个例子中，只要文档批的大小足够大，使得 $\lfloor \rho n \rfloor > 0$，那么平均值大部分都不会受文档批大小的影响。

表 10-3　指定大小的面向主题的批过滤的 P@k

文档批数量	n	P@$\lfloor \rho n \rfloor$					
		$\rho = \frac{10}{N}$	$\rho = \frac{20}{N}$	$\rho = \frac{100}{N}$	$\rho = \frac{200}{N}$	$\rho = \frac{1000}{N}$	$\rho = \frac{2000}{N}$
1000	141	—	—	—	—	0.03	0.02
500	281	—	—	—	—	0.04	0.01
100	1407	—	—	0.05	0.04	0.04	0.02
50	2814	—	—	0.07	0.05	0.04	0.01
10	14070	0.1	0.10	0.07	0.06	0.04	0.03
5	28140	0.2	0.10	0.06	0.06	0.04	0.01
1	140651	0.1	0.15	0.07	0.06	0.04	0.03

P@$\lfloor \rho n \rfloor$假设从每批文档中都取出相同比例的文档呈现给用户。这个假设是不太现实的，因为有些文档批可能比其他的文档批包含更多的相关文档。但这个情况完全是很偶然，或者因为文档是按时间排序的，所以很自然地会受到时事的影响。过滤系统调整这个比例变量可更好地迎合用户的信息需求。假设用户准备以平均比率 ρ 去检查文档集，也就是，逐个检查由过滤器处理的 N 个文档中的 ρN 个文档。如果从每批中恰好返回 ρN 个文档给用户，那么查准率不会得到改进。只有过滤器从那些它认为包含了更多的相关文档的文档批中返回更多的文档给用户时，查准率才有可能提高。

因为 BM25 算法根据相关性得分 s 排名文档，所以我们也可以不使用每批文档中的前 $\lfloor \rho n \rfloor$个文档，而是选择对于固定的阈值 t，文档相关性得分 $s > t$ 的那些文档。总的来说，t 越大返回的文档越少，但查准率也会越高，反之亦然。因此，对于任意 k 值，都存在一个 t 值，使得有 $k = \rho N$ 个文档的相关性得分都满足 $s > t$。姑且我们假设对于任意的 k 都有可能事先确定一个合适的 t 值。基于这个假设，我们用术语**累加** P@k（aggregate P@k）来描述系统在选用阈值 t 时，总共返回 k 个文档（见表 10-4）的有效性。**累加** AP（aggregate AP），就是按正常方法累加得到的 P@k 衍生出来的。正如我们所期望的那样，每一列的结

果几乎都是相同的。它们之间的差别是由于各批文档间的 IDF 值的微小差别而造成的。

表 10-4　面向主题批过滤的平均结果

文档批数量	n	P@10	P@20	P@100	P@200	P@1000	P@2000	AP
1000	141	0.2	0.10	0.07	0.08	0.04	0.02	0.058
500	281	0.1	0.15	0.08	0.08	0.04	0.03	0.056
100	1407	0.1	0.15	0.06	0.07	0.04	0.03	0.053
50	2814	0.1	0.15	0.07	0.06	0.04	0.03	0.053
10	14070	0.1	0.15	0.07	0.06	0.04	0.03	0.053
5	28140	0.1	0.15	0.07	0.06	0.04	0.03	0.053
1	140651	0.1	0.15	0.07	0.06	0.04	0.03	0.053

累加 P@k 和累加 AP 适用于建模现实场景，过滤器通过设置上述 t 值，可满足特定用户的需求。它们也可为另外一种现实场景建模，每批的结果都先后进入到一个优先级队列中，用户可以随意读取。一旦所有的文档批都处理完后，累加 P@k 就是优先队列中前 k 个文档的查准率，同时累加 AP 也就衡量了最后排名结果的有效性。

10.1.2　在线过滤

在线过滤类似于批大小为 $n=1$ 的批过滤。如图 10-3 所示，在线过滤器对队列中的每个信息都立刻处理，而不是对一批信息进行响应。被视为是相关的信息都分发给用户；而被认为是不相关的信息就全部被丢弃掉。分发中介可能是一个文本信息系统，或是一个电子邮件系统，或者是一个便于用户查询的结果存档。这个中介的特点是实时、具备存储和访问的能力。

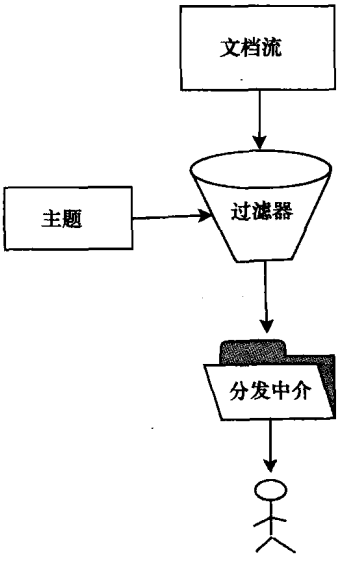


图 10-3　面向主题的在线过滤

正如批过滤一样，在线过滤需要在查准率和查全率间找到一个平衡点。一般来说，可通过返回更多的信息给用户，以提高查全率而牺牲查准率。当今最快的即时通信媒体（如手机短信）都具有大容量存储以及复杂访问的能力，因此造成限制的是用户处理这些信息的能力，而不是媒体。如果可能，一个好方法是按**优先级**（prioritize）将这些信息分发给中介，就像一个**优先级队列**（priority queue）一样。正如排名检索一样，用户检查一定数量高优先级的文档，从而默认地决定了上述平衡点。

如果不可能按优先级排名时，有必要确定一个合适的阈值 t 使得分发适当数量的文档。我们设定合适的 t，使得分发率恰好为 ρ（如前一节的定义），或优化某个可以在查准率和查全率之间达到平衡的函数（如 F_β）。

某些诸如 BM25 这样的排名检索方法中的相关性得分是一个合适的优先级指标，这些得分基于单个文档计算。在这里 BM25 不是很合适，因为它依赖于词频 N_t，而词项数目必须在一个大的文档集上计算。也就是说，我们可将 BM25 用于一个只包含一个待过滤文档的文档集中（近似于空），累加结果并不比批过滤的结果差多少，如表 10-5 所示。从下一节还可以看到，无论是批过滤还是在线过滤，通过使用历史样本，效果都可以大大提高。

表 10-5 在线过滤与面向主题的批过滤的累加结果的比较。第一行来自表 10-4

方法	P@10	P@20	P@100	P@200	P@1000	P@2000	AP
单批文档过滤	0.1	0.15	0.07	0.06	0.04	0.03	0.053
在线过滤	0.2	0.15	0.05	0.05	0.03	0.02	0.041

10.1.3 从历史样本中学习

一般来说，在部署好过滤器后，从同一个或相似的文档源中，总可以得到一些可用的历史样本文档。回顾之前部署在 1993 年及之后的金融时报文档上的过滤器。TREC45 文档集中也包含了 1992 年及之前的金融时报上的 69 508 个文档，可作为历史样本。

由前面对 TREC 的评价结果可知，这些历史样本中有 22 个是与主题 383 相关的。但这些信息在部署过滤器时并不需要知道。我们首先考虑不带相关信息的历史文档，然后再考虑如何利用这些信息（如果有）去大大地提高系统的有效性。

历史样本集的统计信息

对于批过滤，使用历史文档的最简单方法也许就是用它们来增加每批文档的大小。对于给定的某批文档，历史文档与其他文档混合在一起，并对整个文档集排名。然后，从排名列表中去除历史文档，就得到每批中的待过滤文档的得分和排名。采用这个方法最主要的效果就是能得到一些更好的文档统计信息（尤其对 IDF），因而会得到一个更精确的相关性得分。另外一个效果就是不再返回每批文档中固定的前 k 个文档，而是返回出现在混合排名列表中的前 $k'>k$ 个文档。如果某批文档中包含更多得分较高的文档，那返回给用户的文档也较多。对于合适的 k'，系统虽然返回的文档数一样，但查准率会更高。

通过使用历史文档可有效地把在线过滤的问题转化成批过滤的问题，使得只需要计算每个待过滤文档的得分，或许还包括它在历史文档集中的相关排名。为了这个目的，之前章节介绍的索引结构就完全用不上了。需要用来计算 BM25 和类似得分的只是一个词典，包含了每个词项出现的文档数 N_t，如果得分理想，还包括一个历史文档得分的有序列表。为了计算相关性得分，我们简单地取出相关的 N_t 并带入公式；为了计算排名，我们则在历史得分列表上使用一个二分查找。

表 10-6 的第三行列出了在线过滤器上使用历史样本集的统计信息后对累加效果的影响，由表可知，累加效果与采用批过滤器的效果几乎完全相同。

表 10-6 在面向主题在线过滤的累加结果中历史文档集统计数据的影响。表中的前两行来自表 10-5，第三行表示一个使用了历史统计数据来计算 IDF 值的过滤器的效果。最后一行显示的效果是：在部署过滤器之前，通过 BM25 的相关反馈从已标记训练样本中抽取的 20 个词项来扩展查询得到的结果

方法	P@10	P@20	P@100	P@200	P@1000	P@2000	AP
单批文档	0.1	0.15	0.07	0.06	0.04	0.03	0.053
在线过滤（不带历史统计数据）	0.2	0.15	0.05	0.05	0.03	0.02	0.041
在线过滤（带历史统计 IDF 数据）	0.1	0.15	0.07	0.06	0.04	0.03	0.054
在线过滤（带历史统计 IDF 数据＋历史训练样本）	1.0	0.95	0.39	0.24	0.06	0.03	0.555

历史训练样本

因为与过滤器相关的信息需求是持续更新的，所以相关文档的样例是已知的或者不难获取的。这些样例被称为**训练样本**（training example）或**标记数据**（labeled data）。在机器学

习中使用标记数据来建立分类器的方法被称为“监督学习”。本书中已经介绍了一种简单的监督学习方法：BM25 相关反馈法（参见 8.6 节）。

前面已经提到，在**金融时报**的历史样本集中有 22 个相关文档。我们使用 BM25 相关反馈法从这些相关文档中选出最合适的 $m=20$ 个词项，并用这些词项来扩展来自主题中的 3 个词项。利用这些词项，在每个文档上使用 BM25 计算各自的得分，正如在线过滤器一样。

表 10-6 的最后一行列出了上述监督学习方法的累加结果。第一列和最后一列的结果并不是印错了：对于这个特别任务，采用历史训练样本会使 P@10 和 AP 均得到 10 倍的提高。

10.1.4　语言分类

从表 10-7 的 60 个语言版本的维基百科中，我们通过“random article”链接抓取了 8012 篇文章，然后从每篇文章正文中抽取出前 50 字节的内容（见表 10-1）。现在考虑分类与过滤的对偶问题：

- **分类**（categorization）：给定一个文档 d 和一个可能的类别集，确定文档 d 所属的类别（或类别集）。分类可简化为**类别排名**（category ranking），即根据各类别包含该文档的可能性对各个类别进行排名。
- **过滤**（filtering）：给定一个类别 c，确定属于 c 的文档。过滤可简化为**文档排名**（document ranking），即根据各文档属于该类别的可能性对各个文档进行排名。

表 10-7　维基百科 60 个语言版本的标签

标签	语言	标签	语言	标签	语言
ang	盎格鲁撒克逊人的语言	fi	芬兰语	nn	新挪威语
ar	阿拉伯语	fr	法语	no	挪威语
az	阿塞拜疆语	he	希伯来语	pa	旁遮普语
bg	保加利亚语	hi	印地语	pl	波兰语
bn	孟加拉语	hr	克罗地亚语	pt	葡萄牙语
bpy	比什奴普莱利亚语	hu	匈牙利语	ro	罗马尼亚语
br	布列塔尼语	id	印度利西亚语	ru	俄语
bs	波斯利亚语	is	冰岛语	simple	初级英语
ca	加泰罗尼亚语	it	意大利语	sk	斯洛伐克语
cs	捷克语	ja	日语	sl	斯洛文尼亚语
cy	威尔士语	ka	格鲁吉亚语	sq	阿尔巴尼亚语
da	丹麦语	ko	韩语	sr	塞尔维亚语
de	德语	la	拉丁语	sv	瑞典语
el	希腊语	lb	卢森堡语	ta	泰米尔语
en	英语	lt	立陶宛语	th	泰语
eo	世界语	mk	马其顿语	tr	土耳其语
es	西班牙语	mr	马拉塔语	uk	乌克兰语
et	爱沙尼亚语	ms	马来语	vi	越南语
eu	巴斯克语	new	尼泊尔语言	yo	约鲁巴语
fa	波斯语	nl	荷兰语	zh	中文

在我们的例子中，文档就是摘自维基百科的文章片段，类别就是各种语言。根据定义，我们例子中的文档只能属于一个类别，由它来源的维基百科的语言版本而定。在其他一些场合中，考虑一个文档属于若干个类别也是有意义的，例如，一种文档可能包含若干种语言或一种语言有多种方言。如果处理的是整篇文章，那么判定是哪种语言就比较直观了（Mc-

Namee，2005），但在我们的例子中，我们有意将范围限制在短的文章片段上。

当我们需要路由信息或将文档组织为特定语言的文档集时，会遇到语言分类问题。这些问题都不能现成地用一个可由搜索引擎处理的查询来表示。相反，我们完全依赖训练样本来获得信息需求。现在，我们任取 4000 个文章片段作为历史样本，把剩下的 4012 个文章片段作为分类或过滤的对象。在机器学习阶段，历史样本构成了一个**训练集**（training set），剩下的则是**测试集**（test set）。最终目标是分类或者过滤一个远远大于测试集的样本集，测试集可视为是一个**例子**（sample），用于在更大的数据集上评估分类器的有效性。

可以使用带历史样本的面向主题过滤的方法实现过滤和文档排名。我们把每种语言看做一个独立的主题，然后运用 BM25 相关反馈法（参见 8.6 节）。在这个例子中，把在一个主题语言中的每个训练片段都视为一个“相关文档”。对于某种给定的语言，我们从训练片段中构造出一个查询，将每字节 4-gram 作为一个查询词项。

表 10-8 给出了应用 BM25 到每个语言上作为 60 个过滤问题得到的 AP 结果和整体 MAP。用普通的搜索标准来看，MAP＝0.78 已经是非常高的了，但仍然不能反映出将 BM25 用于分类问题的（实际）效果。

表 10-8 采用 BM25 且词项选自 4000 个训练样本的 60 种语言过滤问题的 AP 结果

标签	AP	标签	AP	标签	AP
ang	0.91	ar	0.99	az	0.53
bg	0.67	bn	0.84	bpy	0.81
br	0.94	bs	0.48	ca	0.70
cs	0.67	cy	0.95	da	0.40
de	0.76	el	1.00	en	0.59
eo	0.72	es	0.67	et	0.45
eu	0.82	fa	0.95	fi	0.85
fr	0.79	he	1.00	hi	0.72
hr	0.47	hu	0.75	id	0.67
is	0.93	it	0.79	ja	0.94
ka	0.99	ko	1.00	la	0.79
lb	0.97	lt	0.82	mk	0.81
mr	0.86	ms	0.68	new	0.91
nl	0.84	nn	0.63	no	0.45
pa	0.96	pl	0.83	pt	0.83
ro	0.92	ru	0.75	simple	0.54
sk	0.75	sl	0.59	sq	0.60
sr	0.68	sv	0.76	ta	0.97
th	1.00	tr	0.82	uk	0.84
vi	0.96	yo	0.63	zh	0.87
			MAP：0.78		

为了解决分类问题，我们不应对文档排名，而必须对语言排名，同时为每个文档计算这个排名。可以使用文档排名中的相关性得分。给定文档 d 和语言 l，设 $s(d, l)$为 d 在 l 语言中的文档排名中的相关性得分。现在考虑两种语言 l_1 和 l_2。假设 $s(d, l_1)>s(d, l_2)$表示文档 d 是用语言 l_1 书写的可能性比用语言 l_2 书写的可能性大。在这个假设下，$s(d, l)$ 也是语言 l 关于文档 d 在分类排名中的相关性得分。

一次显示所有的 4012 个分类是不可行的。表 10-9 展示了一个例子：对于德国维基百科摘录的一个文本片段的所有语言排名。我们看到正确分类（l_1＝de）排在首位，其他与德语

类似的分类——日耳曼语言——排在那些与德语不相似的分类之前，例如亚洲语言。严格的分类问题定义就只会有一个相关结果：正确的分类。其余的均不相关。

表 10-9　表的最上面给出了德文片段的语言排名（正确标签：de）。$s(d, l)$指给定语言的 BM25 得分

Werner Haase (* 2. August 1900 in Köthen (Anhalt

标签	$s(d, l)$	标签	$s(d, l)$	标签	$s(d, l)$
de	58.2	lb	35.2	da	33.6
no	29.1	en	28.4	tr	19.4
sk	18.5	hu	17.5	bs	16.4
la	14.7	cs	14.5	et	13.2
fi	13.0	es	12.7	cy	12.0
is	11.8	sl	10.6	simple	10.4
nl	9.3	pl	7.3	yo	7.1
sv	6.0	sq	5.9	ru	5.0
ro	4.6	ang	4.5	lt	3.9
el	3.4	eo	3.3	az	2.4
sr	2.4	nn	2.4	ms	1.7
fr	1.4	zh	0.0	vi	0.0
uk	0.0	th	0.0	ta	0.0
pt	0.0	pa	0.0	new	0.0
mr	0.0	mk	0.0	ko	0.0
ka	0.0	ja	0.0	it	0.0
id	0.0	hr	0.0	hi	0.0
he	0.0	fa	0.0	eu	0.0
ca	0.0	br	0.0	bpy	0.0
bn	0.0	bg	0.0	ar	0.0

AP (RR): 1.00

表 10-10 列出了 4012 种语言排名的整体综合指标。在该表中，P@1 一般被称为**准确率**（accuracy）：正确分类的文档（排第一位）的比例。MAP 等价于**平均倒排名值**（mean reciprocal rank，MRR），每个排名列表中只有一个是相关的（参见 12.1.5 节）。对于单个主题，倒排名值定义为 $RR=\frac{1}{r}$，其中 r 指第一个（也是唯一一个）相关结果的排名。因为 $AP=P@r=\frac{1}{r}$，其中 RR=AP，MRR=MAP。

表中结果可用**微平均**（micro-average）和**宏平均**（macro-average）来表示，定义如下：

- 微平均：所有文档上的综合指标，不分类别。
- 宏平均：分别为每个类别计算的平均综合指标。

表 10-10　4012 个文本片段上关于语言排名的综合度量。P@1 等价于准确率：被正确分类的文本片段的比例。MAP 等价于 MRR，因为问题的定义为：对于每个文本片段，只有一种“相关”语言

	准确率 (P@1)	误检率 (1-准确率)	MRR (MAP)
微平均	0.79	0.21	0.860
宏平均	0.79	0.21	0.857

在这个例子中，微平均和宏平均的差别很小，因为测试集中每种语言的文档数量几乎都是一样的。当样本数不同或各类别的排名效果不同时，微平均和宏平均间才有可能出现较大

的差别。为了说明这个差异，我们把用于计算综合指标的文档集从原来的测试集扩展到整个维基百科。

表 10-11 列出了每个版本的维基百科上的部分文章及其上的 MRR。各个版本的大小相差甚远，最大的版本（英文）比其他版本包含的文章数要多得多，但其 MRR 只是一般水平（0.71）。这个测试集的最终结果，微平均 MRR 要小于宏平均 MRR（0.82：0.86）。在其他应用中，如垃圾信息过滤，这个差别就更重要了。总的来说，宏平均与每个类别的文档比例无关，而微平均则是该比例的加权。为了把微平均从一个文档集扩展到另外一个文档集，需要知道文档比例和与类别相关的综合得分。微平均与宏平均的差别对不同分类来说是特定的：当运用到搜索结果时，MAP 和其他综合指标就是宏平均了。

表 10-11 扩展到 60 个语言版本的维基百科内容中的语言分类结果。“%”列表示某个版本的维基百科上的文章的比重。某些版本（如英文）包含比其他版本多得多（如冰岛语）的文档

标签	%	MRR	标签	%	MRR	标签	%	MRR
ang	0.01	0.90	ar	0.81	0.99	az	0.19	0.91
bg	0.60	0.87	bn	0.16	0.89	bpy	0.20	0.84
br	0.21	0.89	bs	0.22	0.69	ca	1.45	0.80
cs	1.05	0.77	cy	0.19	0.95	da	0.90	0.65
de	7.52	0.88	el	0.35	1.00	en	23.95	0.71
eo	0.95	0.80	es	3.91	0.78	et	0.52	0.73
eu	0.32	0.87	fa	0.50	0.97	fi	1.69	0.88
fr	6.66	0.88	he	0.77	1.00	hi	0.25	0.93
hr	0.49	0.58	hu	1.04	0.87	id	0.86	0.68
is	0.21	0.94	it	4.71	0.81	ja	4.88	0.94
ka	0.25	0.99	ko	0.80	1.00	la	0.23	0.80
lb	0.22	0.99	lt	0.72	0.91	mk	0.24	0.88
mr	0.19	0.81	ms	0.32	0.83	new	0.42	0.85
nl	4.47	0.89	nn	0.40	0.71	no	1.81	0.57
pa	0.01	0.96	pl	5.03	0.89	pt	3.98	0.90
ro	1.04	0.93	ru	3.20	0.81	simple	0.49	0.79
sk	0.90	0.79	sl	0.63	0.78	sq	0.19	0.88
sr	0.62	0.70	sv	2.63	0.85	ta	0.15	0.97
th	0.38	1.00	tr	1.07	0.85	uk	1.21	0.88
vi	0.68	0.96	yo	0.05	0.93	zh	2.10	0.91

MRR：0.82/0.86（微平均/宏平均）

10.1.5 在线自适应垃圾邮件过滤系统

图 10-4 画出了我们在第三个和最后一个例子中提到的垃圾过滤问题的场景。过滤器收到邮件流，并将它们顺序分发到用户的收件箱或一个隔离的文件中。收件箱中的信息可供用户按时间顺序阅读，为了找到被过滤器错误分发的有用信息（非垃圾邮件），用户偶尔也搜索一下隔离文件。在以上两个操作中，当用户在收件箱中发现垃圾邮件或在隔离文件中找到有用信息时，便可向过滤器反馈。在这个应用中，把收件箱视为一个简单的队列，而把垃圾箱视为一个优先队列（即一个动态更新的有序队列）会比较有效。

我们使用 TREC 2005 公开垃圾邮件语料库[⊖]，共有 92 180 个信息，其中 39 399 个是垃圾信息，78 798 个是有用信息。我们不把这些信息分成训练集与测试集，而是采用一种**在线反馈法**（on-line feedback），假设有一个**理想的**（ideal）用户会检测并报告所有的错误。在这个假设前提下，每个信息都能在分发后作为一个历史训练样本：那些被报告有错的信息假设属于用户指定的类别，那些没有报错的信息假设已被过滤器正确分类。尽管这个完美用户的假设是不存在的，可是我们可以将其作为比较评价结果的基准。

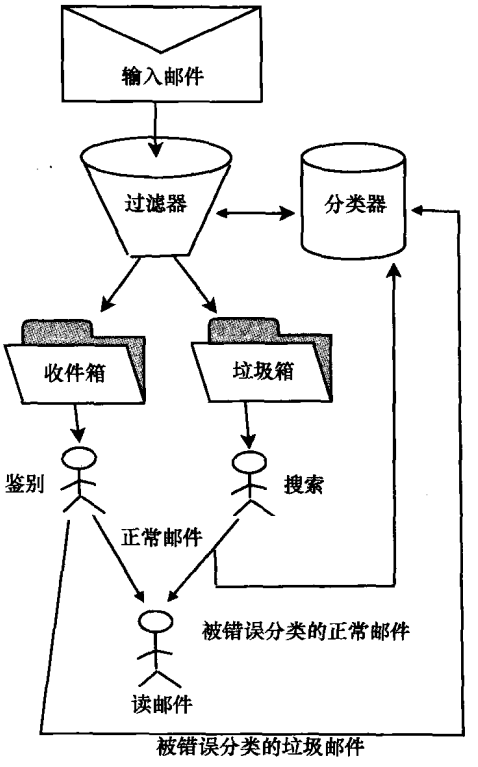


图 10-4　在线垃圾邮件过滤器的部署

在线自适应过滤可通过 TREC 垃圾过滤器评估工具包实现，测试工具、测试语料库和样本过滤器都可从网上[⊜]下载。尽管该工具包所使用的类别刚好也是命名为**垃圾信息**（spam）和**有用信息**（ham），但是它适用于任何**在线二元分类**（on-line binary categorization）任务。本章所有的例子都可使用该工具包。为了使用该工具包评估，过滤器必须先实现表 10-12 中详细列出的 5 条命令。

作为一个能说明问题的例子，我们再次使用 BM25，建立一个在线自适应垃圾信息过滤器。我们再次强调这并不是 BM25 的典型应用，但它为本章后面要讨论的技术提供了一个坚实的基础。垃圾信息过滤是一个**二元分类**（binary categorization）的例子，因为只有两个不同的类别：根据定义，每个信息只能属于其中一个类别。与前面的例子不同，垃圾信息过滤中并没有主题信息也没有历史样本。训练样本从在线反馈中获得。一开始过滤器没有任何信息可作为决策基础，但随着已处理信息越来越多，过滤器就学会了区分类别的信息特征。在训练过程中过滤器提高的速度和行为被称为**学习曲线**（learning curve）。

表 10-12　TREC 垃圾邮件过滤器评估工具包的命令。为了在这个工具包框架内评价一个过滤器，该过滤器必须顺序调用表中的 5 条命令

命令	过滤器操作
initialize	创建一个过滤器来处理一个新的信息流
classify *file*	为 *file* 中的信息返回一个类别以及优先级得分
train spam *file*	将 *file* 作为一个垃圾信息的历史样本
train ham *file*	将 *file* 作为一个有用信息的历史样本
finalize	关闭过滤器

⊖ trec.nist.gov/data

⊜ plg.uwaterloo.ca/~gvcormac/jig

这里使用的基准方法与前面语言分类所使用的方法是一样的。分别对每个文档使用带相关反馈的 BM25 计算相关性排名：根据每个文档的两个得分 $s(d, \text{spam})$和 $s(d, \text{ham})$来判定是垃圾信息还是有用信息。d 的类别排名由差值 $s(d)=s(d, \text{spam})-s(d, \text{ham})$决定。如果 $s(d)>0$，那就认为该信息是垃圾信息，否则是有用信息。

与以前做法一样，查询词项都是从已标记为垃圾信息或有用信息的信息中提取出来的字节 4-gram。就像过滤过程一样，我们为垃圾信息和有用信息分别构造和维护查询。当获得反馈后，在合适的查询中添加新的词项并更新 IDF 值。两个类别的查询初始化都为空，得分都为 0，但当越来越多的信息被处理后，它们的得分就会迅速增长。与前面的实验不同，我们不限制添加到每个查询中的反馈词项的数量，允许查询无限制地增长。

为了进行在线排名，得分 $s(d)$可作为一个优先级。表 10-13 中给出的文档排名问题的有效性指标，并没有为我们的预期目的（垃圾过滤）提供太多的信息。概括说来，是因为搜索的目的在于确定有用的文档，如 P@k 和 AP 等有效性指标反馈给系统的是最容易分类的那些文档。这些度量无法告诉我们过滤器对于那些难以分类的文档的效果如何，但这才是垃圾信息过滤器效果的关键所在。

表 10-13 垃圾信息过滤中文档排名问题的有效性指标。这三行分别给出了使用独立的垃圾信息相关性得分、有用信息相关性得分以及它们的差值的效果

相关性得分	P@10	P@20	P@100	P@200	P@1000	P@2000	AP
$s(d, \text{spam})$	1	1	1	1	1	1	0.991 8
$s(d, \text{ham})$	1	1	1	1	1	1	0.990 6
$s(d, \text{spam}) - s(d, \text{ham})$	1	1	1	1	1	1	0.992 7

一个在线垃圾过滤器中主要的分类问题的有效性指标如表 10-14 所示。因为只有两个类别，所以倒排名值也是多余的，不需要列出。为了计算过滤器的 F_1 值，假设所有垃圾信息组成了一个相关类。这是（垃圾过滤相关）文献中的基本假设。当你发现没有明确说明查全率和查准率是对应有用信息还是垃圾信息的，你都可以假设作者是把垃圾信息作为相关类的（即文献中报告的查全率和查准率都是对应垃圾信息而言的——译者注）。

表 10-14 在一个在线垃圾过滤器中主要的分类问题的有效性指标（面向信息检索）

类别	准确率（P@1）	误检率（%）	查准率	查全率	F_1
垃圾信息	0.999 89	0.011	0.803 08	0.999 89	0.890 74
有用信息	0.671 49	32.860	0.999 77	0.671 49	0.803 39
微平均	0.859 54	14.055	0.859 45	0.859 54	0.859 54
宏平均	0.835 69	16.431	—	—	0.847 07
Logit 平均	0.992 60	0.740	—	—	0.852 33

由表 10-14 可知，垃圾信息的准确率（P@1）很高，出现了大量接近完美的结果，令人难以理解。以百分比的形式展示误检率（1−P@1），看起来更方便与直观。评价过滤器的有效性时，很容易看出误检率为 0.1% 的过滤器比误检率为 1% 的过滤器性能要好 10 倍。当然准确率为 0.999 的过滤器要比准确率为 0.990 的过滤器好 10 倍，但还需要心算和计算大量正确的值。回到刚才那个例子中，垃圾邮件的误检率为 0.01% 意味着 10 000 个垃圾信息中有 1 个会被分发到用户的收件箱中。而有用信息的误检率为 32.9% 就意味着大约 1/3 的有用信息被放到垃圾箱中。用户很难接受这样的误操作；但如果情况相反（指 1/3 的垃圾邮件被放到收件箱——译者注），则用户可能还可以接受。如果有用信息的误检率是 0.01%，

而垃圾信息的误检率是 32.9%，那意味着过滤器会把大量的有用信息分类到用户的收件箱中，并过滤掉 2/3 的垃圾信息。

综合统计指标，无论是微平均还是宏平均，它们都无法反映出某个指定类别的误检率。并且，微平均对整体中垃圾信息的比例非常敏感。例如，假设垃圾信息的数量加倍，而有用信息的数量、过滤器有用信息和垃圾信息的误检率均不变。那微平均误检率会从 14% 降到 8.9%。过滤器的有效性提高了吗？当然没有。微平均把垃圾信息的**普遍度**（prevalence）与过滤器的属性视为同等重要。

logit 平均（logit average，LAM）是在 logit 变换（见公式（10-38））下的宏平均：

$$\mathrm{lam}(x,y)=\mathrm{logit}^{-1}\left(\frac{\mathrm{logit}(x)+\mathrm{logit}(y)}{2}\right) \tag{10-1}$$

从直觉上来说，这个均值将得分解释为概率，并根据它们贡献的权重把它们结合起来。在 logit 平均下，得分 0.1 与 0.01 的差与得分 0.9 与 0.99 的差所产生的效果是一样的。从某种意义上说，这些对值都代表了效果上一个数量级的差异。而 0.5 与 0.51 间的差异几乎不带来什么影响。Cormack（2008）基于**比值比**（odds ratio）给出了 LAM 的更一般的推导（参见 10.2.1 节）。

查准率和查全率，以及基于它们的其他综合指标，都很难用于解释过滤任务且受普遍度的严重影响。对于垃圾信息过滤，为了计算查准率和查全率，我们必须先任意假定垃圾信息或有用信息是“相关的”。一般会假设垃圾信息是“相关的”，表 10-14 列出了在这一假设下的查准率和查全率。注意到微平均查准率和查全率是多余的，它们均等价于准确率。很容易证明，无论类别是否是互斥的，这个等价关系都会成立——正如我们垃圾信息过滤和语言分类的例子一样。宏平均查准率和查全率分开是没有什么意义了，但合在一起可以计算如 F_1 这样的综合查全率/查准率指标的宏平均值，正如表 10-14 所示。在这里，我们很难对查准率、查全率或 F_1 给出一个有意义的解释，所以建议在考察过滤任务时，避免使用它们作为评价指标。

10.1.6　二元分类的阈值选择

当把某排名方法，如 BM25，用于二元分类时，阈值 t 的选择会直接影响到这两个类别的误检率。在垃圾信息过滤的例子中，我们把 $s(d)>0$ 的信息标记为垃圾信息，而剩下的都标记为有用信息。这里，我们用 t 代替 0，标记 $s(d)>t$ 的信息为垃圾信息，其他为有用信息。表 10-15 给出了不同的 t 值对应的误检率。

表 10-15　随阈值设置函数变化的垃圾信息和有用信息的误检率

阈值 t	垃圾信息误检率	有用信息误检率	宏平均误检率	平均误检率	宏平均误检率 F_1
2156	56.8%	0.0%	28.4%	1.1%	0.66
1708	36.2%	0.1%	18.2%	2.3%	0.79
1295	10.0%	0.4%	5.2%	2.0%	0.94
1045	2.2%	1.0%	1.6%	1.5%	0.98
931	1.0%	2.8%	1.9%	1.7%	0.98
713	0.1%	10.0%	5.1%	1.0%	0.95
−180	0.0%	34.9%	17.4%	0.7%	0.84

可以看出，阈值本身是没什么意义的，但是大的阈值产生较低的有用信息误检率，而牺牲了高垃圾信息误检率，反之亦然。对于某个阈值 t，**受试者工作特征曲线**（receiver oper-

ating characteristic curve，ROC）由所有的点（x，y）组成，其中的 x 表示垃圾邮件的误检率，$1-y$ 表示正常邮件的误检率（即 y 表示正常邮件的准确率）。ROC 曲线从几何角度描述了过滤器有效性，并且独立于阈值，同样地，查全率-查准率曲线描述了排名检索的有效性，并且也独立于任何截断参数。

在线性坐标下画出 ROC 曲线（参见图 10-5），并不能很直观地反映出过滤器的性能：曲线太贴近于 x 和 y 轴。因此图 10-6 就提供了一个 logit 变换后的曲线。如果调整 t 以达到某个有用信息的误检率，那么从曲线中很容易就可以确定对应的垃圾信息的误检率，反之亦然。也可以用这个曲线来比较过滤器。如果一条曲线在另一条曲线之上，表示这条曲线代表的过滤器比另一个要好。如果曲线相交，就要根据垃圾信息误检率和有用信息误检率的相对重要性来判断哪个更好。为了进行比较，我们在相同的数据集上对比了 BM25 得到的结果与当前流行的 SpamAssassin[⊖]（版本为 3.02，采用默认参数值）得到的过滤结果。可以看到 BM25 在大部分区间都比 SpamAssassin 要好，但在低有用信息误检率时曲线相交了，SpamAssassin 要稍微好一点。

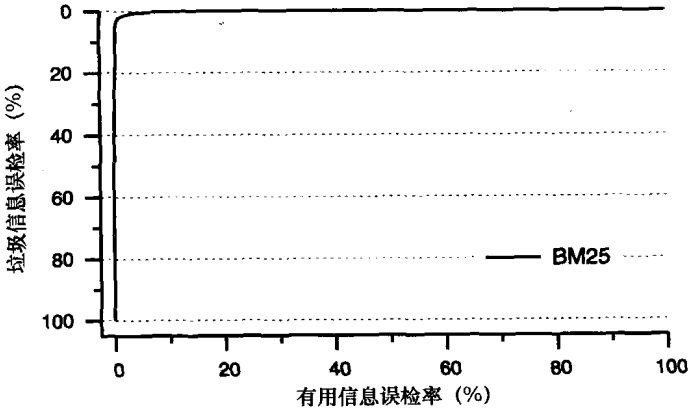


图 10-5 线性坐标下基于 BM25 的垃圾信息过滤器的 ROC 曲线

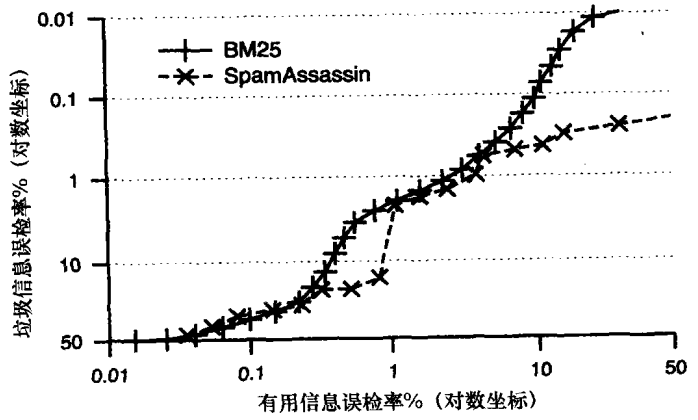


图 10-6 SpamAssassin 和基于 BM25 的垃圾邮件过滤器在对数坐标下的 ROC 曲线

⊖ spamassassin.apache.org

ROC 曲线下的面积（area under the ROC curve，AUC 或 ROCA）是 0～1 之间的一个数，可作为阈值独立的过滤器的性能指标。ROC 曲线越高，则其 AUC 值越大。除了几何意义外，AUC 还有概率意义。它是一个随机选取的垃圾信息比一个随机选取的有用信息的得分高的概率：

$$\mathrm{AUC} = \Pr[s(d_1) > s(d_2) | \ d_1 \in \text{spam},\ d_2 \in \text{ham}] \tag{10-2}$$

我们的 BM25 过滤器的 AUC=0.998 1。为了方便，把 ROC 曲线**上部**（above）的面积用百分比来表示：1－AUC=0.189 6%。对于 SpamAssassin，1－AUC=0.516 3%。

在评价前已经设定好阈值的情况下，ROC 曲线就只包含一个点（x，y），因此 AUC 无定义。在这种情况下，logistic 平均值 LAM 提供了一种综合指标，该指标对阈值的敏感程度要比其他指标小，如表 10-15 所示。微平均误检率、宏平均误检率和 F_1 受阈值的影响很大——所以很少用它们作为评价标准。

如果使用历史数据训练过滤器，一个简单合理的做法是尝试使用多个阈值 t 来获得垃圾信息误检率和有用信息误检率之间的理想平衡点。在一个带反馈的在线设置过程中，如果有用信息误检率过高，可通过小幅增加 t 的值来进行动态调整，反之也可减小 t。但是形式化“理想平衡点”是不容易的，在很多情况下，最好还是由用户指定阈值，无论是显式的还是隐式的。用户界面应可让用户输入阈值调整（例如，一个从“展示少量”到“展示更多”的滚动条，或一组标有“严格过滤”、“适度过滤”、“不过滤”的按钮）。或者可要求用户为每个类别设定一个可以接受的误检率，或者设定每单位时间处理的文档数，或者设定收到邮件的分发比例。这些参数都很容易估计，随着时间的推移，为了将它们维护在指定的水平，t 不断地自动调整。

10.2　分类

上面已经通过详细的例子，采用 BM25 去说明分类器和过滤器的特性，下面将使用其他方法去解决相同的问题。过滤器和分类器的核心是**分类**（classification）。最基本的分类问题是**二元分类**（binary classification），就是一个确定对象是否具有某种性质 P 的过程。**二元分类器**（binary classifier）是二元分类的自动化过程。在我们提出二元分类方法的过程中，还使用垃圾邮件过滤的例子，其中对象就是电子邮件，性质 P 就是“是垃圾邮件”，而分类器就是垃圾邮件过滤器。接着，我们将把该方法应用到其他例子中，其中，对象就是文档，性质就是“相关”，或者如“乌克兰语”之类的某种语言。

分类问题的出现远比计算机的出现要早得多。经典统计理论的早期工作中就关注过二元分类问题，如在一个啤酒厂的质量控制中确定坏批（质量不好的啤酒——译者注）的过程。在这个基础上出现了一些特殊的术语沿用至今。现在，任何真正的分类必然是不完美的。**第一类错误**（type 1 error）是分类器把本应不具有某种性质的对象错误地认为它具有这种性质（把好的啤酒确认为坏的）；**第二类错误**（type 2 error）是当对象具有某种性质时分类器却无法辨识（不能把坏啤酒检测出来）。这些错误的后果依赖于分类本身的目的。对于啤酒分类而言是避免把坏的啤酒卖给客户。如果第一类错误经常发生，就会由于浪费而导致利益损失；如果第二类错误经常发生，后果虽然不那么显然但却严重很多，包括商誉损失和危害公共健康。

通过**信号检测理论**（signal detection theory）——该理论是在发明雷达时提出的，我们可进一步研究分类。信号检测理论广泛应用于医疗诊断测试以及其他领域。实际上，每个人

都对诊断测试的应用比较熟悉。例如，阳性的妊娠测试会被视为是一个人怀孕的证据（不是proof）。一个**阳性**（positive）的测试结果可作为某些条件的证据；一个**阴性**（negative）的测试结果可作为不符合某些条件的证据。**真阳性**（true positive，tp）是指检测正确的阳性结果，**假阳性**（false positive，fp）指检测错误的阳性结果。同样，**真阴性**（true negative，tn）指检测正确的阴性结果，而**假阴性**（false negative，fn）指检测错误的阴性结果。也就是说，假阳性和假阴性分别对应第一类错误和第二类错误。表 10-16 对应列出了 4 种可能的情况，我们把这张表称为关于诊断测试结果的**相依表**（contingency table）。

表 10-16 关于诊断测试结果的相依表

		性质或条件	
		不出现	出现
测试结果	反例	tn	fn
	正例	fp	tp

诊断测试的有效性一般用**敏感度**（sensitivity）和**特异度**（specificity）描述，也称为**真阳率**（true positive rate，tpr）和**真阴率**（true negative rate，tnr）。如果采用表 10-16 中的元素来表示某个测试中的 4 种可能出现情况的频率，有

$$sensitivity = tpr = \frac{tp}{tp + fn}, \quad specificity = tnr = \frac{tn}{tn + fp} \tag{10-3}$$

因为对于好的测试结果，敏感度和特异度都趋于 1，所以把误检率一并考虑进来会更便于比较。**假阴率**（false negative rate，fnr）和**假阳率**（false positive rate，fpr）定义如下：

$$fnr = 1 - sensitivity = \frac{fn}{tp + fn}, \quad fpr = 1 - specificity = \frac{fp}{tn + fp} \tag{10-4}$$

在妊娠测试中，我们只对怀孕这一情况感兴趣。假阴率对应测试结果为阴性的怀孕妇女的比例；假阳率对应测试结果为阳性的未孕妇女的比例。

假阴率和假阳率很容易因为某种原因——称为**检察官的谬论**（prosecutor's fallacy）——而被误解。现在我们做一个妊娠测试，并设 $fpr=1\%$，$fnr=10\%$，并且某人的测试结果呈阳性。检察官的谬论断言这个人怀孕的概率为 $1-fpr=99\%$。如果这个人是男的，那么明显这个断言是一个谬论。无论测试结果如何，一个男人怀孕的几率是微乎其微的。所以，最合理的解释就是测试结果是假阳性。但如果接受测试的是一个有怀孕症状的育龄妇女，那合理的解释就是测试结果是真阳性，而且其概率要远远高于 99%。为了量化这样的先验概率，我们引入比值、比值比和似然比这三个概念。

10.2.1 比值和比值比

把操作概率和条件概率简称为**比值**（odds）和**比值比**（odds ratio，OR）。给定条件 x 下的事件 e，可以定义：

$$\mathrm{Odds}[e] = \frac{\Pr[e]}{\Pr[\overline{e}]} \tag{10-5}$$

$$\mathrm{OR}(e, x) = \frac{\mathrm{Odds}[e \mid x]}{\mathrm{Odds}[e]} \tag{10-6}$$

根据定义，直接可得：

$$\mathrm{Odds}[e \mid x] = \mathrm{Odds}[e] \cdot \mathrm{OR}(e, x) \tag{10-7}$$

对公式（10-7）中的 $\Pr[e \mid x]$ 和 $Pr[\overline{e} \mid x]$ 采用贝叶斯定律，有：

$$\mathrm{OR}(e,x)=\frac{\frac{\Pr[e|x]}{\Pr[\bar{e}|x]}}{\frac{\Pr[e]}{\Pr[\bar{e}]}}=\frac{\frac{\Pr[x|e]\cdot\Pr[e]\cdot\Pr[x]}{\Pr[x|\bar{e}]\cdot\Pr[\bar{e}]\cdot\Pr[x]}}{\frac{\Pr[e]}{\Pr[\bar{e}]}}=\frac{\Pr[x|e]}{\Pr[x|\bar{e}]} \tag{10-8}$$

这种形式的比值比通常也称为已知事件 e 下条件 x 的**似然比**（likelihood ratio，LR）：

$$\mathrm{LR}(e,x)=\frac{\Pr[x|e]}{\Pr[x|\bar{e}]} \tag{10-9}$$

其中，$\Pr[x|e]$是给定事件 e 下条件 x 的似然，$\Pr[x|\bar{e}]$是给定事件 $\bar{e}$ 下条件 x 的似然。

比值这个概念很好理解，也很容易通过统计“赢”和“输”的次数来估计。如果一个运动队赢了 50 次，输了 30 次，就可以估计他们赢的比值为$\frac{50}{30}\approx 1.67$。比值比可以理解为当条件 x 为真时，比值改变的幅度。例如，假设知道新队员 Mika 的加入将会提高队伍的胜算提高到$\frac{51}{29}\approx 1.76$，那么就可以说比值比为：

$$\mathrm{OR}(\text{winning, Mika})=\frac{\frac{51}{29}}{\frac{50}{30}}\approx 1.06 \tag{10-10}$$

一般来说，如果 $\mathrm{OR}(e,x)>1$，就说条件 x 提高了事件 e 的比值；如果 $\mathrm{OR}(e,x)<1$，那就说 x 降低了 e 的比值。现在考虑加入另外一个运动员 Jenson 的情况，这时 OR（winning，Jenson）≈ 1.12。这表明让 Jenson 加入是一个不错的选择。但同时让这两个运动员都加入情况又如何？OR（winning，Mika and Jenson）的值是多少呢？由朴素贝叶斯假设可断言：

$$\mathrm{OR}(e,x\ and\ y)=\mathrm{OR}(e,x)\cdot\mathrm{OR}(e,y) \tag{10-11}$$

使用这个假设，有

$$\mathrm{OR}(\text{winning, Mika } and \text{ Jenson})\approx 1.18 \tag{10-12}$$

但在很多情况下，这个假设是不现实的。如果 Mika 和 Jenson 在队中的位置是相同的，例如都是当守门员的，那么同时雇佣两人得不到太多的累计效益。而且，如果 Mika 和 Jenson 之间有私人矛盾，反而会导致整个队伍取得胜利的比值下降。

现在考虑妊娠测试，我们希望计算怀孕的比值，已知检测结果呈正例：

$$\mathrm{Odds}[\text{pregnant}|\text{positive}]=\mathrm{Odds}[\text{pregnant}]\cdot\mathrm{OR}(\text{pregnant, positive}) \tag{10-13}$$

从公式（10-8）有：

$$\mathrm{OR}(\text{pregnant, positive})=\mathrm{LR}(\text{pregnant, positive})=\frac{tpr}{fpr}=\frac{90\%}{1\%}=90 \tag{10-14}$$

先验比值（prior odds）——Odd［pregnant］，完全由证据决定，而不是测试。如果要求进行妊娠测试的妇女 80% 实际上都已经怀孕了，那就有：

$$\mathrm{Odds}[\text{pregnant}|\text{positive}]=\frac{80\%}{20\%}\cdot\frac{90\%}{1\%}=360 \tag{10-15}$$

同时有，

$$\Pr[\text{pregnant}|\text{positive}]=\frac{360}{1+360}\approx 99.7\% \tag{10-16}$$

但如果测试结果是阴性，有

$$\mathrm{Odds}[\text{pregnant}|\text{negative}]=\frac{80\%}{20\%}\cdot\frac{fnr}{tnr}=\frac{80\%}{20\%}\cdot\frac{10\%}{99\%}\approx 0.404 \tag{10-17}$$

同时有，

$$\Pr[\text{pregnant}\,|\,\text{negative}] \approx \frac{0.404}{1+0.404} = 28.8\% \tag{10-18}$$

在这种情况下，就需要再进行一次妊娠测试。

10.2.2 构造分类器

在分类和过滤应用中，我们希望利用已知所属类别的文档样本自动构造一个二元分类器。当已知两个互斥类别（例如，有用信息和垃圾信息），构造一个二元分类器以判定对象是否属于其中的一个类别。一个正例结果表示属于这个类别，反例结果表示属于另一个类别。如果类别不是互斥的，则可分别为每一个类别构建一个独立的二元分类器。对于 $n>2$ 个类别，可以将多个二元分类器合并或建立一个 n 路分类器（参见 11.6 节）。本节中我们考虑构建一个二元分类器的问题，并考虑二元分类器在分类和过滤中的作用。

分类可形式化为一个学习器 L 构造一个分类器 c 的过程，c 返回正例/反例结果或一个连续的得分。给定一个文档集合 D，用 $P \subset D$ 表示 D 中具有某些让人感兴趣的文档子集。方便起见，用 $\overline{P} = D \setminus P$ 代表 P 的补集，表示不具备这种让人感兴趣性质的文档子集。关键问题是构建一个分类器——一个具体的函数，对于任何文档 d，都能准确回答问题“已知 d，$d \in P$?”。一个理想分类器是一个完全函数

$$isp : D \to \{pos, neg\} \tag{10-19}$$

使得仅当 $d \in P$ 时，$isp(d) = pos$ 成立。但是这样的理想分类器是不存在的，不过，我们可以构造一个近似的分类器 $c \approx isp$。硬分类器和软分类器所使用的近似概念是不同的。对于硬分类器

$$c : D \to \{pos, neg\} \tag{10-20}$$

对于大部分文档 $d \in D$，通过扩展 $c(d) = isp(d)$ 来逼近 isp。对于软分类器

$$c : D \to \mathbb{R} \tag{10-21}$$

对于大部分 $(d, d') \in P \times \overline{P}$，通过扩展 $c(d) > c(d')$ 来逼近 isp。硬分类器 c_h 可以利用软分类器 c_s 和一个固定阈值 t 来定义：

$$c_h(d) = \begin{cases} pos & (c_s(d) > t) \\ neg & (c_s(d) \leqslant t) \end{cases} \tag{10-22}$$

一个度量硬分类器有效性的简单实用的方法是

$$accuracy = 1 - error = \frac{|\{d \,|\, c_h(d) = isp(d)\}|}{|D|} \tag{10-23}$$

在过滤器构造中通常都会使用并优化**准确率**（accuracy）和**误检率**（error），但它们均独立于普遍度。同时使用以下两个误检率度量

$$fpr = \frac{|\overline{P} \cap \{d \,|\, c(d) = pos\}|}{|\overline{P}|}, \quad fnr = \frac{|P \cap \{d \,|\, c(d) = neg\}|}{|P|} \tag{10-24}$$

对于软分类器，使用代价指标

$$1 - \text{AUC} = \frac{|P \times \overline{P} \cap \{(d, d') \,|\, c(d) < c(d')\}| + \frac{1}{2}|P \times \overline{P} \cap \{(d, d') \,|\, c(d) = c(d')\}|}{|P| \cdot |\overline{P}|} \tag{10-25}$$

在过滤过程中，把分类器 c 看做一个固定公式，带一个由已学习的**概要文件**（profile）指定的隐含参数，也就是说，$c(d)$ 是 $c(profile, d)$ 的简写，其中概要文件是从学习器 L 得

到的。

10.2.3　学习模型

学习器 L 根据获得的证据构造一个概要文件。证据的获取方式依赖于**学习模式**（learning mode）。

- **监督学习**（supervised learning）是机器学习分类器中的一种常见模式。学习器的输入（T, *label*）包含一个训练集 $T \subseteq D$ 以及一个函数 *label*：$T \rightarrow \{pos, neg\}$，*label* 在子集 T 上逼近 *isp*。函数 *label* 一般由专家手工输入。假设 T 与样本 D 独立且同分布（independent and identically distributed，i. i. d.），并且对于所有 $d \in T$，有 $label(d) = isp(d)$，则学习器生成一个对于某些效用函数而言优化了的 c 的概要文件（*profile*），如准确率。带上述假设的监督学习非常常见，一般假设也是没有问题的。但要获得一个样本是非常困难的——尤其是一个与被分类文档独立且同分布的样本。实际上，文档集 D 中的很多成员（我们感兴趣分类的文档）只存在于未来，因此要获得这样的样本是不可能的。同样，构造函数 *label* 的过程也相当费力并且很容易出错，因此这个假设的存在性是值得怀疑的。也不应该假设通过优化准确率就能使分类器到达预期的效果。
- **半监督学习**（semi-supervised learning）假设输入为（T，S，*label*），其中，$T \subseteq D$，$S \subset T$，*label*：$S \rightarrow \{pos, neg\}$，也就是说，*label* 仅定义在训练集的一个子集上。半监督学习考虑到获取样本文档比标记它们容易这一事实。正如监督学习一样，它也假设 T 是文档集 D 的一个 *i. i. d.* 样本。这个假设允许学习器从未标记的样本 $T \setminus S$ 中学习到更多关于文档集 D 的分布情况。当输入为（S，S，*label*）时，半监督学习就是监督学习；这个特殊情况为比较半监督学习器提供了一个方便的基准。
- **直推式学习**（transductive learning）与半监督学习类似，使用与前面一样的标记和未标记样本（T，S，*label*）。区别在于未标记样本就是测试样本，也就是 $T = D$。因此，分类器 $c: T \rightarrow \{true, false\}$ 或 $c: T \rightarrow \mathbb{R}$ 只运用在 T 的元素中。典型的信息检索就是直推式学习的一个例子，所有的语料都用于统计文档信息，被分类为相关文档的集合是语料库的一个子集。
- **非监督学习**（unsupervised learning）假设不存在函数 *label*，也就是说，输入仅有 $T \subseteq D$，并且不直接使用它去构造分类器。尽管如此，非监督学习还是可以与其他方法结合在一起使用的。例如，假设每组中的成员都属于同一个类，**聚类**（clustering）方法可为相似信息找到所属的组。
- **在线学习**（on-line learning）并不事先区分训练样本和测试样本。待分类样本构成一个序列 $S = d_1, d_2, \cdots d_n$。当文档 d_k 被分类后，所有之前的文档 $\{d_{i<k}\}$ 就可以作为训练样本。每个样本首先测试分类器，然后再训练分类器。一个在线分类器（不需要高效或有效）可以通过使用 $T_k = \{d_i \mid i < k\}$ 作为训练集，为每个 d_k 构造一个新的批分类器实现。如果每个文档都被标记了，那就是监督在线学习；否者，就是半监督或者无监督。这个方法有两个主要缺点：

1）样本特性可能会随时间而改变，这种现象称为**概念漂移**（concept drift）。已有方法由于忽略了训练样本序列中内在的时空线索，因此无法为这种现象建模。

2）如果使用每个可用训练样本去构造一个新的分类器 c_k 来分类每个 d_k，那对 S 上总的运行时间将是平方级的或更高。当构造 c_k 时，共有 $k-1$ 个训练样本，检查每一个样本

所需的时间就是k的一个比例。因此，构造所有的 c_k 的时间下界是

$$\sum_{i=k}^{n} k-1 \in \Omega(n^2) \tag{10-26}$$

- **增量式学习**（incremental learning）可以降低分类一个样本序列的总开销。一个增量式学习器从为 d_k 构造的分类器c_k 的隐式概要文件 profile_k 中高效构造c_{k+1}，而无需检查 T_k 中所有的样本。为在线过滤选择分类方法时，一个很重要的标准就是学习器具备高效的增量式构造的能力。增量式学习近似于使用批过滤或者滑动窗口的非增量式学习。
- **主动学习**（active learning）允许分类器从有限的未标记训练样本中请求标记。例如，分类器请求用户标记一些特定信息，然后基于这些标记样本再对剩下的信息进行分类。主动学习的原型方法就是**非确定采样**（uncertainty sampling)(Lewis 和 Catlett，1994)，在每一个未标记样本中使用软分类器，请求那些分类结果接近阈值 t 的样本的标记。

10.2.4 特征工程

尽管我们把分类器的处理域 D 定义为一个文档集，但是很少分类器会在文本表示的内容上进行直接操作。一般来说，每个文档都会表示成一组**特征**（feature）集，这些特征提取自文本或与文本相关的外部信息，例如收到电子邮件的时间。定义与提取特征的过程对分类器是很有用的——称为**特征工程**（feature engineering）——它将对过滤器的整体效果产生重要的影响。因此，没有标记为特征表示的特别的学习方法，读者会怀疑这些公布结果的有效性（无论好或坏）。

一个文档 d 通常被表示为一个 n 元特征向量 $x^{[d]}=\langle x_1^{[d]}, x_2^{[d]}, \ldots, x_n^{[d]}\rangle$，其中每个 $x_i^{[d]}$ 是一个离散或连续值，用于量化信息所包含的对分类有用的证据。因此，每个信息可以表示成 n 维**特征空间**（feature space）中的一个点。最明显的特征就是第 3 章中词条和词项的简单的统计信息了。

分词中的特征抽取与搜索中的分类基本是一样的，只是重点有些不同。在过滤和分类应用中，文档都是按序处理的，而不是从一个大的语料库中抓取，因此不需要构建索引。能用于分类的特征远远比能用于搜索引擎的特征要多得多，因为搜索引擎要为每个特征都检索一次位置信息列表。例如，字符或 n-gram（见 3.3 节）这些特征在大量的应用中都是实际且有用的。**稀疏二元组**（sparse bigram）——由 k 划分的词条对或更小的词条——会增加特征空间的维度，但在过滤中取得了不错的效果（Siefkes 等人，2004）。

尽管如此，当 n 很大时，学习器无论在效率上还是分类效果上的性能都不好。现在考虑从一个训练集 $T(|T| \ll n)$中学习一个分类器时会出现的问题。在这里，肯定可以通过得出联立方程的解来为 T 构建一个完美的分类器。但是，这个分类器在 D 上的总体性能会很差，这种现象称为**过度拟合**（overfitting）。所有的分类器上都会出现**泛化误差**（generalization error），导致该分类器在 D 上的性能比 T 上差。不同学习方法的主要区别之一就在于最小化泛化误差的能力。

特征选择（feature selection）用于减少特征的数量 n，从而减少特征空间 X 的维度。更一般的做法是采用**降维**（dimensionality reduction）技术，将特征空间 X 投影或者变换到一个维度更少的空间中。进行降维主要有两个目的：达到更好的时间或空间效率以及降低泛化误差。

在以前，特征选择或降维是构建分类器前的一个单独预处理步骤。随着分类器构建技术的发展，这种观点不合适了。特征选择和分类器之间可以有很多互动，使得二者可以结合起来考虑。我们对 c 的定义并没有规定不允许作为一个中间步骤将它的输入投影到更小的空间。一些分类器，如朴素贝叶斯方法和决策树方法，就很好地采用了这种做法。但一些性能最好的分类器，如 logistic 回归和支持向量机，就没有明确地采用降维，而是通过其他方法解决效率和泛化误差的问题。然而，我们并不是说特征选择和降维是没有用的，而只是说明在采用这些技术时，需要先考虑应用的背景和分类的方法，在很多情况下，是不需要用到这些技术的。

特征选择是最直接的降维方法，停词（见 3.1.3 节）就是一个简单的例子。已有很多统计方法可用于确定最**重要**（important）的特征，并把剩下的去掉。词干提取（见 3.1.2 节）是降维的一个简单例子，它把多个特征合并成一个。Hash 也是一种成功应用的降维技术，它通过任意合并具有相同 hash 的维度来缩小空间。更复杂的方法，如主成分分析，用线性代数将整个特征空间变换到另一个更小的维度中。

一些特征工程选择可能很难与增量式分类器的构造过程融合在一起，因此也很难和在线自适应过滤融合在一起。之前未知的特征会随时出现，因此增加了空间的维度。某个特征的已知值也会随着学到的新样本而增加。如 IDF 这样的全局统计量就得重算，因为每个文档的增加都改变了它包含的所有词项的 IDF 值。而这个改变会大大影响前面已过滤文档的有效得分。对于在线应用，统计特征选择和降维均存在很大的问题，因为统计量是经常变化的。

评价分类器时一种常见的错误是：在训练和测试文档上做特征选择或降维。这种做法很明显是错误的，因为该方法把从测试样本获得的信息直接传入到分类器的概要文件中。

10.3　概率分类器

概率分类器为给定文档 d（形式为 $x^{[d]}=\langle x_1^{[d]}, x_2^{[d]}, \ldots, x_n^{[d]}\rangle$）计算一个估计概率 $p^{[d]} \approx \Pr[d \in P \mid x^{[d]}]$，表示文档 d 具有性质 P 的概率，或者等价地说，属于具有这个性质的文档集合 P 的概率。这个估计可直接用作一个软分类器

$$c(d) = p^{[d]} \tag{10-27}$$

或加上一个阈值 t（$0<t<1$），得到一个硬分类器

$$c(d)=\begin{cases} pos\ (p^{[d]} > t) \\ neg\ (p^{[d]} \leqslant t) \end{cases} \tag{10-28}$$

概率分类有以下两个步骤：

1）为 $x^{[d]}$ 中的每个 $x_i^{[d]}$ 估计 $p_i^{[x]} \approx \Pr[d \in P \mid x_i^{[d]}]$。

2）把上述估计值结合在一起，输出 $p^{[d]} \approx \Pr[d \in P \mid x^{[d]}]$。

10.3.1　概率估计

特征可以是离散或连续的。一个离散特征（也称为**分类特征**（categorical feature））假设可能取值是有限个。而连续特征取值有无限多个。对于分类特征和连续特征，有不同的概率估计方法。

1. 分类特征的概率估计

首先考虑一种特殊的情况：**二元**（binary）特征，只有 0 和 1 两种可能取值。二元特征

的值 $x_i^{[d]}$：{0，1} 没有什么特殊的含义，$x_i^{[d]}=1$ 表示某个词条出现在文档 d 中，而 $x_i^{[d]}=0$ 表示某个词条不出现在文档 d 中。给定 $x_i^{[d]}=k$，文档 d 在 P 中的比值可估计为训练集 T 中正例与反例的比例：

$$\text{Odds}[d\in P\,|\,x_i^{[d]}=k]\approx\frac{\left|\left\{d\in T\,|\,x_i^{[d]}=k\right\}\cap P\right|}{\left|\left\{d\in T\,|\,x_i^{[d]}=k\right\}\cap \overline{P}\right|} \tag{10-29}$$

例如，假设单词“money”在垃圾信息中出现了 100 次，在有用信息中出现了 5 次。那给定一个包含“money”的信息，它是垃圾信息的估计为 $\text{Odds}\,[d\in P\mid x_i^{[d]}=1]\approx\frac{100}{5}=\frac{20}{1}$。同样地估计用概率形式表达为 $\Pr\,[d\in P\mid x_i^{[d]}=1]\approx\frac{20}{1+20}=0.952$。当 $k=1$ 时，没什么特别的。现在假设训练集 T 由 1000 个垃圾信息和 1000 个有用信息组成，并推断其中 900 个垃圾信息和 995 个有用信息都有 $x_i=0$，因此，一个不包含单词“money”的信息是垃圾信息的估计是 $\text{Odds}\,[d\in P\mid x_i^{[d]}=0]=\frac{900}{995}\approx 0.9$，也就是，这两个概率几乎相等。因此，单词“money”不出现对过滤问题几乎是没有贡献的；出于这个原因，过滤器通常会忽略词条不出现所产生的信息。

如果训练样本中的正例数量 a 和反例数量 b 都足够大，那$\frac{a}{b}$将是一个很好的比值估计。如果它们都比较小，那么由于随机性，这个估计是不可靠的；如果其中一个或同时取值为 0，那么估计结果就是$\frac{0}{1}$、$\frac{1}{0}$或$\frac{0}{0}$，都不是有意义的结果。一个缓解这个问题的简单方法是：分别为分子和分母加上一个很小的正常量 γ 和 ϵ，即用$\frac{a+\gamma}{b+\epsilon}$作为概率估计。当 $a=b=0$ 时，估计结果为$\frac{\gamma}{\epsilon}$；当 a 和 b 都很大时，估计结果为$\frac{a}{b}$。一般来说，$\gamma=\epsilon=1$。

我们知道，上面的估计公式可以推广到非二元分类特征的情况，其中 k 可能取值不只是 0 或 1。这时需要分别为每个 k 值计算比值估计。

2. 连续特征的概率估计

如果 $x_i^{[d]}$ 是实数特征，一种直接估计这个特征概率的方法是将它的值与阈值 t 进行比较，变换为二元特征值 $b_i^{[d]}$：{0，1}

$$b_i^{[d]}=\begin{cases}1 & x_i^{[d]}>t\\ 0 & x_i^{[d]}\leqslant t\end{cases} \tag{10-30}$$

然后向上一节所述那样估计 $\text{Odds}\,[d\in P\mid b_i^{[d]}=k]$。尽管是实数值，与离散值一样，也没有什么特别的意义，因此也可处理一下特征，使得 $x_i^{[d]}$ 越大文档 $d\in P$ 的概率越高。换句话说，$x_i^{[d]}$ 本身就是一个软分类器，$b_i^{[d]}$ 就是对应的硬分类器。对于一个 n 元类别值 $b_i^{[d]}\in\{0,1,\cdots,n-1\}$，可以采用 $n-1$ 个阈值分出 n 个桶（bin）：

$$b_i^{[d]}=\begin{cases}n-1 & t_{n-1}<x_i^{[d]}\\ \cdots & \cdots\\ 1 & t_1<x_i^{[d]}\leqslant t_2\\ 0 & x_i^{[d]}\leqslant t_1\end{cases} \tag{10-31}$$

根据连续特征值 $x_i^{[d]}$ 的取值得到相应的概率估计。

这个方法的主要缺点是：随着 n 增大，对于任意的 k，具有性质 $b_i^{[d]}=k$ 的文档的数量会减少，这使概率估计的可靠性下降。另一种方法是定义一个变换函数 $f:\mathbb{R}\to\mathbb{R}$，使得

$$f(k)\approx \text{Odds}[d\in P\,|\,x_i^{[d]}=k] \tag{10-32}$$

使用带参数的模型代替简单统计方法来估计 $x_i^{[d\in P]}$ 和 $x_i^{[d\in\overline{P}]}$ 的分布。例如，假设符合（正态）高斯[⊖]分布，4 个参数——$x_i^{[d\in P]}$ 和 $x_i^{[d\in\overline{P}]}$ 的均值和标准差 $\mu(i,P)$、$\sigma(i,P)$、$\mu(i,\overline{P})$、$\sigma(i,\overline{P})$——足以描述分布的特征。给定这些参数，可以计算似然比

$$\text{LR}(d\in P,\ x_i^{[d]}=k)\approx\frac{g(\mu_{(i,P)},\sigma_{(i,P)},k)}{g(\mu_{(i,\overline{P})},\sigma_{(i,\overline{P})},k)} \tag{10-33}$$

其中，g 是高斯分布的概率密度函数。由公式（10-7）与公式（10-8）有

$$\text{Odds}[d\in P\,|\,x_i^{[d]}=k]\approx\frac{N_P}{N_{\overline{P}}}\cdot\frac{g(\mu_{(i,P)},\sigma_{(i,P)},k)}{g(\mu_{(i,\overline{P})},\sigma_{(i,\overline{P})},k)} \tag{10-34}$$

其中，N_P 和 $N_{\overline{P}}$ 分别对应 T 中的正例和反例的数量。

3. 一个例子

为了说明上面的方法，使用从 TREC 2005 公共垃圾邮件语料库（Cormack 和 Lynam，2005）中提取的两个特征。已知该语料库中的所有信息都是发送给某个组织中的不同成员，这个组织的名字（在这里是 Enron）在垃圾信息和有用信息中的普遍度不同。我们使用的两个特征分别是转换为小写之后每个信息的标题和正文中出现字符串序列“enron”的次数。head：enron 表示“enron”在标题中出现的次数；body：enron 表示“enron”在正文出现的次数。表 10-17 列出了语料库中 18 条信息的这些属性，其中 10 条是垃圾信息，8 条是有用信息。

表 10-17　TREC 2005 公共垃圾邮件语料库中的样本

信息标签	正确分类	垃圾邮件	有用邮件
016/201	spam	12	0
033/101	spam	11	0
050/001	spam	10	0
066/186	ham	7	24
083/101	ham	21	0
083/101	ham	21	0
100/001	ham	27	4
133/101	spam	12	17
148/013	ham	22	5
166/201	ham	13	23
183/101	spam	11	0
200/001	spam	14	4
216/201	ham	25	2
233/101	spam	13	20
250/001	ham	5	0
266/201	spam	12	0
283/101	spam	13	0
300/001	spam	11	22

⊖ 12.3.2 节讨论了高斯分布：$g(\mu,\sigma^2,x)=\varphi_{\mu,\sigma^2}(x)$。

由于二元特征只表明“enron”在信息对应的部分出现与否，因此用处十分有限。词条“enron”出现在**每个**信息的标题中，因此除了垃圾信息与有用信息的比率，“enron”的出现没有提供任何信息（即似然比为 1 的情况）。词条“enron”出现在 4 个垃圾信息和 5 个有用信息的正文中，因此以 4∶5 的比值估计判断一个正文中包含“enron”的信息是垃圾信息。“enron”没有出现在 6 个垃圾信息和 3 个有用信息的正文中，因此，对于这些信息可以得到 2∶1 的比值估计。

表 10-18 将这些估计（改写成概率形式）与我们的“黄金标准”（即在整个语料库上计算得到的**真实**（true）概率的最佳估计）进行比较。

表 10-18 对表 10-17 的样本进行的采样估计和黄金标准估计的比较

特征 f	训练数据		黄金标准	
	词频	Pr [spam | f]	词频	Pr [spam | f]
head : enron $\neq 0$	1.0	0.56	0.9999	0.57
head : enron $= 0$	0.0	0.50	0.0001	0.00
body : enron $\neq 0$	0.5	0.44	0.62	0.45
body : enron $= 0$	0.5	0.67	0.38	0.77
总体	1.0	0.56	1.00	0.57

表 10-19 列出了把 head：enron 的值分成 3 个离散区间（[0，9]，[10，19]和[20，30]）得到的结果。概率估计从训练数据中得到，此时平滑参数分别为 $\gamma=\epsilon=0$ 和 $\gamma=\epsilon=1$（参见 10.3.1 节了解这些平滑参数的作用）。中间区间（10≤head：enron<20）很好地预测了垃圾信息，两端则预测了有用信息。

表 10-19 对表 10-17 中的样本进行离散特征估计

特征 f	训练数据			黄金标准	
	词频	$\gamma=\epsilon=0$	$\gamma=\epsilon=1$	词频	Pr [spam | f]
$0 \leqslant$ head : enron < 10	0.11	0.00	0.25	0.18	0.05
$10 \leqslant$ head : enron < 20	0.61	0.91	0.85	0.74	0.75
$20 \leqslant$ head : enron < 30	0.28	0.00	0.14	0.05	0.19

假设高斯分布下参数 $\mu_P=11.9$，$\sigma_P=1.2$，$\mu_{\overline{P}}=17.6$，$\sigma_{\overline{P}}=8.3$，表 10-20 给出了 head：enron 每个可能取值的预测。对于较小的 k 值，该模型巧妙估计了 Pr [$d \in P$ | head：enron $=k$]，但对于较大的 k 值，该模型则大大低估了这个概率。但是，很少使用到较大的 k 值，因此低估引起的影响也不大；并且这时通常会得到更多正确的分类结果。无论如何，该模型还是有很大的改进空间的。

表 10-20　高斯模型下对表 10-17 的样本进行的采样估计和黄金标准估计的比较

k	训练数据		黄金标准	
	词频	Pr [spam | head: enron=k]	词频	Pr [spam | head: enron=k]
5	0.06	0.000 0	0.00	0.000 0
6	0.00	0.000 0	0.01	0.070 5
7	0.06	0.001 7	0.08	0.000 0
8	0.00	0.031 1	0.05	0.040 9
9	0.00	0.231 5	0.03	0.176 7
10	0.06	0.588 0	0.07	0.619 1
11	0.17	0.773 5	0.28	0.836 6
12	0.17	0.804 9	0.19	0.734 3
13	0.17	0.715 8	0.09	0.783 8
14	0.06	0.437 1	0.04	0.726 9
15	0.00	0.107 9	0.02	0.632 1
16	0.00	0.009 4	0.01	0.468 7
17	0.00	0.000 4	0.01	0.416 2
18	0.00	0.000 0	0.01	0.483 8
19	0.00	0.000 0	0.01	0.353 9
20	0.00	0.000 0	0.01	0.574 5
21	0.11	0.000 0	0.01	0.423 6
22	0.06	0.000 0	0.01	0.400 8
23	0.00	0.000 0	0.00	0.528 1
24	0.00	0.000 0	0.00	0.102 6
25	0.06	0.000 0	0.02	0.011 4
26	0.00	0.000 0	0.00	0.062 9
27	0.06	0.000 0	0.00	0.002 6

10.3.2　联合概率估计

我们想估计

$$p^{[d]} \approx \Pr[d \in P \mid x^{[d]}] \tag{10-35}$$

分别计算文档 d 的每个特征的估计

$$p_i^{[d]} \approx \Pr[d \in P \mid x_i^{[d]}] \quad (1 \leqslant i \leqslant n) \tag{10-36}$$

为了方便，求这些概率估计的**对数比值**（log-odds）估计作为代替

$$l^{[d]} \approx \text{logOdds}[d \in P \mid x^{[d]}] \tag{10-37}$$

其中

$$l^{[d]} = \text{logit}(p^{[d]}) = \log \frac{p^{[d]}}{1-p^{[d]}}, \qquad p^{[d]} = \text{logit}^{-1}(l^{[d]}) = \frac{1}{1+e^{-l^{[d]}}} \tag{10-38}$$

同时定义

$$l_i^{[d]} \approx \text{logOdds}[d \in P \mid x_i^{[d]}] \quad (1 \leqslant i \leqslant n) \tag{10-39}$$

考虑 $n=0$，…，2 这些特殊情况，同时也考虑 $n>2$ 的一般情况：

- 当 $n=0$ 时，表示空向量，因此，估计值 l_0 可以简化为

$$l^{[d]} = l_0^{[d]} = \frac{|P \cap T| + \gamma}{|\overline{P} \cap T| + \epsilon} \approx \text{logOdds}[d \in P] \tag{10-40}$$

（其中，γ 与 ϵ 是平滑参数，跟前面提到的一样）。

- 当 $n=1$ 时，因为向量空间太小了，几乎可以忽略，因此有

$$l^{[d]} = l_1^{[d]} \approx \text{logOdds}[d \in P \mid x_1^{[d]}] \tag{10-41}$$

- 当 $n=2$ 时，就有点难以处理了。因为并没有一般的方法，在不考虑条件依赖 $x_1^{[d]}$ 与 $x_2^{[d]}$ 的情况下将 $l_1^{[d]}$ 与 $l_2^{[d]}$ 结合得到一个估计值。从公式（10-7）与公式（10-40）有

$$\log \text{OR}(d \in P, x_1^{[d]}) \approx l_1^{[d]} - l_0^{[d]} \tag{10-42}$$

$$\log \text{OR}(d \in P, x_2^{[d]}) \approx l_2^{[d]} - l_0^{[d]} \tag{10-43}$$

在朴素贝叶斯假设下（参见公式（10-11））有

$$\log \text{OR}(d \in P, x^{[d]}) = \log \text{OR}(d \in P, x_1^{[d]} \text{ 且 } x_2^{[d]}) \approx l_1^{[d]} - l_0^{[d]} + l_2^{[d]} - l_0^{[d]} \tag{10-44}$$

因此有

$$l^{[d]} = -l_0^{[d]} + l_1^{[d]} + l_2^{[d]} \approx \text{logOdds}[d \in P \mid x^{[d]}] \tag{10-45}$$

但在实际中，朴素贝叶斯假设很少成立。例如当发现邮件中包含词项“sildenafil”时——无论是垃圾邮件还是正常邮件——通常同时也包含词项“Viagra”。除了无效的假设，一般还会使用朴素贝叶斯分类器，因为该分类器简单，而且当使用概率阈值 t=0.5 时，它可以作为一个硬分类器，尽管这时候它们的概率估计还不精确（Domingos 和 Pazzani，1997）。

一个相反的假设是 x_1 与 x_2 是独立的。例如，“sildenafil”与“Viagra”的同时出现表明邮件是垃圾邮件；但只出现二者之一，或这两个单词同时出现在某封特定的邮件都无法判定该邮件就是垃圾邮件。简短地说，一个包含“sildenafil”和“Viagra”的信息并不比只包含其中一个词项的信息更像或更不像是一个垃圾邮件。在这个假设下，l_1 与 l_2 可能是差不多的，因为它们估计的是同一个量并且只有估计误差不同：

$$l^{[d]} = \frac{l_1^{[d]} + l_2^{[d]}}{2} \tag{10-46}$$

对于朴素贝叶斯和 log-odds 平均来说，在 $n \geqslant 2$ 时，一般的解决方案是将各个 $l_i^{[d]}$ 线性组合起来得到 $l^{[d]}$：

$$l^{[d]} = \sum_{i=0}^{n} \beta_i \cdot l_i^{[d]} \tag{10-47}$$

对于朴素贝叶斯，有

$$\beta_i = \begin{cases} 1-n & (i=0) \\ 1 & (i>0) \end{cases} \tag{10-48}$$

对于 log-odds 平均，有

$$\beta_i = \begin{cases} 0 & (i=0) \\ \frac{1}{n} & (i>0) \end{cases} \tag{10-49}$$

表 10-21 用我们例子中两个离散特征的联合值比较了这两个方法。我们看到 log-odds 平均得到的估计相对稳定，趋近于 $p_0=0.55$；而朴素贝叶斯法得到的估计值相对波动较大。对于某些例子，采用 log-odds 平均可能会得到更好的估计值，但对于另外一些，采用朴素贝叶斯会可能得更好的估计值。

表 10-21　用 log-odds 平均和朴素贝叶斯得到的联合概率估计

特征 f_1	特征 f_2	$\Pr[\text{spam} \mid f_1, f_2]$		
		Log-Odds 平均	朴素贝叶斯	黄金标准
$0 \leqslant$ head : enron < 10	body : enron $= 0$	0.45	0.36	0.14
$0 \leqslant$ head : enron < 10	body : enron > 0	0.33	0.16	0.03
$10 \leqslant$ head : enron < 20	body : enron $= 0$	0.77	0.90	0.86
$10 \leqslant$ head : enron < 20	body : enron > 0	0.66	0.76	0.65
$20 \leqslant$ head : enron < 30	body : enron $= 0$	0.36	0.21	0.40
$20 \leqslant$ head : enron < 30	body : enron > 0	0.25	0.08	0.12

β_i 的取值有多种可能。例如，求朴素贝叶斯估计和 log-odds 平均估计的均值，得到一个不同的线性组合，反映了各个 $x_i^{[d]}$ 间的部分条件依赖关系。或者对于某个 β_i，根据对应的 $l_i^{[d]}$ 的精确度，选用合适的权值代替 $\frac{1}{n}$。

给定一组已标记训练样本，logistic **回归**（logistic regression）计算使**似然**（likelihood）最大化的 β_i。假设概率估计为 $p^{[d]}=\frac{1}{1+e^{-l[d]}}=\Pr[d \in P \mid x^{[d]}]$，似然是将样本概率简单联合的值，即

$$likelihood = \prod_{d \in T \cap P} p^{[d]} \cdot \prod_{d \in T \cap \overline{P}} 1 - p^{[d]} \tag{10-50}$$

因为 logistic 回归是为了求 β_i，所以不需要计算每个 $l_i^{[d]}$ 的 log-odds 估计。尤其不需要为 logistic 回归变换分类特征。相反，可以把特征 $x_i^{[d]}$：$\{k_1, k_2, \cdots, k_m\}$ 理解成 m 个不同的二元特征 $x_{i1}^{[d]}$，$x_{i2}^{[d]}$，…，$x_{im}^{[d]}$，其中

$$x_{ij}^{[d]}=\begin{cases} 1 & (x_i^{[d]} = k_j) \\ 0 & (x_i^{[d]} \neq k_j) \end{cases}$$

通常，$x_i^{[d]}$ 是二元值，$x_{i0}^{[d]}$由于 10.3.1 节所述的理由会忽略掉，因此 $x_i^{[d]}$ 可用 $x_{i1}^{[d]}$有效代替。另外，如果连续特征与 log-odds 成正比（或接近于正比），也不需要对它进行变换。

10.3.3　实际考虑

特征的表示方式与结合方法对分类器实现的简单性及其效率有着很大的影响，特别对于在线分类器。前面已经提到，一些特征变换方法，如 TF-IDF 与统计特征选择等，是很难与自适应分类器结合起来使用的。另一种基于概率的理解方式——对全局分布建模——同样也存在类似的困难。因此，从单个信息中抽取离散特征，独立于从训练集中得到的特征，会更适合在线系统。即使对于批过滤，这些简单的特征表示通常也与那些复杂的特征一样，甚至更好，而且基于全局统计信息的特征自适应性也较差。

如果参数已知，朴素的贝叶斯分类器还是很容易实现的，而且批处理版本和在线版本之间也没有什么区别。批处理版本如图 10-7 所示。它统计 $T \cap P$ 与 $T \cap \overline{P}$ 中每个特征的出现次数，并由此计算 log-odds 系数。一个等价的在线自适应版本（没有画出来）只是简单地把统计值与系数计算结合起来。新特征的增量发现也很容易完成：当某个特征第一次出现在文档中，就将其初始值设为 0。

输入：
训练样本集 $T \subset D$，训练样本 $d \in T$，表示为 $x^{[d]} = \langle 1, x_1^{[d]}, x_2^{[d]}, \cdots, x_n^{[d]} \rangle$
标记函数 $label$：$T \to \{0, 1\}$
平滑参数 γ、ϵ
输出：
$\beta \cdot x^{[d]}$是 *Log Odds* [$label$ (d) =1] 的朴素贝叶斯估计，其中 $\beta = \langle \beta_0, \cdots, \beta_n \rangle$

```
1  p ← a ← ⟨0, ... 0⟩
2  for d ∈ T do
3    for i ∈ [0..n] do
4      if x_i^[d] = 1 then
5        if label(i) = 1 then p_i ← p_i + 1 else a_i ← a_i + 1
6  β_0 ← logit((p_0+γ)/(a_0+ε))
7  for i ← 1 to n do
8    β_i ← logit((p_i+γ)/(a_i+ε)) − β_0
```

图 10-7 构造朴素贝叶斯分类器

尽管朴素贝叶斯方法适合做一个硬分类器，但是它依然是一个软分类器，因为它过度估计了各个特征结合所带来的效果。结合的特征越多，过度估计越严重。为了消除这个影响，我们从每个文档中选择固定的词项个数。对于某个固定的 m 值，$x_i^{[d]}$ 中只有最大的 m 个值和最小的 m 个值用于计算 $x^{[d]}$。这个选择过程倾向于规范化在文档中以及在正例和反例特征中过度估计的幅度。

一般将 logistics 回归看做一个批处理算法，但在在线或批处理应用中，采用**梯度下降**（gradient descent）法比使用朴素贝叶斯来实现会更简单和有效。梯度下降法通过在每次迭代中都沿着负梯度方向前进，找到该函数的局部最小值。图 10-8 展示了批处理版本的实现。最简单的增量版本，对于每个接收的文档都执行一次梯度步骤，而不是多次迭代直至收敛。也可以维护当前已知的训练样本的增量历史并在此基础上训练。例如，当训练一个新的正例时，也同时训练一个随机选择的反例，周而复始。最终的结果就是平衡了正例与反例的样本数量，以达到更好的分类效果。

输入：
训练样本集 $T \subset D$，训练样本 $d \in T$，表示为 $x^{[d]} = \langle 1, x_1^{[d]}, x_2^{[d]}, \cdots, x_n^{[d]} \rangle$
标记函数 $label$：$T \to \{0, 1\}$
速率参数 δ
输出：
$\beta \cdot x^{[d]}$是 Pr [$label$ (d) =1] 的最大似然估计，其中 $d \in T$

```
1  β ← ⟨0, ..., 0⟩
2  loop until convergence:
3    for d ∈ T do
4      p ← 1/(1+e^(−β·x^[d]))
5      β ← β + (label(d) − p) · δ · x^[d]
```

图 10-8 使用梯度下降法实现的 logistic 回归

在很多时候，梯度下降法的空间效率对大规模问题相当有吸引力。然而，如果训练样本数量很多，那就需要时间效率更高的算法了。

梯度下降法相对较慢的收敛速度对于克服过度拟合是一个优点。毕竟 logistics 回归是用于解决联立方程的，因此当特征数量超过训练样本数量时，会出现过度拟合的问题。只要选择合适的学习率 δ，梯度下降法就能避免这个问题。另外，批处理方法使用了**正则化**（regularization）的技术来避免过度拟合。正则化不但最大化每个训练样本的似然值，同时还最小化 β 的变化幅度，因为大的系数值容易出现过度拟合问题。在最大化似然值和最小化 $|\beta|$ 之间需要找到一个平衡点，定义为正则化参数，记为 C。

所有主要的统计和数学软件包里都实现了批处理的 logistic 回归，这是科学研究的一个标准工具。对于分类，比较著名的实现包括 Weka（Witten 和 Frank，2005），LR-TRIRLS（Komarek 和 Moore，2003）和 LibLinear[⊖]。

10.4　线性分类器

线性分类器把信息 d 的特征向量 $x^{[d]}$ 看做 n 维空间中的一个点，其中 n 是特征的数量。分类器由一个系数向量 $\beta=\langle\beta_1,\beta_2,\cdots,\beta_n\rangle$ 和一个阈值 t 组成。公式 $\beta\cdot x=t$ 定义了一个超平面，将空间分成两半（half-spaces）。超平面一边（$\beta\cdot x^{[d]}>t$）的所有点都分类为正例，而在另一边（$\beta\cdot x^{[d]}\leqslant t$）的所有点都分类为反例。如果 $\forall_{d\in p}:\beta\cdot x^{[d]}>t$ 并且 $\forall_{d\in\overline{P}}:\beta\cdot x^{[d]}\leqslant t$，那 $\beta\cdot x=t$ 就是一个**分类超平面**（separating hyperplane）。如果存在这个集合的一个分类超平面，那么一组信息集被称为是**线性可分**（linearly separable）。前面一节介绍的概率分类器的 log-odds 公式就是一个线性分类器。本节给出该分类器的一种几何上的理解方式，并给出几种构造方法。

为了方便，我们仅讨论 $n=2$ 的情况。但请记住典型的过滤应用还是会涉及多个特征，从而也有更多的维度。图 10-9 展示了我们例子（表 10-17）中 18 个信息的向量空间表示。其中，x 轴对应高斯模型变换后的 head：enron 特征（表 10-20）。y 轴对应用统计数表示的 body：enron 特征。对角线就是一个分类超平面，因为所有的正例都在该对角线的一边，而所有的反例都在另外一边。因此，它是一个完美的分类器——至少对于样本数据来说是这样的。

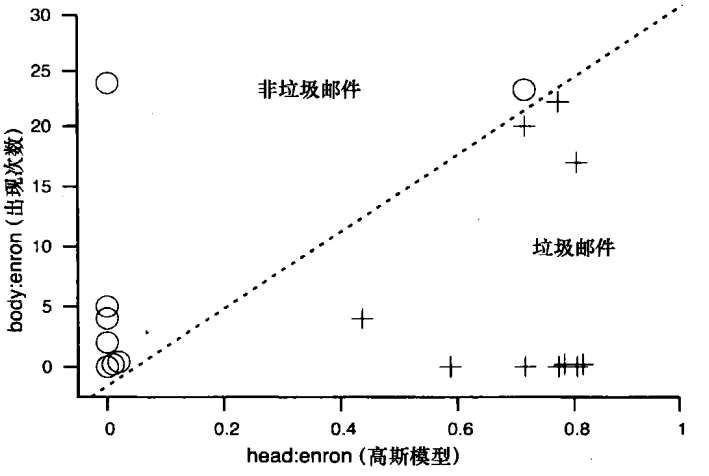


图 10-9　一个线性可分的例子，带一个分类超平面

图 10-10 说明了同一条线在同源样本数量增加后不再是一个分类超平面，实际上，已经不存在一个分类超平面了。但是，越来越多的正例落到了**垃圾邮件**一边，而越来越多的反例落到了另一边。因此这条线仍然是一个合理的分类器。但是它是该向量空间里**最好**（best）的线性分类器吗？如果只使用训练数据，该怎么选择呢？

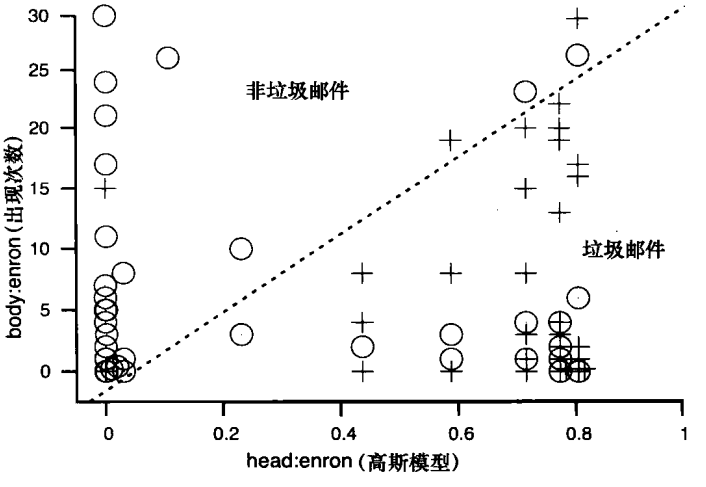


图 10-10　一个数据量更大但线性不可分的例子

⊖ www.csie.ntu.edu.tw/~cjlin/liblinear

答案就在于“**最好**”的定义。

如果所有点都是线性可分的，那一般都可以找到无限多个分类超平面。图 10-11 中给出的所有正例线性组合的极端曲线将正例和反例区分开来。即使存在一个超平面。也不太看得出来最好的分类器是一个分类超平面。如果假设有用信息〈0.72，23〉是独异点——可能在训练数据中这是错误的——可以合理选择图 10-12 中的垂直分割线，它反映了第二个特征是没有实际作用的这一假设。但事实上，原来的分类器（即图 10-10）是更适合的。然而，我们这里只关注基于训练数据的选择。关于什么才是最好的分类器，图 10-9 与图 10-12 展示了两个完全对立的观点：

- 一种是正确分类所有的训练样本，同时最大化超平面与离它最近的样本的距离（图 10-9）。
- 另一种是允许部分样本分类错误，同时增加其他样本到超平面的距离（图 10-12）。

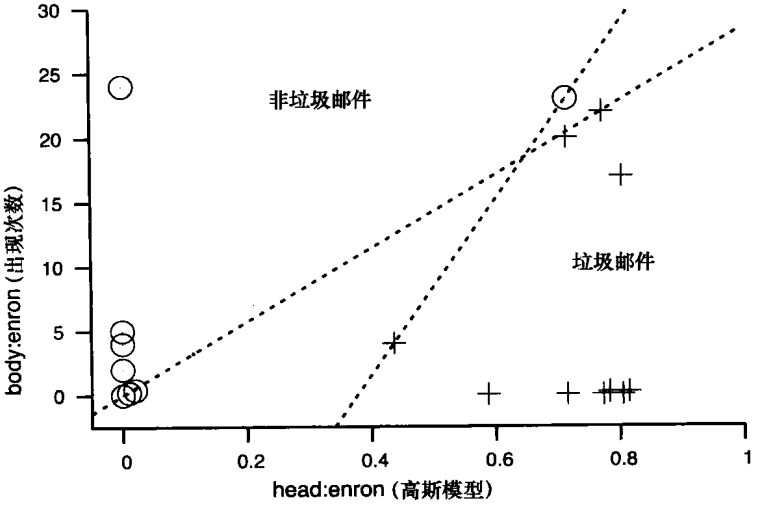


图 10-11　一个线性可分的例子，存在两个分类超平面。哪一个“更好”

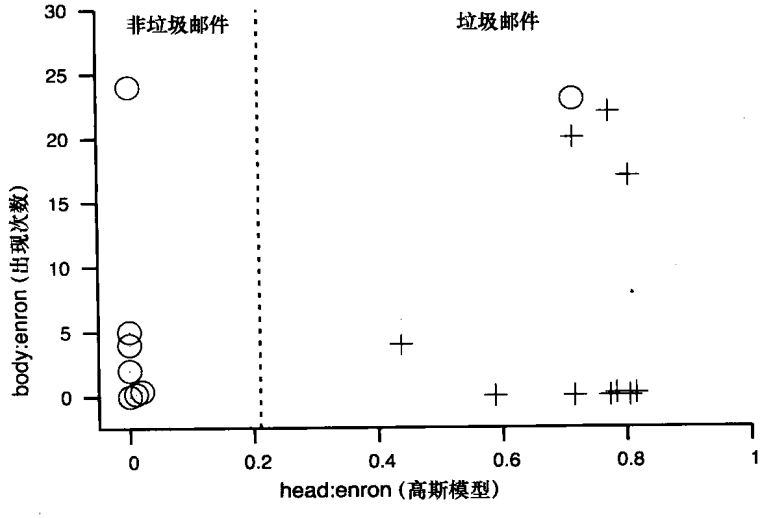


图 10-12　忽略一个点

10.4.1　感知器算法

感知器算法（图 10-13）迭代地找出一个分类超平面——或任何分类超平面，如果存在一个分类超平面——就以一个权重向量 β 开始，在由 β 指定的超平面的错误区域内，对于每一个样本可以是升序的，也可以是降序的。算法忽略已正确分类的样本。如果样本是线性可分的，则算法在有限步内收敛，否则算法不终止。对于实际应用而言，经过一段时间后，假设找到一个足够好的分类器，就算不是在任何意义上都最优，算法也可以停止了。感知器算法对于过滤而言是适合的，因为它简单、可增量并自适应。

输入：
训练样本集 $T \subset D$，训练样本 $d \in T$，表示为 $x^{[d]} = \langle 1, x_1^{[d]}, x_2^{[d]}, \cdots, x_n^{[d]} \rangle$
标记函数 $label$：$T \to \{-1, 1\}$
输出：
如果线性可分，输出 β 使得 $\beta \cdot x^{[d]} > 0$，当且仅当 $label(d) = 1$ 时，
否则，算法无法终止。

```
1  β ← ⟨0, ..., 0⟩
2  while ∃_{d∈T} : β · x^[d] · label(d) < 0 do
3      β ← β + x^[d] · label(d)
```

图 10-13　感知学习算法

间隔感知器（margin perceptron）算法为那些接近超平面且落在分类正确区域上的样本和所有落在错误区域的样本增加值 β。间隔定义为在欧几里得空间中离超平面最近的样本到超平面的距离。间隔感知器（图 10-14；参见 Sculley 等人（2006））加入了一个间隔参数 τ，使该方法偏向于选取间隔值较大的分割线。在标准感知器算法停止时，间隔感知器还会调整超平面直至 $\frac{\tau}{|\beta|}$ 的间隔满足。注意到每一步 $|\beta|$ 都会增大，不断地减少间隔直到找到一个合适的超平面。然而，并不能保证可以找到最大的可能间隔（即最小的可能的 $|\beta|$），但只要 τ 足够大，间隔感知器一般都能找到一个合理的近似。

输入：
训练样本集 $T \subset D$，训练样本 $d \in T$，表示为 $x^{[d]} = \langle 1, x_1^{[d]}, x_2^{[d]}, \cdots, x_n^{[d]} \rangle$
标记函数 $label$：$T \to \{-1, 1\}$
间隔参数 τ
输出：
如果线性可分，输出 β，使得当 $label(d) = 1$ 时，$\beta \cdot x^{[d]} \geqslant \tau$；当 $label(d) = -1$ 时，$\beta \cdot x^{[d]} \leqslant -\tau$。
否则，算法无法终止。

```
1  β ← ⟨0, ..., 0⟩
2  while ∃_{d∈T} : β · x^[d] · label(d) < τ do
3      β ← β + x^[d] · label(d)
```

图 10-14　间隔感知器

10.4.2　支持向量机

支持向量机（support vector machine，SVM）直接计算分类超平面，使离超平面最近样本的间隔或距离最大。到超平面距离相等的若干点被称为**支持向量**（support vector），产生的分类器是这些向量的一个线性组合——其他点可以忽略。支持向量机更倾向得到如图 10-9 的结果（而不是图 10-11），有用信息的支持向量为 ⟨0，0⟩，⟨0.72，23⟩，垃圾信息的支持向量为 ⟨0.72，20⟩。

对于不可分数据，或被少数点严重影响的可分数据（如图 10-12 或我们训练数据中的点

⟨0.72，23⟩)，不要要求所有的训练样本都被正确分类。支持向量机需要在最大化间隔与最小化训练误差间找到平衡点。平衡系数 C 表示后者相对于前者的权重。$C=0$ 表明 SVM 仅仅关注第一项；$C=1$ 表示这两者权重相等，同等重要，一般都是用这个作为默认值；$C=100$ 表示给第二项很大的权重，经验表明，这适用于过滤垃圾信息（参见 Drucker 等人 (1999) 或 Sculley 等人 (2006))。

很多软件包都实现了 SVM，其中包括 Weka[⊖]、SVM-light[⊜] 和 LibSVM[⊜]。Sculley 和 Wachman (2007) 提出了利用 SVM 来实现有效的增量在线过滤的梯度方法。

10.5 基于相似度的分类器

本节将介绍基于相似度的分类器，它利用的假设是相似文档比不相似文档更有可能属于同一类。对于分类，把相似度的概念形式化为一个函数 sim：$D\times D\to\mathbb{R}$，其中 $sim(d_1, d_2) > sim(d_3, d_4)$ 表示在某种意义上 d_1 与 d_2 比 d_3 与 d_4 更相似。与这最相似的例子是第 2 章中介绍的信息检索里的向量空间模型，在那里 sim 是个余弦公式（公式 2-12)：

$$sim(d_1, d_2) = \frac{|x^{[d_1]} \cdot x^{[d_2]}|}{|x^{[d_1]}| \cdot |x^{[d_2]}|} \tag{10-51}$$

对于不带历史样本的面向主题的过滤器，文档可以根据它与查询 q 的相似度进行排名，就与排名检索一样，由此可得到一个软分类器：

$$c(d) = sim(d, q) \tag{10-52}$$

如果有历史样本，就可计算 d 与历史样本中某个文档或全部文档的相似度，然后根据这些结果对 d 进行分类。现在定义一个新的相似度函数以形式化这种方法：Sim：$D\times 2^D\to\mathbb{R}$，其中 $Sim(d, D')$ 表示 d 与集合 $D'\subset D$ 中的文档的相似度。各种基于相似度的分类器的区别主要在于如何定义函数 sim，以及 Sim 如何从 sim 演化出来。

10.5.1 Rocchio 法

Rocchio 法 (Rocchio，1971) 定义

$$Sim(d, D') = sim(d, d'), \quad \text{其中 } x^{[d']} = \frac{1}{|D'|}\sum_{d\in D'} x^{[d]} \tag{10-53}$$

其中，D'用一个虚拟文档 d'表示，d'的特征向量就是 D'中所有文档的质心。形式最简单的 Rocchio 分类器仅使用了训练样本的正例：

$$c(d) = Sim(d, T\cap P) \tag{10-54}$$

对于分类，当正例训练样本和反例训练样本都存在时，可用以下差值得到一个更好的分类器

$$c(d) = Sim(d, T\cap P) - Sim(d, T\cap \overline{P}) \tag{10-55}$$

在向量空间模型中，Rocchio 法曾被广泛用作相关反馈。正如向量空间模型中不再使用余弦指标一样，Rocchio 法也不再使用了，如 BM25 这样的相关反馈方法取得了更好的效果。不难看出 Rocchio 法其实是一个线性分类器，但它的性能不如这里介绍的其他分类器。

⊖ www.cs.waikato.ac.nz/ml/weka

⊜ svmlight.joachims.org

⊜ www.csie.ntu.edu.tw/~cjlin/libsvm

应用在语言分类和垃圾信息过滤（10.1 节）的 BM25 可以看做 Rocchio 法的一个变种，sim 的定义是采用 BM25 相关反馈公式，而不是余弦指标。

10.5.2　基于记忆的方法

基于记忆（memory-based）的方法，也被称为**基于事例**（case-based）的方法，用训练样本本身作为分类器的概要文件。当需要分类某个文档或文档集时，就搜索这些样本。也许，最简单的基于记忆的方法就是**最近邻法**（nearest neighbor，NN），一般认为该分类器属于硬分类器：

$$c_h(d) = label\left(\arg\max_{d' \in T} sim(d, d')\right) \tag{10-56}$$

如果 sim 刚好是前面已经介绍过的标准搜索任务里的排名方法之一，那么最近邻分类器可通过索引训练样本然后用搜索引擎检索最相似的文档来实现。否则，需要为每个 $d' \in \tau$ 计算 sim（d，d'）。

更常见的做法是：我们先计算一个最近邻软分类器，然后推导出一个硬分类器：

$$Sim(d, D') = \max_{d' \in D'} sim(d, d') \tag{10-57}$$

$$c_s(d) = Sim(d, T \cap P) - Sim(d, T \cap \overline{P}) \tag{10-58}$$

$$c_h(d) = \begin{cases} \text{pos} & (c_s(d) > 0) \\ \text{neg} & (c_s(d) \leqslant 0) \end{cases} \tag{10-59}$$

c_s 的有效实现包括为 T_{pos} 与 T_{neg} 构造独立的搜索索引。

一个简单的变形算法是 k **最近邻**（k-nearest neighbor，KNN），其中对于固定的 k，只考虑与 d 最相似的 k 个文档。这样一个硬分类器便可定义为最相似的 k 个文档的主要投票。（即 d 的类别由 k 个文档中最多的类别来决定——译者注）

10.6　广义线性模型

前面已经提到，特征工程与分类器构造这两者之间没有本质的区别。通过选择合适的特征表示 $x^{[d]}$，我们可以将任何分类问题完全转换成一个线性分类器能解决的问题。目前我所用的例子依赖于下面这些特征工程：

- 对于概率分类器，采用 logit 变换——称为**关联函数**（link function）或**变换函数**（transfer function）——将问题转化为线性分类问题。
- 对于线性分类样本，对其中一个维度运用高斯变换。

图 10-15 展示了未经变换的线性分类样本。两个维度的特征表示都是原始词频。在这里，我们将用以下这些概念将特征的原始表示和变换表示区分开来：

- $x^{[d]}$ 是 d 的特征的**原始**（raw）表示。这个例子中，$x^{[d]}$ 是由两个词频组成的向量。通常，在某种意义上 $x^{[d]}$ 是文档 d 的直接表示。
- $x^{[d]} = \varphi(x^{[d]})$ 是文档 d 的特征的**变换**（transformed）表示，其中 φ 是一个映射函数。x 是一个向量空间，它的维度不需要与 x 相同。如果维度比原来的少，则是运用了**降维技术**（dimensionality reduction）。否则维度可能会变大，甚至于无限大。对于后者，可用**核方法**（kernel method）在 X 上构造线性分类器，而不需要计算 $X^{[d]}$。感知器分类法与 SVMs 都可用作核方法。

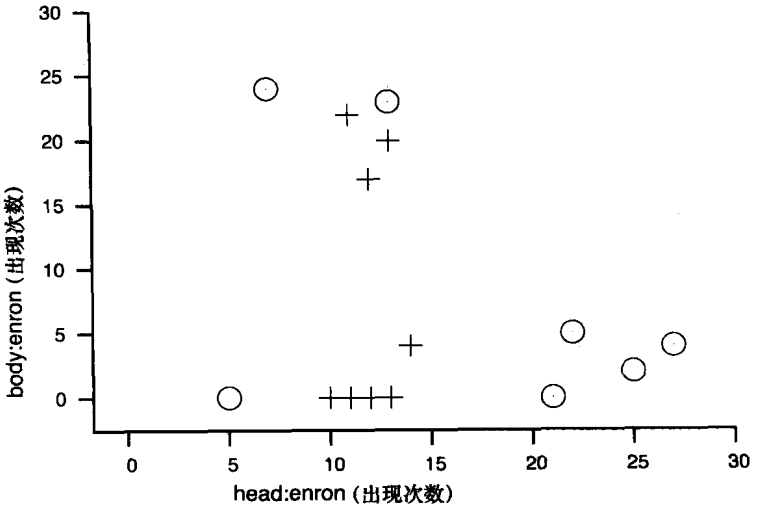


图 10-15　未经变换的特征

使用公式（10-34），为样本得到一个映射

$$X^{[d]}=\varphi(x^{[d]})=\left\langle \frac{N_P}{N_{\overline{P}}}\cdot\frac{g(\mu_{(1,P)},\sigma_{(1,P)},x_1^{[d]})}{g(\mu_{(1,\overline{P})},\sigma_{(1,\overline{P})},x_1^{[d]})},\ x_2^{[d]}\right\rangle \tag{10-60}$$

注意到，可在特征工程的某个步骤中从训练样本里估计得到四个参数：μ（1，P）、σ（1，P）、μ（1，$\overline{P}$）、σ（1，$\overline{P}$）。另外一种方法为这些参数推测几个值，然后看看哪一个效果更好，这个过程称为**参数选择**（parameter selection），更通俗一点的说法是**调整**（tuning）。同样，我们也可以推测 φ。推测 φ 及其参数的过程称为**模型选择**（model selection）。在评价一个构造过程涉及参数或模型选择的分类器时，至关重要的是评价（即测试）数据无法验证（算法的效果——译者注）。一个常用的方法是将训练样本分成训练集与**验证**（validation）集（将会在 11 章详细介绍）。现在我们暂时假设 φ（包括与它有关的任何参数）是已知的。

为了便于说明，我们考虑相同数据上的另外一个映射 φ^+，增加了 x 的维度，变成三维空间：

$$\varphi^+(x^{[d]})=\left\langle x_1^{[d]},\ x_2^{[d]},\ \frac{N_P}{N_{\overline{P}}}\cdot\frac{g(\mu_{(1,P)},\sigma_{(1,P)},x_1^{[d]})}{g(\mu_{(1,\overline{P})},\sigma_{(1,\overline{P})},x_1^{[d]})}\right\rangle \tag{10-61}$$

这个映射使整个空间变得立体了，它沿着新的第三维从平面上凸显了正例，使得反例往相反方向凹陷。在这个新的三维空间中，样本是线性可分的。并且，第一维是不必要的，因为去掉它使空间减为二维空间后，样本依然是线性可分的：

$$\varphi^-(X^{[d]})=\left\langle X_3^{[d]},\ X_2^{[d]}\right\rangle \tag{10-62}$$

原来的映射 φ 其实就是 φ^+ 与 φ^- 的组合。

$$\varphi(x^{[d]})=\varphi^-(\varphi^+(x^{[d]})) \tag{10-63}$$

核方法

一个线性分类器可以重新定义为一个基于相似度的分类器，其中

$$sim(d_1,d_2)=X^{[d_1]}\cdot X^{[d_2]} \tag{10-64}$$

现在考虑特征空间 x 是线性可分的情况。感知器算法可以用于计算权重向量 β 使得

$$c(d)=\beta\cdot X^{[d]} \qquad c(d\in P)>0 \qquad c(d\in \overline{P})<0 \tag{10-65}$$

由图 10-13 可知，β 必须是训练集中的正例与反例的线性结合，其中正例的系数均取非负值，反例的系数均取非正值，即有：

$$\beta=\sum_{d\in T}\alpha_i X^{[d]} \qquad \alpha_{d\in P}\geqslant 0 \qquad \alpha_{d\in \overline{P}}\leqslant 0 \tag{10-66}$$

这里 α 是一个权重向量，T 中的每个文档都有一个这样的向量。联立公式（10-64）、公式（10-65）和公式（10-66），得到分类器的**对偶公式**（dual formulation）：

$$c(d)=(\sum_{d'\in T}\alpha_{d'}\cdot x^{[d']})\cdot x^{[d]}=\sum_{d'\in T}\alpha_{d'}\cdot(x^{[d']}\cdot x^{[d]})=\sum_{d'\in T}\alpha_{d'}\cdot sim(d,d') \tag{10-67}$$

除了在 sim 的定义中，对偶公式并没有用到 $X^{[d]}$。**核感知器**（kernel perceptron）算法（图 10-16）采用这种表示方式并计算 α（而不是 β），因此更新规则变为

$$\beta\leftarrow\beta+X^{[d]}\cdot label(d) \qquad\qquad \alpha_d\leftarrow\alpha_d+label(d) \tag{10-68}$$

因此核感知器需要用到 sim 与 α，但不需要计算 $X^{[d]}$ 或 β。其他方法，包括间隔感知器和 SVM 等，均可形式化为核方法。

输入：
训练样本集 $T\subset D$，训练样本 $d\in T$，样本可任意表示
标记函数 $label$：$T\rightarrow\{-1,1\}$
相似度（核）函数 sim：$D\times D\rightarrow\mathbb{R}$，其中 $sim(d_1,d_2)=X^{[d_1]}\cdot X^{[d_2]}$定义在虚特征空间 X 上
输出：
如果线性可分的，输出 α 使得 $c(d)>0$，当且仅当 $label(d)=1$，其中 $c(d)=\sum_{d'\in T}\alpha_{d'}sim(d',d)$。否则，算法无法终止。

```
1  α ← ⟨0, ..., 0⟩
2  while ∃_{d∈T} : c(d) · label(d) < 0 do
3      α_d ← α_d + label(d)
```

图 10-16　核感知器学习算法

所谓的**核技巧**（kernel trick）就是不需要 X 就能实现 $sim(d_1,d_2)$，因此也避免了计算 X。只要公式（10-64）成立，采用任何的实现方式均可以。这种实现方法称为**核函数**（kernel function）。核技巧允许我们使用大量甚至无限多个虚特征，这在其他技术里是不可能的。

例如，现在有一个文档集 $d\in D$，且它的原始特征表示 $x^{[d]}$，由 n 个词频组成。这里不使用传统的余弦法则直接构造一个线性分类器，而是根据两个文档中词频相同的词项数量定义相似度：

$$sim(d_1,d_2)=|\{i\in[1,n]\,|\,x_i^{[d_1]}=x_i^{[d_2]}\}| \tag{10-69}$$

一般来说，$f=x_i^{[d]}$ 可取无限大的数，但通过定义一个从（f，i）到 N 映射 π，即可枚举所有可能的值：

$$\pi(f,i)=i+n\cdot f \tag{10-70}$$

给定 π，定义虚特征表示 $X^{[d]}$ 为

$$X^{[d]}_{\pi(f,i)}=\begin{cases}1 & (x_i^{[d]}=f)\\ 0 & (x_i^{[d]}\neq f)\end{cases} \tag{10-71}$$

在这个空间中，有 $sim(d_1,d_2)=X^{[d_1]}\cdot X^{[d_2]}$。但是通过枚举 $x^{[d_1]}$ 和 $x^{[d_2]}$，而不是

$X^{[d_1]}$和 $X^{[d_2]}$来实现 sim。

特征空间与核函数的设计只受限于人的想象力。标准例子包括字符串核、多项式核、径向基函数（高斯）核。SVM 包通常支持多种核函数，包括用户自定义的核函数，其中 sim 函数由编程语言决定。

10.7 信息理论模型

一个**模型**（model）就是从已知事件中估计某些事件发生的概率。气象学家可能会使用像湿度与大气压这些依据，从而预测明天下雨的机会为 80%。在这个例子中，需要预测的事件是下雨，气象模型估计的是 $\Pr[\text{rain}] \approx 0.8$ 与 $\Pr[\text{not rain}] \approx 0.2$。

在第 6 章中，我们介绍了几个数据压缩模型，这些模型都假设模型$\mathcal{M}$准确预测了来自字母表$\mathcal{S}$的每个符号 s：

$$\forall_{s \in \mathcal{S}} \Pr[s_i] = \mathcal{M}(s_i) \tag{10-72}$$

这里，我们做相反的假设：任何实际模型$\mathcal{M}$都不必是精确的，如果比其他模型更接近真实概率，那就说$\mathcal{M}$比$\mathcal{M}'$更好。但是判别$\mathcal{M}$和$\mathcal{M}'$哪一个更好，有两个挑战：

- **真实概率**（true probability）是无法得到的，无法将其作为比较时的“黄金标准”。
- 概念“接近”存在多种可能的解释。

假设这些问题都已经解决了并且我们也有一个方法来判断哪一个是更好的模型，则可以通过以下两个方法之一来构造一个分类器：

- 分别根据训练样本中的正例和反例建立**数据压缩模型**（data compression model），并使用它们估计一个未知文档的似然比：

$$\mathcal{M}_P(d) \approx \Pr[d \mid d \in P] \qquad \mathcal{M}_{\overline{P}}(d) \approx \Pr[d \mid d \in \overline{P}] \tag{10-73}$$

$$c_s(d) = \frac{\mathcal{M}_P(d)}{\mathcal{M}_{\overline{P}}(d)} \approx LR(d \in P, d) \tag{10-74}$$

- 从多个候选模型中选出一个最符合训练样本标签的模型$\mathcal{M}$，并使用$\mathcal{M}$本身进行分类：

$$c_s(d) = \mathcal{M}(label(d) = pos \mid d) \approx \Pr[d \in P \mid d] \tag{10-75}$$

这个方法通常用于构造**决策树分类器**（decision tree classifier）。更一般的，这种方法被称为**模型选择**（model selection）。

10.7.1 模型比较

定义$\mathcal{P}(x)$为假设事件 x 为真时的概率，定义$\mathcal{M}$为这个概率的模型：

$$\mathcal{M}(x) \approx \mathcal{P}(x) = \Pr[x] \tag{10-76}$$

$I_{\mathcal{P}}(x)$表示给定 p 后，x 所含的**信息量**（information content），单位是位（bit）：

$$I_{\mathcal{P}}(x) = -\log_2 \mathcal{P}(x) \tag{10-77}$$

$\mathcal{H}(\mathcal{P})$是分布$\mathcal{P}$下随机变量 X 的**熵**（entropy），或期望信息量（用位衡量）：

$$\mathcal{H}(\mathcal{P}) = \mathrm{E}[I_{\mathcal{P}}(X)] = \sum_x \mathcal{P}(x) \cdot I_{\mathcal{P}}(x) \tag{10-78}$$

$\mathcal{H}(\mathcal{P})$表示编码 X 所需的比特位的理论下界。当且仅当用 $I_{\mathcal{P}}(x)$位为每个 x 编码时，才能达到这个下界。

如第 6 章中介绍的那些实际的数据压缩方法，出于以下两个原因是达不到这个下界的：

- 在$\mathcal{M}(x) = \mathcal{P}(x)$的假设下，编码每个 x 大约需要 $I_{\mathcal{M}}(x)$位，因此优化的是$\mathcal{H}(\mathcal{M})$而不是$\mathcal{H}(\mathcal{P})$。

• 一般来说，$I_{\mathcal{M}}(x)$不是一个整数，因此会浪费了部分编码位数。

首先考虑的一个问题是：用个 $I_{\mathcal{M}}(x)$位对 x 编码后，离下界还差多远？分布$\mathcal{P}$下的随机变量 X 的期望长度由$\mathcal{P}$与$\mathcal{M}$的**交叉熵**（cross-entropy）决定：

$$\mathcal{H}(\mathcal{P};\mathcal{M}) = \mathrm{E}[I_{\mathcal{M}}(X)] = \sum_x \mathcal{P}(x)\cdot I_{\mathcal{M}}(x) \tag{10-79}$$

交叉熵（编码长度）与熵（理论下界）的差就是 KL 距离（KL divergence）（见 9.4 节）：

$$D_{KL}(\mathcal{P}\|\mathcal{M}) = \mathcal{H}(\mathcal{P};\mathcal{M}) - \mathcal{H}(\mathcal{P}) \tag{10-80}$$

如果模型$\mathcal{M}_1$ 所需的期望编码长度比模型$\mathcal{M}_2$ 要短，我们就说模型$\mathcal{M}_1$ 比模型$\mathcal{M}_2$ 更好，即

$$\mathcal{H}(\mathcal{P};\mathcal{M}_1) < \mathcal{H}(\mathcal{P};\mathcal{M}_2) \Leftrightarrow D_{KL}(\mathcal{P}\parallel\mathcal{M}_1) < D_{KL}(\mathcal{P}\parallel\mathcal{M}_2) \tag{10-81}$$

$$\Leftrightarrow \sum_x \mathcal{P}(x)\cdot(I_{\mathcal{M}_1}(x) - I_{\mathcal{M}_2}(x)) < 0 \tag{10-82}$$

10.7.2　序列压缩模型

如部分匹配预测（prediction by partial matching，PPM）（Cleary 和 Witten，1984）、动态马尔可夫压缩（dynamic Markov compression，DMC）（Cormack 和 Horspool，1987）与上下文加权树（context-tree weighting，CTW）（Willems 等人，1995），这些都是序列压缩模型，它们把信息 m 看做一个有限字母表中的符号序列 $s_1 s_2 \cdots s_n$。先按序处理这些符号，然后轮流对每个前缀 $s_1 s_2 \cdots s_k$ 单独建立一个模型$\mathcal{M}_k(0 \leqslant k < n)$。每个模型$\mathcal{M}_k$ 用于压缩 s_{k+1}，然后就被丢弃掉。整个信息的模型$\mathcal{M}$为

$$\mathcal{M}(m) = \prod_{k=0}^{n-1} \mathcal{M}_k(s_{k+1}) \tag{10-83}$$

在这个模型中，信息 m 的最优编码长度为

$$I_{\mathcal{M}}(m) = \sum_{k=0}^{n-1} I_{\mathcal{M}_k}(s_{k+1}) \tag{10-84}$$

为了进行分类（Bratko 等人，2006），我们将每个文档 d 表示为符号序列$m^{[d]}$，然后将所有的 $m^{[d\in T\cap P]}$（训练样本的正例）连接起来，构成一个子序列 m^P；也将所有的 $m^{[d\in T\cap \overline{P}]}$（训练样本的反例）连接起来，构成一个子序列 $m^{\overline{P}}$。这样就为这些序列建立了两个序列模型$\mathcal{M}_P$ 与$\mathcal{M}_{\overline{P}}$。当需要对一个新信息 $m^{[d]}$进行分类时，将它分别与 m^P 和 $m^{\overline{P}}$连接，得到两个新的模型$\mathcal{M}_{Pd}$与$\mathcal{M}_{\overline{P}d}$。构造软分类器如下：

$$c(d) \approx \log \frac{\Pr[m^P m^{[d]} \mid d \in P]}{\Pr[m^{\overline{P}} m^{[d]} \mid d \in \overline{P}]} - \log \frac{\Pr[m^P \mid d \in P]}{\Pr[m^{\overline{P}} \mid d \in \overline{P}]} = \log \mathrm{LR}(d \in P, m^{[d]}) \tag{10-85}$$

$$c(d) = I_{\overline{P}d}(m^{\overline{P}} m^{[d]}) - I_{\overline{P}}(m^{\overline{P}}) - I_{Pd}(m^P m^{[d]}) + I_P(m^P) \tag{10-86}$$

也就是说，分类器就是将文档 d 归为反例时信息的增加量与归为正例时信息的增加量的差值。

实际应用中，不需要一开始就计算$\mathcal{M}_{Pd}$与$\mathcal{M}_{\overline{P}d}$。序列压缩法可以从已有的$\mathcal{M}_P$ 与$\mathcal{M}_{\overline{P}}$、以及 $m^{[d]}$中有效构造出$\mathcal{M}_{Pd}$与$\mathcal{M}_{\overline{P}d}$。如果发现 $d \in P$，就用$\mathcal{M}_{Pd}$替代$\mathcal{M}_P$，发现 $d \in \overline{P}$，就用$\mathcal{M}_{\overline{P}d}$替代$\mathcal{M}_{\overline{P}}$，那增量训练过程很容易受到影响。

我们用动态马尔可夫压缩（DMC）来进行解释，这是性能较好的最简单的方法。DMC 将每个信息都看成是一段位序列，文本格式则用 ASCII 或 Unicode。因此，信息表示成一段符号序列（符号为 0 或 1 的）。图 10-17 说明了 DMC 的工作原理。开始，初始马尔可夫模型

（见 1.3.4 节）只有很少几个状态，如图 10-17a 所示。信息都被处理为一段二元位序列，因此状态之间的转换标记为 0 或 1。每个转换上还标记了一个频率，记录经过它的总次数。由此可知，图 10-17a 在训练时使用的是一段长 22 位的序列，其中有 16 位是 1，6 位是 0。状态 A 表示一段最后一位为 0 的序列，状态 B 表示一段最后一位是 1 的序列。从一个状态转移到另外一个状态的概率可以使用以下公式估计：

$$\text{Odds}[1|A]\approx\frac{4}{2}=2 \quad \text{且} \quad \text{Odds}[1|B]\approx\frac{12}{4}=3 \tag{10-87}$$

方法与在线分类器一样：首先使用概率估计公式估计下一位，然后转移到正确的状态，对应的概率也会增大。

如果转换过程中某个状态经常被访问到，那就将其**复制**（cloned）为两个相似的状态。这种情况如右图 b）所示：开始，当将要从状态 A 转移到状态 B 时，把经过这条弧的频率增加到 5；然后，复制状态 B，得到 B′，并把那条高频弧重定向到 B′。状态 B 与 B′的出度按照它们的入度来分配，因此 Odds[1|B′]=Odds[1|B]=3；最后，转移到状态 B′，并增加经过那条弧的频率。通过状态复制，马尔可夫模型逐步增加训练样本中已建模的子字符串的长度。在这个例子中，B′表示以 01 结尾的字符序列模型，B 表示以 11 结尾的字符序列模型。而状态 A，与复制前一样，表示以 0 结尾的字符序列模型。

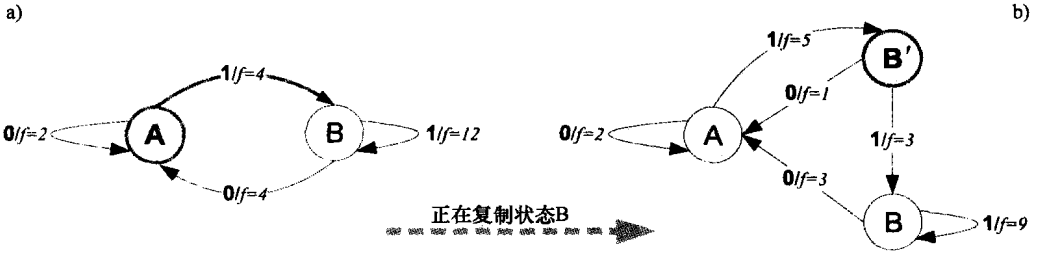


图 10-17 DMC 马尔可夫模型的复制操作

DMC 某个实现版本可以从网上免费下载。[⊖]表 10-22 列出了运用两个 DMC 模型（一个针对有用信息，一个针对垃圾信息）来顺序处理运行样例中的 18 个信息中的 To：field（参见表 10-17）的结果。表 10-22 的第一列列出了信息的正确类别；第二列列出了整体对数似然比 $c(d)$；第三列列出了 To：field 的信息。加深的字符 x_i 表示对数似然比与该字符相关（即字符中每个位的对数似然比的和）：黑色表示垃圾信息（$\log LR(d\in spam, x_i)\gg 0$）；浅灰色表示有用信息（$\log LR(d\in spam, x_i)\ll 0$）；中灰色表示不属于任何一个类别（$\log LR(d\in spam, x_i)\approx 0$）。

第一条信息是灰色的，并且因为它前面没有任何信息，所以对数似然比为 0。这时垃圾信息的模型与有用信息的模型是完全一样的。紧接着的两条信息都被分类为垃圾信息（正确），这并不奇怪，因为暂时没有有用信息样本可作比较。第四条信息——第一条有用信息——被错误地分类为垃圾信息，因为它的对数似然比是正数。随着学习到的样本越来越多，模型从有用信息中识别出垃圾信息的能力越来越强。最后 7 条信息全部被正确分类。可以从各种独立特征中归纳出有用信息的一些关键特征：

⊖ plg.uwaterloo.ca/~ftp/dmc/dmc.c

- 引号
- 全部大写的词条“ENRON”
- 专有姓名，如 Adams 和 pete. davis

垃圾信息的特征有：

- 全部小写的词条“enron”
- 名字 kholst 的各种变形

表 10-22　对信息中的 To: field 运用 DMC 得到的结果，按信息出现的顺序进行在线学习。第一列给出了正确的类别。第二列给出了 DMC 算出的对数似然比。正数表示垃圾信息，负数表示有用信息。第三列给出了信息文本，黑色字体表示高垃圾信息可能，浅灰色字体表示低垃圾信息可能。为了方便讨论，仅采用了 To: field 这部分信息；如果要得到更好的结果，可以使用完整的信息

类别	得分	信息片段
垃圾信息	0.0	To: emclaug@enron.com
垃圾信息	3.9	To: SKean@enron.com
垃圾信息	0.1	To: <joydish@bareed.alburaq.net>
有用信息	1.1	To: "Shapiro, Richard" <Richard.Shapiro@ENRON.com>
有用信息	-0.8	To: "Adams, Jacqueline P." <Jacqueline.P.Adams@ENRON.com>,
有用信息	-4.4	To: "Adams, Jacqueline P." <Jacqueline.P.Adams@ENRON.com>,
有用信息	1.5	To: pete.davis@enron.com
垃圾信息	1.0	To: KAM.KEISER@enron.com
有用信息	-2.1	To: "Abel, Chris" <Chris.Abel@ENRON.com>,
有用信息	-1.8	To: "Moran, Tom" <Tom.Moran@ENRON.com>,
垃圾信息	-0.1	To: nqeb5e6@msn.com
垃圾信息	1.6	To: skean@enron.com
有用信息	-1.4	To: pete.davis@enron.com
垃圾信息	0.1	To: kholst <kholst@enron.com>
有用信息	-1.1	To: "Scott, Susan M." <Susan.M.Scott@ENRON.com>
垃圾信息	0.8	To: mmotley@enron.com
垃圾信息	3.2	To: kholst@enron.com
垃圾信息	3.1	To: keith.holst@enron.com

利用信息相邻符号之间的关系，可将数据压缩模型用于过滤。DMC 使用一个按位动态马尔可夫模型，逐步为越来越长的高频序列建模。PPM 则刚好相反，使用一个 n-gram 字符模型（通常取值 $4 \leqslant n \leqslant 8$），因为后缀树表示比特征向量更合适（特征向量需要用到线性空间）。在特定的理论假设下，采用上下文树加权得到的结果接近于最优，但相比于其他方法，它更复杂，而且得到的结果并不比 DMC 或 PPM 要好。

10.7.3　决策树与树桩

决策树分类器（decision tree classifier）连续地将 D 分成多个子集，使得每个子集中正例的比例或反例的比例比 D 中原来的比例要高。假设存在一棵适当划分 D 的树，那么此树可通过 T 中的样本来构造。

我们考虑二元决策树，划分通过一组布尔公式来表达，每一个公式指定了一个子集上的二元划分。每个公式通常都非常简单，基于某个二元特征或将连续特征与某个阈值进行比较的结果。

最简单的决策树是**二元决策树桩**（binary decision stump），使用一个二元公式 b 将 D 分成两个子集：

$$D^b = \{d \in D | b\}, \qquad D^{\bar{b}} = \{d \in D | \bar{b}\} \tag{10-88}$$

决策树是通过不断地划分 D 的子集而构造出来的。$n-1$ 步后，就得到 n 个不相交的子集：

$$D=D_1 \cup D_2 \cup \cdots \cup D_n \tag{10-89}$$

在 $k=1$ 至 $k=n-1$ 的每一步中，需要进行以下两个决定：

1）确定需要划分的子集 $D_i(1 \leqslant i \leqslant k)$。

2）确定用户将 D_i 划分成 D_i 与 D_k 的 b_{ik}。

挑战就在于如何进行这些决定从而得到一个好的分类器。对于任何合理的“好分类器”的定义，这个问题都会显得很棘手，而且使用某个启发式信息每次只能以“贪心”的方式进行一次决定。这时通常会使用**信息增益**（information gain，IG）。

考虑为一个二元分类问题建立一个模型$\mathcal{M}$，可能的事件分别是 pos 和 neg：

$$\mathcal{M}(\text{pos})=1-\mathcal{M}(\text{neg}) \approx \Pr[d \in P] \tag{10-90}$$

给定一组已标记的训练样本 T，可利用正例的比例建立一个简单的模型：

$$\mathcal{M}(\text{pos})=\frac{|T \cap P|}{|T|} \tag{10-91}$$

训练样本的标签集 $label(T)=\{label(d) \mid d \in T\}$ 中所含的信息量可表示为：

$$I_{\mathcal{M}}(label(T))=\sum_{d \in T} I_{\mathcal{M}}(label(d))=|T \cap P| \cdot I_{\mathcal{M}}(\text{pos})+|T \cap \overline{P}| \cdot I_{\mathcal{M}}(\text{neg}) \tag{10-92}$$

现在分别为 $T^b=T \cap D^b$ 与 $T^{\bar{b}}=T \cap D^{\bar{b}}$ 建立模型：

$$\mathcal{M}_b(\text{pos})=\frac{|T^b \cap P|}{|T^b|} \tag{10-93}$$

$$\mathcal{M}_{\bar{b}}(\text{pos})=\frac{|T^{\bar{b}} \cap P|}{|T^{\bar{b}}|} \approx \Pr[d \in P|\bar{b}] \tag{10-94}$$

假设对于已分类文档，$B \in \{b, \bar{b}\}$ 是已知的，则组合模型为：

$$\mathcal{M}_B=\begin{cases} \mathcal{M}_b & (B=b) \\ \mathcal{M}_{\bar{b}} & (B=\bar{b}) \end{cases} \approx \Pr[d \in P|B] \tag{10-95}$$

给定 B，则训练标签的信息量为：

$$I_{\mathcal{M}_B}(label(T))=I_{\mathcal{M}_b}(label(T^b))+I_{\mathcal{M}_{\bar{b}}}(label(T^{\bar{b}})) \tag{10-96}$$

信息增益与 B 有关，当 B 已知时，信息增益是指标签集所包含的信息量的减少量：

$$IG(B)=I_{\mathcal{M}}(label(T))-I_{\mathcal{M}_B}(label(T)) \tag{10-97}$$

在表 10-17 的 18 个训练样本中，有 10 个是正例，8 个是反例。模型$\mathcal{M}$使用正例所占的比例作为概率估计：

$$\mathcal{M}(\text{pos})=\frac{10}{18} \approx 0.556 \tag{10-98}$$

$$I_{\mathcal{M}}(label(T)) \approx -10 \cdot \log_2(0.556)-8 \cdot \log_2(0.444) \approx 17.84 \text{ (位)} \tag{10-99}$$

现在用公式“$b \Leftrightarrow d$ 的正文中包含 enron”划分 T，有：

$$\mathcal{M}_b(\text{pos})=\frac{4}{9} \approx 0.444 \tag{10-100}$$

$$\mathcal{M}_{\bar{b}}(\text{pos})=\frac{6}{9} \approx 0.667 \tag{10-101}$$

$$I_{\mathcal{M}_b}(label(T^b)) \approx -4 \cdot \log(0.444)-5 \cdot \log(0.556) \approx 8.92 \text{ (位)} \tag{10-102}$$

$$I_{\mathcal{M}_{\bar{b}}}(label(T^{\bar{b}})) \approx -6 \cdot \log(0.667) - 3 \cdot \log(0.333) \approx 8.26 \text{ (位)} \tag{10-103}$$

$$I_{\mathcal{M}_B}(label(T)) \approx 8.92 + 8.26 = 17.18 \text{ (位)} \tag{10-104}$$

$$IG(B) = I_{\mathcal{M}} - I_{\mathcal{M}_B} \approx 0.66 \text{ (位)} \tag{10-105}$$

为了建一棵决策树，每一步都需要搜索可能的公式集，并选择产生最大信息增益值的公式。很重要的一点是，不要搜索出太多的可能公式，或构建过于复杂的公式，或将 D 分成太多的子集。上述任何一种做法都可能导致模型对训练数据过度拟合。这些在信息论中都有详细的解释：在计算信息增益时，我们应该考虑的是公式本身包含的信息量，而不是训练样本标签中包含的信息量。如果从 n 个固定的候选集中选择公式，它的信息量是 $\log_2(n)$ 位。如果公式更复杂，它的信息量可能会更大。当将这些公式的信息量都结合在一起时，训练样本标签的信息增益是个负值，进一步划分可能会适得其反。

10.8　实验对比

本节将前面介绍的各种具有代表性的方法应用到 10.1 节中介绍的 3 个不同的样本集中。为了便于比较，均采用 BM25 得到的结果作为基准。这里得到的结果只是为了让读者对各种方法的效果有个大概的印象。同时这些结果也将作为第 11 章中描述的联合和融合的方法的输入和比较基准。

10.8.1　面向主题的在线过滤器

10.1.2 节详细介绍了面向主题的过滤器会出现的问题，表 10-23 的“历史样本训练”一列中给出了解决这个问题的各种方法。“自适应训练”一列则给出了基于这些方法的自适应训练的效果。自适应训练将历史样本和测试样本视为一个普通序列，并顺序对每个文档进行分类，然后再利用每个文档进行训练。也就是说，该方法与 10.1.5 节中介绍的垃圾信息过滤器是一样的。为了进行比较，综合指标只考虑测试样本的分类结果（即 1993 年及其以后的《**金融时报**》(*Financial Times*)）。

表 10-23 的第一行列出了采用 BM25（在 10.1.2 节中介绍）得到的结果。其中，采用历史样本进行训练的结果就是表 10-6 最后一行的结果。“NB”这一行的结果是采用朴素贝叶斯分类器得到的，该分类器的参数设置如下：

- 二元特征采用字节 4-gram。
- 平滑参数采用 $\gamma = \varepsilon = 1$。
- 从每个文档中提取出 30 个得分最大的特征与 30 个得分最小的特征。

“LR”（梯度下降）这一行的结果是采用在线梯度下降 logistic 回归法得到的（10.3.3 节详细描述了该方法），参数设置如下：

- 二元特征采用字符 4-gram。
- 每个特征向量 $x^{[d]}$ 映射为规范化长度的形式 $X^{[d]} = \dfrac{x^{[d]}}{\sqrt{\left| x^{[d]} \right|}}$。
- 比例参数为 $\delta = 0.004$。
- 采用自适应训练。对于每个新的训练样本，只沿梯度方向前进一步，然后，随机选择一个属于另外一个类别的历史训练样本，并把这个样本作为训练样本。最终的结果是，对相关样本和不相关样本进行训练的次数是一样的。

“DMC”这行是对文本表示的文档采用了 DMC 压缩法的结果。其余均采用批训练的方

法，因此只给出了历史训练样本上的结果。LR（批过滤）是 logistic 回归：

- 使用 LibLinear 软件包（版本为 1.33），使用标记“-s 0”（L2 正则化 logistic 回归），并采用默认参数。
- 采用二元 4-gram 特征。

对于 SVM：

- 使用 SVM^{light}（版本为 6.02），不使用标记（线性核，默认参数）。
- 采用二元 4-gram 特征。

对于 DT：

- 使用 FEST[⊖]（Fast Ensembles of Sparse Trees，稀疏树的快速合并）软件包。标记 $n=2$ 的结果是使用标记“-d 1”（决策树桩，深度为 1）得到的；标记 $n=8$ 的结果是使用标记“-d 3”得到的；标记 $n=256$ 的结果是使用标记“-d 8”得到的。

表 10-23　TREC 主题 383（见图 10-2）的分类结果

方法	历史样本训练		自适应训练	
	P@10	AP	P@10	AP
BM25	1.0	0.56	0.8	0.30
NB	0.0	0.00	0.4	0.06
LR（梯度下降）	0.7	0.39	1.0	0.55
DMC	0.0	0.01	0.1	0.06
LR（批）	0.8	0.48		
SVM（批）	0.8	0.49		
DT（n=2）	0.1	0.03		
DT（n=8）	0.9	0.53		
DT（n=256）	0.8	0.53		

在历史样本上 BM25 的分类效果比其他方法好，这看起来也许不明显，因为 BM25 本来就是为这个目的专门训练的。但是，它的性能要比自适应训练的差。出现这种情况，是因为我们把前 20 个得分最高的特征作为反馈项，而且这个特征集会随着训练样本的增多而改变。相反，带自适应训练的其他的自适应方法的分类效果有所提高。

朴素贝叶斯、DMC 和决策树桩的分类效果都很差。然而，其他的学习方法效果都很好。尽管单凭这个例子就做出哪个算法最好的结论是不恰当的，但这些方法的确值得认真考量。

表 10-24 列出了对语言分类采用了本质上一样的方法后得到的结果。所采用的方法都在 10.1.4 节中介绍了。分别根据 60 种语言对文档排名，并且使用得分结果对每个文档的类别进行排名。为了对文档进行排名，我们列出 60 个排名的 MAP 值，同时为了分类，还列出了 4012 个测试文档上的误检率与 MRR。在文档排名方面，SVM 表现最好，但这个优势没有在分类排名上体现。总的来说，对于提高分类效果的方法，可能的选择也就是 DMC、LR、SVM 与 BM25 了。NB 与 DT（对于任意 n）在文档排名与分类上得到的结果都很差。

⊖ www.cs.cornell.edu/~nk/fest/

表 10-24　语言分类的分类结果

方法	历史样本训练	自适应训练	
	MAP	微平均误检率（%）	MRR
BM25	0.78	20.6	0.86
NB	0.44	27.0	0.80
LR（梯度下降）	0.76	22.9	0.84
LR（批）	0.82	20.0	0.86
SVM（批）	0.83	20.0	0.80
DMC	0.74	20.0	0.86
DT (n=2)	0.33	64.1	0.44
DT (n=8)	0.54	43.7	0.63
DT (n=256)	0.63	34.8	0.70

10.8.2　在线自适应垃圾信息过滤

表 10-25 列出了在线垃圾信息过滤中各种自适应方法的结果，这些方法在 10.1.5 节中进行了介绍。图 10-18 将这些结果以一条 ROC 曲线表示出来。BM25、NB、LR 和 DMC 除了以下的区别都与前面介绍的一样：

- 只使用每条信息的前 2500 字节，因为根据在 TREC 上的实际经验，这样做效果较好。
- 对于 BM25，不像面向主题过滤中将反馈项的数量限制在 20 个，这里对反馈项的数量没有限制。同时，分别计算当垃圾信息与有用信息视为是相关时的得分，然后用二者差值作为总得分。

表 10-25　在线垃圾信息过滤的分类结果

方法	分类错误率（%）					CPU运行时间
	fpr	fnr	宏平均误检率	Logistic 平均误检率	1-AUC	
BM25	32.85	0.01	16.43	0.77	0.190	8 m
NB	1.21	1.21	1.21	1.21	0.062	24 s
LR（梯度下降）	0.41	0.47	0.44	0.44	0.012	12 s
ROSVM	0.32	0.42	0.37	0.37	0.013	4 d
DMC	0.31	0.54	0.42	0.37	0.013	6 m

ROSVM 是指松弛在线 SVM（Sculley 和 Wachman，2007），这是 TREC 2007 垃圾邮件专题中性能最好的两个过滤器之一（Cormack，2007）。ROSVM 是专门设计用于自适应过滤的高效过滤器，改变了经典 SVM 的方法原本不适合的应用。但是，Sculley 为 TREC 会议实现的 ROSVM 工具包比梯度下降 LR 慢 25 000 倍。可以从网上下载 ROSVM 的一个开源的版本。[⊖]

通过比较所有的指标，我们发现 DMC、LR 与 ROSVM 的过滤效果是一样的。从 ROC 曲线可以看出，NB 比 BM25 的效果要好，但在局部效果上这两种方法都次于其他方法。

⊖ www.eecs.tufts.edu/~dsculley/onlineSMO

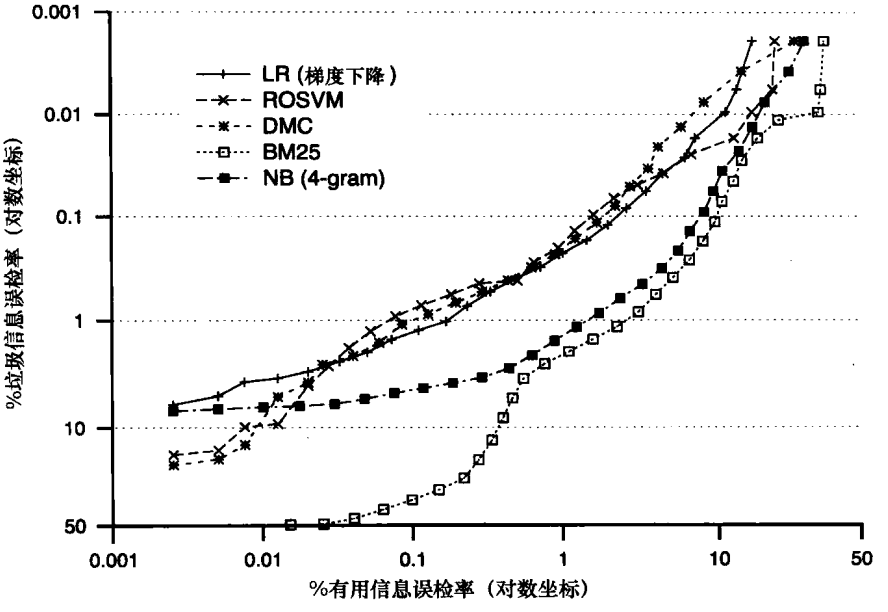


图 10-18　在线垃圾信息过滤器的 ROC 曲线

10.9　延伸阅读

信息检索与统计学习方法涉及多个不同的研究领域。Mitchell（1997）对机器学习给出了一个较好的介绍，但内容可能有点旧。如果想要得到这些方面的一些综合性理论介绍，可以参见 Hastie 等人（2009）的著作。支持向量机是 Joachims（2002）的主要研究方向。Logistic 回归是用于对一般数据进行分析的好方法（Hosmer 和 Lemeshow，2000），并发现该方法对文本分类能得到较好的结果（Komarek 和 Moore，2003），但相比于其他机器学习方法，采用该方法的人不多。

在信息检索领域主要研究检索方面，分类与过滤就显得没那么重要了。Belkin 和 Croft（1992）讨论了检索与过滤之间的关系。在机器学习领域中，文本分类通常都被视为是监督学习分类的一个应用，因为特征恰好就表示了文档。Sebastiani（2002）研究了针对文本分类的学习方法，其内容涵盖了所有方面，而我们前面只提到了 passing。这些降维方法包括：通过词项选择，利用对文献集使用 F_1 得到的比较结果以及从 Reuters news 中提取出来的 RCV1 基准集（Lewis 等人，2004）。

van Rijsbergen（1979）引入了 F_β，将其作为基于集合检索的一种指标。这个指标是根据检索结果的数学性质对结果进行评价，而不是用户的满意程度。Lewis（1991）引入了微平均的概念。引入这个概念是因为他发现，如果每个文档可能属于多个类别，文档 d 属于类别 q，那要让硬分类器从集合 $D\times Q$ 中检索出所有对（d，q），可能会出现问题。基于集合的微平均指标，如查准率、查全率与将检索集 $Rel\subseteq D\times Q$ 与相关集 $Ret\subseteq D\times Q$ 进行比较的 F_β。

Swets（1963，1969）提出了一个用于信息检索评价的信号检测理论。然而，这些方法只适用于部分排名检索方法。真阳率和假阳率对应与信息检索指标中的查全率和漏查率。所以查全率-漏查率曲线与 ROC 曲线是一样的。近年来，机器学习已经把 ROC 曲线分析作为一种标准（Fawcett，2006），并在 TREC 垃圾邮件专题已采用了这种标准（Cormack 和 Ly-

nam，2005）。LAM 是跟垃圾邮件专题一起介绍的，随后被证明与诊断概率比是一样的（Glas 等人，2003；Cormack，2008），该指标一般用于流行病学和医学中。垃圾邮件专题也介绍了一种累加综合指标，该指标是软分类，相当于微平均。

TREC 1 至 TREC 11（1992～2002）中包含了各种路由和过滤任务（Robertson 和 Callan，2005）。TREC 14 至 TREC 16（2005～2007）包括了垃圾邮件专题。TREC 会议上，路由任务就是批过滤：前一年的特定任务（包括文档、主题和 qrels（是指一个查询主题集合及其相应的答案——译者注））就是历史样本，要求系统根据相关性对另外一个新的文档集进行排名。对得到的结果采用标准排名检索指标进行评价。TREC 会议上的过滤，要求系统返回一个无序的结果集，并对这个无序结果集采用多种基于集合的指标进行评价，这些指标包括线性实用性和 F_β。排名检索自己并不会对这些指标进行优化；同时还需要采用一种称为门限（阈值）设置的方法。批过滤使用的历史样本和新文档跟路由中使用的一样。自适应过滤是在线过滤，这里，只有那些在分类后被标记为相关的文档才用作训练样本。因为可作为校对的训练样本很少（Callan，1998；Robertson，2002），所以自适应过滤优化门限设置是很困难的。Sculley（2007）在在线垃圾信息过滤的范畴内解决了类似的问题。垃圾邮件专题上评价了使用本章介绍的方法构建的过滤器。这些任务的延伸任务考察不完整反馈、延时反馈、内部用户反馈以及主动学习带来的影响。Cormack（2008）给出了垃圾邮件专题、相关方法和结果的研究结果。

10.10　练习

练习 10.1　下载 TREC 垃圾过滤器评价工具包，并在样本语料库上运行几个示例过滤器。

练习 10.2　找出 TREC45 里来自《金融时报》的文档。在这些文档中找出更新了的 qrels。[⊖]用垃圾过滤器评价工具包构造一个适用于评价的测试语料库。

练习 10.3　下载一个或多个批过滤器。在你的一个语料库上进行特征工程，为你的分类器准备训练样本和测试样本的输入文件。评价得到的结果。注意，SVMlight、Liblinear、LibSVM 和 FEST 均采用一般的文件格式。像 Weka 和 R 这些系统，它们都实现了很多方法。

练习 10.4　从至少两个不同版本的维基百科上随机抓取一些网页，以构成一个语料库，利用这个语料库去评价各种分类方法。

练习 10.5　使用搜索引擎为文本分类和过滤找出一个或多个经典语料库。例如，近年来用于路由、过滤和垃圾信息任务的路透社新闻、20 Newsgroups、Spambase、Ling Spam 和 TREC 文档集。找出使用这些语料库学习得到的结果。由这些结果，你能知道哪个方法的效果最好吗？你能重复实验，也得到相同的结果吗？当你在你的语料库上测试这些方法的性能时，该怎样比较同一个方法得到的结果？

练习 10.6　实现一个过滤器并对其进行评价。你实现的过滤器不需要太复杂：如果使用的是二元特征，采用朴素贝叶斯、logistic 回归或感知器这些方法会非常简单。实际上，任何数据压缩方法都可能作为一个过滤器使用：把输出长度作为对交叉熵的估计。DMC 以及其他的序列方法的实现可以从 Web 上下载。建立模型的代码可以从这些实现中找出来，就可以省略掉编码的工作。Sculley 的在线 SVM 可以从 Web 上下载。

练习 10.7　在主要的高速公路上，70% 的车会超速。而在这些高速公路上 80% 的两车相撞交通事故中，至少有一台车是超速的。当超速驾驶时，发生事故的比值比是多少？

练习 10.8　50% 的两车相撞事故中，有一辆车能停下来。如果要计算因超速驾驶导致事故发生与在限速范围内驾驶但发生了事故的比值比，还需要什么额外的信息呢？

⊖ trec.nist.gov/data/qrels-eng/qrels.trec8.adhoc.parts1-5.tar.gz

练习 10.9 假设你有经过某一点的每一辆车的速度和车牌的完整日志，还有在经过这点后一小时内发生事故的车牌信息。如何估计某辆车在经过这点后一小时内发生事故的概率？

练习 10.10 高速公路巡逻队确定一辆车牌由“Q”打头的红色小车超速了，但没能抓住它。在法庭上，控方专家证实所有车辆中，有$\frac{1}{36}$的车牌是以“Q”开头的，有$\frac{1}{20}$的小车是红色的，有$\frac{1}{1400}$的小车速度足够快，能逃脱警方的追捕。根据车辆注册数据库，我们发现 Mr.X 拥有一辆红色的法拉利。Mr.X 因为逃避警方而被起诉，并且控方指出 Mr.X 无罪的概率是百万分之一，因为$\frac{1}{20}\cdot\frac{1}{36}\cdot\frac{1}{1400}=\frac{1}{1\ 008\ 000}$。现在你被雇用为辩方专家证人。你觉得你有可能赚到你的证人费吗？

10.11 参考文献

Belkin, N.J., and Croft, W.B. (1992). Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 35(12):29–38.

Bratko, A., Cormack, G.V., Filipič, B., Lynam, T.R., and Zupan, B. (2006). Spam filtering using statistical data compression models. *Journal of Machine Learning Research*, 7:2673–2698.

Callan, J. (1998). Learning while filtering documents. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 224–231. Melbourne, Australia.

Cleary, J.G., and Witten, I.H. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402.

Cormack, G.V. (2007). TREC 2007 Spam Track overview. In *Proceedings of the 16th Text REtrieval Conference*. Gaithersburg, Maryland.

Cormack, G.V. (2008). Email spam filtering: A systematic review. *Foundations and Trends in Information Retrieval*, 1(4):335–455.

Cormack, G.V., and Horspool, R.N.S. (1987). Data compression using dynamic Markov modelling. *The Computer Journal*, 30(6):541–550.

Cormack, G.V., and Lynam, T.R. (2005). TREC 2005 Spam Track overview. In *Proceedings of the 14th Text REtrieval Conference*. Gaithersburg, Maryland.

Domingos, P., and Pazzani, M.J. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130.

Drucker, H., Wu, D., and Vapnik, V.N. (1999). Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874.

Glas, A.S., Lijmer, J.G., Prins, M.H., Bonsel, G.J., and Bossuyt, P.M.M. (2003). The diagnostic odds ratio: A single indicator of test performance. *Journal of Clinical Epidemiology*, 56(11):1129–1135.

Hastie, T., Tibshirani, R., and Friedman, J.H. (2009). *The Elements of Statistical Learning* (2nd ed.). Berlin, Germany: Springer.

Hosmer, D.W., and Lemeshow, S. (2000). *Applied Logistic Regression* (2nd ed.). New York: Wiley-Interscience.

Joachims, T. (2002). *Learning to Classify Text Using Support Vector Machines*. Norwell, Massachusetts: Kluwer Academic.

Komarek, P., and Moore, A. (2003). Fast robust logistic regression for large sparse datasets with binary outputs. In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*. Key West, Florida.

Lewis, D.D. (1991). Evaluating text categorization. In *Human Language Technologies Conference: Proceedings of the Workshop on Speech and Natural Language*, pages 312–318. Pacific Grove, California.

Lewis, D. D., and Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 148–156.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397.

McNamee, P. (2005). Language identification: A solved problem suitable for undergraduate instruction. *Journal of Computing Sciences in Colleges*, 20(3):94–101.

Mitchell, T. M. (1997). *Machine Learning*. Boston, Massachusetts: WCB/McGraw-Hill.

Robertson, S. (2002). Threshold setting and performance optimization in adaptive filtering. *Information Retrieval*, 5(2-3):239–256.

Robertson, S., and Callan, J. (2005). Routing and filtering. In Voorhees, E. M., and Harman, D. K., editors, *TREC — Experiment and Evaluation in Information Retrieval*, chapter 5, pages 99–122. Cambridge, Massachusetts: MIT Press.

Rocchio, J. J. (1971). Relevance feedback in information retrieval. In Salton, G., editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 14, pages 313–323: Prentice-Hall.

Sculley, D. (2007). Practical learning from one-sided feedback. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 609–618. San Jose, California.

Sculley, D., and Wachman, G. M. (2007). Relaxed online support vector machines for spam filtering. In *Proceedings of the 30th ACM SIGIR Conference on Research and Development on Information Retrieval*, pages 415–422. Amsterdam, The Netherlands.

Sculley, D., Wachman, G. M., and Brodley, C. E. (2006). Spam classification with on-line linear classifiers and inexact string matching features. In *Proceedings of the 15th Text REtrieval Conference*. Gaithersburg, Maryland.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47.

Siefkes, C., Assis, F., Chhabra, S., and Yerazunis, W. S. (2004). Combining winnow and orthogonal sparse bigrams for incremental spam filtering. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 410–421. Pisa, Italy.

Swets, J. A. (1963). Information retrieval systems. *Science*, 141(357):245–250.

Swets, J. A. (1969). Effectiveness of information retrieval systems. *American Documentation*, 20:72–89.

van Rijsbergen, C. J. (1979). *Information Retrieval* (2nd ed.). London, England: Butterworths.

Willems, F. M. J., Shtarkov, Y. M., and Tjalkens, T. J. (1995). The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, 41:653–664.

Witten, I. H., and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques* (2nd ed.). San Francisco, California: Morgan Kaufmann.

第11章

Information Retrieval: Implementing and Evaluating Search Engines

融合和元学习

在上一章，我们介绍了多种关于检索、分类和过滤的主要方法。这些方法通常带有参数——如BM25的 k_1、v_{title}、v_{body}、b_{title}和 b_{body}——这些参数都有它们各自的作用。如果我们考虑这些主要方法所有可能设置的参数集，同时考虑其他可能的设计选择，如分词和特征选择，我们将进入一个接近于无穷大的方法空间中，不知道该选择哪种方法。

巨大的方法空间导致出现一个明显的问题：哪些选择组合起来工作效果最好？当然，如果我们已经知道了这个问题的答案，那我们肯定会选择这些方法并将它们组合起来，以得到最好的方法，并忽略其他的方法。你可能会很惊讶地发现，通常可以把多种结果组合起来以提高单个最好方法的结果，而不需要确定哪个方法是最好的。

我们即将要介绍的具体方法如下：

- **融合**（fusion）或**合成**（aggregation），该方法把多个信息检索方法的返回结果组合成一个（参见11.1节）。
- **叠加**（stacking），该方法通过学习，得到多个分类器的最佳组合方式，构成一个元分类器。11.2节将介绍用于自适应过滤的叠加，而在11.3节将介绍适用于非自适应过滤和分类的叠加。
- bagging **或** bootstrap **合成法**（bootstrap aggregation），该方法随机抽取训练样本，得到一个分类器集合，最后取所有结果的平均值（参见11.4节）。
- boosting，通过逐步增加训练样本的权重，以突出那些被错误分类的样本，因此得到一个分类器集合，最后取所有结果的加权平均（参见11.5节）。
- **多类排名**和**分类**（multicategory ranking and categorization），该方法把多个二元分类器的结果组合起来（参见11.6节）。
- **学习排名**（learning to rank），该方法的评分函数是从各个不同的样本排名学习而来的（参见11.7节）。

在介绍各个具体的例子前，我们先给出一些常识。这里介绍的所有方法，都是把各个方法的已知证据合成起来，以得到更强的证据来符合用户的信息需求。尽管某项证据比其他的证据要更严格，通过合理地把这些证据组合起来，把它们看做一个整体，一般来说要比原来任何一项证据都严格。这种现象并不是信息检索中独有的，一般我们称这种现象为**大众智慧**（wisdom of crowds）（Surowiecki，2004）。

我们考虑的证据采用以下3种通用形式中的一种：

1）**类别**（categorical）。它是一个离散的结果，表明一个文档或一组文档集相关于或属于某个特定类别。判断一个类别结果的问题一般称为**分类**（classification）。

2）**序数**（ordinal）。一个排名或得分，根据文档相关于或属于某个类别的证据权重对文档排序后得到。决定一个序数的问题一般称为**排名**（ranking）。

3）**数量**（quantitative）。一个校准得分，通常是一个概率，表明一个文档相关于或属于某个类别的证据权重。估计这个校准得分的问题一般称为**回归**（regression）。

通过使用排名，回归可以化简为排名问题；通过使用部分排名或阈值，排名问题可以化简为分类问题。但每一步都丢失了信息。有时方法本身就包含了简化，因此丢失信息无法避

免。信息丢失量越少，所有证据的权重组合起来的效果就越好。

为了解释组合证据的各种方法，我们考虑前几章提出的问题：排名检索、面向主题的过滤、语言分类和在线垃圾信息过滤。对于每个问题，我们都使用前面已经为各种目的评价过的各种单个方法的结果作为输入，将集成后的结果和单个最好方法的结果进行比较。另外，我们还将使用 LETOR 3 数据集[⊖]说明学习排名这一方法，这是一个由若干个 TREC 语料库组成的基准测试集。

11.1　搜索结果融合

最简单的证据组合方法是**搜索结果融合**（search-result fusion），即把多个搜索引擎返回的文档列表组合成一个。除了文档列表，没有任何其他的信息，在这里，可能会使用列表里的文档排名或得分。我们将考察该方法里的 3 个变量：1）固定临界值检索，即从每个列表中抽出固定数量的文档，而不考虑排名；2）排名合成，即只考虑每个列表内每个文档的排名；3）得分合成，即只考虑每个列表内每个文档的排名和得分。

为了说明搜索结果融合，我们引入以下概念。假设 n 个检索系统响应查询 q 从文档集 D 中返回文档列表 Res_1、Res_2、$\cdots Res_n$。我们希望把这 n 个列表的证据合成起来，得到一个更好的排名列表 Res。为每个 Res_i 定义一个评分函数 $r_i(d)$，表示文档 d 在 Res_i 中的位置，如果 d 不在中 Res_i，则返回一个很大的值 c：

$$r_i(d)=\begin{cases} k & (Res_i[k]=d) \\ c & (d \notin Res_i) \end{cases} \tag{11-1}$$

只要满足 $c>|Res_i|$，c 可取任意值。c 的合理取值为 $c=\infty$、$c=|D|$、$c=\frac{|D|}{2}$ 和 $c=|Res_i|+1$。对于本书的例子，我们使用 $c=\infty$。

对于得分合成，我们假设每个系统返回一个非递增的得分列表 $Score_i$，其中 $Score_i[k]$ 是与 $Res_i[k]$有关的相关性得分。对每个 $Score_i$ 定义一个评分函数 $si(d)$，表示文档 d 在 Rel_i 中的得分，如果文档 d 不在 Rel_i 中，则返回其中最小的一个分数值：

$$s_i(d)=\begin{cases} Score_i[k] & (Res_i[k]=d) \\ Score_i[|Score_i|] & (d \notin Res_i) \end{cases} \tag{11-2}$$

我们一般会假设已经对 $Score_i$ 进行了归一化，以便

$$Score_i[|Score_i|]=\min_d\{s_i(d)\}=0, \qquad Score_i[1]=\max_d\{s_i(d)\}=1 \tag{11-3}$$

我们应用了三种方法把前几章提到的 $n=29$ 种不同的检索方法的结果进行了结果融合，如表 11-1 所示。对于每一个方法，表里都给出了 4 个 TREC 文档集上的 P@10 和 MAP 结果，最好的结果用黑体字显示。

⊖ research.microsoft.com/en-us/um/beijing/projects/letor

表 11-1 表中列出了 29 个用于本章中介绍的融合和学习排名的例子的独立运行结果。黑体的数字表示该结果是对应列中最好的

方法	停词	词干	TREC45				GOV2			
			1998		1999		2004		2005	
			P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
bm25	No	No	0.424	0.178	0.440	0.205	0.471	0.242	0.534	0.277
bm25	No	Yes	0.440	0.199	**0.464**	0.247	**0.500**	0.266	0.600	0.334
bm25	Yes	No	0.424	0.178	0.438	0.205	0.467	0.243	0.538	0.276
bm25	Yes	Yes	0.440	0.199	**0.464**	0.247	**0.500**	0.266	0.592	0.333
bm25_b=0	No	No	0.402	0.177	0.406	0.207	0.418	0.171	0.538	0.207
bm25_notf_b=0	No	No	0.256	0.141	0.224	0.147	0.069	0.050	0.106	0.083
dfr-gb2	No	No	0.426	0.183	0.446	0.216	0.465	0.248	0.550	0.269
dfr-gb2	No	Yes	0.448	**0.204**	0.458	0.253	0.471	0.252	0.584	0.319
dfr-gb2	Yes	No	0.426	0.183	0.446	0.216	0.465	0.248	0.550	0.269
dfr-gb2	Yes	Yes	0.448	**0.204**	0.458	0.253	0.471	0.252	0.584	0.319
lm-dirichlet-1000	No	No	0.450	0.193	0.428	0.226	0.484	0.244	0.580	0.293
lm-dirichlet-1000	No	Yes	**0.464**	**0.204**	0.434	0.262	0.492	0.270	0.600	**0.343**
lm-dirichlet-1000	Yes	No	0.448	0.193	0.430	0.226	0.494	0.247	0.568	0.291
lm-dirichlet-1000	Yes	Yes	0.462	**0.204**	0.436	**0.262**	0.488	**0.272**	**0.602**	0.341
lm-jelinek-0.5	No	No	0.390	0.179	0.432	0.208	0.416	0.211	0.494	0.257
lm-jelinek-0.5	No	Yes	0.406	0.192	0.434	0.248	0.437	0.225	0.522	0.302
lm-jelinek-0.5	Yes	No	0.390	0.179	0.432	0.209	0.414	0.212	0.482	0.253
lm-jelinek-0.5	Yes	Yes	0.406	0.192	0.436	0.249	0.445	0.225	0.508	0.298
lm-unsmoothed	No	No	0.354	0.114	0.396	0.141	0.402	0.171	0.492	0.231
lm-unsmoothed	No	Yes	0.384	0.134	0.416	0.180	0.433	0.196	0.538	0.285
lm-unsmoothed	Yes	No	0.352	0.114	0.394	0.141	0.400	0.172	0.484	0.230
lm-unsmoothed	Yes	Yes	0.384	0.134	0.416	0.180	0.439	0.196	0.518	0.283
prox	No	No	0.396	0.124	0.370	0.146	0.424	0.173	0.560	0.230
prox	No	Yes	0.418	0.139	0.430	0.184	0.453	0.207	0.576	0.283
prox	Yes	No	0.396	0.123	0.370	0.146	0.422	0.173	0.546	0.232
prox	Yes	Yes	0.416	0.139	0.430	0.184	0.447	0.204	0.556	0.282
vsm-lintf-logidf	No	No	0.266	0.106	0.240	0.120	0.298	0.092	0.282	0.097
vsm-logtf-logidf	No	No	0.264	0.126	0.252	0.135	0.120	0.060	0.194	0.092
vsm-logtf-noidf-raw	No	No	0.342	0.132	0.328	0.154	0.400	0.144	0.466	0.151

11.1.1 固定临界值合成

考虑把表 11-1 的结果组合起来以优化 P@10。现在，我们考虑对每一个结果进行分类，每个主题由 10 个无特定序的文档组成。对于 29 个样例方法，29 个结果集合的并集中，每个主题将包括 10～290 个文档，从中我们希望确认 10 个最相关的文档，也就是最大化P@10。

假设最直接的方法就是用“最好”的检索系统返回的 10 个文档作为最相关的文档。但是，哪个检索系统才是最好的呢？也许，基于前面章节的评价结果，我们可以猜想 BM25 或 LMD 是最好的。但是，并没有一个方法或一个参数集对这 4 个语料库同时得到最好的结果。读者可像查询 oracle 一样查询表，从而得到比较后的理想结果：单个最好的结果。请注意

“单个最好的结果”是选择一个结果能达到的最佳上界，在现实中是不太可能实现的上界。

一个简单而又实用的方法就是采用“选举”的策略，从各种方法得到的“候选”文档中选出 10 个文档。选举策略大同小异，但现在我们简单地选出在 29 个结果集中出现得最频繁的 10 个，从而打破每个候选文档都一样的僵局。实际上我们把若干个分类结果合并为一个排名结果，然后应用一个临界值（10 个文档）来得到一个新的分类结果。“选举”策略的结果如表 11-2 所示。对于每个语料库我们都使用了简单的投票方法，但都没有等于或超过理想的最好结果。

表 11-2　本书中讨论的检索方法在 TREC 文档集上的 P@10 分类融合结果

方法	TREC45		GOV2	
	1998 P@10	1999 P@10	2004 P@10	2005 P@10
最好	0.464	0.464	0.500	0.602
一般	0.406	0.430	0.445	0.538
选举	0.452	0.460	0.492	0.554

11.1.2　排名和得分合成

现在考虑对上述 4 个文档集上得到的 MAP 进行优化的问题。每个结果 Res_i 都是一个列表，从中可以得到排名 r_i。将这些排名组合起来，得到一个合成排名 r。该方法称为**排名合成**（rank aggregation）、**融合**（fusion）或**元搜索**（metasearch）。

排名倒数融合（reciprocal rank fusion，RRF）（Cormack 等人，2009）也许是最简单有效的排名合成策略了。RRF 根据一个朴素的得分公式对文档排序

$$\text{RRFscore}(d)=\sum_i \frac{1}{k+r_i(d)} \tag{11-4}$$

对于一般的检索结果，使用 $k=60$ 得到的效果最好。选择这条公式的思想是：尽管排名越高的文档越重要，但也不能忽略排名靠后的文档的重要性。公式中的调和级数具有这一性质。通过强盗方法（rogue method），常数 k 减少了高排名所产生的影响。

考虑更多的方法（却不一定是更有效的方法）基于某种选举理论，在该理论中，选举者只表达他的意愿（喜好），而不是直接选出某些类别。Condorcet 融合（Montague 和 Aslam，2002）根据成对关系 $r(d_1)<r(d_2)$来对文档进行排序，以合并排名，其中每对 (d_1, d_2) 的关系由输入排名中的多数投票决定。

其他不是纯排名合成的方法会使用到得分（数量的，尽管是任意核准的证据）而不是排名。例如，CombMNZ（Lee，1997）根据得分公式排序文档

$$\text{CMNZscore}(d)=|\{i \mid d \in Res_i\}| \cdot \sum_i s_i(d) \tag{11-5}$$

也就是说，CMNZscore 把文档出现的结果集的个数与它的得分和相乘，从而得到一个联合得分用于排名。表 11-3 列出使用这 3 种方法融合 29 种信息检索方法的结果得到的融合结果。对于 4 个语料库中的 3 个，采用 RRF 可以得到最好的 MAP 得分。采用 Condorcet 也能提高 MAP 得分，但当使用的间隔太大时，Condorcet 则不能起到提高 MAP 得分的作用了。尽管使用 CombMNZ 得到的 MAP 得分接近于最好的 MAP 得分，但却不能大大地优于它。为了与前面的结果进行对比，我们还给出 P@10，P@10 是使用排名组合得到的，其中临界值取 10。采用 RRF 得到的 P@10 得分在所有情况下都比采用投票策略得到的 P@10 得分要

高，同时接近于最好的 P@10 得分。

表 11-3 本书中讨论的检索方法在 TREC 文档集上的基于排名的融合结果

方法	TREC45				GOV2			
	1998		1999		2004		2005	
	P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
Best (by MAP)	0.462	0.204	0.434	0.262	0.500	0.272	0.602	0.341
Best (by P@10)	0.464	0.204	0.464	0.247	0.500	0.266	0.602	0.341
RRF (k=60)	0.462	0.215	0.464	0.252	0.543	0.297	0.570	0.352
Condorcet	0.446	0.207	0.462	0.234	0.525	0.281	0.574	0.325
CombMNZ	0.448	0.201	0.448	0.245	0.561	0.270	0.570	0.318

总的来说，排名和得分集成这两种方法都是很有价值的，因为采用这两种方法通常能达到或高于最好排名，而这个过程不需要知道各个独立结果间的相互关系。因此，这两种方法可以与本章中介绍的其他方法结合起来使用，如 bootstrap 集成法（参见 11.4 节）。

11.2 叠加自适应分类器

现在考虑把在线垃圾信息过滤器的分类或数量结果（参见表 10-25）组合起来的问题。分类结果只是简单地指明了每个信息是垃圾信息还是正常信息而已。我们现在简单地使用多数表决法将这些结果组合起来。因为有奇数个过滤器，所以我们不必担心平局的情况。实际上，通过与一个阈值 t 进行比较，投票得到的排名可以转化为一个分类结果。现在假设 5 个结果中的 t 个需要标记为垃圾信息。多数表决法与单个最优过滤器（参见表 11-4）一样好。在实际中，采用 t=4 或 t=5 可能会得到更符合预期目的——过滤掉垃圾信息的满意结果。

表 11-4 通过投票法将在线垃圾信息过滤器的结果组合起来

方法	分类错误（%）		
	假阳率	假阴率	Logistic 平均
最好	0.32	0.42	0.37
多数表决法	0.42	0.34	0.37
1 of 5	32.88	0.00	0.61
2 of 5	1.54	0.02	0.54
3 of 5	0.42	0.34	0.37
4 of 5	0.19	0.61	0.34
5 of 5	0.08	1.50	0.36

排名合成并不适用于在线过滤器，因为该方法要求对得分进行排序。取而代之的是，我们会校准文档 d_i 的得分 s_i，并在已知 d_i 是垃圾信息时，使用历史样本去逼近 log-odds：

$$\text{logOdds}[d_i \in spam] \approx \log \frac{|\{d_{j<i} \in spam \mid s(d_j) \leqslant s(d_i)\}|}{|\{d_{j<i} \notin spam \mid s(d_j) \geqslant s(d_i)\}|} \tag{11-6}$$

这些得分可以使用与概率分类器的特征值一样的方法求平均。使用（元）学习方法把结果组合起来的方法称为**叠加**（stacking）。对于 log-odds 平均，元学习法相当于朴素贝叶斯（参见 10.3.2 节）。也可使用其他自适应的元学习法替代，如梯度下降 logistic 回归。

表 11-5 列出了在运用朴素贝叶斯和 logistic 回归元分类器进行 log-odds 变换后的结果上进行叠加后得到的垃圾信息过滤结果。无论是使用朴素贝叶斯还是 logistic 回归，都比最好的单个过滤器有了很大的提高。

表 11-5　通过 log-odds 平均和回归将在线垃圾信息过滤器的结果组合起来

方法	分类错误（%）			
	假阳率	假阴率	Logistic 平均	1-AUC
最好	0.41	0.47	0.44	0.012
多数表决法	0.42	0.34	0.37	0.095
NB 叠加	0.37	0.34	0.36	0.008
LR 叠加	0.39	0.27	0.33	0.006

现在考虑表 10-23 中列出的面向主题过滤的结果。由于自适应训练结果已经给出了，所以可以直接对表中的 4 种方法采用自适应叠加法。实际上，这些方法都是在整个数据集上按序进行训练的，但表中只给出了它们在测试样本上得到的结果。对叠加垃圾信息过滤器采用同样地方法：把整个数据集的得分按递增方式转换成 log-odds，并采用自适应 logistic 回归对这些得分求平均或组合起来。只给出在测试样本上得到的结果。由表 11-6 可知，叠加的效果与垃圾信息过滤的效果是一样的：这两种方法都比最好的结果有了很大的提高，如果采用 logistic 回归，效果就更显著了。只有一种方法得到的组合结果要比最好的结果差很多。尽管如此，从这些结果我们还是可以看出，只要合理使用叠加，得到的结果都比最好的结果要好。

表 11-6　使用自适应训练，在 TREC 主题 383 上得到的叠加结果。表中的结果可与表 10-23 中“自适应训练”的结果进行比较

方法	历史样本＋在线	
	P@10	AP
单个最好的方法	1.0	0.55
NB 叠加	1.0	0.59
LR 叠加	1.0	0.60

11.3　叠加批过滤器

直接对批训练过滤器返回的结果进行叠加是有问题的，因为很难将这些过滤器返回的得分调校成 log-odds 或其他有意义的数量形式。即使使用如 logistic 回归这样的方法可以直接估计 log-odds，但这也只能在训练集 T 上做。由于过度拟合，不能直接将这些估计推广到测试集上。如果采用如 SVM 和决策树这些方法，情况将更糟糕：采用这些方法，一般不能在训练样本上得到最好的得分；这些得分不能用于测试样本中。

为了解决这个问题，基于训练文档 $d_i \in T$ 的得分，我们采用公式（11-6）去调校文档 $d_i \in D$ 的得分 s_i：

$$\text{logOdds}[d_i \in spam] \approx \log \frac{|\{d_j \in (T \cap spam) \mid s(d_j) \leqslant s(d_i)\}|}{|\{d_j \in (T \backslash spam) \mid s(d_j) \geqslant s(d_i)\}|} \tag{11-7}$$

表 11-7 列出了使用表 10-23 中 9 种方法得到的叠加批训练结果。总体效果与前面的例子相似：朴素贝叶斯叠加能大大地提高效果；采用（批）logistic 回归提高的效果更明显。

表 11-7 使用批训练，在 TREC 主题 383 上得到的叠加结果。表中的结果可与表 10-23 中的“历史训练”结果进行比较

方法	历史训练	
	P@10	AP
单个最好的方法	1.0	0.56
NB 叠加	1.0	0.62
LR 叠加	1.0	0.65

11.3.1 holdout 验证

从训练结果估计 log-odds 就是一个回归的例子——从有效数据中对一个数量结果进行估计。特别的，我们将各种方法的分类有效性作为一个评分函数 $s(d)$，从而估计各个方法的性能。这些估计可以使我们可以得到一种更好的组合排名方法（或分类器）。

然而，前一节中使用的从 $s(d)(d \in T)$ 中估计 log-odds 的技术会受泛化误差的影响。通过使用从一个独立的验证集 $V(V \subset D$ 且 $T \cap V = \emptyset)$ 中得到的 $s(d)$ 来计算 log-odds（或任何其他回归量化），可以避免这个泛化误差。也就是，在 T 上训练该方法，并用它来评分（或分类，或排名）V。

构造 V 的一种最直接的方法是 holdout 法：将大的已标记样本集合 L 进行划分，从 D 中采样，使得 $L = T + V$。假设 L 是 D 的一个独立同分布样本（i. i. d.），T 是 L 的一个随机子集，因此 T 和 V 均认为是 D 的 i. i. d. 样本。因此 log-odds 的计算使用一个与 D 独立的样本，不会受到分类器对 T 过度拟合的影响。（然而，回归可能会对 V 过度拟合，但这是两回事。）V 一般称为**验证集**（validation set），在当前这种特殊背景下，称为 holdout **集**（holdout set）。该方法称为 **holdout 验证**（holdout validation）。

holdout 验证的主要局限是缺少标记样本。L 的大小一般受限于标记开销和可用性，因此，集合 T 的大小与集合 V 的大小会直接互相影响。一般来说，T 越大，得到的基方法越好，而 V 越大，将得到更好的验证。某一方的提高都需要以另一方的降低作为代价。

第二个局限是正例样本在 T 和 V 中的普遍度可能不同。一些方法，特别是依赖于阈值设置的分类方法，对普遍度相当敏感。可以通过使用**分层抽样**（stratified sampling）技术，从 L 中构造出 T（V 也是一样），减轻因普遍度不同而带来的影响。在分层抽样中，对 L 中每个类别里的样本进行单独抽样，以便 T（V 也是一样）中各个类别的比例与 L 大致相同。

11.3.2 交叉验证

交叉验证（cross-validation）也是一种回归方法，该方法将已被标记的样本集 L 分成 k **块**（split）(T_1, V_1)、(T_2, V_2)、…、(T_k, V_k)。在每个训练集 T_i 上对学习方法单独进行训练，并把得到的结果用于为文档 $d \in V_i$ 进行评分。在组合结果上使用回归方法，把所有验证集看成一个大的验证集 $V = \bigcup_{i=1}^{k} V_k$。最常见的交叉验证法是 k **折交叉验证**（k-fold cross-validation），该方法中，各个验证集的大小都一样，且它们之间没有交集：$\forall_{i \neq j} V_i \cap V_j = \emptyset$。另外，每个训练集中的样本不会出现在验证集中：$T_i = L \setminus V_i$。当 k 很大时，有 $T_i \approx L$ 和 $V = L$。因此，V 是对同等大小的独立的 holdout 集的一个合理近似。$k = |L|$ 这一极端情况称为 leave-one-out 验证。

如果使用分层抽样，k 的取值将受普遍度最低的类别的影响。在基于主题过滤的例子中，T 包含 22 个相关文档，因此，我们使用 22 折交叉验证，使每个 V_i 中包含一个相关文

档。当不受分层限制时，k的取值可能受效率问题制约，因为训练开销与k是成正比的。5 折交叉验证（$k=5$）和 10 折交叉验证（$k=10$）是比较常见的。表 11-8 列出了使用 22 折交叉验证后得到的结果，由这些结果可见，该方法大大提高了平均融合与叠加融合的结果（数据采用主题 383）。

表 11-8　使用 22 折交叉验证，在 TREC 主题 383 上得到的叠加结果

方法	历史训练	
	P@10	AP
单个最好的方法	1.0	0.56
NB 叠加	1.0	0.64
LR 叠加	1.0	0.69

交叉验证另外一个重要的作用就是**选择**（selection）或**调和**（tuning）学习方法。在这里，交叉验证用于估计某些总体有效性评价指标，并搜索方法空间和参数集来优化这些指标。正如本章开始所提到的，并且我们已经通过例子证明：选择单个最好的方法或参数集并不能获得最好的总体结果。

11.4　bagging

如果在某个特定的训练集 T 上对学习方法进行训练，并将结果用于 D 中的文档，那得到的分类、排名或回归结果主要会出现以下两种误差：**训练误差**（training error）和**泛化误差**（generalization error）。如果只使用一个训练集 T，那么这两种误差都会下降，对同一个方法，分别在 N 个训练集 T_1、T_2、…、T_N 上进行训练，其中 T_i 是一个独立于 D 的样本，且与 T 的大小相等，对于得到的训练结果，我们将会对这些结果求平均值。

原因很简单。设 $ideal(d)$是由一个理想分类器对文档 $d \in D$ 进行分类时返回的得分。在训练集 T_i 上进行训练的某个特定方法会计算 $c_i(d) \approx ideal(d)$。更具体来说，$c_i(d) = ideal(d) + E_i$，其中 E_i 是一个随机变量，对在训练集上 T_i 进行训练时出现的误差进行建模。现在，假设以这些结果的平均值作为组合结果

$$c(d) = \frac{1}{N} \cdot \sum_{i=1}^{N} (ideal(d) + E_i) = ideal(d) + E \tag{11-8}$$

其中

$$E = \sum_{i=1}^{N} \frac{E_i}{N} \tag{11-9}$$

如果 E_i 两两独立，那么 E 的方差一般来说会比任何一个 E_i 的方差都要小。在 $\sigma_{E_i}^2$ 都相等这一特殊情况下，σ_E^2 要比 $\sigma_{E_i}^2$ 小 N 倍：

$$\sigma_E^2 \approx \frac{\sigma_{E_i}^2}{N} \tag{11-10}$$

如果 T_i 是与 D 独立同分布的样本，那么 E_i 是独立的且它们的方差相同，因此，使用该方法得到的标准误差 σ_E 要比只使用单个训练集 T_i 的训练误差小 $\sqrt{N}$ 倍。

一般来说，把标记样本集 L 分成多个独立的训练集再进行处理所获得的好处并不多，因为在一个大的集合 $T=L$ 上进行训练，效果是一样的（如果不比原来好）。我们会使用另外一种称为 bootstrap 的技术，使用该技术可以模拟出 N 个大小与 L 相同的集合 T_1、

T_2、…、T_N（参见 12.3.2 节）。bootstrap 技术使用 L 代替 D，独立且随机地从 L 中选出 T_i 的每个元素。因为是独立的选择每个元素，因此在 T_i 中的某个元素可能会重复出现，而某些元素可能完全没有出现过。对于 T_i，只要抽样大小（$N>20$）合理，就可以使用泊松公式很好地近似某个给定元素 $d \in L$ 出现 k 次的概率：

$$\Pr[|\{d \in T_i\}| = k] \approx \frac{1}{k! \cdot e} \tag{11-11}$$

即对于任何给定的 T_i，大约 $\frac{1}{e} \approx 36.8\%$ 的文档不会出现，大约 $\frac{1}{e} \approx 36.8\%$ 的文档会出现一次，大约 $\frac{1}{2e} \approx 18.4\%$ 的文档出现两次，以此类推。这些**引导样本**（bootstrap sample）有足够的独立性，可以使误差方差下降。给定的 L 要足够大，能对 D 中的各类文档提供一个好的表示，同时给定的学习方法要把重复的样本看做是独立的样本，只是这些独立的样本刚好具有同样地性质而已，且 bootstrap 训练集近似于内容提取自 D 的训练集（不是提取自 L）（Efron 和 Tibshirani，1993）。

bootstrap **合成**（bootstrap aggregation）或 bagging 技术平均了通过把引导样本看做训练集得到的结果。尽管 bagging 可用在任何学习方法中，但该方法最有效且最常用于**不稳定**（unstable）的学习方法中，这些学习方法得到的结果都高度依赖于某个特定的训练样本。表 11-9 列出了对 TREC 主题 383 采用决策树方法的 bootstrap 合成后的结果。决策树桩（$n=2$）的提高效果最明显。像叶子数为 $n=8$ 和 $n=256$ 这样更大的决策树，N 很小的时候得到的效果不太好，但当 $N \geqslant 32$ 时，得到的结果很好。不过，当 $N \geqslant 128$ 时，提高的效果已经不大了，因为这时得到的结果已经接近于最好的结果了。

表 11-9 对 TREC 主题 383 上决策树 bootstrap 集成法（bagging）结果。表中的数值是 AP，可与表 10-23 中的结果进行比较

方法	bootstrap 集成法采样的数量												
	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
DT (n=2)	0.38	0.22	0.56	0.37	0.52	0.50	0.51	0.51	0.51	0.52	0.52	0.53	0.52
DT (n=8)	0.21	0.33	0.47	0.55	0.59	0.58	0.57	0.58	0.57	0.59	0.58	0.59	0.58
DT (n=256)	0.13	0.44	0.45	0.47	0.55	0.54	0.56	0.55	0.55	0.55	0.55	0.55	0.54

11.5 boosting

boosting 与 bagging 类似，该方法扰动了训练样本，以得到一种**组合**（ensemble）方法，把这些结果组合起来。与 bagging 随机组织训练数据以最小化泛化误差不同，boosting 系统地组织训练数据以最小化训练误差。尽管这两种方法的基本假设有很大的不同，但是它们的最终结果都是得到一个更好的分类器。

boosting 与所有的监督式学习方法类似，整个训练集 T 是从 D 中随机抽样后得到的。但我们不会直接把 T 用于训练；而是对每个 $d \in T$ 赋予一个权重 $w_d \in \mathbb{R}$，使 $1=\sum_d wd$。基学习方法假设：文档的普遍度 ρ_d 与 d 在 D 中的普遍度一样，为 $\rho_d = w_d$，这与原来建议的 T 中的比例不同（即 $\rho_d \approx \frac{1}{|T|}$）。对于像决策树这样的基学习方法，在构建过滤器时，通过把各个文档 d 的贡献值 w_d 都相乘，就能很容易地满足这个假设了。

boosting 的处理过程就是计算 k 个权重向量 $w^{[1]}$、$w^{[2]}$、…、$w^{[k]}$，并通过利用 T 和这些权重向量，把分类器 $c^{[1]}$、$c^{[2]}$、…、$c^{[k]}$ 的学习关联起来。初始时，对于所有 $d \in T$ 有

$w_d^{[1]}=\frac{1}{|T|}$。接下来的步骤中，那些被分类器 $c^{[i]}(d)$错误分类的样本的权重 $w_d^{[i+1]}$ 会增大（相对于 $w_d^{[i]}$），并对其进行归一化，以满足 $1=\sum_d w_d^{[i+1]}$。最终的分类器由前面 k 个分类器的权重和组成：

$$c(d)=\sum_{i=1}^{k}\alpha_i c^{[i]}(d) \tag{11-12}$$

其中，权重 α_i 用于最小化整体的误差。计算 w 和 α 的方法不同，就构成了 boosting 方法系列。也许最著名的就是 AdaBoost：用于决策树。对 TREC 主题 383 采用该方法，得到的结果如表 11-10 所示。boosting 与 logistic 回归以及其他优化方法有很大的相似性（Schapire，2003）。

表 11-10　对 TREC 主题 383 上决策树的 Boosting（AdaBoost）结果。表中的数值是 AP，可与表 10-23 以及表 11-9 中的结果进行比较

方法	boosting 集成法的采样数量												
	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
DT (n=2)	0.19	0.31	0.38	0.52	0.48	0.50	0.49	0.53	0.54	0.59	0.60	0.61	0.59
DT (n=8)	0.21	0.22	0.27	0.41	0.54	0.59	0.60	0.63	0.61	0.62	0.64	0.64	0.63
DT (n=256)	0.60	0.51	0.54	0.54	0.53	0.49	0.55	0.57	0.56	0.56	0.57	0.57	0.57

11.6　多类排名和分类

对于多类排名和分类问题，我们需要一个得分 $s(d,q)$作为替代概率——估计一个文档 $d\in D$ 属于类别 $q\in Q$ 的概率（即 $\Pr[d\in q]$）。对于语言分类的例子（参见 10.1.4 节），类别就是 60 种语言，因此 $q_j\equiv l_j$。对于面向主题的搜索或过滤，类别就是各个主题，因此 $q_j\equiv t_j$。我们首先以语言分类的例子作为引入，然后再转到面向主题的搜索或过滤问题中。“作为替代概率”的意思是：

- 如果 $s(d,q)$是一个类别，那么它比概率的正确性要大。
- 如果 $s(d,q)$是一个序数，那么它暗含的排名与概率是正相关的。
- 如果 $s(d,q)$是一个数量，那么对于某些已知的函数 f

$$f(s(d,q))\approx\Pr[d\in q] \tag{11-13}$$

用得分替代概率并不是唯一一种多类排名方法。该方法只是我们到目前为止介绍的相关性排名和融合方法中关键的一种。这里的区别是：当 $q_j\neq q_k$ 时，假设 $s(d,q_j)$与 $s(d,q_k)$可进行比较：

$$s(d,q_j)<s(d,q_k)\Leftrightarrow\Pr[d\in q_j]<\Pr[d\in q_k] \tag{11-14}$$

即 $s(d,q)$可以实现在文档中对类别进行排名，而不仅仅是在某个类别中对文档排名。

在本节，为了实现多类排名和分类，我们将会介绍几种不同的方法来计算和合并评分函数系列 $\{s^{[f]}:D\times Q\to\mathbb{R}\}$。

11.6.1　文档得分与类别得分

在语言分类的例子中（参见 10.1.4 节），我们为每种语言 l 计算评分函数$s_l(d_i)$，该函数用于替代 $\Pr[d_i\in l]$。为了排名语言，需要某个特定的文档 d 包含 $s_d(l_j)$（用于替代 $\Pr[d\in l_j]$）。到目前为止，我们所使用的方法均简单地设 $s_{d_i}(l_j)=s_{l_j}(d_i)$。在这种选择下的假设是 $s(d_i,l_j)=s_{l_j}(d_i)$，它满足我们前面对 s 规定的标准，即对所有 d_i 和 l_j 的概率

$\Pr[d_i \in l_j]$，均能使用它代替，而不仅仅是某种语言 l。在这个假设下，一个文档 d 最可能使用的语言是

$$\arg\max_{l_j}\{s(d,l_j)\} \tag{11-15}$$

并且，d 可能使用的语言 l_j 或许已经根据 $s(d,l_j)$ 排名了。

对于 logistic 回归，这个假设有一个合理的解释，因为该方法直接估计概率（正如 log-odds 一样）。对于其他线性分类器或广义线性分类器，从某个分离超平面得到的标识距离通常是一个可以接受的得分（如果未经校正）。如最近邻法这种非线性的方法，一般会使用某些内部得分，这些得分也是可以接受的，如与最近邻居的邻近度或在 k 个最相近的邻居中 l_j 的普遍度。决策树将 D 分成多个离散集合：可以利用在 d 所属的离散集中 l_j 的普遍度来定义出一个得分。使用硬分类器，得到的估计不准确；使用 bagging、boosting 或叠加法求这些估计值的平均值，得到的是一个量化的估计。

如果 $s(d,l)$不是一个好的替代，那么得到的结果只是一个好的文档排名，但分类精确度将会很差。从表 10-24 文档排名（MAP）与文档分类的效果（1 -误检率或 MRR）之间的反差，可以看出这种现象。特别的，NB 得到的 MAP 值是 0.44，得到的 MRR 值是 0.8，虽然 DT（$n=8$）得到的 MAP 值（0.54）更高，但 MRR 值（0.64）却比 NB 得到的 MRR 值要小。另外，采用 DMC（配合 LR 一起使用）得到的 MRR（0.86）和误检率（20%）都是最好的，得到的 MAP（0.74）也不错。在上述假设下，无论是文档排名还是类别排名，采用 logistic 回归得到的效果都是最好的。

当考虑 $s(d,l)$的类别或序数值时，一个类别排名的定义不是唯一的。即参数最大这一定义可能是错误的，同时根据 $s(d,l_j)$进行排名，可能会出现平局（多个文档排名相同的情况——译者注）。在这种情况下，需要打破这些平局。如果平局出现不多，那么随机打破这些平局关系是可以接受的。或者，可以利用普遍度最高的类别来打破平局关系，该技术是用于优化微平均（不是宏平均）误检率的。更常见的做法是，通过第二个替代指标 $s^{[2]}(d,l)$来打破平局。对于采用普遍度这一特例，有 $s^{[2]}(d,l)=prev(l)$。

使用第二个替代指标其实是一个更一般化方法的特殊情况：将多个 $s^{[f]}$组合起来得到一个合成评分函数。这种方法的实例如下：

- **文档排名融合**（document rank fusion）。对于每种语言 l，把 k 种文档得分方法得到的结果 $s_l^{[1]}$、$s_l^{[2]}$、…、$s_l^{[k]}$ 组合成一个最终的得分 $s_d(l)$；这里设类别得分方法$s_d(l)=s_l(d)$。
- **类别排名融合**（category rank fusion）。对于每个文档 d，把 k 种类别排名方法得到的结果$s_d^{[1]}$、$s_d^{[2]}$、…、$s_d^{[k]}$ 组合成一个最终的得分$sd(l)$，用于对类别进行排名。
- **多类方法**（multicategory methods）。将文档 d 和语言 l 组合起来以计算$s(d,l)$。

11.6.2 文档排名融合与类别排名融合

表 11-11 给出了 5 种方法组合 9 种特定语言文档排名方法的结果，排名方法的结果如表 10-24 所示。为了进行比较，表中同时列出了使用单个最好的方法得到的结果（使用分类效果指标和文档排名效果指标进行度量）。RRF 一般用于面向主题的检索，使用该方法将 9 个不同的文档排名列表组合成一个。使用 RRF 得到的得分用于为每个文档所使用的语言进行排名。其中 3 个基本排名方法（NB、DMC 和梯度下降 LR）是在线方法，所以可以使用公式（11-6）估计 log-odds。另外 6 个是批方法，所以可以使用公式（11-7）估计 log-odds。

“NB 叠加”这一行的结果是通过求平均后得到的，“LR 叠加”这一行的结果是通过对训练样本使用 logistic 回归组合起来得到的。将 bagging 和 boosting 用于建立一棵节点数为 1024 的决策树（$n=8$）。其他文档排名方法对 bagging 和 boosting 得到的结果均没有作用。

表 11-11　对语言分类进行文档排名融合得到的结果。表中的数据可与表 10-24 中的数据进行比较

方法	双排名	分类	
	MAP	误检率（%）	MRR
类别排名	0.82	20.0	0.86
文档排名	0.83	20.0	0.80
RRF	0.83	18.9	0.87
NB 叠加	0.84	19.2	0.87
LR 叠加	0.82	18.6	0.87
bagging (DT; n=8, N=1024)	0.77	25.9	0.82
boosting (DT; n=8, N=1024)	0.75	35.7	0.69

表 11-12 列出了类别排名融合的结果，该方法基于单个文档将多个得分值组合起来。融合方法有点不同，因为没有把需要排名的语言分成训练集和测试集。因此关于如何将得分转换成概率估计的方法并不是很明显。我们直接对采用各种方法得到的得分使用 RRF、Condorcet 排名合成法和 logistic 回归（这些方法不需要用到估计）。RRF 的效果相对较差，然而 Condorcet 提高了最好的基本方法得到的效果。logistic 回归提高的效果会更大。其实 logisitc 回归是学习排名的一种特殊情况，这将会在 11.7 节中进行详细介绍。

表 11-12　对语言分类使用类别排名融合得到的结果

方法	分类	
	误检率（%）	MRR
排名倒数融合（RRF）	27.0	0.80
Condorcet	19.5	0.87
Logistic 回归（LR）	18.0	0.88

11.6.3　多类方法

二元分类、排名或回归等多种方法都适用于直接解决类别排名问题；我们一般称这些方法为**多类**（multicategory）。一般在文献中称这为**多项式的**（multinomial）或**多分类的**（polytomous），与其相对的是**二项式的**（binomial）或**二叉的**（dichotomous）。

1. 一对多

到目前为止，我们所介绍的方法都是首先解决文档排名问题，然后将这些排名结果组合起来去解决类别排名问题。这种方法一般称为**一对多法**（one versus rest），因为该方法将 n 个二元分类问题的结果组合起来，且每个二元分类问题都是把一个类别从剩下的类别中分离出来。到目前为止，我们已经假设一个软分类器的结果表示的是一个可信度，并从 n 个一对多的结果中选出可信度最高的类别。

一对多法对于二元硬分类器是有问题的。如果恰好某个分类器只有一个正例，也就刚好确定了它的分类。但是如果没有结果是正例，或多个结果都是正例，那就需要一些打破平局的策略了。

2. 一对一

一对一法（one versus one）将原来的问题简化为 $n(n-1)/2$ 个二元分类的问题，为任意两个类别设计一个二元分类器。以**选举**（election）的方式将这 $n(n-1)/2$ 结果组合起来，以为某个文档选取所属的类别或对类别进行排名。人们已经研究过大量的选举策略，但本书不会给出这些策略的综合讨论。最简单的方法也许就是为每个类别统计选票（即正例数），并根据这个统计值排名。但平局（统计值一样——译者注）的情况依然可能出现，不过只有在二元分类器的结果不一致的情况下才会出现。有多种**决胜选举**（runoff）策略可以用于解决这种平局的情况。在很多情况下，解决这种平局的情况可以通过对统计值相同的类别再次进行投票来解决。另外，也可以考虑使用软分类器的可信度。

3. 多类 logistic 回归

logistic 回归特别适合多类回归，从而也适用于排名。回想前面的（二项式）logistic 回归估计：

$$\text{logOdds}[x] = \log\left(\frac{\Pr[x]}{\Pr[\bar{x}]}\right) = \log\left(\frac{\Pr[x]}{1-\Pr[x]}\right) \tag{11-16}$$

其中 x 是我们感兴趣的某些二元属性，而 $\bar{x}$ 是它的补集。

在那个语言分类的例子中，我们对多类排名使用 logistic 回归，方法如下：对于每个类别 q_i（$1 \leqslant i \leqslant n$），用 x_i 表示文档 d 与其所属类别的关系 $d \in q_i$。使用 logistic 回归，单独为每个类别计算 $l_i \approx \text{logOdds}[x_i]$。使用这种方法进行排名也是合理的，因为 l_i 可以合理地表示文档 $d \in q_i$ 这一证据的强度。因此可以推导出一条概率估计公式：

$$p_i = \text{logit}^{-1}(l_i) = \frac{1}{1+e^{-l_i}} \approx \Pr[x_i] \tag{11-17}$$

但这个概率估计不适用于类别是排他的情况（正如它也不适用于那个语言分类的例子一样）。它不能满足所有概率的总和为 1 这一限制：

$$1 = \sum_{i=1}^{n} \Pr[x_i] \tag{11-18}$$

进行验证后再尝试满足这个限制条件得到的估计并不好。这里我们只提出两种建议，以避免这个问题。第一种建议是归一化 p_i 以使它们的和为 1，如公式（11-18）；第二种建议是除了一个 p_i，其余的均使用 logistic 回归进行估计，然后使用公式（11-18）把第 n 个解出来。

一种更好的方法是使用 logistic 回归去估计两个概率间的**比**（ratio）

$$r_{ij} \approx \frac{\Pr[x_i]}{\Pr[x_j]} \tag{11-19}$$

并计算 p_i 以使其满足公式（11-18）和公式（11-19）。这种方法只要求我们计算 $\frac{n(n-1)}{2}$ 个可能需要估计的 r_{ij} 中的（$n-1$）个就可以了；剩下的都是多余的。可任意选择计算哪个 r_{ij}，只要这些值能构成一个基，使所有的 r_{ij} 都能计算出来。

一种标准的选择就是**基准类别** logit **模型**（baseline category logit model）。在这个模型中，我们视 x_1 为基准，并为所有的 i 计算 r_{i1}。对于 $i=1$，由定义有

$$r_{11} = 1 \tag{11-20}$$

对于 $i>1$，当所有其他类别都被排除后，我们首先注意到 r_{ij} 重写为 x_i 与 x_j 的概率比是有可能的：

$$r_{ij} \approx \frac{\Pr[x_i]}{\Pr[x_j]} = \text{Odds}[x_i \mid x_i \text{ 或 } x_j] \tag{11-21}$$

因此，

$$r_{1j}=e^{l_{i1}}\ (i>1) \tag{11-22}$$

其中

$$l_{i1}\approx \log\text{Odds}[x_i\,|\,x_1 \text{ 或 } x_i] \tag{11-23}$$

也就是，通过 logistic 回归使用类别 q_1 或 q_j 训练文档 d 便可计算 l_{i1}，并不是使用剩下的文档：$d\in(q_1\cup q_j)\cap T$。对于 i，$j>1$，

$$r_{ij}=\frac{r_{i1}}{r_{11}}\cdot\frac{r_{11}}{r_{j1}}=\frac{r_{i1}}{r_{j1}}\ \ (i,j>1) \tag{11-24}$$

给定 $\{r_{i1}\}$ 时，计算 p_i 就很直接了：

$$p_i=\frac{r_{i1}}{\sum_{i=1}^{n}r_{i1}} \tag{11-25}$$

出于某些已经超出本书范围以外的原因，无论选择哪个类别 q_i 作为基准，p_i 的计算结果都是一样的（Agresti，2007）。

4. **多类 SVM**

回想起（二元）支持向量机就是要找到两个类别间的可分离超平面，从而找到间隔大小和被错误分类的点的数量（严重程度）的一个平衡。对于文档排名，一个多类 SVM（Crammer 和 Singer，2002）为其构造 n 个超平面，但只把所有的间隔和错误分类的点看成一个普通的优化问题。也就是，多类 SVM 在最大化 n 个最小的间隔与所有被错误分类的点的数量（严重程度）间找到一个平衡。因此，更直接的做法是解决分类错误，而不是使用 logistic 回归，因为 logistic 回归只关心在所有类别上的概率分布。

5. **多类 boosting**

相似的做法可用于 boosting。多类 boosting 针对所有类别优化训练误差（Schapire 和 Singer，2000），而不是单独为每个类别计算 w 和 α。

表 11-13 列出了一对多法和多类语言分类的比较结果，数据采用 TREC 主题 383。总的来说，很难从这些结果中总结出采用哪一种方法会得到好的效果。多类 SVM（SVM-Multiclass[⊖]）的效果会差一点，除了 $c=10\ 000$ 这一极端情况下，它的效果要稍微好于另外一种方法。多类 boosting（Boostexter[⊜]）的效果有明显的提高，但是这是在基于一个很差的基准的基础上。

表 11-13　一对多法与多类语言分类的结果比较

方法	一对多法		多类	
	误检率（%）	MRR	误检率（%）	MRR
LR	20.0	0.86		
SVM (C=0.01)	20.1	0.85	29.9	0.79
SVM (C=1)	20.0	0.86	31.8	0.74
SVM (C=100)	20.0	0.86	22.8	0.84
SVM (C=10 000)	20.0	0.86	19.7	0.86
Boosted stumps (N=1024)	46.4	0.63	32.2	0.76

⊖ svmlight.joachims.org/svm_multiclass.html

⊜ www.cs.princeton.edu/~schapire/boostexter.html

11.7 学习排名

学习排名（learning to rank）这个术语用于描述在对象集 R 中估计正确排序

$$\prec: R \times R$$

的问题。相反，分类和回归估计某些关于 R 的元素的理想函数：

$$ideal: R \to \{true, false\} \quad 或 \quad ideal: R \to \mathbb{R}$$

在这一广泛的特征下，我们已经给出了很多关于学习排名的例子。例如，当 $R \subseteq D$ 时，我们根据某种假设的相关性概念对文档排名；例如，对某些特定查询 q 的二元相关性：

$$d_1 \prec d_2 =_{def} (d_1 \notin rel_q \text{ 且 } d_2 \in rel_q) \tag{11-26}$$

对于文档排名，学习器 l 把一个训练集 $T \subset D$ 和一个标记函数 $label$：$T \to \{rel, non\}$ 作为输入，输出一个评分函数 $s: D \to \mathbb{R}$。通过定义它的输出为 $\precsim$：$D \times D$，（该式子逼近特定的正确排序 $\prec$），结果可根据学习排名进行整合：

$$d_1 \precsim d_2 =_{def} s(d_1) < s(d_2) \tag{11-27}$$

注意到 $\prec$ 是一个典型的偏序。有很多文档对 $d_1 \neq d_2$，使得 $d_1 \not\prec d_2$ 和 $d_2 \not\prec d_1$。对于公式（11-26）来说，无论是 d_1，$d_2 \in rel_q$ 或 d_1，$d_2 \notin rel_q$，上述情况都会出现。估计 $\precsim$ 可能也会是一个偏序。然而，如 MAP 和 P@k 这些信息检索评价指标假设它是全序的。如果不是全序，因为对于某些 $d_1 \neq d_2$，有 $s(d_1) = s(d_2)$，因此像 trec_eval（参见 2.3.2 节）这样的评价软件会随意假设 $d_1 \prec d_2$ 或 $d_2 \prec d_1$。

11.7.1 什么是学习排名

在文献中“学习排名”一词指比我们举出的例子更一般的学习问题，在这些问题中，q 和 D 是固定的，$\prec$ 是指二元相关性。尽管关于什么是学习排名没有一个大家都接受的定义，但是该词还是隐含了以下几个方面的含义：

- 元学习，d 的特征表示 $x^{[d]}$ 是对 q 和 d 采用多种不同的特征工程和排名公式得到的结果。
- $\prec$ 的更一般的特性，与二元相关性相对（公式（11-26））。例如，**分级相关性**（graded relevance）假设存在一个关于 k 个相关类别的有序集 $rel_q^{[1]}$、…、$rel_q^{[k]}$，其中 $rel_q^{[1]}$ 与 q 的相关性最低，而 $rel_q^{[k]}$ 与 q 的相关性最高（参见 12.5.1 节）。在分级相关性下，有

$$d_1 \prec d_2 \Leftrightarrow d_1 \in rel_q^{[i]} \text{ 且 } d_2 \in rel_q^{[j]} \ (i < j) \tag{11-28}$$

- 学习一族排名的问题，其中 $\prec_q$ 和 $\precsim_q$ 均使用 $q \in Q$ 参数化。
- 查询限制训练标记，其中 $d_1 \prec_q d_2$ 只在 $q \in Q_T$ 的情况下才知道，Q_T 是一个查询训练的集合，是 Q 的一个非常小的子集。
- 成对训练，其中 $d_1 \prec_q d_2$ 只在 $(q, d_1, d_2) \subset Q_T \times D_T \times D_T$ 的情况下才知道，且 $D_T \subseteq D$ 是一个查询训练集。
- LETOR 测试数据集[⊖]给出了一个标准的关于信息检索的学习排名的例子。在 LETOR 中，训练集 T 中包含了大量的 3 元组 $(q, x^{[d,q]}, r) \in Q_T \times \mathbb{R}^k \times \mathbb{Z}$，其中 $Q_T \subset Q$ 是一个查询训练集，$x^{[d,q]} = s_1(d, q), s_2(d, q), \cdots, s_k(d, q)$ 是对关于 q 的文档 d 采用 k 个不同的评分函数得到的特征表示，且 r 表示 $\prec$：

⊖ research.microsoft.com/en-us/um/beijing/projects/letor

$$\forall_{q\in Q_T}: r_1 < r_2 \;\Leftrightarrow\; (q,d_1) \prec_q (q,d_2) \tag{11-29}$$

- 一个使用**点击数据**（clickthrough data）（Joachims 等人，2005）来得到成对的、不确定的训练样本的标准例子。假设搜索引擎对查询 q 进行响应时呈现给用户一个排名列表 Res。同时假设用户查看了排名为 k 的文档 $d_k = Res[k]$，但没有查看排名比它更高的文档 $d_j = Res[j]$，其中 $j<k$。从这些信息我们可以推断，根据概率平衡，有

$$d_j \prec_q d_k \tag{11-30}$$

这个推断是不确定的，因为它可能是错误的，同时这个推断是不完整的，因为 $d_1 \prec_q d_2$ 只为一个子集 $(d_1, d_2) \in D_T \times D_T$ 定义。

11.7.2　学习排名的方法

学习排名的方法通常特征化为**逐点**（pointwise）、**成对**（pairwise）或**成列**（listwise），这依赖于它们是如何进行训练的（Cao 等人，2007）。逐点法将问题简化为：根据文档 d 与查询 q 相关的概率给每个 (d,q) 评分。logisitc 回归直接估计这个概率（与 log-odds 一样），并最大化所有标记样本的似然值。其他方法，如线性回归、排名感知和神经网络，当 d 被标记为与 q 相关时，一般会得到一个比 (d,q) 更高的得分，当 d 被标记为与 q 不相关时，一般会得到一个更低的得分。支持向量机在寻找一个超平面，把相关的 (d,q) 样本与不相关的分离开。

逐点支持向量机法说明了逐点法可能存在的缺点。尽管由定义我们有

$$(d_1,q_1) \not\prec (d_2,q_2) \quad (q_1 \neq q_2) \tag{11-31}$$

即使在 $q_1 \neq q_2$ 时，支持向量机还是尝试把所有的相关的 (d_1, q_1) 从不相关的 (d_2, q_2) 中分离出来。排名支持向量机（ranking SVM）只考虑在 $q_1 = q_2$ 的情况下优化间隔和训练误差。实际上，排名 SVM 建立 $|Q_T|$ 用于区分不同的分离超平面，且为每个查询 $q \in Q_T$ 均建立一个。采用类似的做法，其他的逐点法也适用于成对训练。

成列法单独考虑每个查询 $q \in Q_T$ 在整个集合 $D_T \times \{q\}$ 中的排名。这种方法的一个优点是很容易对在每个 D_T 中高排名的文档赋予更多的权重，以此提高查准率或任何其他方面的检索效果。从另外一方面看，$|D_T|!$ 可能是 D_T 的一种组合方式，因此为最好的结果找出一个紧凑和普遍的表示方式是非常具有挑战性的工作。

11.7.3　优化什么

每个学习方法都会优化一些目标函数（最小化一些损失函数），但也有一些限制。在学习排名中，也许最明显的损失函数就是那些反演函数了；学习到的顺序对和理想的顺序对是相反的：

$$inv = |\{(d_1,q_1),(d_2,q_2) \mid (d_1,q_1) \prec (d_2,q_2) \;\text{ 且 }\; s(d_2,q_2) < s(d_1,q_1)\}| \tag{11-32}$$

当 $\prec$ 表示二元相关时，最小化 inv 等同于最小化 AUC，其中 AUC 指 ROC 曲线下的面积。当 $\prec$ 表示一个全序时，最小化 inv 等同于最大化 Kendall 的 τ 相关性：

$$\tau = 2 \cdot (1 - inv) \tag{11-33}$$

最小化 inv 可能不会最大化检索的有效性，但通常会强调查准率而不是查全率。当对排名最高的文档赋予的权重比其他的文档都要高时，如 MAP、P@k 和 nDCG 这些指标会提高。一般来说，直接优化这些指标是不合适的，因为它们是不连续的和非凸的。一些方法使用连续的、凸的函数替代损失函数，以逐渐接近这些指标的上界。

11.7.4 分类的学习排名

我们用于分类的排名融合（参见 11.6 节）就是学习排名的一个例子，尽管 D 和 Q 的意义与上面介绍的相反。我们希望计算 $\precsim:(D\times L)\times(D\times L)$，基本满足

$$(d,l')\prec(d,l)\Leftrightarrow d\text{ 有语言 }l\neq l' \tag{11-34}$$

两种评价“接近程度”的评价指标是精确度（1 -误检率）和 MRR。每个（d，l）的特征表示包含

$$x^{[d,l]}=(r,s_1(d,l),s_2(d,l),\cdots s_k(d,l)) \tag{11-35}$$

其中

$$r=\begin{cases}1 & (d\text{ 有语言 }l)\\ 0 & (d\text{ 有语言 }l'\neq l)\end{cases} \tag{11-36}$$

$$s_m(d,l)=\text{从基本方法 }m\text{ 中计算得到的 }d\text{ 和 }l\text{ 的得分} \tag{11-37}$$

在表 11-12 中，RRF 和 Condorcet 的结果是固定规则，而不是学习方法，但使用它们计算得到的结果与 logistic 回归得到的结果是一样的，这里 logistic 回归是一个逐点学习方法。

为了采用成对或成列法，需要在特征表达式（d，l）中就能区分出某个特定文档 d，以使得一个由普通文档 d 构成的子集为：

$$x^{[d_i,l]}=(r,i,s_1(d_i,l),s_2(d_i,l),\cdots,s_k(d_i,l)) \tag{11-38}$$

表 11-14 再一次给出了逐点 LR 的结果，同时还给出了运用在语言分类排名问题上的排名 SVM（SVM^{light}）的结果，这里使用了两个不同的正则化参数 C。对于两种情况，排名 SVM 得到的结果都稍优于逐点 logistic 回归得到的结果。

表 11-14 对语言分类采用排名 SVM 得到的结果

方法	误检率（%）	MRR
LR（逐点）	18.0	0.88
排名 SVM (C=0.01)	17.3	0.88
排名 SVM (C=0.1)	17.6	0.88

11.7.5 排名检索的学习

表 11-3 展示了固定组合方法有效地把来自各种检索方法的结果组合起来。因为 4 个实验中都采用了相同的方法，我们现在研究采用学习排名法来得到一个更好的组合方法的可能性。为此，把 TREC45 1998、GOV2 2004 runs 和 qrels 看做训练样本和标签，并把 TREC45 1999 和 GOV2 2005 runs 看做相应的测试样本。每个样本（d，q_j）表示成一个查询向量

$$x^{[d,q_j]}=(r,j,s_1(d,q_j),s_2(d,q_j),\cdots,s_k(d,q_j)) \tag{11-39}$$

其中 $k=29$ 表示方法的数量。对于这个问题，我们采用（逐点）logistic 回归和排名 SVM。

表 11-15 列出了采用 logisitc 回归和排名 SVM 得到的结果，并与排名倒数融合这一基准方法得到的结果进行比较。很难说明 LR 的结果本质上是与基准方法的结果不同的。排名 SVM（SVM-Rank[⊖]）的结果相对差一点，并大约需要 4 天的 CPU 时间去处理 GOV2 数据集，而采用 RRF 只需要几秒，采用 LR 仅需几分钟即可。

⊖ www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

表 11-15　logistic 回归学习排名的结果，学习方法是在某年的检索结果上进行训练的，并且在下一年的主题上进行测试

语料库	TREC45		GOV2	
训练	1998		2004	
测试	1999		2005	
指标	P@10	MAP	P@10	MAP
RRF	0.464	0.252	0.570	0.352
LR	0.446	0.266	0.588	0.309
排名 SVM (C=0.02)	0.420	0.234	0.556	0.268

11.7.6　LETOR 数据集

LETOR 是一个针对学习排名的基准数据集（Liu 等人，2007），是用于评价学习排名方法的测试集。在写这本书的时候，第 3 版的 LETOR 数据集是最新的。LETOR 3 由 7 个数据集组成，每个包含如公式（11-39）的格式的样本。这些样本被分成 5 个标准块，为的是可以进行 5 次交叉验证。其中的 6 个数据集选用的文档来自 TREC GOV（不是 GOV2）文档集，选用的主题来自 TREC 2003 和 TREC 2004 网络检索任务。其中一个数据集使用的文档和主题均来自 OHSUMED 文档集。对于 TREC 文档集，相关性是二元的，并且共有 64 种特征以表示查询和文档的不同特征。对于 OHSUMED 文档集，相关性是三元的（不相关、相关和高度相关），并且有 45 个基于内容的特征。标准评价工具由语料库提供。

LETOR 数据集还附带几个最新的学习排名方法得到的原始结果集 Res_i 和 $Score_i$：

- ListNet（Cao 等人，2007）使用梯度下降去优化列表式目标函数。
- AdaRank-MAP（Xu 和 Li，2007）使用带替代目标函数的 AdaBoost，目的是优化平均查准率。
- RankBoost（Freund 等人，2003）使用 AdaBoost 去最小化成对训练样本和排名列表结果间的反演。
- RankSVM 的实现与 SVM^{light} 是一样的，我们把该方法用于语言分类，以最小化成对训练样本间的反演。

这里有 680 个主题，每个主题都包含多达 1000 个文档。尽管是单独在这 7 个数据集上对方法进行训练，表 11-16 列出了对 680 个主题的 P@10 和 AP 得分求平均得到的综合结果。表中还包含了前面例子中使用的逐点 LR（批）法得到的结果。LR（梯度下降）使用带目标函数的梯度下降对式 LR 方法，目标函数增加了某些与高排名不相关文档有关的反演的权重。

表 11-16　对 LETOR 3 语料库中的 583 850 个文档查询对采用学习排名得到的结果。P@10 和 MAP 这两个得分是 680 个主题的得分结果的平均值，其中这 680 个主题来自 7 个 LETOR 3 数据集。RRF、Condorcet 和 CombMNZ 融合了各个学习排名方法的结果

方法	P@10	MAP
ListNet (Cao 等人, 2007)	0.1853	0.5846
LR (梯度下降)	0.1821	0.5837
AdaRank-MAP (Xu 和 Li, 2007)	0.1789	0.5778
排名 SVM (Joachims, 2002)	0.1811	0.5737
LR (批)	0.1780	0.5715
RankBoost (Freund 等人, 2003)	0.1836	0.5622
RRF	0.1902	0.6051
Condorcet	**0.1907**	0.5917
CombMNZ	0.1893	**0.6107**

RRF、Condorcet 和 CombMNZ 这些融合方法将 6 种学习排名方法的结果组合起来，提高了 P@10 和 MAP 的效果。在写这本书的时候，还没有出现能超过这些融合结果的排名学习方法。另外，使用统计分析不能从这 6 种独立的方法中得到任何有意义的差别（Cormack 等人，2009）。

11.8 延伸阅读

Belkin 等人（1995）的报告是早期对不同查询的检索结果进行组合的一种尝试，这些查询都是通过人工或者自动从一个 TREC 主题中提取出来的。本章介绍了一系列的得分组合方法——CombMNZ 是其中一种。对这些方法进行大量的后续研究后发现，使用这些方法把各种检索结果组合起来是非常有效的（如 Lee，1997）。Montague 和 Aslam（2002）研究并评价了得分组合以及选举方法，还提出把 Condorcet 用作选举方法。Voorhees 等人（1995）考虑了测试集融合，这里对无交集的文档集使用同一个查询。**元搜索**（metasearch）通常用于求组合结果的平均，这些组合结果来自自主搜索引擎，Meng 等人（2002）对这进行了研究。

Vogt 和 Cottrell（1999）研究了把线性回归用于组合基于训练样本的检索结果。Wolpert（1992）提出的堆叠泛化（叠加）是一种标准的机器学习方法。在我们例子中特别用到的 log-odds 变换和梯度下降均由 Lynam 和 Cormack（2006）提出。bootstrap 集成法（bagging；Breiman，1996）和 boosting（Schapire，2003）与叠加类似，均是机器学习中的重要方法。大部分的机器学习教科书（如 Hastie 等人，2009）把像叠加、bagging 和 boosting 这些集成方法作为主要介绍的主题。

多类 logistic 回归像 logistic 回归本身一样，是一种普通数据分析的标准技术（Hosmer 和 Lemeshow，2000）。Crammer 和 Singer（2002）为多类支持向量机提出了一种实用算法。Schapire 和 Singer（2002）提出了 boosting 在多类问题中的应用。

在 NIPS 2005、SIGIR 2007 和 SIGIR 2008 的研习会（workshop）中，学习排名已经变成了一个主题。尽管如此，对于学习排名，暂时还没有一个精确的定义。Burges 等人（2005）和 Joachims 等人（2005）的学术论文把学习排名定义为学习和目标排名间的最小反演：Burges 等人使用 Ranknet 这一方法中的梯度下降，然而 Joachims 等人使用支持向量机。

Li 等人（2007）把学习排名看做一个多类分类问题，其中每个分级相关性层表示对应一个类别。Herbrich 等人（2000）和 Freund 等人（2003）描述了成对法。Xu 和 Li（2007）描述了 AdaRank-MAP 列表式方法；Burges 等人（2006）描述了 LambdaRank 列表式方法。Svore 和 Burges（2009）证明了在使用相同的特征集时，LambdaRank 的性能可以比 BM25 好。Yilmaz 和 Robertson（2010）讨论了把信息检索评价指标看做学习排名的优化目标。LETOR 数据集（Liu 等人，2007）为评价学习排名法提供了一个标准基准。Liu（2009）研究了当前的学习排名方法。D.Sculley 给出的 sofia-ml 包为机器学习提供了一套新的快速增量算法。[⊖]

⊖ code.google.com/p/sofia-ml

11.9　练习

练习 11.1　下载一个 TREC 测试集以及一个或多个搜索引擎。在这些主题上，对搜索引擎设置不同的配置并运行，使用 RRF、CombMNZ 和 Condorcet 将这些结果组合起来。比较这些结果。

练习 11.2　下载一个 TREC Spam Filter Evaluation Toolkit，在示例语料库上运行几个示例过滤器。使用选举方法、朴素贝叶斯和 logistic 回归将这些结果组合起来。评价这些结果。

练习 11.3　使用 RRF、CombMNZ 和 Condorcet 将你的垃圾信息过滤器的运行结果组合起来。这个实验会适当为垃圾信息过滤器的用法建模吗？采用这些方法会比采用选举方法、朴素贝叶斯和 logistic 回归得到的结果要好吗？

练习 11.4　下载 LETOR 3 数据集。对该数据集采用一种或多种学习排名法（如 SVM^{light}），将你得到的结果与公布的基准进行比较。

练习 11.5　忽略 qid 域，直接对 LETOR 3 数据集采用 logistic 回归（如 LibLinear）。将这个结果与使用学习排名法得到的结果进行比较。

11.10　参考文献

Agresti, A. (2007). *An Introduction to Categorical Data Analysis* (2nd ed.). New York: Wiley-Interscience.

Belkin, N., Kantor, P., Fox, E., and Shaw, J. (1995). Combining the evidence of multiple query representations for information retrieval. *Information Processing & Management*, 31(3):431–448.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Burges, C. J. C., Ragno, R., and Le, Q. V. (2006). Learning to rank with nonsmooth cost functions. In *Proceedings of the 20th Annual Conference on Neural Information Processing Systems*, pages 193–200. Vancouver, Canada.

Burges, C. J. C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 89–96. Bonn, Germany.

Cao, Z., Qin, T., Liu, T. Y., Tsai, M. F., and Li, H. (2007). Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136. Corvalis, Oregon.

Cormack, G. V., Clarke, C. L. A., and Büttcher, S. (2009). Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759. Boston, Massachusetts.

Crammer, K., and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292.

Efron, B., and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Boca Raton, Florida: Chapman & Hall/CRC.

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The Elements of Statistical Learning* (2nd ed.). Berlin, Germany: Springer.

Herbrich, R., Graepel, T., and Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. In Bartlett, P. J., Schölkopf, B., Schuurmans, D., and Smola, A. J., editors, *Advances in Large Margin Classifiers*, chapter 7, pages 115–132. Cambridge, Massachusetts: MIT Press.

Hosmer, D. W., and Lemeshow, S. (2000). *Applied Logistic Regression* (2nd ed.). New York: Wiley-Interscience.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142. Edmonton, Canada.

Joachims, T., Granka, L., Pan, B., Hembrooke, H., and Gay, G. (2005). Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 154–161. Salvador, Brazil.

Lee, J.H. (1997). Analyses of multiple evidence combination. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–276.

Li, P., Burges, C., and Wu, Q. (2007). McRank: Learning to rank using multiple classification and gradient boosting. In *Proceedings of the 21st Annual Conference on Neural Information Processing Systems*, pages 897–904. Vancouver, Canada.

Liu, T.Y. (2009). Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331.

Liu, T.Y., Xu, J., Qin, T., Xiong, W., and Li, H. (2007). LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, pages 481–490. Amsterdam, The Netherlands.

Lynam, T.R., and Cormack, G.V. (2006). On-line spam filter fusion. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 123–130. Seattle, Washington.

Meng, W., Yu, C., and Liu, K.L. (2002). Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89.

Montague, M., and Aslam, J.A. (2002). Condorcet fusion for improved retrieval. In *Proceedings of the 11th International Conference on Information and Knowledge Management*, pages 538–548. McLean, Virginia.

Schapire, R. (2003). The boosting approach to machine learning: An overview. In Denison, D.D., Hansen, M.H., Holmes, C.C., Mallick, B., and Yu, B., editors, *Nonlinear Estimation and Classification*, volume 171 of *Lecture Notes in Statistics*, pages 149–172. Berlin, Germany: Springer.

Schapire, R., and Singer, Y. (2000). BoosTexter: A boosting-based system for text categorization. *Machine learning*, 39(2):135–168.

Surowiecki, J. (2004). *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations.* New York: Doubleday.

Svore, K.M., and Burges, C.J. (2009). A machine learning approach for improved BM25 retrieval. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 1811–1814. Hong Kong, China.

Vogt, C., and Cottrell, G. (1999). Fusion via a linear combination of scores. *Information Retrieval*, 1(3):151–173.

Voorhees, E.M., Gupta, N.K., and Johnson-Laird, B. (1995). Learning collection fusion strategies. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 172–179. Seattle, Washington.

Wolpert, D.H. (1992). Stacked generalization. *Neural Networks*, 5:241–259.

Xu, J., and Li, H. (2007). Adarank: A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 391–398. Amsterdam, The Netherlands.

Yilmaz, E., and Robertson, S. (2010). On the choice of effectiveness measures for learning to rank. *Information Retrieval.*

第四部分 评 价

第 12 章

Information Retrieval: Implementing and Evaluating Search Engines

度量有效性

评价的目的是度量一个信息检索方法多好地实现了它的预期目标。评价必须能估计在给定情况下某种信息检索方法的有效性，并且在相同情况下与另外一种方法的有效性进行比较，或预测该方法在不同情况下的有效性。如果没有了评价，则很难做出明智的部署决策或发现更好的方法。一个可用的评价方法必须具备以下几项条件：

- 一个能表示信息检索方法预期目标的特征；
- 一个用于量化关于多好地实现了预期目标这一概念的指标；
- 一种准确、精确而又经济的度量技术；
- 一个用于度量误差的估计方法。

本章将涉及以上关于信息检索评价的几个方面，并直接建立在 2.3 节介绍的内容之上。这些内容都是关于基本特定检索任务的评价。

在 12.1 节，我们将回顾在传统信息检索评价中使用的主要有效性度量，并讨论与之对应的检索任务。12.2 节将描述文本检索会议（Text REtrieval Conference，TREC）中使用的评价方法，这代表了信息检索评价中的事实标准，并被大部分研究者所采用。12.3 节的主题是评价结果的统计分析。我们将讨论常用的分析技术用于信息检索时的情况，如置信区间、显著性检验等，以及一些较不常用的方法，例如独立实验的元分析。同时也根据检验力（power）和效度（validity）这两方面比较几个统计检验，也就是：统计检验能多精确地检测出进行对比的检索函数间的差异，以及它们的判断有多准确。在 12.4 节，我们将探索在保持评价结果可靠性的同时减少相关性判断次数的方法。最后，12.5 节将介绍所有扩展了或使用了 12.1 节中介绍的传统评价方法解决问题的有效性评价指标。这些评价方法考虑了多级相关性判定（不仅仅是二元相关/二元不相关评价方法）和不完整判定，还有新颖性和多样性。

除本章外，其他章将从其他方面讨论评价方法。第 10 章包含了用于过滤和分类的评价方法。第 13 章包含了关于 Web 的问题以及评价方法。第 16 章就 XML 信息检索的有效性评价给出了一个概述。

12.1 传统的有效性指标

传统的信息检索评价基于以下两个基本假设：

1）给定一个以检索查询表示的用户**信息需求**（information need），且在一个给定的文档集中，每个文档与此信息需求要么**相关**（relevant）要么**不相关**（nonrelevant）。

2）文档 d 的相关性仅取决于信息需求和 d 本身，独立于文档集中的其他文档的搜索引擎排名。

基于这两个假设，可以定义各种不同的有效性指标。其中的一些指标已在 2.3 节中出现过了。

12.1.1 查全率和查准率

查全率和查准率，或许是信息检索评价中最古老的指标了，用于评价搜索引擎在响应某个查询时检索出的无序文档集。**查全率**（recall）是在该文档集中相关文档所占的比例。设 *Res* 为检索出的文档集，*Rel* 是相关文档集。这样我们有：

$$\text{recall} = \frac{|Res \cap Rel|}{|Rel|} \tag{12-1}$$

作为一个有效性指标，查全率量化了检索结果对用户信息需求的满足程度。查全率对检索任务进行了建模：用户希望找出所有的相关文档，如在文献回顾或法律搜索中。

只需返回文档集中的所有文档，一个搜索引擎可以很容易地实现 1.0 的查全率。当然，如果这样做，结果列表中的大部分文档都是不相关的。**查准率**（precision）从另一方面对结果集进行评价——系统检索出的相关文档占检索出的所有文档的比例：

$$\text{precision} = \frac{|Res \cap Rel|}{|Res|} \tag{12-2}$$

从查准率派生出的用户模型假设：考虑到检查结果中的每个文档需要付出代价，因此用户希望搜索引擎返回数量合适的相关文档。查准率是每单位代价查看的相关文档的数量指标。查准率的倒数等于用户在检索结果集中以某种顺序查看文档，在找到一个相关文档前需要查看的期望文档数。

12.1.2 前 k 个文档的查准率（P@k）

尽管查全率在 20 世纪 60 年代早期的信息检索实验中是一个不错的指标，但随着文档集的逐年增大，它的作用越来越受到怀疑。例如，对于 TREC 主题 426（“law enforcement, dogs”），已知在 TREC 文档集中有 202 个相关文档。只有最好奇的用户才会对全部相关文档感兴趣。在 Web 检索中，查全率的意义就更小了，因为一个主题有数以万计的相关文档是件很平常的事（例如，想想 TREC 的主题“drug abuse”、“vietnam war”和“international space station”）。

前 k 个文档的查准率（precision at k documents，“precision@k”或“P@k”）对用户满意度建模：对于一个较小的 k 值（通常 $k=5$，10 或 20），返回排名最高的 k 个文档的列表给用户。定义如下：

$$\text{P@}k = \frac{|Res[1..k] \cap Rel|}{k} \tag{12-3}$$

其中 $Res[1..k]$ 由系统返回的排名最高的 k 个文档组成。与无约束的查准率一样，P@k 也是假设用户以任意顺序查看结果，并且即使用户发现了一个或多个相关的文档后，她依然会查看**所有的文档**（all of them）。同时还假设，如果搜索引擎返回排名最高的 k 个结果中，没有一个相关的文档，说明查询失败，用户的信息需求没有得到满足。P@k 有时被称做**早期的查准率指标**（early precision measure）。P@k 没有反映查全率，从而可以肯定没有用户会读过关于“drug abuse”或“vietnam war”的所有相关文档。

12.1.3　平均查准率

通常情况下，很难确定 P@k 中的 k 应取何值。建议可取 $k=10$，因为许多搜索引擎默认将前 10 个结果显示给用户。但是，由于我们关注的是对结果排名进行评价，而不是对一个用户界面进行评价，所以 $k=10$ 和取其他值一样，都是任意选取的。

平均查准率（Average Percision，AP）尝试将所有可能的查全率层次上的查准率结合起来以解决这个问题：

$$\mathrm{AP}=\frac{1}{|Rel|}\cdot\sum_{i=1}^{|Res|}\mathrm{relevant}(i)\cdot \mathrm{P@}i \tag{12-4}$$

如果 Res 中的第 i 个文档是相关的（即 $Res[i]\in Rel$），则 relevant（i）为 1；否则为 0。这样，对于每个相关文档 d，AP 计算的是结果列表不断扩大直至包含文档 d 时的查准率。如果一个文档未出现在 Res 中，AP 假设相应的查准率为 0。这样，可以说 AP 中隐式地反映了查全率，因为它考虑了相关文档不在结果列表中的情况。

12.1.4　排名倒数

到目前为止所讨论的指标都假设存在一个相关文档集，其中所有文档都是同样有用的，用户会有兴趣查阅一定数量的相关文档。尽管这个假设对特定检索任务是合理的，但还是存在用户只想看到一个相关文档的情况，因此该文档的排名应该尽可能得高。例如，考虑查询〈“white”，“house”，“official”，“website”〉。对于这个查询，实际上只有一个相关结果（www.whitehouse.gov），返回的结果中将它排在第 1 位明显比将它排在第 5 或第 10 位更好。

排名倒数（Reciprocal Rank，RR）是这样一个指标：它仅关注前几位的结果，并非常偏好相关文档在返回的结果列表中排名非常靠前这一情况。它的定义为

$$\mathrm{RR}=\frac{1}{\min\{k \mid Res[k]\in Rel\}} \tag{12-5}$$

请注意，公式（12-5）允许存在一个以上的相关文档。如果 $|Rel|>1$，上面的假设就变为：一旦用户找到第一个相关文档，她的任务就完成了。另一方面，如果只有一个相关文档，即 $|Rel|=1$，那么排名倒数等价于平均查准率。

12.1.5　算术平均与几何平均

对多个主题的评价结果求平均（计算算术平均值）得到聚合结果。相应的评价指标有**平均查准率均值**（Mean Average Precision，MAP），**平均排名倒数**（Mean Reciprocal Rank，MRR），等等。有趣的是，对于一些指标，“平均”一词并不明确地包含在该指标的名字当中。比如，**前 k 个文档的平均查准率**（mean precision at k）通常简写为 P@k，而不是MP@k。

最近，有人认为计算算术平均值并不是将多个主题的有效性评价值聚合在一起的最佳方式（Robertson，2006）。举个例子说明，考虑两个主题 T_1 和 T_2，每个主题都仅有一个文档与其相关。进一步考虑以下两个结果列表

$$Res(T_1)=\langle -,+,-,-,-,\cdots\rangle,\quad Res(T_2)=\langle -,+,-,-,-,\cdots\rangle \tag{12-6}$$

其中，“+”表示一个相关文档，“−”表示一个不相关文档。两个结果列表的平均查准率

都为 0.5。此时，两个主题的 *MAP* 也均是 0.5。现在，再考虑另一种情况：

$$Res'(T_1) = \langle +, -, -, -, -, \cdots \rangle, \quad Res'(T_2) = \langle -, -, -, -, -, \cdots \rangle \tag{12-7}$$

其中，$Res'(T_1)$的 AP 为 1，$Res'(T_2)$的 AP 为 0。MAP 在这两种情况下（公式（12-6）和公式（12-7））是一样的，但对用户来说，公式（12-6）的总体效果要比公式（12-7）的总体效果好。

由这个观察引出了**几何平均查准率均值**（Geometric Mean Average Precision，GMAP）的定义（Robertson，2006）：

$$\mathrm{GMAP}(AP_1, \cdots, AP_n) = \sqrt[n]{\prod_{i=1}^{n}(AP_i + \varepsilon)} - \varepsilon \tag{12-8}$$

其中 ε 是一个常量，它的作用是消除由于其中一个 AP_i 为 0 所带来的影响。设 $\varepsilon = 0.01$，对于公式（12-6）和公式（12-7）的例子有

$$\mathrm{GMAP}(0.5, 0.5) = 0.5, \quad \mathrm{GMAP}(1.0, 0.0) = \sqrt{1.01 \cdot 0.01} \approx 0.10 \tag{12-9}$$

这个结果与我们的直观感觉是一致的，即从总体效果来看，$Res(T_1)$和 $Res(T_2)$要好于 $Res'(T_1)$和 $Res'(T_2)$。

因为一个 AP 值集合的几何平均与 AP 值取对数后的算术平均是等价的，所以也有人认为这个问题并不是由于采用了算术平均引起的，而是 AP 这一指标本身所引起的。

12.1.6 用户满意度

纵观近 20 年设计的评分函数，几乎都是采用以上介绍的一种或多种有效性指标来进行评价的。鉴于它们在信息检索评价中的重要性，有人可能认为用户满意度和平均查准率间的关系已经得到了彻底的研究，并且已被大家非常好地理解了。遗憾的是，事实并非如此。试图找到用户满意度和各种有效性指标之间的关系，是近期才出现的研究方向。初步的研究结果表明，用户满意度和 AP 间的关系并不密切（Turpin 和 Scholer，2006），但与如 P@10 这样的早期查准率指标间的关系非常密切（Kelly 等人，2007）。尽管如此，AP 仍然是并且将来也是在信息检索评价中使用最广泛的指标。

12.2 TREC

2.3 节简单介绍了用于有效性评价的经典方法，这些方法被编纂在 TREC 中，并被许多其他评价方法所效仿。TREC 提供了一个框架，前面章节介绍的评价方法就可应用到这个框架中。这个框架代表了信息检索评价的事实标准；采取 TREC 的框架得到的评价结果在所有公开的评价结果中占了绝大多数，尤其是在特别检索中。按照这种方法，设计评价实验的组织者们开发了一个主题集，然后将这个主题集和目标文档集一起分发给各个参与这个实验的小组。一般来说，文档集是相对均匀的，且由经专业撰写和编辑的材料组成，如报纸或期刊文章。每个小组根据主题创建查询，并在文档集上执行这些查询。通常这些查询必须自动地根据主题创建出来，不需要人工参与，但在一些实验中，允许人工对查询进行创建或修改。

常用的主题集一般包括 50 或更多个主题。在很多情况下，组织者在创建主题时，会明确地要求参与小组使用主题中的一个域构造查询，且构造查询时不能对这个域进行修改或只允许少量修改。在图 1-8 所示的示例主题中，标题域（“law enforcement，dogs”）就是用作这一目的的。主题的其他部分提供了更多其他的细节，以消除由于标题简短而引起的潜在的

歧义。在理想情况下，主题能准确、完整地描述它对相关文档的要求。TREC 的主题中很少出现印刷或拼写错误，而现实查询中，这种错误却很常见。

在文档集上执行完查询后，每个小组为每个主题返回一个包含 k 个相关文档的排名列表：TREC 中的实验一般采用 $k=1000$；而本书中的实验采用 $k=10\ 000$。关于一个完整主题集的排名列表集在 TREC 的术语中叫做一个**运行实例**（run）。根据实验，组织者可能允许每个小组返回多个运行实例，以便于组织者测试每个小组的系统的各个方面的性能。

当接收到来自各个小组的一个或多个运行实例后，组织者会创建一个用于评价的文档池（pool）。这个文档池由来自每个运行实例的排名靠前的文档联合组成。其中，每个运行实例文档池的深度一般为 100。评判者判定每个文档时使用一个二元值：相关或不相关。[⊖]这些判定在 TREC 的术语中称为 qrels，用于计算前面章节中介绍的查全率、查准率、平均查准率以及其他指标。在计算这些指标时，不在文档池中的文档（因为在任何一个运行实例返回的结果中，它都不属于排名靠前的文档）认为是不相关的。

主题、判定以及文档集构成一个**测试集**（test collection）。TREC 的核心目标是构造一个可重用于后续实验的测试集。例如，如果提出了一种新的信息检索技术或排名公式，提出者可能想将这用于已有的测试集上，并将得到的结果与标准方法得到的结果进行比较。可重用测试集也可用于调整检索公式，即调整其参数以优化其性能。如果一个测试集是可重用的，一般会假设判定尽可能得全面。理想情况下，可以找出所有相关文档。因此，很多评价实验积极鼓励人工运行实例（涉及人工参与），以增加系统找出那些已被标记为相关的文档的数量。

文档池方法需要大量的判定工作。对于 50 个主题，几十个小组和一个深度为 100 的文档池，判定者可能需要做成千上万个判定。对于 1999 年 TREC-8 特定任务，共有 71 个运行实例加入到文档池中（Voorhees 和 Harman，1999）。理论上，如果运行实例间没有重复的结果，这文档池的规模可高达 $71 \cdot 50 \cdot 100 = 355\ 000$。幸运的是，运行实例间有大量的重复结果，尤其是排名靠前的文档。尽管如此，对于这 50 个主题，文档池中依然包含 86,830 个文档。假设给一个判定者判断一个文档的时间只有 30 秒，那完成这 86 830 个文档的判定将需要总共 724 个小时，这足够使一个由 10 个判定者组成的团队忙上两个星期。

12.3　在评价中使用统计

给定一个由文档、主题、qrels 组成的测试集，我们使用 12.1 节中的评价方法在指定测试集上计算某一检索方法 A 的有效性，并将它与其他的检索方法 B 的有效性进行比较。然而，我们只能从这些评价结果中知道这两个方法在给定测试集上的对比结果。理想情况下，我们想知道对于所有由文档、主题和相关性判定组成的文档集中，这两种方法哪种更好，好多少。

统计分析可用于估计一个评价结果多好地预测了系统的性能，而不局限在用于度量它的某个特定测试集。当得到检索方法在某个特定主题上的 AP 值或在某个主题集上的 MAP 值时，我们可以断言，在其他主题上也会得到类似的结果，因为就算离开这些特定的评价主题，系统的预期目的也会扩展得很好。然而，如果没有统计分析，就很难说明该断言成立或那些指标反映出系统多好地满足了它的预期目标。

⊖ 一些 TREC 实验的判定采用三个等级：不相关、相关、高度相关。然而，为了评价单个运行实例，通常把相关和高度相关文档看成是等价的。

统计分析可用于估计某个定量指标的精确度（例如，“方法 A 有多好?”或“A 比 B 好多少?”），或估计支持某个特定假设的证据效力（例如，“方法 A 比方法 B 更好!”）。讲解实验方法论的书中会用很多版面来介绍传统的假设检验，指出当证据效力超过一个任意设定的阈值时，结果就是具有统计显著性的。这样的检验给出的信息比证据效力的定量估计给出的信息要少得多，并且很容易被曲解。例如医药等领域，就很少采用假设检验（Gardner 和 Altman，1986）。在信息检索文献中，统计估计和假设检验通常不被提到或被误用。我们希望能纠正这些行为。

本节较长，使用较大篇幅回顾一些基本知识，同时讨论它们在信息检索评价中的应用。尽管你可能已经熟悉大部分的基本知识，但我们采用了一个不同于数学或应用统计文献的观点。我们将考虑统计在科学方法中的作用，而不是考虑统计本身的目的或作为后验的统计检验——仅仅就为满足出版条件而已。

如果你已经忘记了什么是统计，而你又想避免挖掘这些相关材料的麻烦，你可以跳过或粗读本节剩下的内容。利用这些统计知识要达到的“基本目的”如下：(1) 为你的信息检索实验建立一个合适的基准；(2) 将另一种方法或技术的效果与这一基准进行比较；(3) 报告这两种方法的效果差距，并考虑这个差距对用户是否有重大的影响；(4) 报告一个关于统计置信度的指标值，例如 P-值或置信区间。对于最后一步，通常选用配对 t-检验。

12.3.1 基础和术语

在某种意义上，信息检索评价中的有效性指标与物理度量指标十分相似。想象一下，本书的作者之一想确定他自己的高度以便订购一套西装。他使用一把直尺，测得 5 英尺 8 英寸。为了准确，他再用卷尺测了一次，结果是 173 cm。精确地转换到通用单位后，他分别得到两个值，1727.2 mm 和 1730 mm。他觉得有些迷惑，就用直尺更准确地测了一次，结果为 5 英尺 $8\frac{3}{8}$ 英寸（就是 1736.725 mm），是其中一次测量出错了吗？作者的真实身高是多少呢？答案是“没出错，测量结果都与预期相符”，而且“这并没关系，因为不管怎样，他得到的会是同样尺码的西服”。统计方法就是帮助我们回答类似于这样的问题的。

如果需要更加精确的答案，他也许会用计算器计算这些测量结果的均值，得到 1 731.308 333 mm。如果对足够多的独立测量结果求均值，他可以得到一个关于他身高的精确估计值，但是没有一个估计值会精确到像上面一样，用 10 位数字来表示估计值，因为观察得如此细致入微并不符合我们对一个人身高的认识。我们需要在早上或晚上，穿或不穿厚重的冬衣来测量一个人的身高吗？身高包括头发吗？试图测量得比我们理解中的身高更精确是完全没有意义的。然而，“真实身高”在描述我们估计的准确程度时是一个有用的抽象概念，即要解决这样一个问题：“估计值与真实值有多接近?”为此，把“真实身高”定义为无限多个测量值的均值，其中每个测量值间的差异是偶然的，这一定义非常有用。然后我们把单个测量值与真实值的差异看做**随机误差**（random error）。与随机误差相对的是**系统误差**（systematic error）或**偏差**（bias），系统误差是有可能会发生的，例如，如果卷尺用得太多了，测量值将变大。

指标的查准率[⊖]（precision of measurement）是一个测量不受随机误差影响的程度。**效度**

⊖ 指标的查准率在统计学文献中一般简称为查准率。为清晰起见，在需要时，我们会使用全称以区分信息检索中所使用的有效性指标。

(validity) 是衡量该测量本身对它所测量的内容的真实反映程度（对于我们的例子，就是总体检索效果）。本节我们尤其关注的是，信息检索实验中得到的指标的查准率。如果在一个评价信息检索系统有效性的实验中重复使用完全相同的技术，但却使用不同的主题、文档或相关性判定，那得到的结果将会有多相似呢？如果可能，是否能不重复实验，而预估这个相似程度呢？尽管这些问题大部分是服从统计推论的，但只能从包括效度在内的一般探索框架上下文来理解这些问题，而不是从统计学必须的角度来考虑。取而代之，我们可以利用一些科学探索方法——观察、归纳、演绎、实验——来解决它们。

总体 (population) 这一概念是任何关于统计分析的讨论的核心，并且也是过去和现在一直争论不休的话题 (Lenhard, 2006)。我们采用由 Ronald Fisher (Fisher, 1925) 提出的无限假设总体这一概念：

如果在孟德尔遗传实验中，我们说在经过某种交配后，是白老鼠的概率为 1/2，我们必须把经过这些交配后得到的老鼠看做是一个无限的总体。总体的数量必须是无限大的，因为如果从一个有限的总体中抽样，那已知抽出的一只老鼠是白老鼠后，这一事实会影响到以后抽出的是白老鼠的概率，这并不是我们想要考虑的假设情况；另外，概率并不总是一个有理数。尽管“总体是无限的”明显只能是个假设情况，因为不但老鼠所生的孩子的实际数量是有限的，而且我们还可能希望考虑概率受父母的年龄或它们的营养情况的影响。然而，在我们的实验中，我们可以假设老鼠生出的孩子的数量是无限的，即父母的情况都类似，年龄相当且在相同的环境下。白老鼠的比例在这个假设的总体中具有实际的意义，可以用于说明我们的概率。简单来说，这个假设的总体是由我们研究的条件而得到的概念。这个概率与其他的统计参数一样，是那个总体的一个数值特征。

回到测量作者身高的例子，我们关注的是使用一把直尺或卷尺去度量得到的结果。当前感兴趣的这个总体就是一个相似度量集（不是大家随意所想的，相似作者集）。对于信息检索评价来说，我们感兴趣的结果是一些有效性指标，如 MAP 或 P@k，我们把这些指标用于评价一个系统（或比较两个系统间的 MAP 或 P@k 的不同），对于信息检索来说，总体就是相似指标集。我们描述相似指标集的方式，与描述采用相同的检索方法，但采用不同的文档、主题或相关性判定得到的实验结果的方式是一样的。不同的文档、主题和相关性判定本身可以表示不同的假设总体：主题提交给系统后，我们希望系统从文档集中检索出相关文档，和相关性判定集。

对于身高这个例子，“真实有效性”表示所有指标的均值。同时有效性这个概念不必是准确的，这是因为我们不能准确地说明这些总体。有效的定义可以是“所有在过去、现在、将来要提交给特定信息检索系统的主题”；“所有在过去、现在、将来提交给过滤器的文档”；“某个文档关于某个过去、现在、将来的主题的所有相关性判定”。因为部分总体不存在，在评价时肯定是不能把它们都枚举出来的。

相对于尝试枚举一个假设的总体，我们采用容易获得的数据和观察（如，TREC 集中的主题、文档和 qrels），并把它们看做是从假设总体中获得的：“所有的数据就像是我们采集回来的一样”。相对于我们感兴趣的真实（但是难以度量的）总体（真实总体被称做**目标总体** (target population)），这个假设总体被称做**源总体** (source population)。源总体接近于假设目标总体。越是接近目标总体，源总体得到的指标值越好地反映了目标总体得到的假设的指标值。衡量指标的准确率可以从三个不同的方面考虑：度量指标的查准率；度量指标关于由源总体定义的真实值的效度；指标关于目标总体的效度。我们把这两种效度分别称为**内部效度** (internal validity) 和**外部效度** (external validity)。外部效度也被称为**可转换性**

(transferability) 或**普遍性** (generalizability)。

统计推断只关注查准率和内部效度，即指标多好地反映了由源总体定义的真实值？外部效度不是基于统计推断的，而是基于科学探究的：1) 找出源总体和目标总体间的特性差异，且这些差异值会影响到外部效度；2) 确定一个具有这些特性差异的新的源总体；3) 就新的源总体评价其有效性；4) 评价这些差异带来的影响。例如，原来的 TREC 文档集主要由新闻在线和类似的文章组成，且它们的主题都是关于当前时事的。TREC Web 专题研究由于 Web 网页和 Web 查询的使用而可能导致的差异。现在的思路是，不去找能最好地表示目标总体的最终源总体，而是去找出能解释可能差异的总体。如果度量结果相似，我们对二者的外部效度的置信度都提高了。如果度量结果不一致，那我们就需要进一步地进行科学探究了。

任何实验技术都有假设和局限性，从而影响了它的外部效度。我们并不能因为这些局限性而丢弃某种技术。相反，我们应该找出这些局限性，并采用科学探究的方法对它们造成的影响进行评估。偏差一直存在的情况下，由这些实验得到的全部证据旨在不断地增加我们对信息检索有效性的理解。可以把全部的证据看做一个总体，并采用一种称为**元分析** (meta-analysis) 的技术从中得到统计推断。当前应用元分析最好的领域，如医药，把所有关于某种治疗方案的已知结果都考虑进去。[⊖]

和很多其他的学科一样，评价外部效度在信息检索评价中也是一个挑战。对于一个实验室实验，我们关注的要点是：可能无法获得一个真实的测试集。例如，TREC 中使用的大部分文档集中不包含垃圾文档 (由判定者判断的看似相关却不需要的文档)；主题通常是为评价而构造的，而不是作为出现在实际环境中的信息需求的例子；这些主题通常经过标准信息检索方法的检查；主题的标题域通常用作搜索查询，很少出现拼写错误。并且，即使可以获得查询的真实样本，给定查询的意图也不总是那么明显，也无法保证相关性判断与用户的信息需求是一致的。另一方面，就算不考虑隐私问题，涉及在线系统和人的实验从逻辑上也是很难实现的，因为需要考虑大量的因素：实验和用户行为间的交互、难以确定的信息需求和相关性，以及为了重复和系统比较而进行的差异控制的难度，等等。最后的结果就是在线评价有它们的局限性：实验代价昂贵且非常耗时，而且通常得不到精确的结果。也就是说，这里介绍的统计查准率估计技术可用于估计实验室和在线度量指标的查准率。

12.3.2 置信区间

置信区间 (confidence interval) $c=[l, u]$是一个区间范围，该区间“可能”包含了一个实验度量值 m 的假设“真实值” (例如，使用某个指定测试集计算得到的 P@k 或 MAP)。“可能”由**置信水平** (confidence level) $1-\alpha$ 进行量化，其中 α 被称做**显著性水平** (significance level)。“真实值” $t=E[M]$ 是 M 的期望值，它是一个随机变量，描述了在所有实质上类似的实验中 m 的可取值的特性，这些实验的不同之处仅仅是它们从目标总体中选择样本的方式不同。设 C 是一个随机变量，描述了由相似的指标得到 c 的可取值的特性。置信水平为 $1-\alpha$ 的置信区间断言

$$\Pr[t \in C] \geqslant 1-\alpha \tag{12-10}$$

⊖ www.gradeworkinggroup.org

一个置信区间可以看做由一个**置信下限**（lower confidence limit）和一个**置信上限**（upper confidence limit）组成的 $[l, u]$ 对，这个 $[l, u]$ 对限制了区间的范围，或者可以把置信区间理解成对估计值的容限（如 $\pm\delta$）。置信水平与置信区间有关系，置信水平一般为 95%，这是必须予以说明的。表 12-1 复制了表 2-5 中第一列和最后一列给出的实验结果，同时对于每个估计值，使用以下将详细讲解的经典方法计算置信水平为 95% 时的置信区间，一并在表中列出。我们可以看到，对于所有结果，置信区间的边界大约为有效性指标值的 ±20%。

表 12-1　有效性评价结果，选用的检索方法（都是本书讨论的）的置信区间的置信水平为 95%。在计算区间时，我们假设主题都是从假设总体中抽样得到的，同时指标误差服从正态（即高斯）分布

方法	TREC45（1998）		GOV2（2005）	
	P@10	MAP	P@10	MAP
余弦（2.2.1）	0.264 (0.19-0.34)	0.126 (0.09-0.16)	0.194 (0.13-0.26)	0.092 (0.06-0.12)
邻近度（2.2.2）	0.396 (0.30-0.49)	0.124 (0.08-0.17)	0.560 (0.48-0.64)	0.230 (0.18-0.28)
余弦（原始TF值）	0.266 (0.19-0.34)	0.106 (0.07-0.14)	0.282 (0.20-0.36)	0.097 (0.07-0.13)
邻近度（TF文档）	0.342 (0.27-0.42)	0.132 (0.10-0.17)	0.466 (0.37-0.56)	0.151 (0.11-0.19)
BM25 (Ch. 8)	0.424 (0.34-0.51)	0.178 (0.14-0.22)	0.534 (0.46-0.61)	0.277 (0.23-0.32)
LMD (Ch. 9)	0.450 (0.37-0.53)	0.193 (0.15-0.24)	0.580 (0.50-0.66)	0.293 (0.25-0.34)
DFR (Ch. 9)	0.426 (0.34-0.51)	0.183 (0.14-0.23)	0.550 (0.47-0.63)	0.269 (0.22-0.32)

置信区间反映了用于度量的实验技术的查准率。当重复使用同一种技术时，结果区间包含真实值的期望概率至少等于置信水平 $1-\alpha$。也就是说，置信区间中不包含真实值的概率不大于 α。因此，我们希望表 12-1 的 28 个区间中大约 5% 的区间（即大约 1 或 2 个）不包含真实值。如果没有额外的信息，我们是无法判断哪些区间不包含真实值的。

对于一个给定的置信水平 $1-\alpha$，一个能得到更小的置信区间的指标会更精确。指标的查准率依赖于样本大小、估计的指标以及估计时使用的方法。设计一个评价信息检索系统有效性的实验时需要权衡样本大小、开销，还需要一方面兼顾指标的实用性，另一方面兼顾指标的查准率和效度。

1. 计算置信区间

在描述如何计算一个置信区间前，我们先简要回顾一下 Fisher 的固定交配得到新生小鼠的实验。新生小鼠的颜色只能是白色或黑色，因此结果完全可以描述为

$$\Pr[\text{Color} = \text{white}] = 1 - \Pr[\text{Color} = \text{black}] \tag{12-11}$$

如果我们的实验输出 m 有很多可能的取值，那就有必要为所有可能的 m 估计 $\Pr[M=m]$。这种估计被称做**概率分布**（probability distribution）。当 m 是一个连续量时，它就是一个可取任何值的无限数。这一分布可以用**概率密度函数**（probability density function）描述。为了计算置信区间，我们感兴趣的不是这个分布自身，而是由**累积密度函数**（cumulative density function）描述的累积分布

$$\text{cdf}(x) = \Pr[M \leqslant x] \tag{12-12}$$

通过使用对 M 的分布的估计计算置信区间。各种计算置信区间的方法的区别是它们如何描述和估计这个分布。

- 一般假设取值是自然数的指标服从**正态分布**（也被称做**高斯分布**（Gaussian distribution）），且均值为 μ，方差为 σ^2。这个分布的概率密度函数是高斯函数

$$\varphi_{\mu,\sigma^2}(x)=\frac{1}{\sigma\sqrt{2\pi}}\cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{12-13}$$

图 12-1 画出了这一函数及其相应的累积密度函数

$$\Phi_{\mu,\sigma^2}(y)=\int_{-\infty}^{y}\varphi_{\mu,\sigma^2}(x)\,dx \tag{12-14}$$

Φ 的计算很复杂，最好使用数学软件完成计算。传统的做法是，通过查询统计表获得 Φ 值。由于经常用到某些 Φ 值，为了很快对这个分布有一个清晰的了解，它们都是很容易被认出和记住的：

$$\Phi_{\mu=0,\sigma=1}(-1.96)\approx 0.025,\qquad \Phi_{\mu=0,\sigma=1}(1.96)\approx 0.975 \tag{12-15}$$

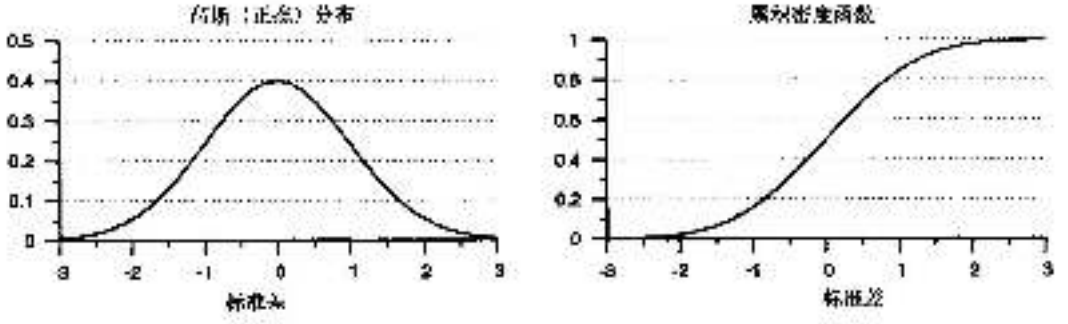


图 12-1 高斯分布的概率密度函数和累积概率密度函数，其均值为 $\mu=0$，方差为 $\sigma^2=1$

- **二项分布**（binomial distribution）来源于统计一系列独立测试得到的正例数。这些独立测试被称做伯努利试验（Bernoulli trials），且对于每个试验，正例出现的概率 q 是一样的。例如，我们现在观察第 $N=100$ 只出生的老鼠，并统计白老鼠的数量 n。参数 N 是试验的次数，q 是正例出现的固定概率，这两个数值完整地描述了二项分布。给定 N 和 q，任意给定值 n 的出现概率就是

$$\binom{N}{n}\cdot q^n\cdot(1-q)^{N-n} \tag{12-16}$$

因为 n 取离散值，所以累积概率是一个求和：

$$\mathrm{cdf}(y)=\sum_{n<y}\Pr(B=n) \tag{12-17}$$

其中，B 是服从某个指定分布的随机变量（在我们的例子中，就是老鼠出生的分布）。图 12-2 画出了二项分布和累积二项分布。

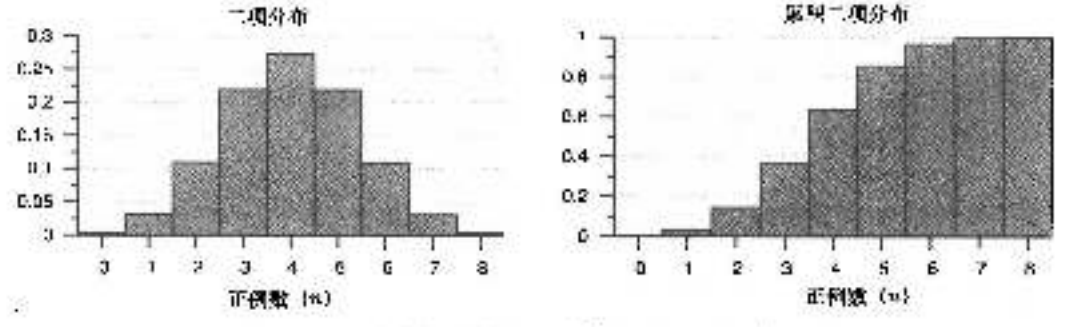


图 12-2 二项分布和累积二项分布，其 $N=8$，$q=0.5$

当 20 世纪初经典统计法得以发展时，计算都是采用手工或者其他非常简单的计算工具，那时计算 N 的常见取值的累积概率是很困难的。由此发展了简单的近似方法以减轻计算负担。虽然现代计算机能准确地计算现实中可能出现的任意 N 值的分布，但这些方法仍常被使用。

其中一种近似方法值得我们注意，而且发现该方法的应用不仅仅是降低计算负担。给定 $0\neq q\neq 1$，当 $N\to\infty$ 时，二项分布接近于均值 $\mu=N\cdot q$、方差 $\sigma^2=N\cdot q\cdot(1-q)$ 的正态分布。根据经验，如果 $\sigma^2>10$，正态分布可以看做是二项分布的一个合理近似。也就是说，一个合理的样本大小是很重要的，同时，一个合理的正例和反例数也是很重要的。

- 正态分布和二项分布都被称做参数分布，因为用一些参数就能完整地描述任一具体分布。因此估计分布的问题就简化为参数估计的问题。另外，一个**经验分布**（empirical distribution）只是一个观察值的多重集，例如，我们的身高测量值 $H=\{1727.2, 1730, 1736.725\}$，或新生老鼠的颜色 $B=\{\text{black}、\text{white}、\text{white}、\text{black}、\text{white}\}$。在经验分布中，任何值的概率就是它本身的频率，因此对于身高，我们有

$$\Pr[H=1727.2]=\Pr[H=1730]=\Pr[H=1736.725]=\frac{1}{3} \tag{12-18}$$

而对于老鼠，我们有

$$\Pr[B=\text{white}]=\frac{3}{5},\qquad \Pr[B=\text{black}]=\frac{2}{5} \tag{12-19}$$

如果这些值是有序的，累积概率是一个简单的求和，例如，

$$\Pr[H\leqslant 1735]=\frac{2}{3} \tag{12-20}$$

更正式的，经验分布可以用离散多重集（或样本），即 s 的值来描述，且假设每个元素出现的概率相等。对于服从这个分布的随机变量 X，我们有：

$$\Pr[X=x]=\frac{|\{i\in s\mid i=x\}|}{|s|} \tag{12-21}$$

累积概率是

$$\Pr[X\leqslant x]=\frac{|\{i\in s\mid i\leqslant x\}|}{|s|} \tag{12-22}$$

经验分布的均值和方差由以下公式给出

$$\mu_s=\frac{1}{|s|}\cdot\sum_{i\in s} i \tag{12-23}$$

和

$$\sigma_s^2=\frac{1}{|s|}\cdot\sum_{i\in S}(i-\mu_s)^2 \tag{12-24}$$

度量一个自然现象（如，人的身高）而产生的经验分布常常与具有相同参数的正态分布相似。

2. 从分布到置信区间

假设已知分布的均值 μ_M 是确定的，计算置信区间这个问题是没有实际意义的：$t=\mu_M$，因此 $c=[\mu_M, \mu_M]$ 的置信度是 100%。为了更好地说明，假设该分布除了 μ_M 以外，所有的信息都是已知的。定义 E 为一个描述指标误差的随机变量，且使得 $M=t+E$。除了均值

$\mu_E=0$，E 的分布与 M 的分布相同。我们首先考虑如何构造置信区间，假设除了 t，分布是已知的，接下来就是如何估计分布。

对于任意 a，$b\geqslant 0$，使得

$$\Pr[E > a] + \Pr[E < -b] \leqslant \alpha \tag{12-25}$$

$c=[m-a, m+b]$是置信水平为 $1-\alpha$ 的置信区间。为了证明这个构造满足这个定义，观察可知，实验度量值 m 是 M 的一个例子，所以 $C=[M-a, M+b]$。重新整理上述不等式，我们有

$$\Pr[t + E > t + a] + \Pr[t + E < t - b] \leqslant \alpha \tag{12-26}$$

$$\Pr[M > t + a] + \Pr[M < t - b] \leqslant \alpha \tag{12-27}$$

$$\Pr[t < M - a] + \Pr[t > M + b] \leqslant \alpha \tag{12-28}$$

一般选择 a 和 b 使得 $a=b$，得到一个关于 m 对称的区间。在某些情况下，可能会把 a 或 b 设成一个已知的极限值，如 0 或 ∞，得到一个**单侧**（one-sided）区间或**单尾**（single-tailed）区间。

给定分布 E 的估计，计算置信区间就很简单了。对于一个对称的区间，会假设分布是对称的，我们发现 $a=b$ 时使得

$$\Pr[E < -b] \leqslant \frac{\alpha}{2} \tag{12-29}$$

对于一个不对称的分布，半对称区间通过找出满足下式的 a 和 b 得到

$$\Pr[E < -b] \leqslant \frac{\alpha}{2}, \qquad \Pr[-E < -a] \leqslant \frac{\alpha}{2} \tag{12-30}$$

对于正态分布，这些值直接通过逆累积密度函数 Φ^{-1} 得到。对于二项分布和经验分布，可以通过使用二分查找找出合适的值。

3. 估计分布

绝大部分的信息检索结果报告只考虑了主题在各种指标上的结果，而假设文档和相关性判断不变。这个假设——主题的选择代表了可能性的唯一来源——使经典的查准率估计方法可用于这些问题。更复杂的估计方法需要考虑文档或相关性判断中的可能性差异。信息检索中关于这些方面的方法已经被提出了（Savoy，1997；Cormack 和 Lynam，2006；Voorhees，2000），但不常使用。

我们这里研究的估计技术假设在某个指标中使用的文档、主题或相关性判断都是独立且随机地从一个假设是无限的总体中选择出来的，并且总体由相似的文档、主题或相关性判断组成。为了便于说明，我们假设各个指标所使用的总体中只有一类（文档、主题或相关性判断）是不同的，而其他的均相同。我们将每个元素看做一个独立的个体（标识为 i），把在某个指标中使用的个体集看做样本 s，且简单地把全部个体视为总体 P。

在上述假设下，可以把指标 m 描述成关于样本 s 的函数 f：

$$m = f(s) \tag{12-31}$$

可以把一个来自于所有可能样本集的样本看做一个服从均匀分布的随机变量 S，同时把 $M=f(S)$看做一个独立变量。在这里，我们不对 f 作任何假设，它仅仅是数学意义上的一个函数：它的取值仅依赖于样本。

4. 经典估计

经典方法假设 f 是某些初等函数 g 的均值，且这些初等函数是关于每个个体的函数，即，

$$f(s)=\frac{1}{|s|}\cdot\sum_{i\in s}g(i) \tag{12-32}$$

例如，如果我们把主题看做是个体，通过令 f=MAP 和 $g=AP$，这一假设得以满足。当不能满足经典假设时，使用一个双射**传递函数**（transfer function）t，可以把 f 转化成合适的 f'和 g'函数对：

$$f'(y)=t(f(y)),\qquad g'(x)=t(g(x)) \tag{12-33}$$

例如，GMAP 是样本 AP 值的几何平均值，而不是算术平均值。这个算法是一个合适的传递函数，因此 f'=log（MAP）和 g'=log（AP）满足假设。传递函数，如 log 和 logit，通常用于各种值上，如取值范围为［0，1］的 AP 值。另一种做法就是使用二项分布（见 12.3.6 节）。

如果 f 满足这个假设，那么总体均值 $E[g(P)]$就等价于真实值 $t=E[M]=E[f(S)]=E[g(P)]$。假设总体分布 $g(P)$的总体方差是 $\sigma^2_{g(P)}$。总体标准差 $\sigma_{g(P)}$ 通常简称为“标准差”。

从样本 s 得到经验分布$\{g(i):i\in s\}$，由这个经验分布我们估计（总体）标准差。转而将这个估计用于计算**标准误差**（standard error）σ_E，由此得到置信区间。一个简单的方法就是采用实验采样方差作为总体方差 $\sigma^2_s=\sigma^2_{g(P)}$ 的估计。那么误差的方差就是 $\sigma^2_E=\sigma^2_M=\frac{\sigma^2_{g(P)}}{|s|}=\frac{\sigma^2_s}{|s|}$。给定这个方差的估计值，那就可以使用逆正态分布计算置信区间。

这个简单的方法涉及两个近似误差，对于小样本空间，这两个误差就比较明显了。首先，抽样方差并不是总体方差的最好估计，可以通过样本大小$|s|=1$ 为这个极端的例子来说明。经验分布 s 的方差是$\sigma^2=0$，这肯定不是一个好的估计值。行话“样本空间大小为 1”等价于“查准率未知的指标”。对于小样本空间$|s|>1$，依然存在问题，可以用$|s|=2$ 这一情况来说明。在这种情况下，不难证明 $\sigma_s\approx\frac{1}{2}\sigma_{g(P)}$。概括来说，我们得到了更好的估计，与介绍的统计文本类似：

$$\sigma^2_P\approx\frac{\sum_{i\in s}(i-\mu_s)^2}{|s|-1} \tag{12-34}$$

注意，这条公式与那条公式的不同之处：σ^2_s 中分母是$\nu=|s|-1$，而这里是$|s|$，其中ν表示自由度（degrees of freedom）。

第二种近似误差是来源于度量 σ^2_s 产生的不确定性。上述公式假设 σ^2_s 是“真实”方差，这里真实就是前面所述的概念。如果我们使用任何指定值以及相应的正态分布 Φ_0，σ^2_s，那么置信区间会反映该分布关于 σ^2_s 的误差。对于 $N=|s|>30$ 的情况，可能误差不显著，但对于 $N\leqslant 30$ 的情况，误差是显著的。由 William Sealy Gosset 提出的 ***t*-分布**（t-distribution）弥补了这一缺陷。Gosset 在发表这篇论文时，采用了化名为 The Student，因为他在 Guinness brewing 公司工作，而他的这一发现被认为是商业秘密。Student 的 t-分布（图 12-3)是所有的 Φ_0，σ^2_s 权重的均值，且每个的权重由自由度为ν的 σ^2_s 的概率分布确定。也就是说，t-评分是一个“真实”分布的估计，而且就是我们所说的真实。回想采样空间大小为 $N=\nu+1$。

计算置信区间的程序与那些简单方法是一样的，除了在累积 t-分布中采用了$\nu=N-1$ 替代 Φ。就像计算 Φ 一样，t 的计算工作也最好交给数学软件完成。除非特别说明，假设我们所给出的结果和统计软件包都是使用这种经典的估计方法的。对于 $N\leqslant 30$ 的情况，应该

使用 t-分布；对于 $N>30$ 的情况，使用 Φ 很好。

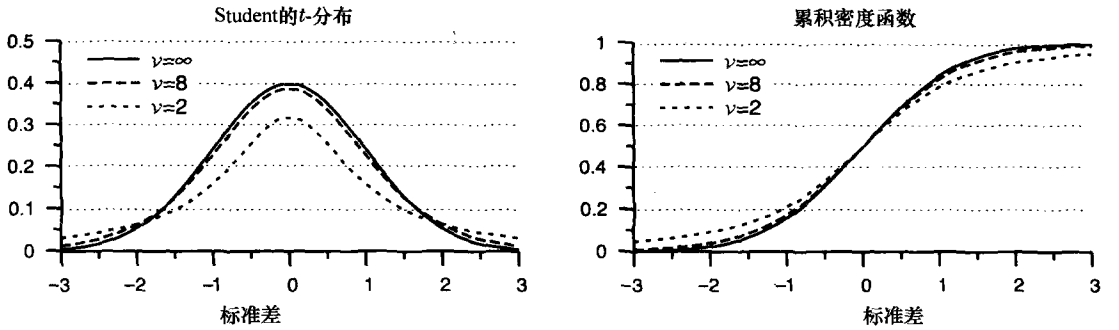


图 12-3 Student 的 t-分布的概率密度函数和累积密度函数，自由度分别为$\nu=2$，$\nu=8$ 和$\nu=\infty$（自由度为$\nu=\infty$的 t-分布等价于高斯分布）

5. bootstrapping

bootstrap（Efron 和 Tibshirani，1993）是一种通过采样样本 s 来模拟经验分布模型 $f(S)$的方法。因为“采样样本”听起来有点拗口，那我们就把这个过程称为**重采样**（resampling）。尽管 bootstrap 可以得到与经典方法类似的结果，但是它不受关于 f 或总体的假设的限制，因此它可应用于无法使用经典方法的情况中。

重采样的过程很简单：我们用样本 s 替代源总体。然后构造一个与 s 独立同分布的多重集$R=\{r_j \mid 1\leqslant j\leqslant k\}$，且每个集合的大小都与 $s(|r_j|=|s|)$一样。独立条件隐含着来自 s 的一些样本可能在某些 r_j 中重复出现，然而某些样本可能没有出现。也就是说，r_j 中的个体由抽取自 s 中的所替代。假设重复样本替代了假设无限的总体中的其他样本，且在计算 f 时是一样的。例如，在统计白老鼠和黑老鼠时，是统计某只黑老鼠 10 次，还是统计 10 只不同的黑老鼠一次是没有区别的。

一旦 R 构造好，可以通过以下两种方法中的一种从经验分布 $f(R)$中计算置信区间：

1）通过估计 $\sigma_M=\sigma_{f(S)}\approx\sigma_{f(R)}$并使用上述的累积正态分布 Φ。

2）通过计算大量的辅助样本（$|R|\gg100$）并直接使用累积经验分布（即$\Pr[f(s)<x]\approx\Pr[f(R)<x]$）。

12.3.3 比较评价

让我们回到原来的主要目标：评价信息检索方法有效性并使用统计推断扩大所得结果的运用。实际中，我们通常对单独评价一个系统的有效性这一做法并不感兴趣，我们更感兴趣的是将这些结果与其他系统的结果进行比较。表 12-1 给出的结果和置信区间让我们对列表中的各种方法的相对性能有了一个总体印象。然而，除了表中最后一行的 Cosine（raw TF）（2.2.1 节）的性能要明显好于其他三种方法，我们不能从这些结果肯定地推断出哪个方法比其他的方法更好。

设 m_A 为对系统 A 的有效性评价，m_B 为对系统 B 的有效性评价。已知 m_A 和 m_B 是在类似的条件下进行评价的，如果 $m_A>m_B$，那么我们可以认为系统 A 比系统 B 更有效。但我们不能知道以下这些方面：

- 系统 A 比系统 B 有效多少？
- 这一差异是实质性的吗？

• 证明这一差异是实质性的证据有多强？

指标的差 $m_{A-B}=m_A-m_B$ 可用于解决第一个问题，而对于第二个问题，根据预期的目标，可以考虑需要多大的差值来区分 A 和 B 来解决。例如，当比较两个人的身高时，很难想象身高差值小于 1mm 的情况能说明什么问题，因此我们会把任何类似这样的差值认为是不重要的。如果度量得足够仔细，也许可以判断这两人的身高差不足 1 mm，但在大多数情况下都不能说“A 比 B 高”。第三个问题——考虑证据的强度——可以通过度量差值的置信区间来解决。如果置信区间只包含了实质性差值，我们可以认为 A 确实比 B 好的置信度为 $1-\alpha$。如果该区间仅包含了非实质性差值，我们可以认为 A 与 B 的差别不是实质性的置信度为 $1-\alpha$。如果该区间包含实质性差值和非实质性差值，那它们的比例就反映出相应概率的平衡度。

表 12-2 根据表 12-1 的内容给出了各个系统的对比结果，且表 12-2 中的内容根据 TREC 主题 351～400 的 MAP 值排名。表中给出了两个连续排名系统间的差值，带置信区间和 p-值，其中 p-值是使用以下讨论的经典方法估计得到的。

表 12-2　对各种评分函数的 MAP 结果两两进行对比，且置信区间的置信水平是 95%，同时 p-值表示各种方法间的差值的显著性水平

方法 A	方法 B	TREC45 (1998)	
		MAP_{A-B} (95% c. i.)	p-值
LMD	DFR	0.010 (-0.002 - 0.021)	0.09
DFR	BM25	0.005 (-0.004 - 0.013)	0.29
BM25	Cosine (TF docs)	0.046 (0.018 - 0.075)	0.002
余弦 (TF文档)	Cosine	0.006 (-0.021 - 0.033)	0.66
余弦	Proximity	0.002 (-0.036 - 0.040)	0.92
邻近度	Cosine (raw TF)	0.018 (-0.021 - 0.056)	0.36

1. 显著性

证据的强度有时也称为**统计显著性**（statistical significance），或更常见是**显著性**（significance）。“显著性差值”意思是置信水平为 $1-\alpha$ 的置信区间不包含 0；即有很强的证据表明它们之间存在差别，不管这差别是否是实质性的。如果没有对差值或实质性的估计，从显著性得到的有用信息就微乎其微了。显著性并不意味着实质性：一个“显著性”指标可能是一个强的非实质性证据，或者是一个强的实质性证据，或都不是。

2. P-值

另一种报告一个带固定显著性水平 α（一般是 0.05）的置信水平的方法是报告一个显著性估计或 p-**值**（p-value）。给定一个指标 p 是使得结果在显著性水平 α 下是显著的最小的。换句话说，这是最大的不包括 0 的置信区间可取的 α 值。与指标的差 m_A-m_B 一起使用，p-值的估计查准率差不多与置信区间一样。置信区间保持 α 不变并计算区间的大小；p-值为一个特定区间计算 α，该特定区间包括 m_A-m_B 且有一区间边界为 0。使用相同的累积概率估计计算置信区间和 p-值。同时报告这两个估计并没有坏处，因为它们从不同的角度解释了同样地查准率估计。由于上述原因（将在下节展开），永远不要报告一个不带 m_A-m_B 这一量化估计的 p-值。

尽管 p-值一般用于度量差别，但也可以为任何指标 m 计算 p。一个双侧的 p-值表示关于任一包含 m 的对称或半对称置信区间 $c=(0, x]$ 或 $c=[x, 0)$ 的显著性水平 α。对于

对称区间，我们有 $x=2m$。给定从一个普通的总体中独立地选出的 A 和 B，那么差值 m_{A-B}必须得到一个对称区间。

单侧 p-值假设 $t\geqslant 0$（即，它假设度量效果的“真实”值大于 0）且选择一个 α 使得 $c=(0,\infty]$。一般来说（假设服从对称或半对称分布），对于相同的实验，双侧 p-值正好是单侧 p-值的两倍。知道 p-值是单侧还是双侧是很重要的，同时，单侧 p-值仅当满足在评价前已明确声明 $t\geqslant 0$ 这一假设时才能使用，这点也是很重要的。一旦知道度量结果 m，再去构造这一假设就太晚了。特别是，如果随机从一个普通总体中选出 A 和 B，那么使用单侧p-值去估计 m_{A-B}的显著性永远都不是一种合适的做法。

除了这些问题，单侧 p-值对评价实质性还是有用的。考虑以下情况：我们希望判定是否 $t_{A-B}>\delta$，其中 δ 是我们认为是实际上的最小数量。关于 $t'=t_{A-B}-\delta$ 的单侧 p-值使我们能判断 A 实质上比 B 好这一证据是否显著。

这里介绍的方法在某些领域中已经得到大家的认可，如医学。度量差值称为**效果量**（effect size），实质性也称为**实质显著性**（substantive significance）或**临床显著性**（clinical significance）。在医学中，结果不再是可供发表的，除了假设、预测结果、用于测试结果的指标和方法要在做实验前提交到一个公共注册处（De Angelis 等人，2004）。这个标准帮助保证假设和用于测试这些假设的方法符合实际且经得起检查。这些收集回来的证据记录在公共注册处，随着实验的增多，它们的可信度也不断增大。

12.3.4 被认为有害的假设检验

正如前一节所述，对比起仅是估计系统差异是否“以显著性水平 α 是显著的（对于任意给定的 α）”而言，系统之间差异的度量以及对该度量查准率的估计（例如：置信度区间的形式）能提供更多的信息。如此使用估计的做法被称做**假设检验**（hypothesis test），我们应该避免假设检验，且在如医药这样的领域中已不再接受这种做法了。

假设检验起源于对大约 1 个世纪前 Ronald Fisher 和 Karl Pearson 间的一场哲学辩论的误解，他们是现代统计学的先驱。这场辩论的基本思想是合理的：在做任何评价前，你应该确定你需要的是什么。简言之，做出你的决定。更正式地来说，Karl Popper 这一有影响力的哲学科学家信奉的科学方法是：要求你构造一个证伪**研究假设**（research hypothesis），然后，如果假设是非真的，使用这个研究假设去预测一个不可能事件。如果对事件的预测正确，那在研究假设中置信度增加。但事件通常可能因为某些其他原因发生，而不是因为假设是正确的；α 是事件随机发生的概率的估计上界。

研究假设通常也被称为**对立假设**（alternative hypothesis），用符号 H_A 表示。对立假设与**零假设**（null hypothesis）对立，零假设用符号 H_0 表示，它假设预测事件 e 随机发生。一个显著性结果断言

$$\Pr[e\,|\,H_0]\leqslant\alpha \tag{12-35}$$

假设检验被大量滥用的原因有以下几点：

1）空假设假定 $t_A\neq t_B$（即 $t_{A-B}\neq 0$）。差值 t_{A-B}一般是一个实数，且服从一个连续分布，因此 $\Pr[t_A=t_B\,|\,A\neq B]\approx 0$。也就是说，$H_0$ 的定义是错的。你所需做的就是做一个实验去证明，其中所采用的评价指标要有足够的查准率。

2）相对好一点的对立假设是 $t_A>t_B$，这至少给空假设 $H_0\equiv t_A\leqslant t_B$ 留了一线机会。首先评价 m_A 和 m_B，然后基于观察 $m_A>m_B$ 构造假设，最后基于单侧检验断言显著性，这种做法太普遍了。这种断言是误导人的，因为实际上它检验复合假设 $m_A>m_b\vee m_A<m_B$，

这等价于我们的空假设 $m_A \neq m_B$，α 被低估了 2 倍了。这并不是说单侧检验的不正确可以通过扩大 α 两倍就能纠正：没有先验假设 $t_A > t_B$，单侧检验基本没有用，就像双侧检验一样。一个不对的科学结果并不能通过任何简单的调整就能修正好。

3）根据固定的显著性阈值 α 判断“显著的”或“不显著的”，就算证据是非常极端的，在“不是显著的”这一情况下也不能表明证据的权重。也就是说，只要是证据就行，不管是弱证据，还是支持或证伪对立假设的证据。

4）隐情是：仅一个分类判断（“显著的”或“不是显著的”）是无法将其查准率估计推导出的各种信息联系在一起的，因此容易受很多似是而非的理解的影响就很正常了：

- 断定一个显著的结果证明了对立假设是不正确的。
- 断定一个显著的结果证明了对立假设为真的概率$\geqslant 1-\alpha$ 是不正确的。
- 断定一个显著的结果证明了空假设为假的概率$\geqslant 1-\alpha$ 是不正确的。
- 断定一个不显著的结果是对立假设的对立证据是不正确的。
- 断定一个不显著的结果不会给对立假设提供任何正面证据是不正确的。
- 给一个对立假设做总体评价时忽略不显著的结果是不正确的。
- 把显著性作为效应量的估计是不正确的。
- 把显著性作为结果的实质性估计是不正确的。

正确使用对立假设要避免对全部显著性描述作明确判断。度量差别的指标，加上一个置信区间或显著性的量化估计（p-值），肯定可以提供更多的信息。遗憾的是，大部分公布的信息检索研究都报告了假设检验的结果，但是却没有报告 p-值或置信区间。当我们尝试理解这些结果时，需要修正一些信息：

- 寻找差值幅度的估计，如果为真，判断这个差值是否是显著的。如果可以得到各个指标的值，那差值估计就可以简单地通过求这些指标的差值获得。
- 确定从相同的学习或实验中丢弃的不显著的结果的数量 u。采用 Bonferroni **校正**（Bonferroni correction）使得 $p < (u+1) \cdot \alpha$。这个校正假设 H 中的任何一个 $u+1$ 结果都为真，记这假设为 H_0，那就表示没有一个结果为假。假设 H_0 为真，那么可认为每个结果“是显著的”的概率是 α。所以当 α 取值较小时，某些“是显著的”的结果的组合概率可能高达

$$1-(1-\alpha)^{u+1} \approx (u+1) \cdot \alpha \tag{12-36}$$

因此，关于 H 的证据的组合权重是 $p < (u+1) \cdot \alpha$。如果很多结果是显著的，或结果是相关的（就如相关评价或排名结果间的两两差别），更复杂的方法，如 Bonferroni-Holm 校正（Holm，1979），可能会得到一个更严格的估计。

12.3.5　配对和未配对差值

度量 m_{A-B} 的一种简单方法是分别计算 m_A 和 m_B，然后计算 $m_{A-B} = m_A - m_B$。要估计 m_{A-B} 的精确度，可以先分别对 m_A 和 m_B 作参数估计，然后对它们的误差分布的方差求和：

$$\sigma^2_{E_{A-B}} = \sigma^2_{E_A} + \sigma^2_{E_B} \tag{12-37}$$

双侧 p-值的粗略估计可以通过评价指标，以及它们对应置信水平为 $1-\alpha$ 的置信区间 m_A、c_A、m_B 和 c_B 获得。如果 $m_A \in c_B$ 或 $m_B \in c_A$，我们就可知道 $p \gg \alpha$。如果 $c_A \cap c_B = \emptyset$，我们就可知道 $p \ll \alpha$。给定 c_A 和 c_B，我么可以根据它们的大小分别估计它们的标准差 σ_{E_A}

和σ_{E_B}，有效地逆转了它们间的计算。根据经验，如果$|c_A \cap c_B| < \frac{1}{2\sqrt{2}}|m_A - m_B|$（即，重叠区间少于$m_A$与$m_B$之间的距离的$\frac{1}{3}$），我们可以假设$p<\alpha$。更准确地说，如果每个区间的大小除以$\sqrt{2}$，那么当$p=\alpha$时，这些小区间会首尾连接但不会有重叠。

配对差值利用m_A和m_B都同是样本s的均值这一标准假设；即，

$$m_A = f_A(s) = \frac{1}{|s|} \cdot \sum_{i \in s} g_A(i) \tag{12-38}$$

$$m_B = f_B(s) = \frac{1}{|s|} \cdot \sum_{i \in s} g_B(i) \tag{12-39}$$

在这一假设下，差值可以用两种不同的方法计算，得到估计m_{A-B}和m'_{A-B}，对样本s来说它们都是一样的，但是它们的误差E和E'的分布不同。

$$m_{A-B} = f_A(s) - f_B(s) = \frac{1}{|s|} \cdot \sum_{i \in s} g_A(i) - \frac{1}{|s|} \cdot \sum_{i \in s} g_B(i) = \frac{1}{|s|} \cdot \sum_{i \in s} g_{A-B}(i) \tag{12-40}$$

$$m'_{A-B} = f(s) = \frac{1}{|s|} \cdot \sum_{i \in s} (g_A(i) - g_B(i)) = \frac{1}{|s|} \cdot \sum_{i \in s} g_{A-B}(i) \tag{12-41}$$

其中，$g_{A-B}(x) = g_A(x) - g_B(x)$。

在大部分情况下，$g_A(i)$和$g_B(i)$间有很强的正相关，所以尽管$m'_{A-B} = m_{A-B}$，但误差的方差是非常小的：

$$\sigma^2_{E'} \approx \frac{\sigma^2_{g_{A-B}(s)}}{|s|-1} \ll \sigma^2_E \approx \frac{\sigma^2_{g_A(s)} + \sigma^2_{g_B(s)}}{|s|-1} \tag{12-42}$$

对于相同的α，配对差值法得到更紧的置信区间，同时对于相同的差值，会得到更小的p-值。

12.3.6 显著性检验

表12-3给出的关于评价结果的例子让我们知道计算p-值的配对差值的三种经典方法：t-检验、符号检验和Wilcoxon符号秩检验。一般来说，计算关于某个差值的p-值的方法与计算关于某个差值的置信区间的方法是一样的：通过估计累积误差概率并用它来解出p。对于单侧显著性

$$p = \Pr[E_{A-B} \geqslant m_{A-B}] \tag{12-43}$$

如以前一样，E_{A-B}是误差的指标，同时对于双侧显著性

$$p = \Pr[|E_{A-B}| \geqslant |m_{A-B}|] \tag{12-44}$$

即，

$$p = \Pr[E_{A-B} \geqslant |m_{A-B}|] + \Pr[E_{A-B} \leqslant -|m_{A-B}|] \tag{12-45}$$

由这些公式可以明显看出，单侧p-值表示正差异值至少与评价值m_{A-B}一样大的概率，而双侧p-值表示任何差异值至少与评价值m_{A-B}一样大的概率。另外更明显的是，对于对称分布，双侧p-值是单侧p-值的两倍，这种情况对于两个相似指标间的差异值的分布是很常见的。

表 12-3　在 TREC 主题 351～358 上比较 BM25 和 LMD。m_{BM25} 和 m_{LMD} 分别表示对应方法关于 8 个主题的有效性评价值（AP）。$m_{BM25-LMD}$ 表示各个主题间的差值，且这些值是采用 t-检验得到的。w_{AB} 和 r_{AB} 分别指胜率和符号秩，这两个值分别是采用符号检验和 Wilcoxon 检验得到的

主题 Id	351	352	353	354	355	356	357	358
m_{BM25}	0.343	0.040	0.223	0.114	0.078	0.012	0.294	0.134
m_{LMD}	0.409	0.045	0.311	0.105	0.149	0.019	0.311	0.105
$m_{BM25-LMD}$	−0.066	−0.005	−0.088	+0.009	−0.071	−0.007	−0.017	+0.029
w_{AB}	0	0	0	1	0	0	0	1
r_{AB}	−6	−1	−8	+3	−7	−2	−4	+5

1. t-检验

计算 E_{A-B} 的方差的简单方法（公式（12-37））可用于计算上述的 p-值。当无法获得原始数据或指标值不是样本均值的时候，这个方法就可用于把结果组合起来。但如果 m_A 和 m_B 是样本均值，同时如果可以获得经验分布的均值（正在表 12-3 所示），可以将它们组合起来得到对 $\sigma^2_{E_{A-B}}$ 的更好估计，从而得到更小的 p-值。（非配对）t-检验有效地计算出两个联合分布的均值，这是对 m_B 的一个补充。相比于上述的那个简单方法得到的结果，经验分布得到的结果是：大小是单个分布大小的两倍（假设两个样本的大小都一样），差不多是 $\sigma^2_{E_{A-B}}$ 的一半，且自由度是 $\mathcal{V}_{E_{A-B}}$ 的两倍。表 12-4 给出了参数估计和 p-值的对比结果。最终结果是，通过使用 t-检验，上述简单方法从 $p=0.9$ 降到 $p=0.7$。

表 12-4　来自表 12-3 中各个系统间进行比较的统计摘要结果和检验结果。给出单独的均值和分布的估计作为参考。简单方法、配对 t-检验和非配对 t-检验，使用 t-分布和不同的方差估计来估计差值的精确度。符号检验使用胜率和二项分布。Wilcoxon 检验使用符号秩和以及对应的 t-分布。有意义的查准率估计以 95% 的置信区间和 p-值（双侧）的形式给出

指标	误差分布	估计	检验	p-value
m_{BM25}	t ($\nu=7, \sigma=0.12$)	0.155 (0.05, 0.26)		
m_{LMD}	t ($\nu=7, \sigma=0.14$)	0.182 (0.06, 0.30)		
$m_{BM25-LMD}$	t ($\nu=7, \sigma=0.18$)	−0.027 (−0.4, 0.33)	公式 (12-37)	0.9
$m_{BM25-LMD}$	t ($\nu=14, \sigma=0.07$)	−0.027 (−0.17, 0.11)	非配对 t	0.7
$m_{BM25-LMD}$	t ($\nu=7, \sigma=0.015$)	−0.027 (−0.06, 0.01)	配对 t	0.1
$w_{BM25-LMD}$	二项分布 ($N=8, q=0.5$)	0.25 (0.03, 0.65)	符号	0.3
$r_{BM25-LMD}$	Wilcoxon T ($N=8$)	−20	Wilcoxon	0.2

配对 t-检验使用的就是 12.3.5 节中详细讲述的配对差值法。如果满足使用它所需的条件，那几乎可以肯定得到的估计的精确度会比非配对 t-检验好（表 12-4，$p=0.1$）。由于这个原因，配对 t-检验是信息检索中最常见的一种检验。

t-检验和配对 t-检验都依赖于假设：误差 E 服从正态分布。当 m 是一个大样本的均值时，这个假设是合理的，因为由**中心极限定理**（central limit theorem）有：当 N 趋于无穷时，N 个服从相同分布（不必非得正态）且方差 $\sigma<\infty$ 的独立变量的均值接近方差为 $\frac{\sigma}{N}$ 的正态分布。

不能使用 t-检验的唯一原因是，m 不是一个样本均值或 N 太小以致不能对正态分布得

到一个好的近似。在大部分这些情况下，可行的替代方法是 bootstrap。bootstrap 不属于经典检验方法的原因是：尽管与这些经典方法同样地古老，但把 bootstrap 用作正式的统计工具还是最近的事；也许是因为该方法偏向计算密集型，所以在经典方法得以发展后，才发现 bootstrap 具有实用性。

2. 符号检验和 Wilcoxon 符号秩检验

符号检验和 Wilcoxon 符号秩检验使用指标 $\hat{m}_{A-B} \approx t_{A-B}$ 替代 m_{A-B}，其中可把≈简单理解为

$$\Pr[\operatorname{sgn}(\hat{m}_{A-B}) = \operatorname{sgn}(t_{A-B})] \gg 0.5 \tag{12-46}$$

因为这个原因，相比于估计差值的幅度，这些检验更符合假设检验的原来目的。它们量化了差别存在的证据强度，但对差别的幅度做出了细致的观察。尽管符号检验和 Wilcoxon 检验度量错误量级，也就是说，当无法满足配对 t-检验的假设时，可以使用这两种方法。

对于符号检验，以下的指标是信息检索系统的相对有效性的指标。以下是前面定义的 g_{A-B} 的**胜率**（win rate）

$$w_{AB} = \frac{|\{i \in s \mid g_{A-B}(i) > 0\}|}{|\{i \in s \mid g_{A-B}(i) \neq 0\}|} \tag{12-47}$$

$w_{AB} > 0.5$ 的意思是在超过一半指标中，A 比 B 更有效。由 w_{AB} 可以很容易地得到一个合适的替代指标：

$$\hat{m}_{A-B} = w_{AB} - 0.5 \tag{12-48}$$

随机变量 w_{AB} 服从参数为 $N = |s|$ 和 $q = t_{AB} = E[W_{AB}]$ 的二项分布，这是真正的赢率，其中 W_{AB} 是一个例子。误差 $E_{AB} = W_{AB} - t_{AB}$ 的分布主要依赖于未知的 t_{AB}。为了估计单侧 p-值，我们假设最差的情况，同时选择 t_{AB} 以便最大化

$$p = \Pr[t_{AB} + E_{AB} \geqslant w_{AB}] \tag{12-49}$$

不难证明当 $t_{AB} = q = 0.5$ 时，取 p 最大值，这也是单侧符号检验的结果。对于双侧检验，将这个结果翻倍即可。

我们知道 $W_{AB} = 0$ 和 $W_{AB} = 1$ 是特殊情况，在这两种特殊情况中，即使当已用上双侧 p-值，单侧 p-值也是合适的。这是因为不可能预先知道 $W_{AB} < 0$ 和 $W_{AB} > 1$。当使用置信区间时，应该计算单侧置信区间以得到指定的置信水平 $1-\alpha$。

对于表 12-3 中列出的 8 个主题，有两个主题的 $m_{\text{BM25}-\text{LMD}} > 0$，有 6 个主题的 $m_{\text{BM25}-\text{LMD}} < 0$，所以 $W_{AB} = 0.25$ 且 $\hat{m}_{A-B} = -0.25$。对于 $N = 8$ 且 $q = 0.5$ 的二项分布，我们有

$$p = \Pr[|E| \geqslant |-0.25|] = 2 \cdot \Pr[W_{AB} \leqslant 0.25] \approx 0.3 \tag{12-50}$$

Wilcoxon 检验用符号秩差值替代 m_{A-B}

$$r_{AB} = \sum_{i \in s} rnk(g_{A-B}(i)) \cdot \operatorname{sgn}(g_{A-B}(i)) \tag{12-51}$$

其中 $rnk(x) = |\{i \mid x \leqslant g_{A-B}(i)\}|$，如果 $i \neq j$，那在这一简化假设下，有 $g_{A-B}(i) \neq g_{A-B}(j)$。放松这个假设将给分析带来意想不到的复杂性，但是为了实际应用，通常随机打破平局或赋予一个平均排名是有意义的：

$$rnk(x) = \frac{1}{2} + \frac{1}{2} \cdot |\{i \mid x = g_{A-B}(i)\}| + |\{i \mid x < g_{A-B}(i)\}| \tag{12-52}$$

检验仿照假设：差值的相对幅度是非常重要的（尽管不是它们的精确值），因此，例如（主题 351）$m_{\text{BM25}-\text{LMD}} = -0.066$ 比 $m_{\text{BM25}-\text{LMD}} = +0.009$（主题 354）更重要。处理过程如下：对所有差值的绝对值大小按升序进行排序，并根据它们在排名列表中的排名赋予相应的

整数权重。例如，赋予主题 352 的差值权重为 1，因为在表中的 6 个主题中它的差值是最小的；而主题 353 的差值权重是 6。然后这个检验计算权重的总和，分别按权重是正和负对其进行统计。统计时根据差值的符号进行：

$$r_{AB} = -6+1-8+3-7-2-4-\cdots-20 \tag{12-53}$$

对于给定的 N，它可能的符号秩差值的分布称为 Wilcoxon T 分布（Wilcoxon T distribution）（如图 12.4 所示）。可以通过枚举所有 2^N 种可能的结果来计算累积概率。对于这个例子，共有 $2^8=256$ 种可能的结果，主题度值为 20 或更大的有 50 个。Wilcoxon 检验因此报告 p 值为 $50/256\approx0.2$。

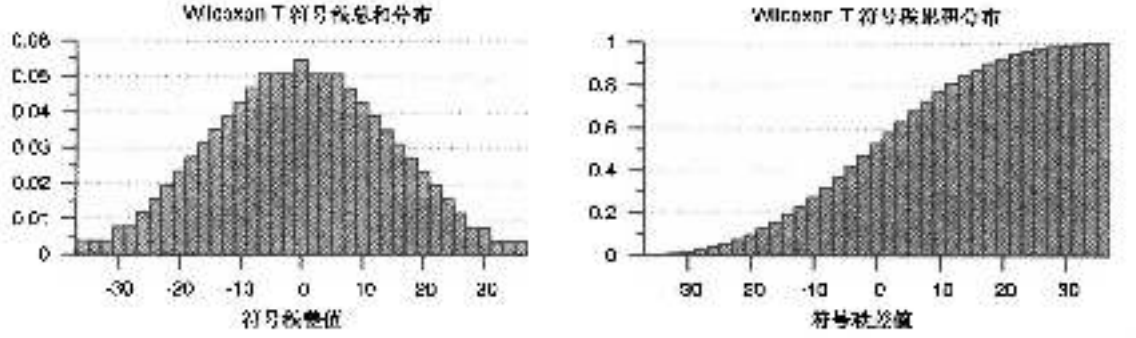


图 12-4　Wilcoxon T 符号秩总和分布以及累积分布，其中 $N=8$

一个基于替代指标的 p-值应该带有一个真实差值的估计。作者应该明确指出使用的什么检验，同时读者应该注意真实差值的估计查准率是未知的。连同实质性阈值 δ，替代检验能提供更多有用的信息。一个确定置信区间为 $m_{A-B}>\delta$ 的替代方法比一个确定置信区间为 $m_{A-B}>0$ 的替代方法更有用。

12.3.7　统计检验的效度和检验力

统计检验的目的是估计似然值：某个指标 m 可用概率来解释。如果估计得到的似然值与预期的一样，那么这个检验就是有效的；如果当指标实际上不是由概率决定时，估计值会很小，那么这个检验就是有检验力的。效度和检验力一起刻画了检验对其预期目标的适用性。效度通常简单地假定，而检验力则使用**检验力分析**（power analysis）去估计，这里，本身假设是有效的。为了比较信息检索系统，我们考虑设计一个元实验去判定统计检验的效度和检验力。

考虑这样一个信息检索实验 x (A_x, B_x, s)：使用主题集 s 评价两个系统 A_x 和 B_x 的有效性差值。设 m_x 是根据某种如 MAP 这样的有效性指标得到的测量差值。设 p_x 或 c_x 对应由某些统计检验方法计算得到的 p-值或置信区间。

设 $X_n=(A_x,B_x,S)$ 是一个独立的实验，使用相同的方法不同的主题集 S 来比较两个相同的系统。其中 $S|=|s|$。并且，让 M_x 为使用 S 得到的两个系统 A_x 和 B_x 间的差值。这个差值的真实值 t_x 为

$$t_x = \mathbb{E}[M_x] \tag{12-54}$$

使用可能的模型 M_x 计算 p_x。该模型把误差为 $E_x=M_x - t_x$ 的累积概率分布近似为：

$$M_x(e)\approx \Pr[E_x \leqslant e] \tag{12-55}$$

如果公式（12-55）在一个合理的范围内对有所的 x 和 e 成立，那么由这条公式得到的 p-值

或置信区间是正确的，同时我们可以说统计检验是有效的。

一般来说，t_x 是未知的，所以不能直接建立效度。假设有独立于 X_x 的副本 $X_x^{[1]}$ 和 $X_x^{[2]}$。对应的指标 $M_x^{[1]}$ 和 $M_x^{[2]}$ 有相同的分布，误差项是 $E_x^{[1]}$ 和 $E_x^{[2]}$。现在考虑差值

$$D_x = E_x^{[1]} - E_x^{[2]} = (M_x^{[1]} - t_x) - (M_x^{[2]} - t_x) = M_x^{[1]} - M_x^{[2]} \tag{12-56}$$

给定由 $\mathcal{M}_x^{[1]}$ 得到的模型 $X_x^{[1]}$，我们可以为 D^x 建立一个模型 $D_x^{[1]}$，并将其与通过元实验得到的经验分布进行比较，以验证 $D_x^{[1]}$ 的有效性。如果对于相应的 x 和 d，有 $D_x^{[1]}(d) \approx Pr[D_x \leqslant d]$，我们可以肯定地推断：$D_x$ 对所有 x 都是有效的，同时 $\mathcal{M}_x$ 也是一样。

假设检验是有效的，那对应于显著性水平为 α 的检验力是

$$power_\alpha = \Pr[p_x \leqslant \alpha \mid \overline{H_0}] \tag{12-57}$$

对于双侧检验 $H_0 \equiv t_x = 0$，根据经验，这个检验力可以从实验的一个样本（且 $t_x \neq 0$）估计得到。采用一个 $m_x \neq 0$ 的实验可以很好地近似这样的样本，因为当 $m_x \neq 0$ 时，不大可能出现 $t_x = 0$，因此可能性可能不准确。对于一个单侧检验，必须预先选择 $H_0 \equiv t_x \leqslant 0$ 或 $H_0 \equiv t_x \geqslant 0$。在 $t_x \geqslant 0$ 和 $t_x \leqslant 0$ 是等概率这一简单假设下，显著性水平为 α 的单侧置信区间的有效性与显著性水平为 2α 的双侧置信区间的有效性是一样的。

比起统计检验的相对有效性来说，检验力的经验估计是有用的，因为在一个总体上进行实验时，发现某种方法的检验力比另外一种方法更好，那在另外一个不同的总体上进行实验得到的结果，很大可能也是该方法的检验力比另外一种方法更好。为了预测某个实验的检验力，必须使用检验力分析从假设的总体分布中构造一个模型 $\mathcal{M}_x$，其中主题数为 $|s|$，估计的真实差值为 t_x，且估计的样本方差为 σ_s^2。

1. 度量效度和检验力

把 TREC 2004 Robust 专题上的结果作为实验的一个总体，每次从 110 个运行实例中取出两个进行比较，每个运行实例都提交给专题以计算 MAP，这里每个运行实例使用 249 个主题的一个子集作为评价材料。为了计算 t-检验、符号检验和 Wilcoxon 检验的效度和检验力，我们构造相似实验对 $\langle x^{[1]}, x^{[2]} \rangle$，其中 $x^{[1]} = \langle A_x, B_x, s^{[1]} \rangle$，$x^{[2]} = \langle A_x, B_x, s^{[2]} \rangle$；$s^{[1]}$ 和 $s^{[2]}$ 每个都由 124 个主题组成，这些主题都是从专题使用的 249 个主题中以不放回的方式随机抽出的，因此它们之间没有交集。对于总共 191 840 个样本，构造这些实验对以便每个不同的实验对出现 32 次。在这里，不取 32，因为 $m_x^{[1]} = 0$，剩下 191 808 对实验。

为每对实验构造 $\mathcal{M}_x^{[1]}$ 并得到 $D_x^{[1]}$，是在不使用 x_2 的情况下用于估计 $m_x^{[1]}$ 和 $m_x^{[2]}$ 间的**不一致**（discordance）的概率。

$$d_x = \Pr[m_x^{[1]} \cdot M_x^{[2]} < 0] \approx \mathcal{D}_x^{[1]}(-|m_x^{[1]}|) \tag{12-58}$$

尽管不可能验证任何一个 d_x，如果计算 d_x 是有效的，那么肯定是在一个大的样本集 $\mathbb{X}$ 上，且该样本集的预测和实际不一致率是几乎相等的：

$$\hat{d} = \sum_{x \in \mathbb{X}} d_x \approx d = \frac{\left\{ x \in \mathbb{X} \mid m_x^{[1]} \cdot m_x^{[2]} < 0 \right\}}{|\mathbb{X}|} \tag{12-59}$$

我们使用 $\mathcal{M}_x^{[1]}$ 计算单侧 p-值 p_x，并根据它把 191 808 个例子分成 6 组。为了保证检验的有效性，对于每一组，预测和实际不一致必须几乎相等。如果不是几乎相等，那么这个检验就是无效的。如果结果是几乎相等，那么我们得出高置信度的结论：检验对于每一组所表示的 p-值的范围是有效的。

表 12-5 至表 12-7 表明 t-检验、符号检验和 Wilcoxon 检验的结果是有效的。每个表分别展示了关于每组的 p、$\hat{d}_p$ 和 d_p 的范围，并把预测误差写成百分比形式：

$$error_p = \frac{\hat{d} - d}{d} \tag{12-60}$$

RMS 概括了 6 个范围的预测误差：

$$\text{RMS误差} = \sqrt{\sum_p error_p^2} \tag{12-61}$$

总的来说，关于 t-检验的预测不一致和实际不一致的比率是合理的，尽管 t-检验对于 $p<0.01$是最优的，得到的估计值是 13.4%，比经验结果还要低。符号和 Wilcoxon 检验对于 $p<0.01$ 的情况就差得多了，对于 p 的其他取值范围，情况也一样糟糕。总的来说，t-检验的预测最精确且 RMS 误差率最低。

表 12-5　t-检验预测不一致，观察不一致以及关于 p 的检验力函数。误差就是预测不一致与观察不一致的比不为 1 的数量。RMS 误差概括了这 6 个误差估计

范围	对	预测不一致	实际不一致	误差
$0.00 \leqslant p < 0.01$	131567(68.6%)	676.5	781	−13.4%
$0.01 \leqslant p < 0.02$	7153(3.7%)	427.6	458	−6.6%
$0.02 \leqslant p < 0.05$	10775(5.6%)	1025.7	981	4.6%
$0.05 \leqslant p < 0.10$	9542(5.0%)	1433.2	1397	2.6%
$0.10 \leqslant p < 0.20$	11436(6.0%)	2592.4	2455	5.6%
$0.20 \leqslant p < 0.50$	21335(11.1%)	8182.5	7837	4.4%
RMS 误差：7.1%				

表 12-6　符号检验预测不一致、观察不一致以及关于 p 的检验力函数。误差就是预测不一致与观察不一致的比不为 1 的数量。RMS 误差概括了这 6 个误差估计

范围	对	预测不一致	实际不一致	误差
$0.00 \leqslant p < 0.01$	125875(65.6%)	675.0	822	−17.9%
$0.01 \leqslant p < 0.02$	9318(4.9%)	581.1	550	5.6%
$0.02 \leqslant p < 0.05$	10068(5.2%)	1009.9	965	4.6%
$0.05 \leqslant p < 0.10$	9949(5.2%)	1524.3	1402	8.7%
$0.10 \leqslant p < 0.20$	10300(5.4%)	2294.6	2107	8.9%
$0.20 \leqslant p < 0.50$	26298(13.7%)	10293.7	8063	27.7%
RMS 误差：14.7%				

表 12-7　Wilcoxon 符号秩检验预测不一致、观察不一致以及关于 p 的检验力函数。误差就是预测不一致与观察不一致的比不为 1 的数量。RMS 误差概括了这 6 个误差估计

范围	对	预测不一致	实际不一致	误差
$0.00 \leqslant p < 0.01$	137008(71.4%)	655.2	981	−33.2%
$0.01 \leqslant p < 0.02$	6723(3.5%)	402.1	466	−13.7%
$0.02 \leqslant p < 0.05$	10096(5.3%)	956.6	1027	−6.9%
$0.05 \leqslant p < 0.10$	8737(4.6%)	1311.7	1382	−5.1%
$0.10 \leqslant p < 0.20$	10267(5.4%)	2320.1	2444	−5.1%
$0.20 \leqslant p < 0.50$	18977(9.9%)	7287.5	7609	−4.2%
RMS 误差：15.3%				

表 12-8 给出了三个检验的 RMS 误差以及 $\alpha=0.01$ 和 $\alpha=0.05$ 时的单侧检验力。Wilcoxon 检验的检验力最高。但是由于 Wilcoxon 一贯过于乐观，使得它所具备的真实检验力并不是数值显示的那么高：如果 p-值更精确，那检验力就更低。符号检验比 t-检验的检验力要低，但误差要更高。

关于这些结果不支持 t-检验的效度引起了大家的忧虑；相反，更低的误差率表明：在比较信息检索系统时，应选择 t-检验。

表 12-8 将 TREC 2004 Robust 专题中的运行实例两两进行比较：分别计算它们在 124 个主题上的 MAP 差值，然后计算统计检验的效度和单侧检验力，比较两个结果。正如表 12-7 所示，Wilcoxon 系统性地低估了 p-值，因此高估了检验力

	RMS 误差	检验力	
		$\alpha=0.1$	$\alpha=0.05$
T 检验	7.1%	0.69	0.78
符号检验	14.7%	0.66	0.76
Wilcoxon 检验	15.3%	0.71	0.80

2. 不同的样本大小

上述实验的结果在一个主题数 $n=124$ 的文档集上度量了 MAP 评分的差别。为了评价选择不同的 n 对效度和 t-检验的检验力的影响，我们采用这样的实验：使 $X_x^{[1]}=\langle A_x, B_x, s\rangle$ 和 $X_x^{[2]}=\langle A_x, B_x, S\rangle$，其中 s 和 S 包含的主题数量不同。我们把 249 个 TREC 主题的子集作为 s，其余的主题作为 S，因此有 $|s|=n$，$|S|=249-n$。在这一情况下，$E_x^{[1]}$ 和 $E_x^{[2]}$ 就不是同分布了。在 t-检验模型下，我们有 $\mu_{E_x^{[1]}}=\mu_{E_x^{[2]}}=0$，然而

$$\frac{\sigma^2_{E_x^{[1]}}}{\sigma^2_{E_x^{[2]}}}\approx\frac{n-1}{249-n-1} \tag{12-62}$$

因此，$\mu_{D_x}=0$

$$\sigma^2_{D_x}\approx(1+\frac{249-n-1}{n-1})\cdot\sigma^2_{E_x^{[1]}} \tag{12-63}$$

表 12-9 展示了在 n 取不同值时得到的 RMS 误差和检验力，其中每次增加 25。已知 t-检验一般对任意 n 均有效，而 RMS 误差一般随着 n 的增加而降低。RMS 误差在 n 取较小值时会增加这是合理的，因为预测不一致性是基于少量信息的。同样当 n 取较大值时（$n=225$），RMS 误差增加也不奇怪，因为 S 所包含的主题较少将导致在计算实际不一致性时出现不确定性。对于 $n=249$ 的情况，RMS 误差没有定义，因为 $|S|=0$。RMS 误差结果与 t-检验的效度是一致的，且效度在 $25\leqslant n\leqslant 249$ 时保持不变。尽管效度保持不变，但随着 n 的增加，检验力如预期的一样也显著增大。

表 12-9 使用不同主题数 n 的 t-检验得到的效度和检验力的对比

n	RMS 误差	检验力	
		$\alpha=0.1$	$\alpha=0.05$
25	11.4%	0.39	0.56
50	10.0%	0.53	0.67
75	6.5%	0.61	0.72
100	7.7%	0.65	0.76
125	6.8%	0.69	0.78
150	6.0%	0.71	0.80
175	3.9%	0.73	0.81
200	3.4%	0.75	0.83
225	9.2%	0.77	0.84
249	—	0.78	0.85

12.3.8 报告指标的查准率

统计检验给出了明确的定义，但一般会被误解，指标的查准率的量化估计表示成一个置

信边界或 p-值。因此，这只是扩展了量化估计而不是替代了它。我们认为置信区间比 p-值更好，因为它们包含了同样地信息，但置信区间更直观，且把指标放到了前面或者中间。

假设检验永远不是绝对的，且把指标和查准率估计放到了“显著的”或“不显著的”的明确声明的后面，这是不正确的。“没有差别”的空假设不为我们提供任何信息。一个合适的假设应该预测一个被认为是实质性的非零差别；查准率估计量化了证据强度，该证据强度指一个指标反映或没有反映这一差别。

如果可能，我们应该选用配对 t-检验。如果对于只有赢/输的情况，符号检验的效果很好。Wilcoxon 检验不大直观，尽管它比符号检验强大且明显比配对 t-检验强大，但对于较小的 p-值，它就完全是错误的。它所服从的分布是不会自然出现的，且随着样本大小的增加，它的方差是无界的，因此，不存在替代差别的真实（总体）值：没有采用中心极限理论。尽管在不采用配对 t-检验时一般会采用 Wilcoxon 检验，但可能选用 bootstrap 会更好。

12.3.9　元分析

元分析是一种将若干个估计值组合成一个，从而得到一个置信度更高的总体估计的技术。[⊖]一种简单的方法就是对一系列的统计独立的实验或者评价采用符号检验（12.3.6 节），且使用二元判断（是/不是）对每个实验或者评价进行判断。例如，如果我们构造一个假设 $A>B$，并进行 4 次独立度量后都有 $A-B>0$，我们可以使用单侧符号检验得出结论 $A>B$（$p<0.06$）。在进行单侧检验时采用标准警告。如果我们简单地观察同一个结果 4 次，且没有构造任何先验假设，那采用双侧检验是合适的，会得到同样地结论但置信度会更低：$A>B$（$p<0.13$）。当要比较各个实验的某种性能但无法获得效应大小和查准率估计时，一般会采用符号检验将这些不同实验的结果组合起来。一般的方法——将各个实验的结果组合起来构造出一个带查准率估计的通用结果——称为**元分析**（meta-analysis）。在如医药这些领域，生命和大量的金钱都承受风险，因此法律上规定使用系统元分析。

下面将详细介绍的**固定效应模型**（fixed-effect model）元分析是一种更强大的方法，因为该方法综合了各个研究的效应大小和查准率估计——在符号检验中丢失的信息——来得到一个更精确的总体估计。表 12-10 的前 4 行给出了 LMD 和 DFR 的对比结果。这 4 个结果中有 3 个结果表明 LMD 更优；另外一个结果则不然。没有一个结果是显著的（$\alpha=0.05$）。报告的 p-值 0.05 是对 0.051 四舍五入后得到的。暂且不管烦琐的细节，对于相反的证据（尽管很弱），筛选掉这些结果是不合适的。符号检验得到 $p\approx0.13$，这不是一个令人信服的证据。另一方面，固定效应模型给出了差异的估计和查准率估计指出 $p\approx0.02$。

表 12-10　LMD（公式(9-32)）和 DFR（公式（9-51）和公式（9-52））的 MAP 的差别的元分析：$MAP_{LMD-DFR}$。所有实验的组合显著性水平（0.02）都比单个实验的显著性水平要低

实　　验	$MAP_{LMD-DFR}$	p（双侧t-检验）
TREC 1998	0.010 (-0.002 - 0.021)	0.09
TREC 1999	0.009 (-0.006 - 0.025)	0.24
GOV2 2005	-0.004 (-0.027 - 0.020)	0.75
GOV2 2006	0.023 (-0.000 - 0.047)	0.05
符号检验	> 0 -	0.13
固定效应模型	0.010 (0.002 - 0.018)	0.02

⊖ glass.ed.asu.edu/gene/papers/meta25.html

固定效应模型将结果集组合起来 $R=\{(m_i,\pi_i)\}$，其中 m_i 是一个带常见真实值 t 的量值的指标，π_i 是它的查准率。在我们的例子中，这个量是指在一个也许更多样化的假设总体上的 $\text{MAP}_{\text{LMD-DFR}}$。在元分析中，查准率一般表示为指标的方差的倒数。

$$\pi_i=\sigma_i^{-2} \tag{12-64}$$

如置信水平或 p-值一样，方差的倒数只是传达同样信息的另一种方法而已；如果假设分布是有效的，二者可以互换。在元分析中，我们在使用“查准率”一词时，并没有给它赋予一个数值，读者可以假设该词就是指方差的倒数。

元分析的第一步是保证每个指标使用的是相同的度量方法和度量单位，且实际上度量的是同样地东西。第二步是把置信水平或 p-值改写成方差的倒数的形式，这可以通过逆转它们的计算过程来实现。常用的估计和它的查准率通过计算各个指标的加权查准率的均值来得到：

$$m=\frac{\sum_i m_i\cdot\sigma_i^{-2}}{\sum_i \sigma_i^{-2}},\quad \sigma^{-2}=(\sum_i \sigma_i^{-2})^{-1} \tag{12-65}$$

可以使用一些常规的方法，如使用 σ 和累积概率分布，计算组合后的结果的置信水平或 p-值。

2008 年美国总统选举就为我们提供了一个正确使用以及错误使用统计方法的案例。尽管调查报告一致认为奥巴马是领先的，但权威人士反复强调调查结果是“在误差范围内”或“统计白热”或“不分胜负”。相反，通过对比各国的调查结果的两两间的差异的元分析指出奥巴马的领先是显著的，也是实质性的，并预测最终奥巴马将以轻微优势胜出。[⊖]

12.4 最小化判定工作

人的实质工作就是对文档进行判断，为信息检索评价得到一个判定集。对本章中的这点而言，我们假设判定是完备的：每个文档对于每个查询而言都存在一个二元相关性判定。文档池方法试图满足这一假设：它要求池中的每个文档都有一个判定结果，且把池外的文档都认为是不相关的。本节将弱化这一假设，首先修改文档池方法以减少判定的次数，然后通过修改我们估计 MAP 的方法，将池中还没有进行判定的文档考虑进来。

用于两种不同评价而言，作为相关性黄金标准的判定（采用 qrels 的形式）是：

1）系统的评价结果影响了应该对哪些文档进行判定（如，参与指定 TREC 任务的系统）。

2）系统的评价结果不会影响应该对哪些文档进行判定。

为了以上两个目的，我们这里关心判定策略的效率和有效性。当使用 qrels 作为黄金标准时，效率可以用人的工作量来度量，有效性可以使用指标的效度和查准率来度量。

我们假设 MAP 是这一小节的主要的有效性指标。也就是说，本节根据各种不同策略度量系统的“真实” AP 的能力来对它们进行考察。

在我们讨论各个策略之前，值得先考虑一个理想 qrels 集的性质，使得在给定的用于度量的一组主题、文档和系统上，尽可能地达到最精确的有效度量。理想 qrels 集将表示每个文档关于每个主题的真实相关性，且独立于所评价的系统。因此上述的两种标准在这里都是一样的。

任何真实的 qrels 集都会由于以下两个原因而不可能是理想的：定义相关性时的不精确性，以及给定某个定义下估计相关性的方法的效度和查准率。真实相关性这一概念与其他真实

⊖ election.princeton.edu/history-of-electoral-votes-for-obama

概念一样难以描述。“满足用户的信息需求”也许是最接近的一种描述方式了，但是我们难以确切地知道用户的信息需求是什么或用户的需求是否得到满足了。在未知效度和查准率的情况下，可要求用户给出一个近似。一份由墨西哥苦杏仁研究所发表的报告对于美国人检索“癌症治疗方法”的信息需求可能是满足的，也可能是不满足的。假设用户是指“已有的癌症治疗方法”，同时如果我们认为苦杏仁在美国不是一种已有的治疗方法，那用户的需求就没有得到满足。但是用户可能不清楚苦杏仁是不是一种已有的治疗方法，因此会导致作相关性判定时出错。第三方的判定者可能更了解苦杏仁的功效，但是他们并不知道用户是指“已有的癌症治疗”还是“最后能尝试的癌症治疗方法”。

尽管存在这些问题，但是人工判定还是相关性判定最好的做法。无论判定者是用户还是第三方人员，判定工作带来的影响是深远的，因为任何一个主要的判定因素都会限制到信息检索评价工作的应用范围。**穷举判定**（exhaustive adjudication）是最直接的方法，该方法让一个判定者或一队判定者对文档集中的每个文档关于每个主题都进行相关或不相关的标记。对于现在的文档集，使用穷举判定的工作量是大得令人难以承受的。一个小型的 TREC 语料库就包含了 50 个主题和大约 500 000 个文档。使用穷举判定将需要进行 2500 万次判定：每个判定 30 秒，需要大约 20 个人年。尽管如此，qrels 也会不精确，因为判定者会犯错或遇到很多不确定的情况。让 3 个独立的判定者对文档进行标记并采用多数人的意见作为 qrels，这样做可以得到更精确的 qrels 集，但需要的工作量将超过 40 个人年。一般来说，人类工作量越大，得到的 qrels 集越精确。但是要达到近似或更好的判定效果，有很多比穷举判定更高效的策略。

暂且不管工作量，qrels 中的某些误差是不可避免的。通过观察在使用它们时对信息检索的有效性指标的效度和查准率所带来的影响，可判断这些误差是不是重要的。

12.4.1　为判定选择合适的文档

对于每个主题，TREC 的文档池方法将每个参加测试的系统返回的前 k 个文档构建为一个文档池，从而避免了采用穷举判定时所带来的大量工作。只有池中的文档才需要判定；其他文档都认为是不相关的。$k=100$ 这一常用取值在由 50 个主题和 500 000 个文档组成的测试集上效果很好。然而即使对于这种规模的文档集，进行评价的工作量还是巨大的（回顾 12.2 节，当中提到对 TREC-8 进行 86 830 次相关性判定，即使每次判定只需耗时 30 秒，仍将大约需要花费判定者 724 小时）。对于文档数量更多、主题数更大的大文档集，工作量将会大得令人望而却步。然而，无论对于模拟大规模的信息检索任务还是提高指标的查准率，都需要更大的文档集。我们研究其他策略的效率和有效性，为判定选择合适的文档。

可以采用真正随机从文档集中抽取出来的样本集，但是除非样本空间很大，否则将导致样本集中只包含非常少的相关文档，反而会降低了样本集对度量信息检索有效性的作用。取而代之的是，采用更容易得到相关文档的偏差抽样技术。文档池方法就是该技术的一个实例：系统在某个测试中检索出的高排名的文档（即那些在文档池中的文档）比那些随机选择的文档更有可能是相关的。并且，这些文档也比其他那些排名较低的相关文档更容易区分各个系统。

采用任何的偏差抽样技术都存在风险：得到的指标反映的不是整个系统的有效性，而是整个系统与样本偏差的一致程度。特别是，结果中包含了文档池中文档的系统可能会比那些效果一样但结果中没有包含文档池中文档的系统要获得更高的评分。对于由 50 个主题和 500 000 个文档组成的 TREC 文档集，偏差还不明显；实验表明同个系统无论对文档池有无贡献，所获得的评分是差不多的。最终结论是，TREC 文档集适用作档案基准：可以用已有

的 qrels 测试新的系统或方法，同时使用相同的文档集进行指标间的比较。TREC 中的方法作为与其他方法进行比较的标准。

1. 交互搜索与判定

交互搜索与判定（interactive search and judging，ISJ）是一种用于选择并判定文档的简单且有效的方法。一个熟练的检索者使用搜索引擎找出并标记尽可能多的相关文档。使用哪个搜索引擎并没有关系。有用的特征包括相关反馈；修正查询去探索不同方面的主题的能力；记录判定的机制；避免对已判定的文档重复判定的机制。在 TREC 6 之前，Cormack 等人（1998）使用 ISJ 为特定任务构造了 qrels 集，用到邻近度排名和一个用户界面来支持上述特征。每个主题的处理过程大概需要 2 个小时，总共需要 100 个小时——大约是对 TREC 进行判定的 1/7 的工作量。Cormack 等人（1998）和 Voorhees（2000）的研究表明：采用这种方法得到的 qrels 集的评价结果与 TREC 的官方结果相近。通过只考虑原来做法得到的（按时间先后）第一个文档的 qrels，可以降低 ISJ 所带来的工作量。这样少一个数量级的工作量所得到的评价结果也与 TREC 的官方结果很相近。“多相近才足够?”这一问题将在后面解答。

2. Move-to-front 文档池

move-to-front 文档池（MTF）采用增量式的方法为判定选择文档，该方法优先选择来自性能较好的系统的高排名的文档（Cormack 等人，1998）。概念上，把系统返回的文档按序放进一个队列中进行考虑。每个文档被依次判定，如果该文档之前未被判定，那么就为其记录一个 qrel。对队列中来自系统的文档不断被处理，直至连续遇到 r 个不相关文档。实际上，我们采用一个优先队列对所有系统交错进行处理。一旦选中某个相关文档，贡献这个文档的系统就把它移到队列的顶部。MTF 选出的相关文档的比例要比文档池方法高得多；对采用这两种方法得到的 qrels 分别进行评价，得到的结果相近。下面将介绍有多相近。

3. 浅文档池

调整文档池方法的工作量的一种简单方法就是修改文档池的深度 k。也就是说，不考虑系统的性能，从每个系统中选出 k 个文档，而 MTF 会从性能更好的系统中选择更多的文档。采用深度为 k 的文档池得到的相关文档的密度要比 MTF 低，但在深度为 k 的文档池的偏差会更小这一假设下，该方法还是很常用的。特别地，当 k 较小时，推荐使用 P@k 替代 MAP，这可能会比直接使用 MAP 产生更低的偏差（Sanderson 和 Zobel，2005）。另外，当使用深度为 k 的文档池计算 P@k 时，我们知道 P@k 所依赖的每个文档的相关性是已知的。因此，使用 P@k 替代 MAP 可能会得到更高的查准率和更低的偏差。由下面的例子可以知道，这一说法并不完全是经验之谈。

4. 更多还是更少主题

一种相对没有那么直接的调整文档池工作量的方法是，使用一个包含更多或者更少主题的文档集（Sanderson 和 Zobel，2005）。判定的总工作量大约正比于 $k \cdot n$，其中 k 是池的深度，n 是主题数（尽管不同系统的结果重叠通常是排名越高重叠数量越大）。在每个主题判定的文档数与主题数之间存在一个权衡关系。正如表 12-9 所示，系统比较的效度和检验力均随 n 的增加而增大。另外，这些量也随着池子深度的增加而增大。增大 k 和增大 n 均能增大指标的查准率。问题是：对于固定判定工作量，使用较多的主题和较浅的池，还是使用较少的主题和较深的池更好？同样地问题也出现在其他方法中，如 ISJ 和 MTF。

5. 评价文档池策略

在评价选择判定文档的策略时，我们必须考虑度量系统有效性的效果。如果已知真实的有效性，我们可以把它用作比较基准。我们已有的最佳黄金标准是采用文档池方法在某个较

大的 k 值下得到的一个结果集。传统的做法是，只根据文档池和判定策略对系统有效性进行排名的能力来评价它们，使更有效的系统排在那些没那么有效的系统前面。将排名进行两两比较，如果排名相似，那么对应系统之间的差异就不是那么重要。这个假设——我们不完全认同——只是为了度量系统有效性而对系统进行排名，而且指标本身（如，根据某种给定策略计算得到的 MAP）是不相关的（Buckley 和 Voorhees，2005）。无论是否认同这一假设，好的系统排名是一个方法的一般效度的证据。

一般使用 Kendall τ 排名相关系数（以它的提出者——英国统计学家 Maurice Kendall 命名的）来比较两个不同的系统排名。τ 表示两个排名间的**反演**（inversion）数：一个反演是指一对（x，y），表示在一个排名中 x 排名比 y 靠前，而在另一个排名中，x 比 y 排名靠后。如果两个排名等价，则 $\tau=1$；如果排名相反，则 $\tau=-1$；如果它们不相关，则 $\tau\approx 0$。

在信息检索评价中，考虑一个由 TREC 实验度量的系统集。Res_1 表示当采用深度为 $k=100$的传统的文档池方法进行评价时，根据 MAP 得到的黄金标准排名。Res_2 表示采用其他不同的判定策略时得到的排名。没有正式的理由下，我们认为 $\tau>0.9$（即少于 5% 的反演）**很好地等同于**（good agreement）黄金标准（Voorhees，2000）。

使用 τ 带来了一个严重的问题：它不考虑指标的查准率。由表 12-9 可见，对于 $n=50$，$k=100$（TREC 2004 Robust 专题中采用的池深度）的情况，度量 MAP 差异的检验力大约为 67%（对于 $\alpha=0.05$）。也就是说，12.3.6 节介绍的 t-检验用于 $p<0.05$ 的检验时，只能区分全部系统对（两两对比）的 67%。因此，即使采用黄金标准，系统对被错误排名的期望至少为 2.8%。另一种误差率同样是 2.8% 的方法与黄金标准完全吻合（$\tau=1$），或者高达 5.6% 的反演（$\tau=0.89$）。误差率为 5.6% 的方法导致的反演可低至 2.8%（$\tau=0.94$），或高至 8.4%（$\tau=0.83$）。Kendall 的 τ 不一定能把这些情况区分开来。

作为 Kendall τ 的替代方法，我们采用显著水平为 α（参见 Cormack 和 Lynam，2007；Carterette，2009b）的**显著反演率**（significant inversion rate）来比较排名间的差异。系统的**显著对**（significant pair）就是采用可选方法（文档池/判定法）与黄金标准（$p<\alpha$）。进行比较时，在 MAP 上得到显著性差异一个**显著反演**（significant inversion）就是一个显著对，由可选方法与黄金标准组成的反演。显著反演率就是显著反演占全部显著对的比例。一个显著反演率小于 α 的方法是有效的，因为任何由于偏差所导致的误差都小于随机误差。可选方法的查准率由它的检验力来度量。

总的来说，如果文档池方法的检验力较高且观察偏差相对于随机误差不是实质性的，那么选用文档池方法较有利。

6. 示例实验

我们将给出一个示例实验来说明判定工作量和效度间的权衡，以及度量结果的查准率（Cormack 和 Lynam，2007）。上面已经进行了可选方法间的比较，除了 ISJ，要进行该模拟并不简单（但可参见 Sanderson 和 Joho，2004）。作为一个判定文档数量的函数，同时该函数使用主题数作为参数，可以对各种判定策略的检验力和偏差进行度量。黄金标准是采用深度 $k=100$ 的池，根据 MAP 对系统进行排名得到的。其他的文档池方法如下：

- 深度 $k\leqslant 100$ 的浅池；
- 使用 P@k 替代 MAP，且池深度 $k\leqslant 100$；
- move-to front 文档池，停止前移条件是连续遇到 r 个不相关文档。

使用提交到 TREC 2004 Robust Retrieval 专题（由 249 个主题和 311 410 个 qrels 组成）的运行实例去模拟这三种方法。为了评价浅池和 move-to-front 文档池，选出的 qrels 可用于

根据 MAP 对系统进行排名，然后将排名结果与黄金标准进行比较。为了将 P@k 作为替代指标进行评价，系统根据 P@k 而不是 MAP 进行排名，池深为 k，仍与根据 MAP 排名的黄金标准进行比较。使用配对 t-检验为所有比较对计算单侧 p-值。把显著结果（$p \leqslant 0.05$）所占的比例看做检验力的估计，同时把显著反演比较对所占的比例看做偏差指标。如果这个比例少于约 $\alpha = 0.05$，那么在估计的效度中，偏差是可忽略因素（相比于随机误差），并且可以把它看做是不重要的。

图 12-5 展示了使用 TREC 中的标准浅池法时，采用不同的 k（池深）和 n（主题数）对判定工作量和检验力（$\alpha = 0.05$）的影响。y 轴表示检验力，x 轴表示对于给定的 n，达到该检验力时需要的相关性判定次数。图中的每个点表示不同的 k 值。由图 12-6（连同图 12-5）可知，当给定判定次数时，move-to-front 文档池的检验力与传统的前 k 个文档池策略的检验力相比差别细微。另外，由图 12-7 可知，对于池深为 k 的情况，根据 P@k 得到的排名结果明显弱于根据 MAP 得到的排名结果。也就是说，当给定相关性判定次数时，不易用P@k区分两个系统。图 12-8 展示了每种方法的观察误差，通过与黄金标准作对比的显著反演进行度量，作为一个判定工作量的函数。由图观察可知，move-to-front 文档池表现出的偏差明显低于其他被视为更**公平**（fair）的方法。对于度量来说，观察的感觉是无可取代的。

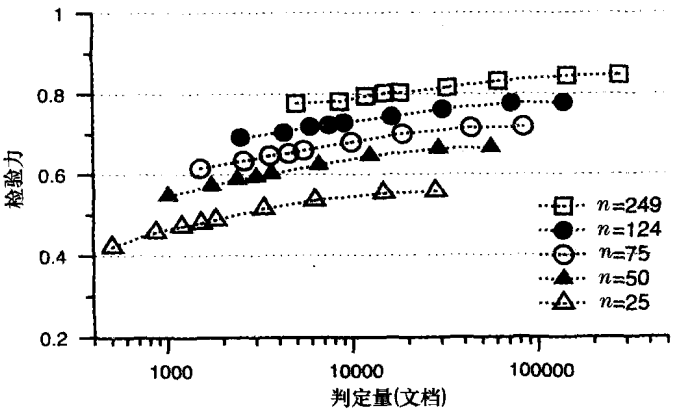


图 12-5 对浅池赋予不同的深度时，关于判定文档数量的检验力函数

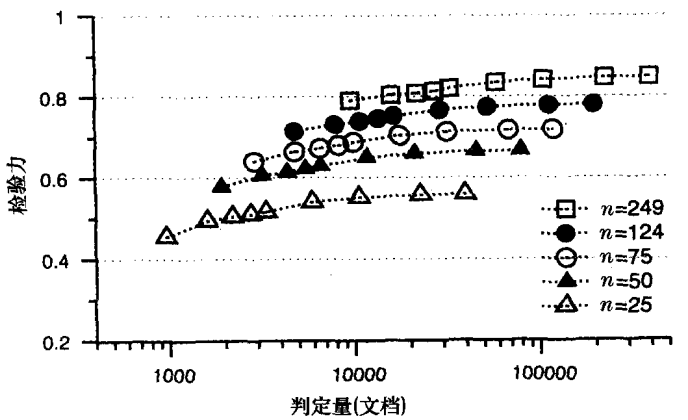


图 12-6 对 move-to-front 文档池赋予不同的深度时，关于判定文档数量的检验力函数

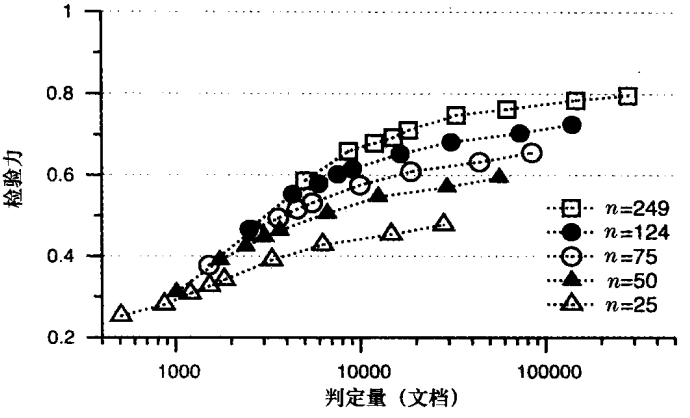


图 12-7　对浅池赋予不同的深度 k 且用 P@k 替代 MAP 时，关于判定文档数量的检验力函数

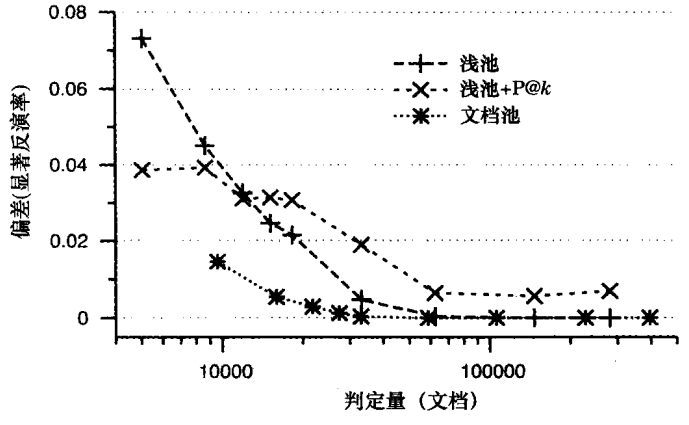


图 12-8　浅池、move-to-front 文档池以及使用 P@k 作为替代指标来作为判定效用的函数上的偏差。对 $\alpha=0.05$ 的情况，使用显著反演率度量偏差

图 12-5 至图 12-8 的结合结果表明，实验设计时使用更多的主题，同时对每个主题使用更少的判定（相比于采用 $n=50$ 和 $k=100$ 的 TREC 标准）将会带来更高的效率，这也是 Sanderson 和 Zobel（2005）建议的做法。然而，他们并不同意以下观点：采用固定池深或一个“完全判定”的指标，如 P@k，将使检验力更高或偏差更低。我们建议在评价新的文档池策略和评价指标时，使用检验力和偏差分析替代排名相关（Buckley 和 Voorhees，2004）。

12.4.2　对池进行抽样

作为一种不同于前一节介绍的选择判定文档法（ISJ 或 move-to-front 文档池）的方法，我们可以保持文档池创建策略不变，但只对池中一个随机文档子集进行判定（而不是整个池）。对文档池进行抽样使得我们可以降低人工相关性判定的次数，同时避免了偏差。如果只选用文档池中 10%～20% 的文档，那么判定工作量可以减少 1/10～1/5。

当然，没有对整个文档池进行判定，就无法实际计算 AP 或任何基于穷举相关性判定这一假设的有效性指标。然而，我们可以基于随机样本**估计**（estimate）这些度量指标。对于

AP，有效性结果被称做**推断平均查准率**（inferred average precision）或infAP。它是平均查准率的无偏估计（Yilmaz和Aslam，2008）。

回顾平均查准率的公式（公式（12-4）），为了方便，在这里再次给出：

$$\mathrm{AP}=\frac{1}{|Rel|}\cdot\sum_{i=1}^{|Res|}\mathrm{relevant}(i)\cdot\mathrm{P@}i \tag{12-66}$$

其中，$|Res|$是指由系统返回的排名列表的长度（对于TREC实验，通常为1000或10 000），如果排名为i的文档是相关的，那么relevant(i)=1；如果是不相关的，则为0。实际上，当且仅当第i个文档**被判定**（judged）为相关时，才有relevant(i)=1。类似可得，Rel是**被判定**（judged）为相关的文档的集合。

为了计算infAP，用期望P@i替代P@i：

$$\mathrm{infAP}=\frac{1}{|Rel|}\cdot\sum_{i=1}^{|Res|}\mathrm{relevant}(i)\cdot\mathrm{E[P@}i]\tag{12-67}$$

这条公式中的期望P@i可以用以下方式计算：首先考虑排名为i的文档。我们可以假设relevant(i)=1，否则就没有必要估计P@i了。因此，

$$\mathrm{E[P@}i]=\frac{1}{i}+\frac{i-1}{i}\cdot\mathrm{E[P@}(i-1)]\tag{12-68}$$

现在为了计算E[P@(i−1)]，对排名为1到i−1的文档逐个进行考虑。对于排名为j(1≤j<i)的文档，有4种可能：

1）如果在排名列表中，j的排名足够靠后，那么该文档可能不是文档池的一部分，在这种情况下，可以假设该文档是不相关的。

2）该文档可能是文档池的一部分，但没有被抽作样本，在这种情况下，认为还未对该文档进行判定。

3）该文档可能是样本集的一部分且被判定为相关。

4）该文档可能是样本集的一部分且被判定为不相关。

现在令Jud为已判定文档集，同时令$Pool$为在池中文档组成的集合。$|Jud|/|Pool|$的值表示随机样本集占整个文档池的比例。我们把P@(i−1)估计为

$$\mathrm{E[P@}(i-1)]=\frac{|Pool\cap Res[1..i-1]|}{i-1}\cdot\frac{|Rel\cap Res[1..i-1]|}{|Jud\cap Res[1..i-1]|}\tag{12-69}$$

其中Res[1..i−1]指出现在排名列表的前i−1项所组成的文档集。公式的右边由两个分式组成。第一个代表了构成部分池的前i−1个文档的比例；第二个代表了前i−1个文档被判定为相关的比例。第二个分式是包括在文档池中的前i−1个文档的查准率的无偏估计。而第一个分式是对这个估计的调整，把不在文档池中的文档也考虑进来（因此假设这些文档是不相关的）。

还有一个小问题：有可能$Jud\cap Res$[1..i]是空集，即前i−1个文档中没有任何一个是已被判定的。在这种情况下，第二个分式将是0/0。为了解决这个问题，我们采用平滑（1.3.4节）得到公式

$$\mathrm{E[P@}(i-1)]=\frac{|Pool\cap Res[1..i-1]|}{i-1}\cdot\frac{|Rel\cap Res[1..i-1]|+\varepsilon}{|Jud\cap Res[1..i-1]|+2\varepsilon}\tag{12-70}$$

其中ε是一个小的常量。结合公式（12-68）与公式（12-70），并去掉i−1，有

$$\mathrm{E[P@}i]=\frac{1}{i}+\frac{|Pool\cap Res[1..i-1]|}{i}\cdot\frac{|Rel\cap Res[1..i-1]|+\varepsilon}{|Jud\cap Res[1..i-1]|+2\varepsilon}\tag{12-71}$$

考虑这样一个例子：用向量形式表示文档的排名列表

$$Res=\langle +,?,-,+,-,X,X,?,X,+,\cdots\rangle \tag{12-72}$$

向量对排名列表中排名靠前的文档的判定进行编码，其中“+”表示已判定相关文档，“−”表示已判定不相关文档，“?”表示在文档池中但未进行判定的文档，X 表示不在文档池中（文档池深度为 5）的文档。假设 $\epsilon=0.000\ 01$，并将这个值用于 NIST 的 trec_eval[⊖]程序。对于这个例子有

$$E[P@10]=\frac{1}{10}+\frac{7}{10}\cdot\frac{2+\varepsilon}{4+2\varepsilon}\approx 0.45 \tag{12-73}$$

当然，在更接近于真实的情况下，在这样的排名列表中未判定文档的相对数量会更多。在传统的假设下，因为出现在列表中的文档只有 3 个被判定为相关文档，因此我们有 P@10＝0.3。EP［@10］是一个估计值，表示如果文档池中剩下的文档都已被判定了，查准率将达到什么水平。

12.5　非传统的有效性指标

12.1 节和 12.2 节中介绍的传统评价方法持续地证明了它们在信息检索历史上的价值。历年以来，很多当前标准的理论和技术在它们刚被提出和改进时，都是采用这套方法来证明的。这些技术包括概率模型、语言模型、随机差分和伪相关反馈（所有这些都在第三部分中介绍了）。随着大文档集的引入，这些方法起到了巨大的作用，用于那些规模和差异性与 GOV2 相当的文档集上时能得到不错的结果。尽管如此，当面对最新的要求时，这些方法就显得力不从心了。这些要求包括：分级相关性判定、处理判定丢失以及包容新颖性和多样性。本节将探讨许多尝试中的一些，以满足这些要求。

12.5.1　分级相关性

Web 的规模远远大于最大的测试文档集。Web 中包含很多质量一般的网页，同时也包含很多质量非常好的网页。有些网页是具有主动侵害性的，包含不可信的广告或危险的产品，或尝试用恶意软件或病毒感染用户的计算机。从企业层面来说，一般的文档管理平台或公司内部网可能都比大部分的测试文档集要大且更多样化。从个人层面来说，很多用户（包括作者在内）都会在他们的个人存储空间中维护着 GB 级的邮件和其他文档，规模很容易就相当于 TREC 中较小的测试文档集，且比它们提供的内容要丰富。

为了反映一个给定文档对某个用户的需求的满足程度，同时解决文档质量不一的问题，可以采用分级相关性代替二元相关性。例如，判定者在判定文档时可采用 6 级评分法：“精确”、“很好”、“较好”、“一般”、“差”、“极差”（Najork 等人，2007）。在这个评分法则中，“差”相应于“不相关”，而“极差”表明文档内容是怀有恶意的，或是误导人的，其他的等级表示不同的相关程度。

分级相关性评定在混合导航/信息检索任务中特别有用（关于这种检索类型的详细介绍，请参见 15.2 节）。例如，考虑检索查询“IBM”。对于这个查询，IBM 的维基百科网页上提供关于该公司的有价值的信息，可认为这个网页的质量是“较好”或“很好”，而 IBM 的官方网站 www.ibm.com 必然就是“非常好”了。

⊖ trec.nist.gov/trec_eval

归一化惩罚累积增益（nDCG）评价方法直接利用了分级相关性判定的优点（Järvelin 和 Kekäläinen，2002）。该方法将排名结果列表的相关性值与“理想”结果列表的值进行比较，其中“理想”结果列表通过如下方式获得：首先对全部最可能相关的文档进行排名，然后对全部次最可能相关的文档进行排名，如此类推。为了计算 nDCG，必须对每个相关性评分等级赋予一个**增益值**（gain value）。这些增益值把相关性评分等级映射到一个可用于计算评分的数值。选择增益值时，应该让其反映出各个评分等级间的相对差异。例如，假设评价文档时采用四级评分制：“高度相关”、“相关”、“基本相关”和“不相关”。我们可以选择如下的增益值

高度相关→ 10
相关→ 5
基本相关→ 1
不相关→ 0

给定文档的排名列表，计算 nDCG 的第一步是构造一个**增益向量**（gain vector）G。例如，假设排名列表的前 6 个文档的判定如下：（1）相关，（2）高度相关，（3）不相关，（4）相关，（5）基本相关，（6）高度相关。其余文档均不相关。这个列表对应的增益向量是

$$G=\langle 5,10,0,5,1,10,0,0,\cdots\rangle \tag{12-74}$$

计算 nDCG 的第二步是计算**累积增益向量**（cumulative gain vector）CG。CG 中第 k 个元素的值是 G 中第 1～k 个元素的总和

$$CG[k] = \sum_{i=1}^{k} G[i] \tag{12-75}$$

对于我们的例子有

$$CG=\langle 5,15,15,20,21,31,31,31,\cdots\rangle \tag{12-76}$$

在计算累积增益函数前，先对每个排名采用一个惩罚函数来惩罚排名较低的文档，以此反映要找到这些文档，用户需要付出的额外工作量。这个惩罚反映了用户不大愿意查看排名较低的文档，因为这将需要更多的时间而且更麻烦。尽管可以采用其他的惩罚函数，但一般都采用 $\log_2(1+i)$，因为这可以更好地反映出用户的工作量（Järvelin 和 Kekäläinen，2002）。这个**惩罚累积增益**（discounted cumulative gain）定义如下：

$$DCG[k]=\sum_{i=1}^{k}\frac{G[i]}{\log_2(1+i)} \tag{12-77}$$

对于我们的例子有

$$DCG=\langle 5.0,11.3,11.3,13.5,13.8,17.4,17.4,17.4,\cdots\rangle \tag{12-78}$$

下一步就是针对“理想”增益向量对惩罚累积增益向量进行归一化。理想排名就是在各级评分上最大化累积增益后的顺序。假设文档集中包含两个高度相关的文档，两个相关文档，以及两个基本相关文档。其余的文档都是不相关的。因此，理想增益向量为

$$G'=\langle 10,10,5,5,1,1,0,0,\cdots\rangle \tag{12-79}$$

理想累积增益向量是

$$CG'=\langle 10,20,25,30,31,32,32,32,\cdots\rangle \tag{12-80}$$

同时，理想惩罚累积增益向量是

$$DCG'=\langle 10.0,16.3,18.8,21.0,21.3,21.7,21.7,21.7,\cdots\rangle \tag{12-81}$$

计算 nDCG 的最后一步就是用理想惩罚累积增益向量归一化惩罚累积增益：

$$\mathrm{nDCG}[k]=\frac{\mathrm{DCG}[k]}{\mathrm{DCG}'[k]} \tag{12-82}$$

对于我们的例子有

$$\mathrm{nDCG}=\langle 0.50,0.69,0.60,0.64,0.65,0.80,0.80,0.80,\ldots\rangle \tag{12-83}$$

通过求主题集中的各个主题的 nDCG 值的算术平均，得到某个主题集的 nDCG，这也是在信息检索评价方法中很常见的做法。类似于查准率和查全率，通常根据不同的检索深度报告 nDCG。在我们的例子中，nDCG@4=0.64 以及 nDCG@8=0.80。

nDCG 评价方法是 Web 检索的常用方法，因为 Web 中的网页质量参差不齐，且存在大量或多或少相关的网页（Najork 等人，2007）。该方法也被应用于 XML 信息检索评价中（参见 16.5 节）。除了 nDCG，CG 和 DCG 都可以作为评价方法直接使用。一个基于 Web 检索结果的研究结果表明（由 Al-Maskari 等人提供，2007），CG 和 DCG 比 nDCG 更好地反映了用户的满意度。

12.5.2　不完整判定和偏差判定

未判定文档为大部分有效性带来了一个问题。当重用大的测试文档集时，不管是测试新的评价方法还是调整某个已知方法，未判定文档（即不在文档池中的文档）可能会出现在高排名的位置上。按照惯例，认为这些文档是不相关的（参见 12.2 节）。实际上，如果判定者遇到这些文档，可能很多已经被判定为是相关的。因此，文档池方法对那些没有对文档池贡献的运行实例存在固有偏差。当重用测试集时，这个偏差将会扭曲关于运行实例的相对质量的结果，并导致得出不正确的结论。

大部分处理不完整判定的方法假设一个**无偏**（unbiased）样本，在该样本中任一系统或方法都不比其他的具有优势。例如，推断平均查准率（参见 12.4.2 节）允许我们在不对整个文档池进行判定的情况下评价有效性，但该方法要求有一个无偏样本来对各级查准率进行无偏估计。

在面对不完整和有偏差的判定时进行有效性评价的能力具有重大的现实意义。处理丢失判定的能力也是当前在动态文档集上进行信息检索评价所必备的。在规模较大的动态文档集上的实验通常会碰到未判定文档。在很多情况下，由于时间和费用的限制，我们不能根据要求对这些文档进行判定以修补测试文档集上的漏洞。例如，Web 搜索服务的经营者可能会通过周期性地在它的搜索引擎上运行一个固定的查询集来评价该引擎的性能。尽管爬虫不断地往文档集中加入新的网页，同时这些网页的判定结果可能还不存在，但是当我们将今天得到的有效性结果与昨天的结果进行对比时，我们依然能获得有意义的对比结果。

尽管不完整和有偏差的判定所引起的问题很重要，但用于解决这个问题的相关工作很少。在这些工作中，有几个研究表明：在系统排名稳定性上，**排名效果**（rank effectiveness）指标或 RankEff 这一评价指标的性能要比其他的建议指标好（Büttcher 等人，2007；Ahlgren 和 Grönqvist，2008）。这个稳定性表明，如果一个信息检索系统的 RankEff 要优于另一系统，那么在判定完整的情况下，关于它们的传统指标展现出的优劣也一样。然而，要注意：这一稳定性的性质不是绝对成立的，随着丢失的判定数量的增加，它的确信度下降。

RankEff 的公式使我们想起了平均查准率的公式（公式（12-4）），除了平均查准率的公式忽略了未判定文档：

$$\mathrm{RankEff}=\frac{1}{|Rel|}\cdot\sum_{i=1}^{|Res|}\mathrm{relevant}(i)\cdot\left(1-\frac{|Res[1..i]\cap Non|}{|Non|}\right) \tag{12-84}$$

如果排名为 i 的文档被判定为相关，则有 relevant(i)=1，如果不相关，则为 0。在这条公式中，*Rel* 是被判定为相关的文档所组成的集合，*Non* 是被判定为不相关的文档所组成的集合，*Res*[1..i]表示出现在排名列表前 k 位的文档所组成的集合。

$$\frac{|Res[1..i] \cap Non|}{|Non|} \tag{12-85}$$

的值实质上就是**已判定不相关**（judged nonrelevant）文档的 recall@i。对于排名列表中的已判定相关文档，利用排名比它们靠前的已判定不相关文档所占的比例对它们做惩罚。这条公式把没有出现在 *Res* 中的已判定相关文档看做出现在所有已判定不相关文档的后面，因此对评分没有任何贡献。

考虑这样一个例子：用向量形式表示文档的排名列表

$$Res=\langle +,?,-,+,-,?,?,?,?,+\rangle \tag{12-86}$$

向量对排名列表中排名前 10 位的文档的判定进行编码，其中“+”表示已判定相关文档，“−”表示已判定不相关文档，“?”表示未判定文档。假设有 4 个已判定相关文档(|*Rel*|=4)以及 6 个已判定不相关文档（|*Non*|=6)。然后计算 RankEff

$$\frac{1}{4}\cdot\left(\left(1-\frac{0}{6}\right)+\left(1-\frac{1}{6}\right)+\left(1-\frac{2}{6}\right)\right)=\frac{5}{8}=0.625 \tag{12-87}$$

为了进行比较，该向量的 AP 值为

$$\frac{1}{4}\cdot\left(\frac{1}{1}+\frac{2}{4}+\frac{3}{10}\right)=\frac{9}{20}=0.450 \tag{12-88}$$

12.5.3 新颖性和多样性

经典的信息检索评价方法假设每个文档的相关性是独立于其他文档而判定的，并且鼓励冗余。假设一个测试集里包含大量几乎相同的文档（很多文档就是完全相同的）。如果这些文档中有一个是相关的，那它们全部都是相关的。如果返回的结果中，它们全部出现在排名靠前的位置上，这将导致使用经典评价方法得到的评分值较高，如 AP，但这种做法肯定不受实际用户的欢迎。尤其对于从 Web 中抓取回的文档集，更会有这样的问题。Bernstein 和 Zobel（2005）研究了在 GOV2 文档集上近似重复的文档的数量，并发现该文档集中超过 17%的文档与其他文档是基本重复的。权宜的办法就是删去文档集中的重复文档，但这仅仅是在回避这个问题而已。取而代之的是，评价方法本身应该直接把重复文档的概率考虑进去，对那些能提供**新颖**（novel）信息的文档进行奖励，而不是奖励那些重复提供用户已知的前面的（即排名更高的）文档。

经典信息检索评价的另外一个有问题的假设是：相关性是根据一个由主题所表达的详细说明的信息需求进行判定的。细看图 1-8 中的 TREC 主题 426，该主题期望把它的标题（“law enforcerment，dogs”）作为查询。在那些专门设计的经典评价实验中，对这个查询的一种理解是：仅当文档提供的“信息是关于全球用于执法的狗”时，才认为该文档是相关的。然而，由 8.6.2 节中的例子可以知道，对于这个查询有第二种理解：一个关于狗执法信息（如规章制度的约束）的请求。尽管第二种理解是完全合理的，并且可能一个实际用户也是这么期望的，但是反映了这种理解的文档将被判定为不相关。要知道，只要简单地改变一下描述或说明，我们就能从一种理解转变到另一种理解了（从一种相对容易理解的查询转变为一种相对难以理解的查询）。在写这本书的时候，我将这个查询输入到 Web 搜索引擎，得到的返回结果中包含各种文档，但却主要偏向于第一种理解。

这个查询的解释都可能对应某个用户的信息需求所指定的方式。例如，某个用户可能只关心关于狗执行禁毒法的信息；另外一个用户可能只关心关于他们社区的规章制度的约束。理想情况下，一个信息检索系统所返回的结果会反映出查询中的内在歧义，并让这些结果尽可能地覆盖所有的方面。为了满足这个要求，有效性评价方法应该在检索结果中合理地奖励**多样性**（diversity）。

对于一个给定的查询，信息检索系统返回的排名列表应该能反映出查询中的有用信息的广度以及查询中的任何内在歧义。查询“jaguar”（美洲豹——译者注）就是模糊查询的一个标准例子。信息检索系统响应这个查询时，最好能返回多种文档，其中包含讨论车、豹以及经典的 Fender 吉他的文档。把所有这些文档组合起来，应该能涵盖所有的理解方式。理想情况下，由这个查询得到的文档有序表应该合理地把所有用户的兴趣考虑进去。如果车比豹更受欢迎，合适的做法是：在转换主题前把前几个文档返回给用户。排名较前的文档可能涵盖了各个主题的主要方面。排名较后的文档应该对这些基本信息作补充，而不是不断地重复同样地内容。

在描述一个同时考虑了新颖性和多样性的评价方法前，我们先介绍两个来自 Clarke 等人（2008）的例子作为引入该方法的动机。第一个例子是一个 Web 检索的例子。第二个例子基于一个在 TREC 2005 上进行的实验性的问答任务（Voorhees 和 Dang，2005）。

1. Web 检索例子

表 12-11 展示了关于 Web 查询“UPS”的前 10 个结果中的 5 个，是在撰写本书期间由一个商用搜索引擎获返回的。我们保留了由搜索引擎指定的文档顺序，但去掉了一些结果使例子更精简。模糊性是明显的。当一个用户输入这个查询时，他可能想追踪由联邦包裹服务公司送出的包裹，计划购买一个不间断电源，或检索普吉特湾大学的主页。该缩写的正确扩展与用户的需求有关。

表 12-11　对于 Web 查询“UPS”可能返回的结果，其中涵盖了对该查询的各种理解

排名	网页标题	URL
1	UPS 全球首页	www.ups.com
2	快件信息追踪	www.ups.com/tracking/tracking.html
3	不间断电源	en.wikipedia.org/wiki/Uninterruptible_power…
4	The UPS Store：Retail packing，shipping，…	www.theupsstore.com
5	普吉特湾大学：主页	www.ups.edu

这很难说这五个网页中的任何一个要比其他的更相关。对于这些网页，总有一群用户认为其中一个是最好的结果。在经典的评价方法中，相关性判定会依赖于主题说明的细节，这些细节对信息检索系统来说是不可见的。根据这些不可见的细节，这些文档中的任何文档都可能被判为是相关的或不相关的。因为这个查询具有明显的模糊性，一个基于这个查询的主题很自然地不会被 TREC 所接受，因此使得这样的问题得以避免。遗憾的是，这样的问题在实际中是不可避免的。

一种可能的排名这些网页的方法是根据认为这些网页是相关的可能的用户组的相对大小对网页进行排名。凭猜测，对联邦包裹服务公司感兴趣的用户群要远远大于对普吉特湾大学感兴趣的用户群，即使在华盛顿州内也是这样。对不间断电源感兴趣的用户数应该介于这两者之间。表 12-11 中的顺序与这一猜测一致。但要注意到一个关于不间断电源的网页（#3）落在两个关于联邦包裹服务公司的网页之间。假设对不间断电源感兴趣的用户构成了其中一

个多元化用户群，且继续在该点上浏览结果，那么这是合理的。对于第五个结果，对大学感兴趣的用户可能构成了另一个多元化用户群。因此，结果中的多样性直接根据用户群的需求进行处理。

假设表 12-11 给出了最优的排名（可能不是），可以对它进行非正式且直观的调整。我们的评价方法可以通过精确地对这个排名赋予最高的评分来反映这种直观性。

2. 问答例子

第二个例子基于一个来自 TREC 2005 上的问答任务的主题（Voorhees 和 Dang，2005）。相比于传统检索任务，问答（QA）的目标是为某个具体问题返回确切答案，常常通过组合来自不同源的信息来实现。在 TREC 2005 任务中，问题被分为多个系列，且每个系列都与一个目标相关联。参与的系统都能获得该目标以及问题，并且应为每个问题返回一个答案。表 12-12 给出了主题 85 的目标和问题："挪威邮轮公司（NCL)"。

表 12-12 TREC 2005 问答主题 85。主题由多个问题组成，每个问题关注整个主题的某一方面

85：挪威邮轮公司
85.1：给 NCL 的船命名
85.2：在 1999 年，哪一家邮轮公司尝试接管 NCL?
85.3：NCL 的私人岛屿名称是什么?
85.4：NCL 在邮轮公司中按规模排名是多少?
85.5：为什么 Grand Cayman 避开一艘 NCL 的邮轮?
85.6：NCL 发起的所谓主题邮轮

我们可以从不同的角度看待这个主题，把目标看做查询，同时把问题看做一个用户可能寻求的具体信息块的表示或例子。表 12-13 中的结果是把目标看做查询，同时采用 Wumpus 实现的 BM25 评分公式得到。语料库与 TREC 2005 QA 任务中的一样，由新闻文章组成。前 10 个文档的标题如表 12-13 中所示。对于每篇文章，表中都指示了根据官方的 TREC 判定，它是否能回答该问题。对于这个例子而言，如果一篇文章列出了任何 NCL 船只的名字，我们就认为它能回答问题 85.1。最后一列给出了能回答的问题的总数。

表 12-13 查询"挪威邮轮公司（NCL)" BM25 返回的前 10 个结果，检索的文档集为 TREC 2005 QA 任务语料库。对能回答问题的每个文档进行了标记

文档标题	85.1	85.2	85.3	85.4	85.5	85.6	总计
a. Carnival Re-Enters Norway Bidding		X		X			2
b. NORWEGIAN CRUISE LINE SAYS···		X					1
c. Carnival, Star Increase NCL Stake		X					1
d. Carnival, Star Solidify Control							0
e. HOUSTON CRUISE INDUSTRY GETS···	X					X	2
f. TRAVELERS WIN IN CRUISE···	X						1
g. ARMCHAIR QUARTERBACKS NEED···			X				1
h. EUROPE, CHRISTMAS ON SALE	X						1
i. TRAVEL DEALS AND DISCOUNTS							0
j. HAVE IT YOUR WAY ON THIS SHIP							0

尽管这些问题肯定不能涵盖一个主题的所有方面，但我们可以把它们看做一个合理的代表。从这个观点出发，我们可以将所有的文档相关性基于这些问题来判定，把回答的问题数作为分级的相关性分值。因此，如果我们只考虑文档能回答的问题数，一个关于文档的"理想"排名就是 a-e-b-c-f-g-h-d-i-j-d，能回答 2 个问题的文档要比只能回答一个问题的文档排名靠前。

如果我们同时考虑新颖性，理想排名将把文档 g 放到第三位，高于其他只回答了一个问题的文档，这是因为只有文档 g 回答了问题 85.3。此外，除了没有被任何文档回答的问题 85.5 外，a-e-g 排名回答了所有问题。可以认为其他文档是不相关的，因为它们没有增添任何新信息。然而，因为其他文档可能包含了超出问题范围的方面，我们不应该在第三个文档就停住了。另外，判定可能包含错误，或文档也许不能完全回答一个指定问题。给定已知信息，我们可以通过考虑每个问题被回答的次数完成文档排名。文档 b（回答了问题 85.2）可以排在文档 g 后面，紧跟其后的是文档 f（回答了问题 85.1），然后是文档 c 和 h（第三次回答了这些问题）。最终的排名是 a-e-g-b-f-c-h-i-j-d。

3. 评价新颖性和多样性

新颖性和多样性的有效性指标的发展和验证仍是一个开放的研究问题。下面我们介绍一种由 Clarke 等人（2008）提出的指标。其他方法将在延伸阅读中提及。

为了概括上面的例子，我们可以考虑一个表示为**金块**（nugget）集的信息需求。从概念上说，一块金块表示一个与信息需求相关的具体事实，或某个具体问题的答案。它也可表示一个结构或导航要求，指示一个文档属于某个具体的文档集，是在某个具体时段写的或出现在某个具体的 Web 网站中。然后，根据文档包含的金块数赋予相应的分级相关性分值：金块数越多越好。除此之外，给定一个文档的排名列表，如果一块金块出现在一个文档中，那它在后面文档的分值可能要降低，因此，相比于多样性，更偏向于新颖性。在获得分级相关性分值后，可采用如 nDCG 这样的评价方法度量有效性。使用这些增益值的扩展 nDCG 被称做 α-nDCG。

更正式的说法是，我们使用金块集 $\{n_1, \cdots, n_m\}$ 对用户信息需求进行建模。给定文档 $\langle d_1, d_2, \cdots\rangle$ 的排名列表，如果判定文档 d_i 包含金块 n_j，则令 $N(d_i, n_j)=1$；否则，$N(d_i, n_j)=0$。因此文档 d_i 包含金块的数量为

$$\sum_{j=1}^{m} N(d_i, n_j) \tag{12-89}$$

如果不管冗余，那么可以直接把这个数值用作分级相关性分值。然而，随着我们深入研究排名列表，我们可能希望随着金块的重复来调整这个数值，以反映冗余信息的价值在下降。为了进行这个调整，我们首先定义

$$r_{j,k-1}=\sum_{i=1}^{k-1} N(d_i, n_j) \tag{12-90}$$

为排名在前 $k-1$ 个的文档中被判定包含金块 n_j 的文档的数量。为了方便，我们定义 $r_{j,0}=0$。然后我们定义增益向量 G 中的第 k 个元素为

$$\mathrm{G}[k]=\sum_{j=1}^{m} N(d_k, n_j)\alpha^{r_{j,k-1}} \tag{12-91}$$

其中，α 是一个常量，$0<\alpha\leqslant 1$。这个常量表示由于金块重复而导致的增益值的减少。例如，如果我们令 $\alpha=1/2$，那么表 12-13 的有序列表的增益向量为

$$\mathrm{G}=\langle 2, \tfrac{1}{2}, \tfrac{1}{4}, 0, 2, \tfrac{1}{2}, 1, \tfrac{1}{4}, \cdots\rangle \tag{12-92}$$

在这个例子中，每个文档平分了所包含金块的增益值。当 $\alpha=1$ 时，无论金块重复出现多少次，增益值保持不变。当 $\alpha=0$ 时，金块出现一次后，增益值就减为 0。

现在，nDCG 的计算就可以用 12.5.1 节描述的方法。为了进行归一化，理想有序表就是各个级别的最大化累积增益值的顺序。对于我们的例子，理想有序表是 a-e-g-b-f-c-h-i-j-d，

同时相应的增益向量是

$$G'=\langle 2,2,1,\frac{1}{2},\frac{1}{2},\frac{1}{4},\frac{1}{4},\cdots\rangle \tag{12-93}$$

12.6 延伸阅读

第一个信息检索测试集是由 Cyril Cleverdon 以及他的同事一起在 20 世纪 60 年代创建的，这是 Cranfield 检验（Cleverdon，1967）里一系列实验的一部分。通过使用一个固定的由文档、查询以及人工判定组成的集合的基本信息检索系统评价方法一般被称为 Cranfield **范式**（Cranfield paradigm）。这个方法早在 20 世纪 90 年代初就被 TREC 所采用，并在后来扩展到大文档集上（规模超过 GB 级）。TREC 以及它使用 Cranfield 范式的历史可以参考 Voorhees 和 Harman（2005）。

在信息检索评价中统计方法的合理使用一直是争论不休的话题。本章中介绍的一些统计方法在如医药这些领域中已成为了标准，但在信息检索领域中还没有得到普遍使用。Sanderson 和 Zobel（2005）研究了统计检验在信息检索中的使用，并得出结论：对大量的主题进行浅判定要比对少量主题进行深判定好。Webber 等人（2008b）通过对信息检索系统两两进行比较的实验，即一个关于效果量、主题数以及池深的函数，来评价统计检验的检验力。通过这些实验，他们推知：在对 TREC 2004 Robust 专题上的系统两两进行对比以找出它们之间的区别时，采用主题数为 617 且池深度 $k=5$ 时，能得到 80% 的检验力，这与图 12-5 相符。Smucker 等人（2007）使用随机化检验作为黄金标准来评价 t-检验、符号检验、Wilcoxon 检验以及 bootstrap。他们得出结论：随机化检验、t-检验以及 bootstrap 计算得到的 p-值相近，而其他检验的 p-值差别巨大。Thomas（1997）讨论了信息检索领域外的回溯性能力分析的争议性，并比较了用于计算回溯性能力的方法。**英国医疗学报**（British Medical Journal）中有很多关于元分析的文章摘要。⊖

已经有很多通过选择性判定来减少判定工作量的方案。Cormack 等人（1998）研究了通过交互搜索、判定以及人工重构查询的过程来创建测试集。他们的目标是在给定的一段时间内找出尽量多的相关文档。Sanderson 和 Joho（2004）在这个方向上研究得更深入，他们描述了三种不需使用文档池来创建测试集的方法。Zobel（1998）和 Cormack 等人（1998）都提出了在池中对文档进行动态排序以增加下一个被判定的是相关文档的概率的方法。Carterette 等人（2006）描述了一种选择下一个判定文档的算法，该算法能提高判定对区分系统的能力的影响。Moffat 等人（2007）基于排名偏差查准率（rank-biased precision）指标研究自适应判定，该评价指标根据文档在排名列表中的排名对它们赋予指数下降的权重。Aslam 等人（2006）以及 Yilmaz 和 Aslam（2008）描述了包括 infAP（12.4.2 节）的抽样方法。Yilmaz 等人（2008）对这些工作进行了改进并扩展成 nDCG。Soboroff 等人（2001）提出了一种避免对全部文档进行相关性判定的有趣方法。

过去几年提出了大量的信息检索评价方法。有部分方法是来自其他领域的，并将它用于为信息检索服务。nDCG 指标是首先由 Järvelin 和 Kekäläinen（2002）提出并对其进行描述的。该方法一提出就被应用到很多领域中，包括 Web 检索（Burges 等人，2005；Najork 等人，2007）和 XML 信息检索（Al-Maskari 等人，2007）。Buckley 和 Voorhees（2004）提出了 bpref 指标，该指标在结构与目的上均与 RankEff 相似。Shah 和 Croft（2004）将倒数排名（公式（12-5））评价方法应用到评价高查准率检索中，其中高查准率检索主要关注在高

⊖ www.bmj.com/collections/ma.htm

排名上要高查准率。Aslam 等人（2005）提出了一种量化有效性指标的方法。Moffat 和 Zobel（2008）描述并评价了一个基于简单用户行为模型的排名偏差查准率。Chapelle 等人（2009）描述了一种用于分级相关性的倒数排名指标，其中分级相关性判定根据一个简单的用户模型进行，且该用户模型中的用户行为符合某个商业搜索引擎的日志。

Amitay 等人（2004）和 Büttcher 等人（2007）都提出了通过提取相关文档的特征来创建大规模测试集的方法，这种方法使得判定新文档的工作可以自动完成。Carterette（2007）正式定义了可重用测试集的概念。Custis 和 Al-Kofahi（2007）考虑了查询扩展技术的评价方法。Webber 等人（2008a）研究了比较测试集之间的评分的方法。Sakai 和 Kando（2008）研究了相关性判定丢失所带来的影响。

早在 20 世纪 60 年代，就有大量的研究者已经开始研究信息检索评价中的新颖性和多样性（Goffman，1964；Boyce，1982）。Carbonell 和 Goldstein（1998）描述了**最大边界相关法**（maximal marginal relevance），该方法尝试在最大化相关性的同时最小化高排名文档间的相似性。Zhai 等人（2003）提出并验证了基于一个风险最小化框架的子主题检索的方法，同时介绍了对应于子主题检索的查全率和查准率。Chen 和 Karger（2006）描述了一种利用负反馈的检索方法，即一旦结果列表中包含某个文档，就假设该文档是不相关的，这样做是为了实现最大化多样性的目标。Agrawal 等人（2009）应用分类系统去提高检索结果的多样性，并将这种做法推广到几种传统的有效性评价方法上（使其考虑多样性），包括 nDCG、MAP 以及 MRR。Spärck Jones 等人（2007）要求大家在提出评价方法以及创建测试集时考虑多样性。Vee 等人（2008）在购物检索这一背景下考虑多样性。Vee 等人（2008）在购物搜索的应用中考虑了多样性；van Zwol 等人在图像搜索中考虑了多样性。Clarke 等人（2009）将由 Moffat 和 Zobel（2008）提出的排名偏差精度与由 Clarke 等人（2008）和 Agrawal 等人（2009）提出的新颖性和多样性指标结合在一起，由此扩展了排名偏差查准率，使其通过一个简单的用户需求和行为的模型来考虑未指定的查询。Carterette（2009a）考虑了为新颖性和多样性指标计算理想结果。

12.7　练习

练习 12.1　在公式（12-71）中，假设 $|Jud \cap Res[1..i-1]|=0$。在这种情况下，$E[\mathrm{P@}i]$的值是多少？

练习 12.2　考虑以向量形式表示文档的排名列表

$$Res=\langle 0,-,+,0,-,\mathrm{X},0,\mathrm{X},\mathrm{X},+,\cdots\rangle \tag{12-94}$$

其中，“+”表示一个已判定相关文档，“−”表示一个已判定不相关文档，0 表示一个在池中的未判定文档，X 表示不在池中的文档。计算 $E[\mathrm{P@1}]$和 $E[\mathrm{P@10}]$。假设 $\epsilon=0.000\,01$。

练习 12.3　根据给定的增益向量

$$\mathrm{G}=\langle 1,3,0,2,1,3,0,0,\cdots\rangle \tag{12-95}$$

和给定的理想增益向量

$$\mathrm{G}'=\langle 3,3,2,2,2,1,1,0,\cdots\rangle \tag{12-96}$$

计算 nDCG 向量。

练习 12.4　考虑以向量形式表示文档的排名列表

$$Res=\langle 0,-,+,0,-,0,0,0,0,+\rangle \tag{12-97}$$

向量对排名列表中(列表长度为 $k=10$)的文档判定进行编码，其中“+”表示已判定相关文档，“−”表示已判定不相关文档，0 表示未判定文档。假设有 3 个已判定相关文档($|Rel|=3$) 以及 4 个已判定不相关文档($|Non|=4$)。为该向量计算 RankEff（公式（12-84））。

练习 12.5 在商用 Web 搜索引擎上搜索以下模糊的指定查询：（a）“jaguar”，（b）“windows”，（c）“hello”，以及（d）“charles clarke”。在返回的前 10 个结果中，对这些查询一共有多少种不同的理解方式。

练习 12.6 假设你要计算一个指定系统的 MAP，该指定系统使用一个由 n 个主题组成的集合 s，且该集合在置信水平为 95% 的置信区间 $c=[l,u]$ 上的估计是 m。使用 n 个特征相似但不同的主题得到第二个估计 m'。这里没有 m' 的查准率估计。区间 c 包含 m' 的概率是多少？假设 $\overline{m}=\frac{m+m'}{2}$。在没有其他额外信息的情况下，计算置信水平为 95% 且包含 $\overline{m}$ 的最小置信区间。如果第二个主题集的大小 $n'\neq n$，那么最小的置信区间是什么？

12.8 参考文献

Agrawal, R., Gollapudi, S., Halverson, A., and Ieong, S. (2009). Diversifying search results. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining*, pages 5–14. Barcelona, Spain.

Ahlgren, P., and Grönqvist, L. (2008). Evaluation of retrieval effectiveness with incomplete relevance data: Theoretical and experimental comparison of three measures. *Information Processing & Management*, 44(1):212–225.

Al-Maskari, A., Sanderson, M., and Clough, P. (2007). The relationship between IR effectiveness measures and user satisfaction. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 773–774. Amsterdam, The Netherlands.

Amitay, E., Carmel, D., Lempel, R., and Soffer, A. (2004). Scaling IR-system evaluation using term relevance sets. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 10–17. Sheffield, England.

Aslam, J. A., Pavlu, V., and Yilmaz, E. (2006). A statistical method for system evaluation using incomplete judgments. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 541–548. Seattle, Washington.

Aslam, J. A., Yilmaz, E., and Pavlu, V. (2005). The maximum entropy method for analyzing retrieval measures. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 27–34. Salvador, Brazil.

Bernstein, Y., and Zobel, J. (2005). Redundant documents and search effectiveness. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pages 736–743. Bremen, Germany.

Boyce, B. (1982). Beyond topicality: A two stage view of relevance and the retrieval process. *Information Processing & Management*, 18(3):105–109.

Buckley, C., and Voorhees, E. (2005). Retrieval system evaluation. In Voorhees, E. M., and Harman, D. K., editors, *TREC — Experiment and Evaluation in Information Retrieval*, chapter 3, pages 53–75. Cambridge, Massachusetts: MIT Press.

Buckley, C., and Voorhees, E. M. (2004). Retrieval evaluation with incomplete information. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 25–32. Sheffield, England.

Burges, C. J. C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 89–96. Bonn, Germany.

Büttcher, S., Clarke, C. L. A., Yeung, P. C. K., and Soboroff, I. (2007). Reliable information retrieval evaluation with incomplete and biased judgements. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 63–70. Amsterdam, The Netherlands.

Carbonell, J., and Goldstein, J. (1998). The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 335–336. Melbourne, Australia.

Carterette, B. (2007). Robust test collections for retrieval evaluation. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 55–62. Amsterdam, The Netherlands.

Carterette, B. (2009a). An analysis of NP-completeness in novelty and diversity ranking. In *Proceedings of the 2nd International Conference on the Theory of Information Retrieval*, pages 200–211. Cambridge, England.

Carterette, B. (2009b). On rank correlation and the distance between rankings. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 436–443. Boston, Massachusetts.

Carterette, B., Allan, J., and Sitaraman, R. (2006). Minimal test collections for retrieval evaluation. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 268–275. Seattle, Washington.

Chapelle, O., Metzler, D., Zhang, Y., and Grinspan, P. (2009). Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 621–630. Hong Kong, China.

Chen, H., and Karger, D. R. (2006). Less is more: Probabilistic models for retrieving fewer relevant documents. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 429–436. Seattle, Washington.

Clarke, C. L., Kolla, M., Cormack, G. V., Vechtomova, O., Ashkann, A., Büttcher, S., and MacKinnon, I. (2008). Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 659–666. Singapore.

Clarke, C. L. A., Kolla, M., and Vechtomova, O. (2009). An effectiveness measure for ambiguous and underspecified queries. In *Proceedings of the 2nd International Conference on the Theory of Information Retrieval*, pages 188–199. Cambridge, England.

Cleverdon, C. W. (1967). The Cranfield tests on index language devices. *AsLib proceedings*, 19(6):173–193. Reprinted as Cleverdon (1997).

Cleverdon, C. W. (1997). The Cranfield tests on index language devices. In *Readings in Information Retrieval*, pages 47–59. San Francisco, California: Morgan Kaufmann.

Cormack, G. V., and Lynam, T. R. (2006). Statistical precision of information retrieval evaluation. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 533–540. Seattle, Washington.

Cormack, G. V., and Lynam, T. R. (2007). Power and bias of subset pooling strategies. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 837–838. Amsterdam, The Netherlands.

Cormack, G. V., Palmer, C. R., and Clarke, C. L. A. (1998). Efficient construction of large test collections. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 282–289. Melbourne, Australia.

Custis, T., and Al-Kofahi, K. (2007). A new approach for evaluating query expansion: Query-document term mismatch. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 575–582. Amsterdam, The Netherlands.

De Angelis, C., Drazen, J., Frizelle, F., Haug, C., Hoey, J., Horton, R., Kotzin, S., Laine, C., Marusic, A., Overbeke, A., et al. (2004). Clinical trial registration: A statement from the International Committee of Medical Journal Editors. *Journal of the American Medical Association*, 292(11):1363–1364.

Efron, B., and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Boca Raton, Florida: Chapman & Hall/CRC.

Fisher, R. A. (1925). Theory of statistical estimation. *Proceedings of the Cambridge Philosophical Society*, 22:700–725.

Gardner, M. J., and Altman, D. G. (1986). Confidence intervals rather than p values: Estimation rather than hypothesis testing. *British Medical Journal*, 292(6522):746–750.

Goffman, W. (1964). A searching procedure for information retrieval. *Information Storage and Retrieval*, 2:73–78.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70.

Järvelin, K., and Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446.

Kelly, D., Fu, X., and Shah, C. (2007). *Effects of Rank and Precision of Search Results on Users' Evaluations of System Performance.* Technical Report 2007-02. University of North Carolina, Chapel Hill.

Lenhard, J. (2006). Models and statistical inference: The controversy between Fisher and Neyman-Pearson. *British Journal for the Philosophy of Science*, 57(1).

Moffat, A., Webber, W., and Zobel, J. (2007). Strategic system comparisons via targeted relevance judgments. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 375–382. Amsterdam, The Netherlands.

Moffat, A., and Zobel, J. (2008). Rank-biased precision for measurement of retrieval effectiveness. *ACM Transactions on Information Systems*, 27(1):1–27.

Najork, M. A., Zaragoza, H., and Taylor, M. J. (2007). HITS on the Web: How does it compare? In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 471–478. Amsterdam, The Netherlands.

Robertson, S. (2006). On GMAP – and other transformations. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 78–83. Arlington, Virginia.

Sakai, T., and Kando, N. (2008). On information retrieval metrics designed for evaluation with incomplete relevance assessments. *Information Retrieval*, 11(5):447–470.

Sanderson, M., and Joho, H. (2004). Forming test collections with no system pooling. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 33–40. Sheffield, England.

Sanderson, M., and Zobel, J. (2005). Information retrieval system evaluation: effort, sensitivity, and reliability. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 162–169. Salvador, Brazil.

Savoy, J. (1997). Statistical inference in retrieval effectiveness evaluation. *Information Processing & Management*, 33(4):495–512.

Shah, C., and Croft, W. B. (2004). Evaluating high accuracy retrieval techniques. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2–9. Sheffield, England.

Smucker, M., Allan, J., and Carterette, B. (2007). A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the 16th ACM conference on Conference on Information and Knowledge Management*, pages 623–632. Lisbon, Portugal.

Soboroff, I., Nicholas, C., and Cahan, P. (2001). Ranking retrieval systems without relevance judgments. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 66–73. New Orleans, Louisiana.

Spärck Jones, K., Robertson, S. E., and Sanderson, M. (2007). Ambiguous requests: Implications for retrieval tests. *ACM SIGIR Forum*, 41(2):8–17.

Thomas, L. (1997). Retrospective power analysis. *Conservation Biology*, 11(1):276–280.

Turpin, A., and Scholer, F. (2006). User performance versus precision measures for simple search tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 11–18. Seattle, Washington.

van Zwol, R., Murdock, V., Garcia Pueyo, L., and Ramirez, G. (2008). Diversifying image search with user generated content. In *Proceedings of the 1st ACM International Conference on Multimedia Information Retrieval*, pages 67–74. Vancouver, Canada.

Vee, E., Srivastava, U., Shanmugasundaram, J., Bhat, P., and Amer-Yahia, A. (2008). Efficient computation of diverse query results. In *Proceedings of the 24th IEEE International Conference on Data Engineering*, pages 228–236. Cancun, Mexico.

Voorhees, E., and Harman, D. (1999). Overview of the eighth text retrieval conference. In *Proceedings of the 8th Text REtrieval Conference*, pages 1–24. Gaithersburg, Maryland.

Voorhees, E. M. (2000). Variations in relevance judgments and the measurement of retrieval effectiveness. *Information Processing & Management*, 36(5):697–716.

Voorhees, E. M. (2004). Overview of the TREC 2004 Robust Track. In *Proceedings of the 13th Text REtrieval Conference*. Gaithersburg, Maryland.

Voorhees, E. M., and Dang, H. T. (2005). Overview of the TREC 2005 Question Answering track. In *Proceedings of the 14th Text REtrieval Conference*. Gaithersburg, Maryland.

Voorhees, E. M., and Harman, D. K. (2005). The Text REtrieval Conference. In Voorhees, E. M., and Harman, D. K., editors, *TREC — Experiment and Evaluation in Information Retrieval*, chapter 1, pages 3–20. Cambridge, Massachusetts: MIT Press.

Webber, W., Moffat, A., and Zobel, J. (2008a). Score standardization for inter-collection comparison of retrieval systems. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 51–58. Singapore.

Webber, W., Moffat, A., and Zobel, J. (2008b). Statistical power in retrieval experimentation. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 571–580. Napa, California.

Yilmaz, E., and Aslam, J. A. (2008). Estimating average precision when judgments are incomplete. *International Journal of Knowledge and Information Systems*, 16(2):173–211.

Yilmaz, E., Kanoulas, E., and Aslam, J. A. (2008). A simple and efficient sampling method for estimating AP and NDCG. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 603–610. Singapore.

Zhai, C., Cohen, W. W., and Lafferty, J. (2003). Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 10–17. Toronto, Canada.

Zobel, J. (1998). How reliable are the results of large-scale information retrieval experiments? In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 307–314. Melbourne, Australia.

第13章

Information Retrieval: Implementing and Evaluating Search Engines

度量效率

第12章讨论了搜索引擎有效性的度量方法，也就是，它们产生的搜索结果的质量。但是，结果的质量只是一个检索系统整体实用性中的一个指标。第二个也是同样重要的一个指标是效率。**有效性**（effectiveness）描述专门为完成某项工作而设计的系统所完成工作的好坏程度，而**效率**（efficiency）则关注于工作时系统的资源消耗情况。

在搜索引擎中，效率通常指以下三个不同方面中的一个：时间效率（多快?）、空间效率（需要多少内存/磁盘?）或开销效率（启动和维持系统运行需要多少开销?）。这三个方面彼此紧密影响。例如，花费更多的金钱会增加系统的内存资源，但因此也可以让更多的数据存放在RAM中，从而让系统运行得更快；通过复杂索引压缩技术，可以降低对内存的需求，这样做可能会降低全局开销，但是由于采用了更复杂的解压程序，因此可能会使查询处理变慢。

本章将主要讨论时间效率，因为时间效率是用户唯一可感觉的指标，甚至也可以将它看做是有效性的一个方面：越快越好（多数情况下）。贯穿本章，我们将**性能**（performance）等同于**时间效率**（time efficiency），并把它们作为评价查询速度的标准（如，秒/查询，查询/秒）。

本章由三部分组成。第一部分（13.1节）关注性能指标：一个搜索引擎很快是什么意思？我们引入**吞吐量**（throughput）和**延迟**（latency）作为两个重要的性能指标。第二部分（13.2节）通过排队论来分析系统在高负荷下的运作情况，从而探讨这两个指标的关系。最后一部分（13.3节和13.4节）关注两种不同类型的性能优化方法：查询调度和缓存。尽管它们都与度量过程无直接关联，但都对评价结果有很大影响。另外理解它们在设计性能测试实验时的作用也很重要。

13.1 效率标准

度量一个计算机系统的性能是很困难的。个人计算机在20世纪80年代中期成为一种商品，那时大部分制造商在宣传他们的产品时，还在使用MHz频率作为评价CPU效率的主要标准。随着时间流逝，处理器频率作为评价系统性能的指标变得越来越不可靠了，新一代的微处理器总是宣称比上一代更快，尽管它们有着相似甚至更低的时钟频率。当前CPU生产商的注意力已经从CPU频率上转移了，不再关注CPU频率和每时钟周期执行指令的次数，而是转向关注让每台计算机拥有更多的CPU核，所以在这种情况下要评价一个计算机系统的性能就更困难了。

人们尝试用一些实际的性能指标来告诉我们一台计算机有多快或多慢，从而规避了直接评价CPU性能这个难题。第一个提议是由**标准性能评价机构**（Standard Performance Evaluation Corporation，SPEC）带头提出的，该机构是一个在1988年建立的非营利性组织。SPEC已经着手研发用于评价计算机系统各性能方面的多种基准。例如，他们的SPECfp基准就主要关注计算机的浮点单元。SPECfp得分是一项衡量对于需要扩展数值计算的任务计算机性能的指标，如物理模拟或是3D绘图。另外，SPECweb2005基准设计成用来衡量Web服务器这样的计算机性能的指标（它们的工作包括提供HTML网页，使用PHP生成

动态内容等）。完整的基准列表可以在 SPEC 的网站[⊖]上找到。

适合每个搜索引擎性能基准的标准化序列是不可能存在的。即便存在一个，也不可能明确地知道它到底有多大的作用，鉴于一个特定排名算法的部署决策通常代表效率和有效性之间的权衡。例如 8.6 节中的伪相关反馈机制。由表 8-2 中的评价结果可知，通过采用伪相关性反馈机制总是可以大幅提高查准率。但是，这么做要求有一秒的时间来传输数据，那么搜索引擎的响应就变慢了。如果没有采用用户学习的方法，那么要知道伪相关性反馈是否实际上提升了搜索引擎的整体性能就很困难了。实际中，因为大量时间和开销都和用户学习关联，所以如何权衡效率和有效性常常是由策略决策决定的，且这些策略决策不是由实验数据得出的。由于对于这个问题没有一般性的解决方案，因此我们可以忽略它而假设搜索引擎的排名函数是固定的，这样我们就可以专注于系统效率。

13.1.1　吞吐量和延迟

当微调搜索引擎的性能时，我们常常会陷于左右为难的境地——两组不相上下的实现方案（它们的搜索结果是一样的），我们想知道哪一个更好。两种最常见的性能指标——吞吐量（throughput）和延迟（latency）可以回答这个问题。

- 因此指给定周期内搜索引擎处理查询的数量。这个值常常以**每秒查询数量**（queries per second，qps）作为衡量单位。如果搜索引擎在 5 秒内处理了 700 条查询，那它的吞吐量就是 140 qps。

当提及吞吐量时，我们常常指的是**理论吞吐量**（theoretical throughput），又称为**服务速率**（service rate），它表示系统处理查询可能达到的最快速率。一个紧密关联的指标是**服务时间**（service time），即处理器在活跃状态下处理一条查询的时间。服务时间和服务速率可通过下面的公式联系起来

$$serviceRate = \frac{m}{serviceTime} \tag{13-1}$$

其中，m 表示系统中处理器的数量。

- **延迟**，也称做**响应时间**（response time），即搜索引擎从接收查询到返回用户结果所用的时间。它的度量单位是**每条查询的秒数**（seconds per query）。

可以度量在服务器端或在客户端上的延迟。后者称为**端到端延迟**（end-to-end latency），它表示从用户发出查询到用户收到搜索结果这段时间。它包含了网络延迟，比搜索引擎处理查询造成延迟更加能够反映用户的满意程度。但是，进行性能比较评价时，只关注实际的搜索延迟就足够了，因为网络带来的额外的延迟是一个无关于实现方案的常量。

如果只是针对单个查询，那么吞吐量和延迟是毫无意义的。当查询的总量非常大时，达到上千乃至百万级时，就可用一个均值来代表它们：**平均吞吐量**（mean throughput）或**平均延迟**（mean latency），这与我们报告例如平均查准率均值这样的有效性指标是类似的（见 12.1 节）。

回到之前如何判定两种实现方案哪个更好的问题上来，我们通过使用在有效性评价中使用的工具来解决这个问题：在两种实现方案的系统上处理同一个查询集$\mathcal{Q}=\{q_1,\cdots,q_n\}$，记录下每一条查询 q_i 的延迟（或服务时间），就可以计算出置信区间来分辨这两种方案间的不同之处（可以回顾一下 12.3.2 节置信区间的计算和解释）。如果置信区间包含 0，我们就可以

⊖ www.spec.org

由此推断出这两种方案间没有多大的区别。否则，我们就可以知道哪个方案更快，并且丢弃另外一个。

相比于有效性评价，性能评价有它特有的优势，即不需要任何人工的判断。我们可以轻易地度量搜索引擎处理成千上万条查询的服务时间，这样可以在低于 1% 的误差下可靠地检测出它们性能间的差别。要在有效性评价中也达到这样敏感的级别将会带来巨额开销。

延迟≠服务时间

人们可能会认为延迟和服务时间（或延迟和吞吐量）是相同的度量。但是这两者是非常不同的。系统延迟绝对大于它的服务时间，而且通常会大大超过。除了实际处理查询开销的时间，延迟还包括等待周期，如：

- 如果索引不是存储在内存中，那就需要等待硬盘传输位置信息数据。这个等待的周期并不计入系统服务时间中，因为等待从磁盘读取位置信息数据时可以处理其他的查询。
- 查询队列等待直到 CPU 可用。如果在系统高负载的情况下，排队效应可能是延迟的一个主要原因（见 13.2 节）。
- 一个分布式搜索引擎有 n 个独立的索引服务器，当前 i 个服务器完成任务后，需等待余下的 $n-i$ 个服务器返回它们的结果。

除了这种下限关系，吞吐量和延迟间还常常存在一种**对抗性**（adversarial）关系，即通过提升其中一个来改进系统性能，往往会导致另一个指标的下降。如果将我们的思路扩展到并行化的情况，那么这种对抗性就更为明显。假设我们有一个搜索引擎，它有两个索引服务器 I_1 和 I_2。那至少有两种方法可以分割查询处理负载：

- **索引复制**（index replication）：I_2 是 I_1 的副本。它们都维护着全部索引的一个副本。当一个查询到来时，它传送给 I_1 或 I_2 的概率都是 50%。
- **索引划分**（index partitioning）：I_1 和 I_2 各索引着 50% 的文档集合。当一个查询到来时，它被转发到这两个索引服务器。I_1 和 I_2 各自根据它们索引的内容计算前 k 个检索结果。当这两个索引服务器都完成各自的任务后，这两个结果集合并成最终结果并返回给用户。

我们比较这两种方法，均给它们只配置一台服务器，并分析吞吐量和延迟。索引复制可能增加一倍的吞吐量，因为每个索引服务器只处理 50% 的查询。它对延迟没有影响，因为单个查询仍然只被一个服务器处理。索引划分同时将吞吐量和延迟提高到一样的程度，但却因为下述几个原因，使加速度小于 100%：

- 一些操作涉及的查询处理开销独立于索引的大小。例如，如果索引存储在磁盘中，系统需要为每个查询词项至少执行一次磁盘寻道。这个开销不会因为将整个集合分半而减少。
- 搜索/排名处理的计算复杂度与索引规模的关系不是线性的。5.1.1 节中的 MaxScore启发策略可以用于避免为那些不在靠前的搜索结果里的文档进行评分。如果使用了MaxScore，索引越大，那些需要评分的文档就会越小。

因此，如果我们的注意力主要放在吞吐量上，那么就没有必要提升延迟，采用索引复制就是正确的选择[⊖]。否则，索引划分就更加合适。一个大规模的搜索引擎，同时面对延迟和吞吐量问题，可以考虑结合这两种方法。

⊖ 除了纯粹性能的考虑，复制相对于划分的优点在于它提供了容错：如果索引服务器 I_1 或 I_2 失败了，尽管速率下降了，但系统仍能处理查询。使用索引划分，如果一个索引服务器失败了，系统就极有可能不能用了，因为它将丢失搜索结果中 50% 的内容。第 14 章将讨论分布式搜索引擎中的容错。

13.1.2　汇总统计和用户满意度

和有效性指标一样，效率指标的汇总统计（如平均吞吐量或平均延迟）不总是特别有意义。考虑两个假定的搜索系统 A 和 B，系统 A 常常需要 500 ms 来处理一条查询，而系统 B 采用了成熟的缓存策略，所以 90% 的查询可以在 10 ms 内处理完毕，而其余的 10%，则平均每条查询需要 4.9 s。两种系统都达到了同样的平均延迟，但是 A 提供的用户体验远比 B 要好，因为 10% 的用户将无法忍受不可预料的长响应时间。

固然上面的例子是虚构的，这样极端的差别在现实中是很少遇到的。尽管如此，它也足以说明如果我们仅用平均性能度量是会陷入相似的问题中的。为了克服这个问题，人们通常使用百分比来取代平均值。例如，一个搜索引擎公司的技术目标是 99% 的查询都可以在 1 s 内完成。从用户的角度看，这种指标远远比平均延迟指标更有意义。它还有另外受欢迎的一面，它可以回答在前一节中提到的延迟和吞吐量之间的权衡问题。延迟现在不再是个自由变量。我们可以微调系统来优化吞吐量，以使 99% 的查询都可以在 1 s 内完成。

13.2　排队论

搜索引擎的操作常常面对着如何决定搜索引擎的查询容量的问题，即估计当满足特定的延迟需求时，例如前面所说的 99% 的查询在 1 s 内完成的标准，最大的吞吐量可以达到多少。一种解决的方法是执行性能测试实验：通过传送查询给搜索引擎（例如，重现之前记录的查询日志），并逐渐增加查询到达搜索系统的频率，直至达到预定义的延迟极限。遗憾的是，不总是有合适的时机来运行这样的测试。首先，它耗费了珍贵的计算资源而不是用这些资源来处理实际的查询。其次，如果搜索引擎还没有投入使用，也很难获取准确反映查询到来时间分布的查询日志。

另一种获取实际性能的测试实验是使用数学模型来预测实验的输出，这种方法基于对搜索引擎查询处理性能的估计。假设我们每条查询都逼近于平均服务时间。那我们在已知平均查询到达速率的情况下，可以用排队论的方法来计算系统的平均延迟。通过反向计算，就可以计算出在满足预定义延迟需求时可能达到的最大吞吐量。

给定平均服务时间和查询到达速率，我们用于估计延迟的模型是基于以下三个基本假设：

1）两个连续查询到来的时间差（**到达间隔时间**（inter-arrival time））是一个随机变量 A，它服从指数分布，即密度函数为

$$f_A(x) = \lambda \cdot \mathrm{e}^{-\lambda x} \quad (x \geqslant 0) \tag{13-2}$$

其中，λ 是**查询到达速率**（query arrival rate），$E[A] = \int_0^\infty x \cdot f(x,\lambda)\ \mathrm{d}x = 1/\lambda$ 是两个查询先后到达的平均时间差。这个模型有时可以称为**泊松过程**（Poisson process），因为在固定时间间隔内到来的查询数量服从**泊松分布**（Poisson distribution）公式（8-34）。

2）每条查询的服务时间是一个随机变量 s，它也服从指数分布，密度函数为

$$f_S(x) = \mu \cdot \mathrm{e}^{-\mu x} \quad (x \geqslant 0) \tag{13-3}$$

其中，μ 是系统的服务速率（即它的理论吞吐量）。$E[S] = 1/\mu$ 是每条查询的平均服务时间。我们假设 $\mu > \lambda$，否则搜索引擎就不能处理所有到来的查询。

3）查询以**先来先服务**（first-come-first-served，FCFS）的模式处理。系统通常一次处理一条查询。

假设 1 基于的观点是：大量查询都是相互独立的，并且在时间上服从均匀分布。这一假设通常是有效的，至少在考虑短周期的情况下是成立的，比如几分钟或半小时。对于较长的周期（几天或是数周），假设 1 就不能成立了，因为用户一般白天比晚上要活跃，工作日比周末要活跃，诸如此类。

如果从理论上考虑，假设 2 是不成立的，但是却满足实际的服务时间分布。例如图13-1给出了在 TREC TB 2006 高效任务和 GOV2 文档集的模式独立索引中运行 10 000 条查询时，邻近度排名（图 2-10）和 Okapi BM25（公式（8-48））的服务时间分布（只是 CPU 时间，不含等待硬件的时间）。尽管两种排名算法非常不同，但是两种算法中每条查询的服务时间都可以使用密度函数为$f_S(x)=\mu\cdot e^{-\mu x}$（其中邻近度排名中的$\mu=4.37$ qps，而 BM25 的$\mu=0.75$ qps）的指数分布合理地建模。

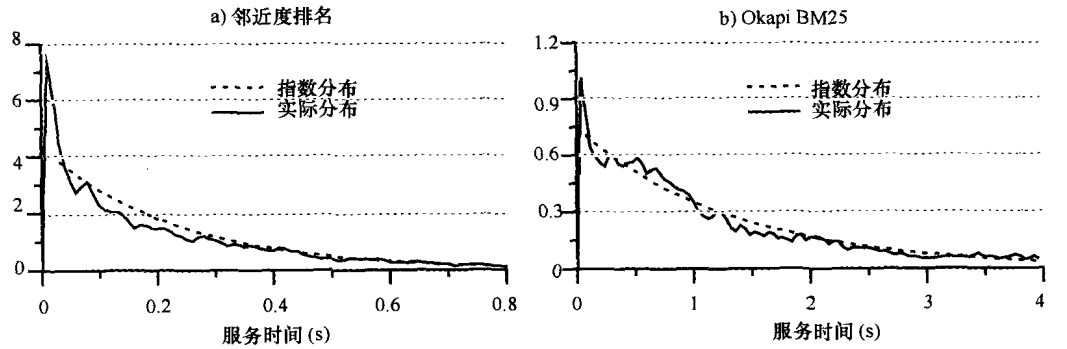


图 13-1 以每条查询的服务时间（即 CPU 时间）为随机变量建立的指数分布模型
a）平均服务时间$E[S]=0.23$s（μ=4.37 qps）的邻近度排名（图 2-10）；
b）平均服务时间为$E[S]$=1.34s（μ=0.75 qps）的 Okapi BM25(公式(8-48))

假设 3 是为了简化模型。大规模搜索引擎常常使用很多复杂的调度算法。当系统在高负荷下运行时，选择正确的调度算法会对延迟有很大的影响。但是，复杂的调度算法越多，分析就越困难，所以我们暂时忽略它们。到 13.3 节我们再重新回过头来探讨这个主题。

13.2.1 肯德尔符号

上面描述的排队模型对应于**肯德尔符号**（Kendall's notation）中的$M/M/c/\infty/\infty/F$模型，这是以英国统计学家 David G.Kendall 的名字命名的（不是另一个发明了 Kendall 符号τ的英国统计学家 Maurice G.Kendall）。

- 第一个“M”指查询到达的过程，这个过程我们假设为**马尔可夫过程**（Markovian）（即指数）。
- 第二个“M”指服务时间分布，假设也是一个马尔可夫过程。
- “c”指的是系统中**服务通道**（service channel）（即服务器）的数量。为了简单起见，我们仅讨论 $c=1$ 的情况。但是，注意到这样的情形也包括由 n 个并行服务器组成的分布式系统，其中每个服务器负责 $1/n$ 的索引。
- 第一个“∞”指的是队列中槽的数量，这里我们假设是无限的。
- 第二个“∞”指的是正在查询的人数，这里我们也认为是无限的。
- “F”是**排队规则**（queue discipline），也就是系统处理查询的次序。如前所述，我们采用 FCFS 的策略。

因为完整的肯德尔符号的最后三个参数常常都是相同的（$\infty/\infty/F$），所以时常使用精简的符号$M/M/c$来取代累赘的符号$M/M/c/\infty/\infty/F$。

13.2.2　M/M/1 排队模型

现在我们来描述如何通过$M/M/1$排队模型来计算搜索引擎的延迟分布。在我们深入讨论计算细节之前，让我们先重述一个常用于排队分析的理论。

Little 定律（Little，1961）

设 n 为系统在某个随机时间点的查询平均数（包含了队列中正在处理或等待的查询）。再者，令λ为查询到达速率。那么系统的平均查询响应时间 r 是

$$r = \frac{n}{\lambda} \tag{13-4}$$

公式（13-4）的证明过程比较简单（在 Little 的论文中可以找到正式的证明过程）。考虑在足够长的时间间隔 t 内有q 条查询到来（$q=\lambda \cdot t$）。那么系统处理这 q 条查询的总时间就是 $r_{\text{total}}=r \cdot q$。由于当一条查询在系统中的时候（包含了队列中正在处理或等待的查询）才被计入总时间 r_{total}，因此有 $r_{\text{total}}=n \cdot t$，由此推出 $r=n/\lambda$。注意到我们对于服务时间和到达时间间隔的分布没有做任何假设。那么 Little 定律就可以应用到任何的排队模型上，而不仅仅是 $M/M/1$ 模型。

为了计算搜索引擎的平均延迟，我们首先计算系统在任一时间点上处理的期望查询数量。我们可以把搜索引擎看做是一个概率状态机，其中 I_i 状态表示“系统中有 i 条查询”（即 1 条查询正在被处理，而 $i-1$ 条查询在队列中等待）。每个状态与概率 p_i 相关：系统在某一时刻处于状态 Z_i 的概率。这种类型的状态机称为**连续马尔可夫链**（continuous Markov chain），如图 13-2 所示。

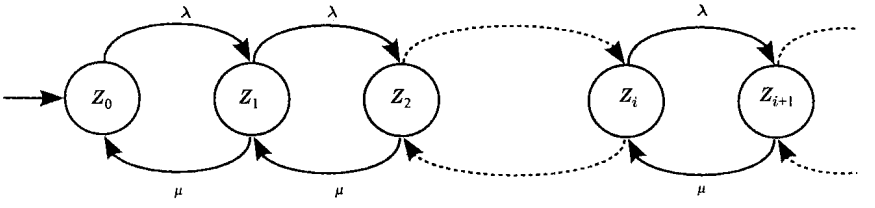


图 13-2　一个连续马尔可夫链，把系统的查询队列描述为带有无限个状态的状态机。状态 Z_i 表示“系统中有 i 条查询”。状态转移 $Z_i \rightarrow Z_{i+1}$ 和 $Z_{i+1} \rightarrow Z_i$ 的相对频率由查询到达速率λ和服务速率μ 分别给出

我们对两个相邻状态 Z_i 和Z_{i+1}间的转移以及它们之间的相对频率感兴趣。让 T^+表示从 Z_i 转移到Z_{i+1}，T^-表示反方向的转移。在平均情况下，单位时间内 T^+发生$\lambda \cdot p_i$ 次，而 T^-发生 $\mu \cdot p_{i+1}$次。因为我们假设查询到达过程和服务过程都是马尔可夫过程（即无记忆的），它们各自的数量都不取决于系统中的查询数量。所以，分数

$$\frac{\lambda \cdot p_i}{\mu \cdot p_{i+1}} \tag{13-5}$$

对于所有 $i \geqslant 0$ 都是一样的。这一观察的本质是：平均来说，两种类型的状态转移发生的频率都是一样的。很明显，T^-不可能比 T^+发生得更频繁，因为只有一条查询到达后才有可能被处理。另外，T^+也不可能比 T^-发生得更加频繁；如果是，那么队列中查询的平均数量就是个无穷值了（同时我们会有$\lambda \geqslant \mu$）。因此，这两项必须具有相同的值：

$$\lambda \cdot p_i = \mu \cdot p_{i+1} \tag{13-6}$$

或等价于：

$$p_{i+1} = \frac{\lambda}{\mu} \cdot p_i \tag{13-7}$$

为了方便，我们定义$\rho = \lambda/\mu$（ρ称为**流量强度**（traffic intensity）或**利用率**（utilization））。那我们就可以得到

$$p_i = p_0 \cdot \rho^i \; \forall \; i \geqslant 0 \tag{13-8}$$

由于 p_i 构成了概率分布，因此它们的和为 1。由此我们可以计算 p_0：

$$p_0 \cdot \sum_{i=0}^{\infty} \rho^i = 1 \Leftrightarrow p_0 \cdot \frac{1}{1-\rho} = 1 \Leftrightarrow p_0 = 1 - \rho = 1 - \lambda/\mu \tag{13-9}$$

系统中的查询数量 N 遵循下式：

$$E[N] = \sum_{i=0}^{\infty} i \cdot p_i = \sum_{i=0}^{\infty} i \cdot (1-\rho) \cdot \rho^i = \frac{\rho}{1-\rho} \tag{13-10}$$

最后，通过应用 Little 定律（公式（13-4））我们可以得到

$$E[R] = \frac{E[N]}{\lambda} = \frac{1}{\mu(1-\rho)} = \frac{1}{\mu - \lambda} \tag{13-11}$$

其中，R 是一个随机变量，表示每条查询的响应时间（即延迟）。如果系统服务时间 S 和到达间隔时间A 都服从指数分布，那么响应时间 R 也服从指数分布（Harrison，1993）：

$$f_R(x) = (\mu - \lambda) \cdot \mathrm{e}^{-(\mu-\lambda)x} \tag{13-12}$$

为了方便，表 13-1 总结了从 $M/M/1$ 模型派生出来的重要公式。

表 13-1 $M/M/1$ 排队模型的关键公式（λ：到达速率；μ：服务速率）

统计量	公式
平均利用率	$\rho = \lambda/\mu$
系统中查询的平均数量	$E[N] = \rho/(1-\rho)$
平均响应时间	$E[R] = 1/(\mu - \lambda)$
响应时间分布	$f_R(x) = (\mu - \lambda) \cdot \mathrm{e}^{-(\mu-\lambda)x}$

13.2.3 延迟量和平均利用率

让我们回到 13.1.2 节中假设的目标状态：对于 99% 的查询，搜索引擎的延迟小于 1 s。假设引擎处理每条查询平均需要 100 ms（即 $\mu = 10$）。那么引擎响应时间的密度函数为

$$f_R(x) = (10 - \lambda) \cdot \mathrm{e}^{-(10-\lambda)x} \tag{13-13}$$

而延迟低于 1 s 的那部分查询为

$$\int_0^1 f_R(x)\,\mathrm{d}x \; = \; 1 - \mathrm{e}^{\lambda - 10} \tag{13-14}$$

对于 99% 的查询，我们有 $\mathrm{e}^{\lambda-10} = 1 - 0.99$，这意味着$\lambda \approx 5.4$。也就是说，我们可以保持搜索引擎的平均利用率在 54% 上，并且可以达到我们的目标——99% 的查询延迟小于1 s。如果更进一步提升目标，我们希望 99.9% 的查询延迟都小于 1 s，那么结果是$\lambda \approx 3.1$，对应 31% 的平均利用率。

上述的例子说明了搜索引擎的理论吞吐量（与它的服务速率一样）和它实际上获得的吞吐量（不会导致不合理的高延迟）间的不同。实际上，高于 50% 的平均利用率就被认为是

非常高的，而且不太可能。并且，注意到我们所有的计算都是基于到达速率λ为常数这一假设。这个假设通常只对短时间周期有效，也许是几分钟，但对较长的时间间隔就失效了，比如一整天，因为用户在长周期中的活动波动很大（大部分人都是晚上睡觉的）。因为查询到达速率的变化，长时间的利用率水平大概为 20% 或是更低都是很常见的。因此，相比于用不考虑延迟的简单吞吐量分析去预测，不如在硬件上花费 5 倍乃至更多的资源。

13.3　查询调度

在上一节，我们对搜索引擎延迟 R 作了如下假设：查询以先来先服务的方式处理。这种排队规则背后的理由是公平性：我们不希望 Alice 在 Bob 前提交了查询，却在 Bob 后接收到她的搜索结果。FCFS 就保证了这种次序，搜索结果的返回次序总是和用户查询到来的次序一致。但是，如果我们主要的目的不是公平，而是最小化延迟，FCFS 就不是一种最优策略了。

考虑以下情景：有两条查询 q_i 和 q_{i+1} 在队列中等待处理。假设我们知道它们各自的服务时间 s_i 和 s_{i+1}，并且 $s_i = s_{i+1} + \delta$ ($\delta > 0$)。我们可以交换两个查询的次序，将 q_i 的延迟增加到 s_{i+1}，同时将 q_{i+1} 的延迟减少为 s_i，从而它们平均的延迟降低到 $\delta/2$。

改变搜索引擎处理查询次序的机制称为**调度算法**（scheduling algorithm）。为了简单起见，我们只讨论以下三种算法：

- **FCFS**（先到先服务）。这是我们目前的讨论中假设使用的调度算法。
- **SJF**（最短任务优先）。每当搜索引擎完成一条查询处理，同时从队列中选择下一条查询来处理时，总是选择那条具有最短预测服务时间的查询。
- **DDS**（截止驱动调度）。对于系统中的每个查询，我们定义它的截止时间为它到来后 1 s。每当搜索引擎处理完一条查询后，就对查询队列重新排序，最小化超过截止时间的查询数量。如果出现平局（即两种排序都得到同样数目的超时的查询），引擎就选择平均延迟最小的那个排序。

注意到预测查询 q 的服务时间是很困难的。但是一般使用粗估计就足够了，可以利用查询词项的位置信息列表的长度和查询中词项的数目获得这个粗估计。

图 13-3 展示了我们模拟在服务速率 $\mu = 10$ qps 时搜索引擎的实验结果。在做这个实验时，我们让查询到达速率在 $\lambda = 1$ qps 到 $\lambda = 8$ qps（即平均利用率在 10%～80%）。每个模拟都持续数小时（模拟时间），处理几十万条查询。图 13-3 描绘了三种调度算法中搜索引擎的平均延迟和查询时间大于 1 秒的那部分查询。你可以看出，在高负载下，选择正确的调度算法将有很大的影响。例如当 $\rho = 0.7$ 时，FCFS 导致了 5% 的查询超时。但如果使用 DDS，它的目标就是最小化超时查询的数目，那么就只有 1% 的查询超时。

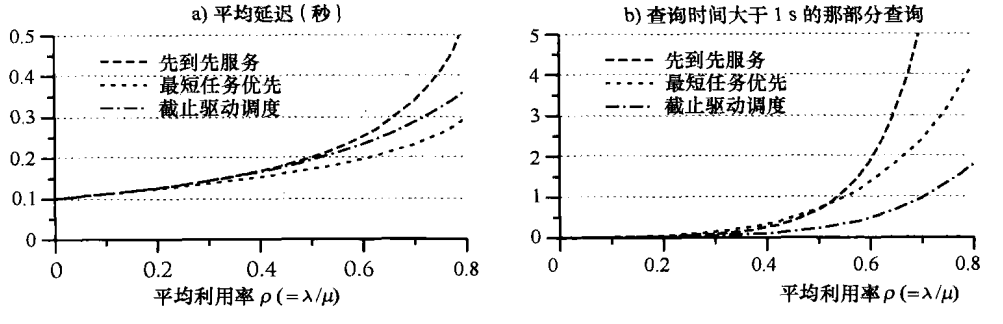


图 13-3　模拟在不同的到达速率λ下，不同的查询调度算法产生的延迟。服务速率固定为 $\mu = 10$ qps

延迟开销函数

暂不讨论各个算法降低全局查询延迟的能力，SJF 和 DDS 都有一个明显的缺点：**饥饿**（starvation）。如果某条查询的服务时间被预测得很高，那么它在被处理前可能需要等待一段很长的时间。例如，尽管 SJF 把延时 1 秒以上的查询比例从 5% 降低到 2.5%（$\rho=0.7$），但是把延时超过 2 秒的查询比例从 0.02% 提高到了 0.06%。如果使用 DDS，这一效果相对没有那么明显，但是延时超过 2 秒的查询比例依然达到 0.05%。

一般来说，单独一个数值，比如低于某个延迟阈值的查询百分比是不能完整地描述搜索引擎的性能和它对用户满意度的影响的。在降低某一种类型的查询延迟前，我们就不可以避免地要增加另一种类型的查询延迟。因此，在选择具体调度算法前，必须明确我们的全局目的，或是定义一个开销函数来帮助我们分辨一种给定延迟水平的不良影响程度和最小化每个查询的平均开销。

一个线性的开销函数 $c(x)=x$ 表示最小化平均延迟的目标。一个多项式的开销函数 $c(x)=x^q(q>1)$ 考虑了搜索引擎的延迟在处理长查询时会引起大幅变化，从而导致用户烦躁的情况。一个恰当的开销函数只能通过用户学习获取。当选择了一个查询后，我们就可以选择一种最优的调度算法来处理它。

13.4 缓存

前面的章节我们已经探讨了不改变平均服务时间的情况下降低搜索引擎延迟的方法。我们讨论的查询调度算法不需要修改任何查询处理逻辑；只是通过简单地改变搜索引擎处理查询的调度次序来实现降低延迟的目标。在本节中，我们将主要探讨另一种优化策略：通过缓存最频繁使用的数据片来降低平均服务时间。

缓存是一种经典的**时间与空间的权衡**（time-versus-space trade-off）方法。如果我们给定系统额外的内存资源，那么系统就可能使用这些内存来存储前面查询的搜索结果。每当搜索引擎接到一条查询时，它首先检查它的内部缓存来判断是否已经包含了搜索结果。仅当没在缓存中找到搜索结果时，系统才需要真正地处理查询来获得搜索结果。

刚开始，好像缓存和构造可靠的、有意义的性能指标这一问题是没有关系的。但是，明白缓存在查询流中的特性是非常重要的。例如，在 AOL 查询日志[⊖]中，平均查询长度大概是 2.8 个词项。如果我们假设一个结果缓存的容量是无限的，那么每个不同的查询都只需要处理一次就足够了，那么平均每个查询的长度增加到 3.5 个词项，因为越长的查询重复的可能就越小。因此，我们评价搜索引擎的服务速率时，是评价缓存前还是缓存后的服务速率有很大的差别。而且，一些缓存只有在一段时间后（**预热期**（warmup period））才能达到它们的预期命中率，这意味着重启后搜索引擎的有效服务速率可能比预期的低很多。

13.4.1 三级缓存

当为搜索引擎设计缓存系统时，我们首先必须弄清楚哪些类型的数据是值得缓存的。很显然答案是“搜索结果”。如果用户 A 输入一条查询，而用户 B 在几分钟后同样输入这条查询，那么就没有必要重新处理该查询了。相反的，搜索引擎应该缓存用户 A 最后获得的搜

⊖ AOL 查询日志包含了 2006 年 3 月到 5 月 AOL 搜索引擎上接收到的近 2 千万条查询。它在 2006 年 8 月初被公开，因其涉及私人信息问题遭到大家的热议，并在发布数日后被迫下线。通过在自己喜欢的搜索引擎上输入“aol query log”关键字，仍可在网上找到数据的副本。

索结果，并将缓存的版本传送给 B。

缓存搜索结果对处理常见查询的效果是非常好的，比如“选举”和“全球变暖”这类常常被议论的话题。但是，缓存本身也有局限性。就像文本集中的词频，查询频率大概服从齐夫分布，其中第 i 个最常见词项的频率 $f(i)$ 为

$$f(i) = c \cdot i^{-\alpha} \tag{13-15}$$

其中 c 是某个常数，$\alpha > 1$（具体可见 1.3.3 节中对齐夫分布的详细描述）。对于查询分布，参数 α 通常非常接近于 1，这意味着分布图形上伴随着大量的散点（只出现一次的查询）将出现一条“长长的尾巴”。Baeza-Yates 等人（2007）对从 www.yahoo.co.uk 网站上得到的一年的查询日志进行分析，发现 44% 的查询都只出现一次。平均来说，日志中的每条查询在整个周期内只被请求过两次。因此，即使有无限的缓存空间，缓存的命中率也不会高于 50%。无需多说，在缓存中保存一条查询多天，对诸如 Web 这样多变的文本集来说是不可接受的。因此，缓存命中率低于 50% 的情况也是很常见的。

Long 和 Suel（2005）提出了**多级缓存**（cache hierarchy）策略，该策略中缓存从三个不同的层面影响搜索引擎，从而解决了上述问题。

1. 搜索结果。从每条查询的观点来看，这是最有效的缓存策略。当在缓存中找到一条查询时，搜索引擎几乎马上就可以返回结果，节省几毫秒到数百毫秒不等的服务时间。在分布式的搜索引擎中，缓存一般位于查询代理或是专用服务器上。所以当查询结果在缓存中找到时，就没有必要连接索引服务器了。

遗憾的是，如前所述，一般缓存的搜索结果不可能超过查询流的 50%。因为多数长查询都是散点，因此可被缓存的查询流基本都是包含 2～3 个词项的短查询，这进一步限制了缓存的作用。因此，使用结果缓存总共节省的时间比期望缓存命中率要低。Long 和 Suel 的实验指出，因为缓存的查询都较短，50% 的命中率只能将每条查询的平均开销降低 33%，因此相对于没有缓存的查询来说，它们的处理只是快一点而已。

2. 交叉列表。很多搜索引擎都采用合取检索模型（布尔 AND）。为了找到包含所有查询词项的文档集，它们必须交叉所有查询词项的位置信息列表。这可以用“term-at-a-time”的模式处理。第一步，对前两个查询词项的位置信息列表进行交叉处理；第二步，将第一步得到的结果和第三个词项位置信息列表合并起来；如此类推。

通过缓存共有查询词项对的交叉列表，我们可以减少对每个查询所做的工作量。例如，处理查询〈“global”，“warming”，“facts”〉，我们缓存“global”和“warming”这两个词项的交叉列表。一段时间后，当我们遇到查询〈“global”，“warming”，“controversy”〉时，我们就可以重用由前面查询得到的交叉列表，从而降低执行开销。Long 和 Suel 对基于磁盘索引做交叉列表缓存作了估计，发现可以降低大概 50% 的平均服务时间。如果索引是在内存中，加速度可能会更小。

3. 位置信息列表。除了在前面两个层次上的缓存策略，我们还可以通过在内存中维护频繁被访问的位置信息列表，来降低搜索引擎的查询处理开销。这种方法的效果依赖于是否采用了交叉列表缓存。按照 Long 和 Suel 的说法，如果引擎只缓存搜索结果而不能缓存交叉列表，那么增加列表缓存可以降低 45% 的平均查询开销。然而，如果同时缓存了交叉列表，位置信息列表缓存只可以节省约 20% 的开销。

Turpin 等人（2007）增加了第四项缓存级别：文档。尽管缓存文档对搜索处理没有影响，但是可以加快显示给用户的文档摘要的产生——这也是常被（学术上的）信息检索研究忽视的一个方面。

13.4.2 缓存策略

在我们决定了缓存什么样的数据后，我们仍必须解决最后一个问题：哪些项目应该被保存在缓存中？既然缓存不可能有足够的空间来保存所有可能的搜索结果、交叉列表和位置信息列表，那么我们就不得不考虑哪些是值得被保存在缓存中的，而哪些是不值得的。

缓存策略（cache policy）（或**缓存算法**（cache algorithm)），就是在给定了项目集合和缓存空间容量时，用来决定哪些项目应该存储在缓存中。缓存策略可以是**静态**（static）的也可以是**动态**（dynamic）的。当使用静态策略时，缓存中的项目集是事先确定的，在搜索引擎启动后也不会改变。但它可以周期性的更新，例如在更新索引时一同更新。在动态策略中，每当系统发现一个值得缓存的新对象（如，位置信息列表），缓存中的项目集就会相应发生改变。动态缓存策略又称为**替代算法**（replacement algorithm），因为往缓存中增加一个项目常常意味着要剔除另外一个。

1. 缓存策略的一般目标

缓存是一个几乎在所有计算机科学领域中使用到的概念。因此，我们希望重用那些在其他领域中已有的且已被证明成功的缓存策略。有以下两种这样的策略：

- **最近最少使用**（LRU）。每当一个新项目所加入的缓存空间已满时，就将那个最长时间未被访问的项目从缓存中剔除。
- **最不经常使用**（LFU）。每当一个新项目所加入的缓存空间已满时，就将那个最不经常使用的项目（既然它被加载到内存中）从缓存中剔除。

当用在一个大小（几乎）固定的对象集上时，例如一组查询的前 k 个搜索结果集，LRU 和 LFU 的效果都相当不错，但是当把这两种策略用于大小差异很大的对象集上时，很快就会达到它们的极限。例如，如果简单地将 LRU 应用到上一节所述的三级缓存中，那将会有超过 90% 的缓存被位置信息列表占用，原因很简单：位置信息列表比交叉列表或是搜索结果都要大得多。这对缓存来说不是最经济的做法。

2. 开销感知缓存策略

一种解决上述问题的方法是：将缓存分割成三个桶，每个对应一级缓存。每个桶的大小都根据以往相应缓存类型的经验进行分配。如果桶的大小选择得很好，这种方法会得到令人满意的性能。但是，我们还能做得比这更好。基本观点是：将每个可缓存的项目 x 表示成以下三种属性：

- C_x 表示把 x 加载到缓存所导致的开销。与 x 的大小（以字节为单位）成一定比例。
- g_x 表示缓存 x（比不缓存 x）所带来的期待增益。根据 x 的类型（即它在多级缓存中的层次），这可能是一个查询的服务时间，也可能是计算一个交叉列表所需的时间，或是内存载入一个位置信息列表所需的时间。
- p_x 表示在固定周期内 x 被使用次数的期望值。你可以认为 p_x 是一个概率。

把 x 载入缓存的期望净收益为：

$$\mathrm{ENB}(x) = \frac{p_x \cdot g_x}{c_x} \tag{13-16}$$

当然，这条公式不受搜索引擎限制。开销感知缓存在很多应用中都是很有用的，包括 WWW 代理（Cao 和 Irani，1997）。

公式（13-16）隐式地定义了缓存策略：按 ENB 排序所有的项目 x；将前 n 个项目存储到缓存，选择适当的 n 使得前 n 条项目的联合大小不超过缓存容量。这种缓存策略有两个

便利的性质。第一，它顺理成章地包含了 13.4.1 节中讨论的三种项目类型。第二，参数 C_x、g_x 和 p_x 可由以前的查询日志估计得到。第二个性质很重要，因为这允许我们将 ENB 转变为一个静态策略，由此可以在初始化阶段就把前 n 个项目预加载到缓存中。假设查询的分布和查询词项只随着时间缓慢变化，那么采用静态策略可以接近最优性能。

有时，有些人会争论上述策略是次优的，因为它可能会留下小部分未使用的缓存（**内部碎片**（internal fragmentation））。但是，这种争论仅是理论层面上的。实际上，每个缓存的项目相对缓存总容量来说都是很小的，因此碎片是可以忽略的。

13.4.3　预取搜索结果

如果我们能够访问一个已经存在的查询日志，就可以用它来预取我们期望不久就会被请求的查询的结果，即使它们目前还不在缓存中。例如，假设我们现在用搜索引擎查询新闻、博客或是其他对时间敏感的信息，这使得我们无法重用缓存超过一两个小时的搜索结果。每一天结束后，缓存命中率常常达到 30%～40%，但这时我们的用户去睡觉了，流量就会降低，到了早上，缓存命中率就可能接近 0 了。

假设明天的查询与今天的类似，我们可以主动提交一组较常使用的查询，使得当用户早上醒来的时候这组查询的结果位于缓存中。结果预取不会降低搜索引擎的整体负载，但由于提高了用户查询的缓存命中率而降低了平均延迟。并且，也有助于减少日间闲时和忙时的负载差异，当搜索引擎接近理论吞吐量时，这是非常重要的。

我们可以再进一步扩展这个想法，不断预取我们预料在一天中的某个时段中将被发出的查询的结果。Beitzel 等人（2004）对来自商业 Web 搜索引擎中长达一周的查询日志进行了分析，发现“例如，私人财务查询，在早上 7～10 点变得较多，而音乐查询就变得较少”。当然，如果在第二天发起相同查询前，缓存项目到期或已从缓存中剔除，预取策略才是有用的。

13.5　延伸阅读

Lilja（2000）就评价计算机各方面性能作出了一个易于理解的介绍。对于排队论更深入的讨论，包括比 13.2 节中简单的 $M/M/1$ 模型更高级的模型，请参见 Gross 等人（2008）或 Kleinrock（1975）的论文。

在 13.4.1 节中讨论的三级缓存是由 Long 和 Suel（2005）首次提出的。三级缓存是基于 Saraiva 等人（2001）研究的二级缓存的。

Baeza-Yates 等人（2007）研究了几个关于缓存结果和位置信息列表的算法，包括 Q_{TF} D_F 算法，一种近似于 13.4.2 节中 ENB 策略的算法。Garcia（2007）研究了大量的缓存策略，包括开销感知算法。Fagni 等人（2006）讨论如何将搜索引擎的缓存分为静态部分和动态部分，并对各个部分使用不同的算法。他们证明了混合静态/动态缓存比纯粹的动态或静态策略更优，因为混合静态/动态缓存既利用了以前的静态查询，又能适应不断变换的查询分布。Zhang 等人（2008）研究了同时采用交叉列表缓存和索引压缩的效果。

在其他的情况下，TREC 组织者将针对效率的任务加入到 TREC 提出的普通的基于有效性的评价中。这样的努力得到的其中两个成果是 VLC（very large corpus，非常大的语料库）和 TB（terabyte，百万兆字节）级检索。它们均出现在 Hawking 和 Thistlewaite（1997）、Hawking 等人（1998）、Clarke 等人（2005）和 Büttcher 等人（2006）的论文中。

13.6　练习

练习 13.1　假设搜索引擎处理查询的速率为 $\mu=20$ qps。查询的到达速率为 $\lambda=15$ qps。设想服务时间和

间隔到达时间都服从指数分布，那么搜索引擎的平均延迟是多少？延迟超过 1 秒的查询所占的百分比是多少？当搜索引擎处理 99.9%的查询的延迟均低于 1 秒时，最大吞吐量是多少？

练习 13.2 对两个调度算法 SJF 和 DDS（13.3 节中讨论的），请分别给出一个延迟开销函数

$$c:[0,\infty)\rightarrow[0,\infty)$$

将延迟映射到开销，并使每条查询的平均开销最小。

练习 13.3 考虑一个无限查询流的查询分布服从指数定律

$$p(i)=\frac{c}{i^{1.05}}$$

其中 $p(i)$是第 i 个最频繁的查询出现的概率，c 是一个常数，使得$\sum_{i=1}^{\infty}p(i)=1$。使用静态缓存策略，选择 1 000 000 个最频繁出现的查询，那么可以通过缓存回答的查询比例是多少？下面这个近似可能是有用的：

$$\sum_{i=1}^{\infty}i^{-\alpha}\approx 0.5772+\int_{1}^{\infty}x^{-\alpha}\mathrm{d}x$$

练习 13.4 考虑静态缓存策略 $Q_{TF}D_F$。该策略缓存了前 n 个得分最高的词项的位置信息列表，其中词项 t 的得分定义如下：

$$score(t)=\frac{f_Q(t)}{N_t}$$

其中 $f_Q(t)$是词项 t 在查询（通过现有的查询日志获取）中出现的次数，而 N_t 是包含词项 t 的文档数量。在什么条件下，$Q_{TF}D_F$ 等价于 13.4.2 节中的 ENB 策略？

13.7 参考文献

Baeza-Yates, R., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., and Silvestri, F. (2007). The impact of caching on search engines. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 183–190. Amsterdam, The Netherlands.

Beitzel, S. M., Jensen, E. C., Chowdhury, A., Grossman, D., and Frieder, O. (2004). Hourly analysis of a very large topically categorized Web query log. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 321–328. Sheffield, England.

Büttcher, S., Clarke, C. L. A., and Soboroff, I. (2006). The TREC 2006 terabyte track. In *Proceedings of the 15th Text REtrieval Conference (TREC 2006)*. Gaithersburg, Maryland.

Cao, P., and Irani, S. (1997). Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, pages 193–206. Monterey, California.

Clarke, C. L. A., Scholer, F., and Soboroff, I. (2005). The TREC 2005 terabyte track. In *Proceedings of the 14th Text REtrieval Conference*. Gaithersburg, Maryland.

Fagni, T., Perego, R., Silvestri, F., and Orlando, S. (2006). Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Transactions on Information Systems*, 24(1):51–78.

Garcia, S. (2007). *Search Engine Optimisation Using Past Queries*. Ph.D. thesis, RMIT University, Melbourne, Australia.

Gross, D., Shortle, J., Thompson, J., and Harris, C. (2008). *Fundamentals of Queueing Theory* (4th ed.). New York: Wiley-Interscience.

Harrison, P. G. (1993). Response time distributions in queueing network models. In *Performance Evaluation of Computer and Communication Systems, Joint Tutorial Papers of Performance '93 and Sigmetrics '93*, pages 147–164. Santa Clara, California.

Hawking, D., Craswell, N., and Thistlewaite, P. (1998). Overview of TREC-7 very large collection track. In *Proceedings of the 7th Text REtrieval Conference*. Gaithersburg, Maryland.

Hawking, D., and Thistlewaite, P. (1997). Overview of TREC-6 very large collection track. In *Proceedings of the 6th Text REtrieval Conference*, pages 93–106. Gaithersburg, Maryland.

Kleinrock, L. (1975). *Queueing Systems. Volume 1: Theory.* New York: Wiley-Interscience.

Lilja, D. J. (2000). *Measuring Computer Performance: A Practitioner's Guide.* New York: Cambridge University Press.

Little, J. D. C. (1961). A proof for the queueing formula L=λW. *Operations Research*, 9(3):383–387.

Long, X., and Suel, T. (2005). Three-level caching for efficient query processing in large web search engines. In *Proceedings of the 14th International Conference on World Wide Web*, pages 257–266. Chiba, Japan.

Saraiva, P. C., de Moura, E. S., Ziviani, N., Meira, W., Fonseca, R., and Riberio-Neto, B. (2001). Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 51–58. New Orleans, Louisiana.

Turpin, A., Tsegay, Y., Hawking, D., and Williams, H. E. (2007). Fast generation of result snippets in web search. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 127–134. Amsterdam, The Netherlands.

Zhang, J., Long, X., and Suel, T. (2008). Performance of compressed inverted list caching in search engines. In *Proceeding of the 17th International Conference on World Wide Web*, pages 387–396. Beijing, China.

第五部分　应用和扩展

第 14 章

Information Retrieval: Implementing and Evaluating Search Engines

并行信息检索

信息检索系统常常得处理大规模的数据。这些系统需要处理数千兆乃至数兆兆字节级的文本，并建立和维护数百万文档的索引。虽然第 5～8 章介绍的技术可以满足这种需求，但在某种程度上，单单靠复杂的数据结构和巧妙的优化方法是明显不够的。仅仅一台计算机所拥有的计算和存储能力甚至不足以索引万维网（Word Wide Web[1]）上的一小部分网页。

在本章中，我们将介绍各种大规模文本集（如 Web）上的信息检索系统的构建方法。第一部分（14.1 节）关注的是并行查询处理，即通过使用多个索引服务器来处理多个同时到达的查询，从而提高搜索引擎的服务效率。这部分也将讨论分布式搜索引擎中的冗余和容错问题。在第二部分（14.2 节），我们将注意力转到离线任务的并行执行上，例如文本语料库的索引构造和统计分析。之后我们将介绍一种用于处理大量数据的大规模并行计算框架 MapReduce 的基础。

14.1　并行查询处理

并行化技术可以通过多种方式来帮助搜索引擎更快地处理查询，两种最常用的方法是索引划分（index partitioning）和复制（replication）。假设有 n 个索引服务器，以标准的术语来说，将索引服务器称为节点（node）。通过将索引复制 n 份并分发到 n 个独立的节点上，便可将搜索引擎的服务速度提高 n 倍（在理论和计量下），而且不会影响单个查询所需的时间。这种并行方式称为查询间并行（inter-query parallelism），因为多个查询并行处理，但单个查询还是顺序处理的。另一种方式，我们将索引分成 n 份，每个节点只处理属于它的那一小部分。这种方法称为查询内并行（intra-query parallelism），因为每个查询都由多个服务器并行处理。这样便提高了搜索引擎的服务速度，以及每次查询的平均时间。

这一节我们主要关注查询内并行，我们研究将索引划分为多个独立的部分的索引划分方案，使得每个节点可以处理整体索引中的一小部分。

有两种主要的索引划分方案：文档划分（document partitioning）和词项划分（term partitioning）（如图 14-1 所示）。在文档划分索引中，每个节点只保存文档集中一个子集的索引。例如在图 14-1a 中，节点 2 维护的索引包含以下文档编号列表：

$$L_1=\langle 4,6\rangle,\ L_2=\langle 5\rangle,\ L_3=\langle 4\rangle,\ L_5=\langle 6\rangle,\ L_7=\langle 4,6\rangle$$

---

[1] 没有人知道万维网上可索引网页的确切大小。估计至少有 100 亿个网页。2005 年 8 月，根据 Yahoo! 最新公布的数据，它索引的全部文档数达到了 19.2 亿个（http://www.ysearchblog.com/archives/000172.html）。

在词项划分索引中，每个节点只负责文档词项集的一个子集。如图 14-1b 中的节点 1 包含以下列表：

$$L_1 = \langle 1, 3, 4, 6, 9 \rangle, \quad L_2 = \langle 2, 5 \rangle, \quad L_3 = \langle 2, 3, 8 \rangle$$

这两种划分策略会使系统处理查询的方式大大不同。在文档划分搜索引擎中，n 个节点中的每一个都要处理引擎收到的所有查询。而在词项划分中，只有当节点上的索引包含查询中的至少一个词项时，该节点才会处理这个查询。

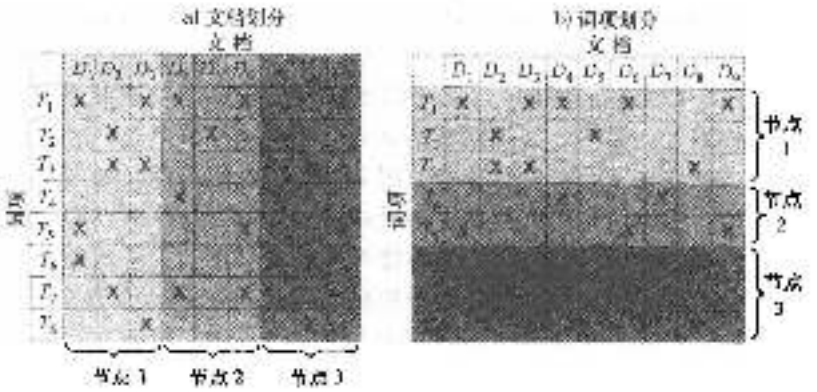


图 14-1　两种流行的索引划分方案：文档划分和词项划分（假设索引包含 8 个词项和 9 个文档）

14.1.1　文档划分

在文档划分索引中，每个索引服务器只负责文档集中的一个子集。一个前端服务器接收来自用户的查询，然后接收服务器（receptionist）将查询转发给所有 n 个索引节点，等待它们处理查询，并把索引节点上返回的搜索结果合并，把最后的结果列表返回给用户。一个文档划分搜索引擎示意图如图 14-2 所示。

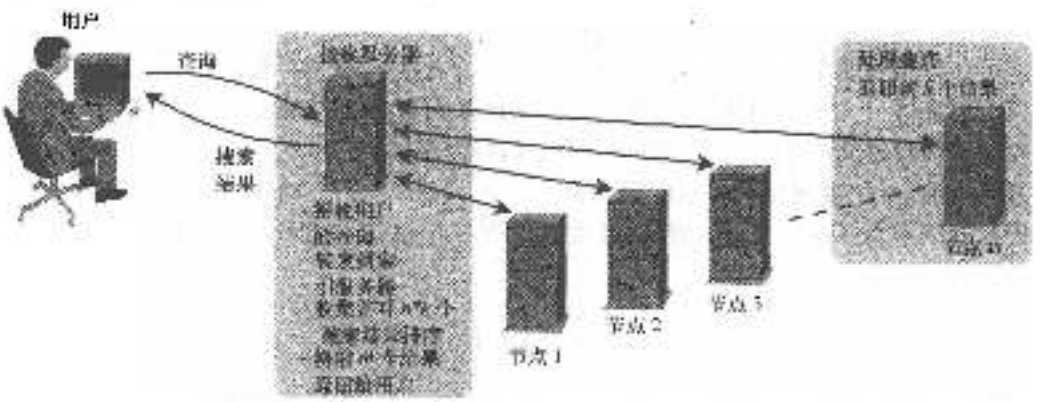


图 14-2　带一个接收服务器和若干个索引服务器的文档划分查询处理过程。每个接收到的查询都转发给所有的索引服务器

文档划分方法最大的优点在于简单。因为所有索引服务器都独立运行，并且在低层次的查询处理事务中也没有额外的复杂需求。所需工作也就是接收服务器把查询转发给后台，然后接收分别从 n 个节点中返回的前 k 个搜索结果，从中选出前 m 个返回给用户（m 的值通常由用户指定，而 k 的值通常由搜索引擎使用者决定）。除了转发查询和搜索结果外，接

收服务器还维护一个缓冲区，用来存放最近/最频繁的查询结果。

如果搜索引擎维护的是一个允许更新的动态索引（如文档插入/删除），那么甚至有可能在分布式的方式下执行更新，每个节点只需要更新各自的索引就可以了。这种方法避免了复杂的涉及整个索引的集中式索引构造/维护过程。但是，只有当各文档之间是彼此独立的情况下，这种方法才是可行的；如果跨文档信息，如超链接和锚文本，也是索引的一部分内容，那这种方法就不可行了。

当决定如何将文档集划分到 n 个索引节点上时，一种可能的方式是文档-节点分配方式——例如，将相似的文档保存在同一节点中。在 6.3.7 节中，我们已经知道如果根据文档的 URL 对文档集中的文档进行重排序，就可以更好地压缩索引，因此具有相似 URL 的文档就会有相邻的文档编号。显然，如果同域网页被分配到同一节点上，那么这种方法是最有效的。但这种方法的问题在于有可能造成索引服务器的负载不均衡。如果给定节点主要包含某一主题的相关文档，当这个主题突然变得很受欢迎时，那么这个节点的查询处理负载就会大大高于其他节点。最后就会导致可用资源不能得到最大利用。为了避免这个问题，通常最好的做法就是不要什么额外的操作，只是简单地将文档集划分为 n 个完全随机子集。

当索引划分好后，每个子索引就加载到其中一个节点中去，接下来就要决定每个节点的返回集大小 k 的值。如何根据用户指定的返回结果个数 m 来选择 k 呢？假设用户要求返回 $m=100$ 个结果。为了保险起见，我们可以让每个索引节点都返回 $k=100$ 个结果，确保接收服务器能接收到整体文档集上排名前 100 的结果。但是，因为以下两个原因，这不是一个好的决定：

1）不太可能所有前 100 个结果都来自同一个节点。每个索引服务器都返回 100 个结果给接收服务器，就得耗费许多不必要的负载。

2）对于执行如 MAXSCORE（参见 5.1.1 节）这样的启发式策略，k 的取值对于查询处理性能有很大的影响。表 5-1 说明了当在 GOV2 的词频索引上进行查询时，结果集大小从 $k=100$ 减少到 $k=10$ 时，每个查询的平均 CPU 时间减少了 15%左右。

给定索引服务器的个数 n 和每节点返回集大小 k，Clarke 和 Terra（2004）提出了一种方法，可计算接收服务器能收到整体文档集前 m 个结果的概率。他们的方法基于这样的假设：当文档集被分为 n 个子集后，每个文档被随机分配到一个索引节点上，则每个节点以相同概率返回整体文档集上前 1、2、3……个最好的结果。

考虑由前 m 个搜索结果组成的集合 $R_m=\{r_1, r_2 \cdots, r_m\}$。对于每一个文档 r_i，被某个节点找到的概率是 $1/n$。因此，前 m 个结果中的 l 个被该节点找到的概率满足如下的二项分布：

$$b(n,m,l)=\binom{m}{l}\cdot\left(\frac{1}{n}\right)^{l}\cdot\left(1-\frac{1}{n}\right)^{m-l} \tag{14-1}$$

如果要求 n 个索引节点均返回前 k 个结果，集合 R_m 中所有成员被找到的概率可以通过以下递归公式计算：

$$p(n,m,k)=\begin{cases}1 & \text{若 } m\leqslant k\\ 0 & \text{若 } m>k \text{ 且 } n=1\\ \sum_{l=0}^{k} b(n,m,l)\cdot p(n-1,m-l,k) & \text{若 } m>k \text{ 且 } n>1\end{cases} \tag{14-2}$$

前两种基本情况显而易见。在递归情况中，考虑系统的第一个节点，计算由这个节点返回整

体文档集上前 m 个结果中的 $l=0$，1，2，…，k 个的概率（即 $b(n,m,l)$）。对于 l 的每个可能取值，这个概率都要乘以 $p(n-1,m-l,k)$，即 $\mathcal{R}_m$ 中剩下的 $m-l$ 个文档被其他的 $n-1$ 个索引节点发现的概率。公式（14-2）看起来没有一个闭合形式的解，但可以通过复杂度为 $\Theta(n,m,k)$ 的动态规划来解决。

图 14-3 给出了以 99.9%以上的概率找到前 m 个结果时单个节点所需检索的深度 k。对于 $m=100$ 和 $n=4$，单个节点检索深度 $k=41$ 就能达到预期概率水平。如果放松正确率要求，从 99.9%降到 95%，那么 k 就可以从 41 减少到 35。

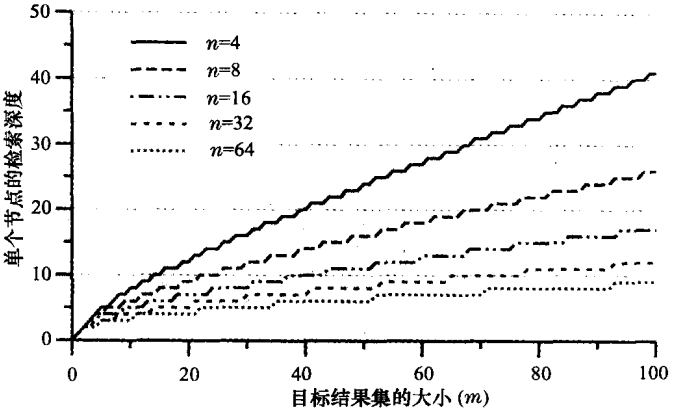


图 14-3　选择最小的检索深度 k，使得返回前 m 个结果的概率 $p(n,m,k)>99.9\%$，其中 n 为文档划分索引中的节点数量

有时候根据不同的 m 值来优化检索深度 k 会更好，而不是由用户指定。例如，如果接收服务器为最近发起的查询维护一个缓冲区，对于一个给定查询，即使用户指定只要前 10 个结果，还是有必要维护前 20 个结果，这样点击“下一页”时可以直接从缓冲区里得到结果。允许接收服务器检查结果集 $\mathcal{R}_{m'}$（$m'>m$）也是非常有用的，因为这样更便于应用多样性搜索重排技术，如 Carbonell 和 Goldstein（1998）提出的**最大边界相关法**（maximal marginal relevance）或是 Google 的启发式**密集主机**[⊖]（host crowding）。

14.1.2　词项划分

虽然文档划分通常是不错的选择，而且与节点数量成线性关系，但只有单个节点上的索引数据都存储在内存或低延迟的可随机访问的存储介质中（如闪存），这个方法才能发挥最大的潜力，如果数据存储在磁盘里，情况就不是这样了。

考虑一个文档划分搜索引擎，所有的位置信息列表都存储在磁盘上。假设每个查询平均包含 3 个单词，我们希望搜索引擎可以处理的峰值查询负载为每秒钟 100 个查询（100 qps）。回忆 13.2.3 节的内容，因为排队效应，利用率通常不能超过 50%，除非可以忍受经常性的延迟跳动。因此，查询负载 100 qps 转换成对应的服务器速度就是至少 200 qps 或等价于每秒钟 600 次的随机访问操作（每个操作对应一个查询词项）。假设平均磁盘寻道延迟是 10 ms，单个硬盘驱动器每秒不能超过 100 次随机访问操作，只达到期望吞吐量的 1/6。我们试图通过为每个节点增加硬盘来突破这个限制，但是为每个服务器都装配 6 个硬盘不太实际。而且，不管增加多少节点到系统中，系统都处理不了每秒几百个查询的负载。

词项划分通过把文档集划分成词项集，而不是文档集，来解决磁盘寻道问题。每个索引节点 v_i 只负责某个特定词项集 $\mathcal{T}_i$，只有当给定查询包含的词项属于 $\mathcal{T}_i$ 时，该节点才会处理这个查询。我们关于词项划分查询处理的讨论是基于 Moffat 等人（2007）提出的流水线架构。这种架构中，查询以“term-at-a-time”的方式来处理（具体细节可参见 5.1.2 节中介绍

⊖ www.mattcutts.com/blog/subdomains-and-subdirectories/

的 term at a time 的查询处理策略)。

假设查询包含 q 个词项 t_1，t_2，…，t_q，那么接收服务器把查询转发给负责词项 t_1 的节点 $v(t_1)$。从 t_1 的位置信息列表中创建一组文档得分累加器后，$v(t_1)$将查询和这组累加器一起转发给词项 t_2 的节点 $v(t_2)$，$v(t_2)$更新累加器集合，然后传给 $v(t_3)$，依次类推。当流水线的最后一个节点 $v(t_q)$也完成后，就把最终的累加器集合传给接收服务器。接收服务器从中选择前 m 个搜索结果返回给用户。这个方法的示意图如图 14-4 所示。

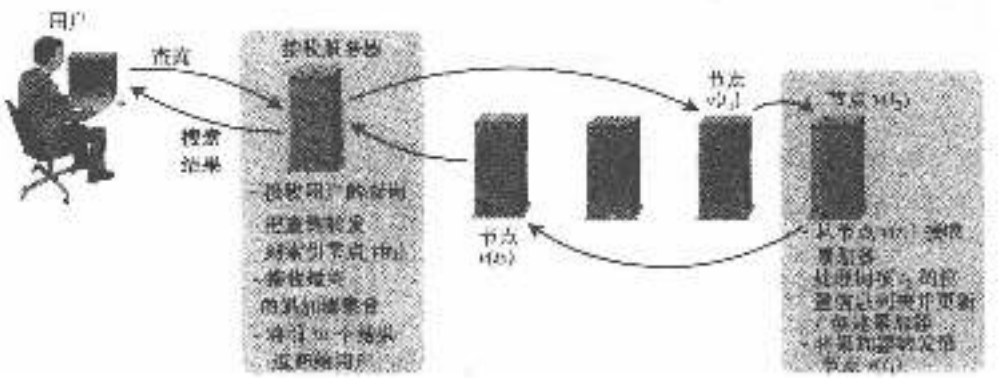


图 14-4 词项划分查询处理方案：带 1 个接收服务器和 4 个索引服务器，根据查询中的词项（图中查询包含 3 个词项），每个查询在索引服务器之间传递

很多用于顺序 term-at-a-time 查询处理的优化也同样适用于词项划分查询处理：首先处理不频繁词项；可用累加器裁剪策略维护累加器集合的大小，从而使全局网络流量得到控制；使用影响力排序（impact ordering）可以有效确定一个给定词项最重要的位置信息。

注意到，上述流水线查询处理架构并不适用于查询内并行，因为每个查询在每个阶段只由一个节点负责。因此，尽管这里描述的简单形式的词项划分有助于提高系统的理论吞吐量，但却不能降低搜索引擎的响应时间。如果接收服务器转发查询给 $v(t_1)$的同时也预取指令传给 $v(t_2)$，…，$v(t_q)$，可在一定程度上解决这个问题。当节点 $v(t_i)$接收到累加器集合时，部分 t_i 的位置信息可以已经加载到内存，从而查询的处理就更快了。

尽管词项划分潜在的性能优于文档划分方法（至少在磁盘索引上是这样），但也存在一些缺点使得这种方法在实际应用中难以采用：

- **可扩展性。**当文档集更大时，每个位置信息列表也会变大。对于一个包含十亿个文档的语料库来说，一个频繁词项的位置信息列表很容易就达到几百兆字节，如 "computer" 或 "internet"。处理一个包含几个类似这样的高频词的查询至少需要几秒，这是大多数用户所不能忍受的。为了解决这个问题，需要将长位置信息列表分割为多个小块并分到不同节点上去，每个节点负责位置信息列表的一小部分且并行工作。但是，这使得查询处理逻辑变得复杂了。
- **负载均衡。**词项划分受到索引节点间负载不均衡的困扰。对应单个词项的负载是该词项在文档集和用户查询中词频的函数。如果词项的位置信息列表非常长，并且在检索查询中也常出现，那么对应的索引节点的负载会比系统中的平均负载要高。为了解决这个问题，引起高负载的位置信息列表就要复制并分发到多个节点上。这样，与该词项相关的计算负载就由多台机器共同负担，但这样增加了存储需求的成本。
- **term-at-a-time。**词项划分方法最严重的问题可能在于它无法支持高效的 document-

at-a-time 查询处理。为了在词项划分索引的顶部计算 document-at-a-time 的得分，与裁剪累加器集合相反，整个位置信息列表都要在网络中传递。由于位置信息列表非常庞大，因此这是非常不实际的。因此，document-at-a-time 的排名方法，如 2.2.2 节中的邻近度排名函数，也不适用于词项划分索引。

即便有这些缺点，词项划分有时也是一个不错的选择。例如，回忆 13.4.1 节中的三级缓冲模型。第二级缓冲的是列表的交集（参见 2.2 节，带布尔 AND 查询语义的搜索引擎）。我们不选用文档划分方法，因为需要为每一个节点配备一个交集缓冲区，这时便可选择词项划分索引并把缓存的列表交集视为普通的位置信息列表一样处理。如图 14-4 展现的例子，如果我们之前查询过 $\langle t_1, t_2 \rangle$，那么在节点 $v(t_2)$的缓冲区中就有 $\langle t_1 \wedge t_2 \rangle$ 的交集列表；当处理一个新的查询 $\langle t_1, t_2, t_3 \rangle$ 时，就可以跳过节点 $v(t_1)$。

更一般的情况，如果位置信息列表相对比较短，无论是本身所代表的信息比较短（例如，列表交集）还是人为缩短了（例如，运用了索引裁剪技术，具体可参见 5.1.5 节），词项划分更是一个不错的选择。

14.1.3　混合方案

考虑一个 n 个节点的分布式索引和大小为 $|c|$ 的文档集。当 $|c|/n$ 很小时，磁盘寻道是整个查询处理中的主要耗费，此时文档划分的效率就比较低。但是，当 $|c|$ 很大时，用词项划分处理一个查询所需时间又长得难以忍受。

Xi 等人（2002）提出了一种混合的架构，通过一种标准的文档划分模式将文档集划分为 p 个子集。每个子集的索引在 n/p 个节点上进行词项划分，使得系统中的每个节点只负责 p 个子集之一的词项集的出现。在合适的负载均衡策略下，可以使吞吐量增加 n 倍，寻道延迟减少 p 倍。

作为混合词项/文档划分的一个替代方案，我们也可考虑混合文档划分和复制。注意词项划分的主要目标是增加吞吐量，而不是减少延迟。因此，我们不在 p 个子索引上进行词项划分，而是通过简单地将每个子索引复制 n/p 次并在 n/p 个完全一样的副本中实现查询的负载均衡，也能获得同样的性能水平。在一些高端的应用中，2003 年 Google 就采用了这种索引布局（Barroso 等人，2003）。通过这种方法，全局最大吞吐量和平均延迟就大致和混合文档/词项划分的情况相同。但由于需要 n/p 次复制，存储需求就有点高。如果索引存储在磁盘上，那就不成问题了。

14.1.4　冗余和容错

对于一个拥有成千上万用户的大规模搜索引擎，可靠性和容错与响应时间和检索质量同样重要。为了得到更高的查询负载，我们给搜索引擎增加机器数量，那么越来越有可能在某一时刻某台机器会出错。如果设计系统时考虑了容错功能，那么单个机器的出错对吞吐量和检索质量只会有微小的影响。而如果设计系统时没有考虑容错功能，一个错误就可能使搜索引擎无法工作。

我们在一个 32 个节点的分布式搜索引擎中比较简单的无复制的文档划分和词项划分方案。如果词项划分索引中的某个节点出错了，那么搜索引擎就不能处理包含任何一个由这个节点管理的词项的查询了。不包含这些词项的查询则不受影响。对于一个包含 3 个词项的随机查询 q（Web 查询的平均词项个数），q 仍能被剩下的 31 个节点处理的概率为：

$$\left(\frac{31}{32}\right)^3 \approx 90.9\% \tag{14-3}$$

如果文档划分索引中的某个节点出错了，搜索引擎仍能处理所有的查询，但是会丢失一些搜索结果。如果划分是无偏的，则对于一个随机查询而言，前 k 个结果中遗漏了 j 个的概率是

$$\binom{k}{j} \cdot \left(\frac{1}{32}\right)^j \cdot \left(\frac{31}{32}\right)^{k-j} \tag{14-4}$$

因此前 10 个结果不受这个出错影响的概率为

$$\left(\frac{31}{32}\right)^{10} \approx 72.8\% \tag{14-5}$$

由此可以看出，机器出错对词项划分索引的影响比文档划分索引的影响要低。然而，这样的比较有些不公平，因为不能处理 1 个查询比丢失前 10 个结果中的 1 个要严重得多。如果我们考察因为索引节点失效而导致前 10 个结果中至少丢失 2 个的概率，有：

$$1 - \left(\frac{31}{32}\right)^{10} - \binom{10}{1} \cdot \left(\frac{1}{32}\right) \cdot \left(\frac{31}{32}\right)^9 \approx 3.7\% \tag{14-6}$$

因此，由单个节点出错而对一个查询造成严重影响的概率是非常小的，大多数用户不太会注意到其中的差别。根据这一论点，单节点失效的情况下文档划分索引比词项划分索引性能下降慢。

对于多个相关结果的信息式查询，这种方式足够好了。但对于导航式查询则不然（参见 15.2 节介绍的关于信息式查询和导航式查询的不同）。以一个导航式查询〈“white”，“house”，“website”〉为例。这个查询只有一个重要结果（http：//www.whitehouse.gov/），**必须**（must）出现在搜索结果的最前面。如果文档划分索引的 32 个节点中有一个出错了，则对于每个导航式查询都有 3.2% 的概率会丢失这个最重要的结果（如果对于该查询只有一个最重要的结果）。有很多方法可以解决这个问题，常用的有以下 3 种：

- **复制**（Replication）：如前一节所提到的混合划分方案，我们可为同一个索引节点维护多个副本，并让它们并行地处理查询。如果给定索引节点的 r 个副本中有一个失效了，剩下的 $r-1$ 个节点就会分担丢失副本的负载。这种方法优点在于简单易行（并可同时提高吞吐量和容错能力）。缺点在于如果正在运行的系统已接近理论吞吐量，并且 r 值很小时，剩下的 $r-1$ 个副本将会超载。
- **部分复制**（Partial Replication）：不是将整个索引复制 r 份，而是只复制那些重要文档的索引信息。这个策略背后的意义是大多数搜索结果并不重要，很容易使用等价的相关文档取代它们。这种方法的缺点是很难预测哪些文档是导航式查询的目标。如 PageRank（15.3.1 节）这样的查询独立的信号还是能提供一些指引的。
- **休眠复制**（Dormant Replication）：假设搜索引擎由 n 个节点组成。我们将每个节点 v_i 上的索引都分为 $n-1$ 个片段并平均分发到其他的 $n-1$ 个节点中，但不使用它们来处理查询，而是让它们（在磁盘上）休眠。只有当节点 v_i 失效时，相应的 $n-1$ 个片段才被激活并被载入内存以处理查询。将片段载入到内存中是非常重要的，否则每次查询引起的磁盘寻道总数将加倍。由于每个节点的副本都存储到另外 $n-1$ 个节点上，休眠的副本将使存储开销增加一倍。

联合上面的策略也是有可能的——例如休眠部分索引的副本。我们不是复制给定节点上

的所有索引，而是只为该节点上的重要文档复制副本。这就既减少了存储开销，又减少了因节点失效而对搜索引擎的吞吐量造成的影响。

14.2 MapReduce

除了处理搜索查询，还有很多数据密集型任务需要用到大规模搜索引擎。这些任务包括：建立并更新索引，找出语料库中的重复文档，分析文档集的链接结构（如 PageRank，参见 15.3.1 节）。

MapReduce 是由 Google 开发的一个框架，为大量数据（多个 TB 级数据）的高度并行计算（上千台机器）而设计，可完成以上各种任务。MapReduce 最初由 Dean 和 Ghemawat (2004) 提出。他们的论文除了对 MapReduce 的框架提出了高度的概括，还包含了很多有趣的实施细节和性能优化的信息。

14.2.1 基本框架

MapReduce 的灵感来自于编程语言（如 Lisp）中的 map 和 reduce 函数。map 函数以一个函数 f 和一个元素列表 $l=\langle l_1, l_2, \cdots, l_n\rangle$ 作为输入参数，它返回一个新的列表：

$$map(f,l) = \langle f(l_1),\ f(l_2),\ \cdots,\ f(l_n)\rangle \tag{14-7}$$

reduce 函数（也称为**折叠**（fold）或**累积**（accumulate)），以一个函数 g 和一个元素列表 $l=\langle l_1, l_2, \cdots, l_n\rangle$ 作为输入参数，并返回一个新的元素 l'：

$$l'=reduce(g,l)=g(l_1,\ g(l_2,\ g(l_3,\ \cdots))) \tag{14-8}$$

当讨论到 MapReduce 中的 map 函数时，通常是指传入 map（map 由框架提供）的函数 f。同样，当提到 reduce 函数时，通常是指传入 reduce 的函数 g。下面我们将会遵照这个约定。

概括来说，一个 MapReduce 程序（通常简称 MapReduce）的流程就是先读入一个〈键/值〉对序列，然后再执行一些运算，输出另一个〈键/值〉对序列。关键字和特征值通常是字符串，但实际上可以是任何类型。MapReduce 由三个步骤组成：

- **map 阶段**，读入〈键/值〉对，在两个字段上分别运用 map 函数。函数的一般形式如下：

$$map:(k,v)\mapsto\langle(k_1,v_1),(k_2,v_2),\ \cdots\rangle \tag{14-9}$$

也就是对于每个〈键/值〉对，map 输出一个〈键/值〉对序列。这个序列有可能为空，也有可能不为空，并且输出的关键字也不一定就和输入的关键字不一样（虽然通常是不一样的）。

- **shuffle 阶段**，根据关键字对 map 阶段输出的每个〈键/值〉对进行排序，并且将关键字相同的特征值都聚集在一起。
- **reduce 阶段**，在每一个关键字和它对应的特征值上运用 reduce 函数。函数的一般形式如下：

$$reduce:(k,\ \langle v_1,v_2,\cdots\rangle)\mapsto(k,\ \langle v_1',v_2',\cdots\rangle) \tag{14-10}$$

也就是，对每一个关键字，reduce 函数处理对应的特征值列表并输出另一个特征值列表。输出特征值和输入特征值有可能相同，也有可能不同。但输出关键字通常和输入关键字必须是一样的，这依赖于具体实现。

图 14-5 给出了一个统计给定的文本语料库中所有词项出现次数的 MapReduce 的 map 和 reduce 函数。在 reduce 函数中，如果输出关键字和输入关键字一样，则可以忽略。

```
map (k, v) =                          reduce (k, [v1, v2, ...vn]) =
1   split v into tokens               4   count ← 0
2   for each token t do               5   for i ← 1 to n do
3     output(t, 1)                    6     count ← count + vi
4   return                            7   output(count)
                                      8   return
```

图 14-5 对于给定的文本语料库，MapReduce统计每个词项出现的次数。map函数处理的输入值是文档或其他文本片段，这里忽略了输入关键字。MapReduce的输出是一个 (t, f_t) 元组的序列，其中，t 是词项，f_t 是 t 在输入中出现的次数

MapReduce是可以高度并行化的，因为map和reduce都可以在很多不同的机器上并行执行。假设共有 $n=m+r$ 台机器，其中 m 为map工作机群（map worker）的数量，r 为reduce工作机群（reduce worker）的数量。MapReduce的输入被划分成多个称为map碎片（map shard）的小块，每个碎片通常为16 MB~64 MB。每个碎片都被分派给 m 个map工作机群中的一个来单独处理。在大型的MapReduce中，每个map工作机通常需要处理数十乃至上百个碎片。一个工作机通常一次只处理一个碎片，所以同个工作机中的碎片只能被顺序处理。但是，如果工作机是多核的，就可以并行处理多个碎片以提高性能。

在相似的模式下，输出也被划分成独立的reduce碎片（reduce shard），其中reduce碎片数量通常和reduce工作机群数量 r 相同。每个由map函数产生的〈键/值〉对都会传给一个reduce碎片，通常是根据关键字来将给定的〈键/值〉对分配给对应的碎片。例如，如果我们有 r 个reduce碎片，每个〈键/值〉对分配的目标碎片可以通过下面的式子来选择

$$shard(key, value) = hash(key) \bmod r \tag{14-11}$$

其中hash函数可以随意选择。以这种方式将map的输出分配给reduce碎片可以保证同一个关键字的所有特征值都在同一个reduce碎片中。在每个reduce碎片内部，根据关键字对输入的〈键/值〉对进行排序（在shuffle阶段中），并把排序结果传给reduce函数，产生MapReduce的最终结果。

图14-6中展示了图14-5中相应的MapReduce的数据流。数据源是摘自莎士比亚文集的三个小文本片段。每个片段代表一个独立的map碎片。把map工作机群输出的〈键/值〉对，根据关键字的hash值划分到三个reduce碎片中。在该例中，假设hash（"heart"）mod 3=0，hash（"soul"）mod 3=1，依次类推。

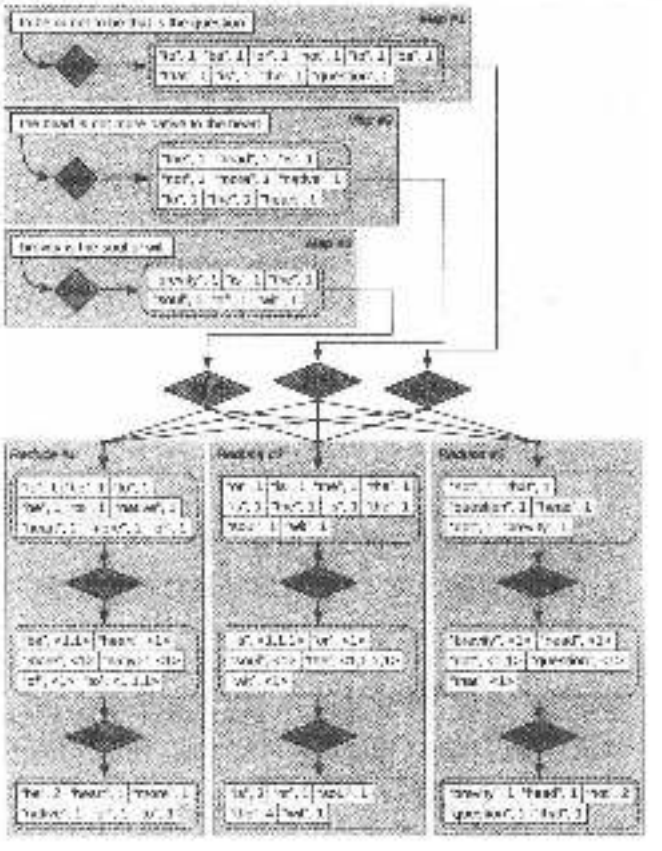


图 14-6 对应图14-5中的MapReduce的数据流，使用3个map碎片和3个reduce碎片

map 阶段的工作可能和 shuffle 阶段重叠，shuffle 阶段的工作也可能和 reduce 阶段重叠。但是 map 阶段和 reduce 阶段的工作是不可能重叠的。因为 redcue 函数只有在给定关键字的所有特征值都已获得的情况下，才会被调用。由于一般不可以预测 map 工作机群传输的关键字，所以 reduce 阶段就不可能在 map 阶段完成前开始。

14.2.2　合并

在很多 MapReduce 任务中，对应同一个关键字的一个 map 碎片可能产生大量的〈键/值〉对。例如，如果我们使用图 14-5 中的 MapReduce 程序来处理一个典型的英语文本语料，那么 6%～7%的关键字将是“the”。转发所有类似“the”这样的元组给 reduce 工作机群，将造成网络资源和存储资源的浪费。更重要的是，它造成了整个负载分布的极不平衡。不管有多少 reduce 工作机群，它们中的某个将需要承载总体 reduce 工作的 7%。

为了解决上述问题，我们修改 map 工作机群，让它们可以累加在本地 hash 表中每个碎片的词项数量；并在结束当前碎片时以（t，f_t）的格式输出取代一共需要输出 f_t 次的（t，1）。这种方法不足之处在于需要额外实现这部分程序。

另一种方法是使用 hash 表，转发数据给 reduce 工作机群前，可以为每一个 map 碎片执行一个本地的 shuffle/reduce 阶段。这种方法相当于在本地 hash 表里统计本地计数，但由于不需要修改任何程序而得到开发者的喜爱。一个使用在 map 碎片上而不是 reduce 碎片上的 reduce 函数被称为一个**合并器**（combiner）。每个 reduce 函数只要保证它的输入值和输出值是同一类型，即可用在它自己的输出上，就可作为一个合并器。在统计词项的 MapReduce 例子中，刚好满足上述的要求，所有输入和输出的值在 reduce 阶段都是整数类型。

14.2.3　辅助关键字

迄今为止描述的 MapReduce 基本框架，都不能保证传入到 reduce 函数的同一个关键字的值保持一个相对的序。通常这并不要紧，因为 reduce 函数可以通过它希望的任何方式来检查和重排特征值。但是，对于特定的任务，一个给定的关键字的特征值的数量可能非常大，以至于它们不能被同时载入到内存中，这样在 reduce 函数中实现重排序就很困难了。在这种情况下，在把关键字传给 reduce 函数前，MapReduce 框架对每个关键字的特征值按某种方式排好序会比较好。

这种任务的一个例子是索引构造，特征值在传入 reduce 函数前必须按某种预定义的序排好。为了为给定文本集合创建文档编号索引，我们可以定义这样的 map 函数，对每个出现在文档 d 中的词项 t，输出一个〈键/值〉对$(t, docid(d))$。reduce 函数则通过合并 t 的所有位置信息来构造它的位置信息列表（如果可以，则对它们压缩）。显然，能这样处理的前提是 reduce 的输入按 $docid$（d）升序排列。

MapReduce 支持**辅助关键字**（secondary key）的概念，即用于定义传入 reduce 函数的同一个关键字的特征值的次序。在 shuffle 阶段，〈键/值〉对一般按它们的关键字排序。然而，当给定关键字对应的特征值不止一个的时候，同一个关键字的特征值就可以根据它们的辅助关键字排序。在 MapReduce 的索引构造过程中，〈键/值〉对的辅助关键字就是特征值（即包含给定词项的文档编号）。

14.2.4　机器失效

当在成百上千台计算机上运行 MapReduce 程序时，一些机器可能会偶尔出现问题而不

得不关闭。MapReduce 框架假设 map 函数的行为是严格确定的，并且 map 函数对给定 map 碎片的输出只取决于该碎片（即同一个 map 工作机处理的两个碎片上不存在信息交换）。如果这个假设成立，一个 map 工作机失效了，那就把它的碎片分配给另一个机器重新处理。

处理 reduce 工作机失效就有一点复杂了。因为每个 reduce 碎片上的数据依赖于每个 map 碎片上的数据，如果把 reduce 碎片分配给另一个工作机，就需要重新处理所有的 map 碎片。为了避免这个问题，map 阶段的输出一般不直接传给 reduce 工作机群，而是临时存放在一个可靠的存储层上，例如专用的存储服务器或是 Google 文件系统（Ghemawat 等人，2003）。如果一个 reduce 工作机失效了，新的 reduce 工作机就从这个存储层上重新读入数据。但是，就算 map 的输出以可靠方式存储，当一个 reduce 工作机失效时，还是需要为失效碎片重新执行 shuffle 阶段。如果我们也想避免这个问题，那么 reduce 工作机就要在进入 reduce 阶段前将 shuffle 阶段的输出传送到存储服务器上。因此，对于给定碎片，shuffle 阶段和 reduce 阶段在同一机器上运行，除非机器机器故障频繁发生，否则额外的网络传输开销可以忽略。

14.3 延伸阅读

相比该书中的其他课题，并行信息检索的著作还是比较少的。已有的出版刊物大都局限于由数十乃至更少机器组成的中小型计算集群，只有一些主要的搜索引擎公司才会不定期地出版这方面的刊物。所以，在某些情形下，我们很难来对一个新架构的可扩展性进行合理的评估。例如，尽管在 14.1.2 节介绍的基本形式的词项划分在 8 节点的集群上可以很好的工作，但是它肯定无法扩展到拥有成百上千个节点的集群。尽管如此，一些从小规模实验中获得的结果可能会适用于大规模并行搜索引擎。

Moffat 等人（2006）对词项划分查询造成的负载均衡问题进行研究后，得到结论：如果在适当的地方使用负载均衡策略，词项划分的查询性能可以与文档划分一样好。Marín 和 Gil-Costa（2007）也进行了相似的研究，并指出：词项划分索引的性能有时能优于文档划分。Abusukhon 等人（2008）比较了词项划分的变种（带长位置信息列表的词项分布在多个索引节点中）。

Puppin 等人（2006）讨论了在不随机分配文档而是基于查询的排名把文档分配给节点的文档划分方案（通过已知的查询日志）。对同一查询集中排名高的文档通常会被分配到同一节点。而每个到来的查询都通常转发给那些可能返回好的查询结果的索引节点。另外，Xi 等人（2002）以及 Marín 和 Gil-Gosta（2007）给出了关于混合划分方案（联合词项划分和文档划分）的实验结果。在并行查询处理上，他们得到的结果与 Marín 和 Navarro（2003）的结果有些许不同（基于后缀数组而非倒排文件的分布式查询处理）。

Barroso 等人（2003）对 Google 的分布式查询处理作出了概述。其他关于 Google 的大规模数据处理的介绍由下述学者提出：Ghemawat 等人（2003）、Dean 和 Ghemawat（2004，2008）以及 Chang 等人（2008）。

Hadoop[⊖]是一套开源的并行计算架构，这个架构开发的灵感来源于 Google 的 MapReduce 和 GFS 技术。在其他组件中，Hadoop 还包含了 HDFS（一种分布式的文件系统）和 MapReduce 的执行方案。Hadoop 计划是由 Doug Cutting（Lucene 搜索引擎的创建者）提出的。Yahoo 是这个计划中的一个主要贡献者，并且 Yahoo 也有望成为拥有世界上最大的 Hadoop 集群的公司（包含几千台机器）。

⊖ hadoop.apache.org

近年在处理通用、非图像相关计算时使用的图形处理器（graphics processing units，GPUs）已经引起了一些关注。由于它们高度并行的性质，在长序列数据需要被顺序或倒序处理的情况下，如排序（Govindaraju 等人，2006；Sintorn 和 Assarsson，2008）与析取（即布尔 OR）查询处理（Ding 等人，2009），GPUs 的性能远远优于普通的 CPU。

14.4　练习

练习 14.1　当复制一个分布式的搜索引擎时，复制可以在节点级别（即一个具有 $2n$ 个索引节点的集群，两个节点将共享给定索引碎片的查询负载）或集群级别（即两个相同集群，但彼此间没有共同的副本）。讨论它们各自的优缺点。

练习 14.2　描述文档-划分索引可能出现的扩展性问题（就算索引存储在内存中）。（提示：查询处理操作的复杂度与索引大小是次线性关系。）

练习 14.3　给定一个节点个数 $n=200$ 的文档-划分索引，目标结果集大小 $m=50$，为了以 99% 的概率得到正确的前 m 个结果，每个节点返回的结果集大小 k 最小是多少？

练习 14.4　一般化 14.1.4 节中的休眠复制策略，使之可以处理 k 个机器同时失效的情况。这时的总体存储需求是多少？

练习 14.5　请描述面向词项划分索引和面向文档划分索引的休眠复制的不同。

练习 14.6　(a) 为给定的文本语料库设计一个计算平均文档长度（每个文档的词条数）的 MapReduce 程序（即一个 map 函数与一个 reduce 函数）。(b) 修改你的 reduce 函数使它可以作为一个合并器。

练习 14.7　设计一个 MapRduce 程序，计算一个包含词项 t_2 的文档也包含词项 t_1 的条件概率 $\Pr[t_1 \mid t_2]$。你会发现使用了辅助关键字的 map 函数很有用，它会使同一个关键字的所有特征值都按某种顺序进行排序。

练习 14.8（项目练习）　采用练习 5.9 中实现的 BM25 评分函数，来构造一个文档-划分搜索引擎。分别为 100%、50%、25% 和 12.5% 规模的 GOV2 文档集构建索引。对每种索引规模，计算每个查询的平均时间（用一些标准的查询集）。你观察到什么？这是否影响文档划分的可扩展性？

14.5　参考文献

Abusukhon, A., Talib, M., and Oakes, M.P. (2008). An investigation into improving the load balance for term-based partitioning. In *Proceedings of the 2nd International United Information Systems Conference*, pages 380–392. Klagenfurt, Austria.

Barroso, L.A., Dean, J., and Hölzle, U. (2003). Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28.

Carbonell, J.G., and Goldstein, J. (1998). The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 335–336. Melbourne, Australia.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R.E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, 26(2):1–26.

Clarke, C.L.A., and Terra, E.L. (2004). Approximating the top-m passages in a parallel question answering system. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 454–462. Washington, D.C.

Dean, J., and Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation*, pages 137–150. San Francisco, California.

Dean, J., and Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Ding, S., He, J., Yan, H., and Suel, T. (2009). Using graphics processors for high performance

IR query processing. In *Proceedings of the 18th International Conference on World Wide Web*, pages 421–430. Madrid, Spain.

Ghemawat, S., Gobioff, H., and Leung, S. T. (2003). The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 29–43. Bolton Landing, New York.

Govindaraju, N., Gray, J., Kumar, R., and Manocha, D. (2006). GPUTeraSort: High performance graphics co-processor sorting for large database management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 325–336. Chicago, Illinois.

Marín, M., and Gil-Costa, V. (2007). High-performance distributed inverted files. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management*, pages 935–938. Lisbon, Portugal.

Marín, M., and Navarro, G. (2003). Distributed query processing using suffix arrays. In *Proceedings of the 10th International Symposium on String Processing and Information Retrieval*, pages 311–325. Manaus, Brazil.

Moffat, A., Webber, W., and Zobel, J. (2006). Load balancing for term-distributed parallel retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 348–355. Seattle, Washington.

Moffat, A., Webber, W., Zobel, J., and Baeza-Yates, R. (2007). A pipelined architecture for distributed text query evaluation. *Information Retrieval*, 10(3):205–231.

Puppin, D., Silvestri, F., and Laforenza, D. (2006). Query-driven document partitioning and collection selection. In *Proceedings of the 1st International Conference on Scalable Information Systems*. Hong Kong, China.

Sintorn, E., and Assarsson, U. (2008). Fast parallel GPU-sorting using a hybrid algorithm. *Journal of Parallel and Distributed Computing*, 68(10):1381–1388.

Xi, W., Sornil, O., Luo, M., and Fox, E. A. (2002). Hybrid partition inverted files: Experimental validation. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries*, pages 422–431. Rome, Italy.

第15章

Information Retrieval: Implementing and Evaluating Search Engines

Web搜索

前述章节摒弃特殊的例子和实验，从一般化的角度介绍了信息检索。假设信息检索系统包含一个文档集，其中每个文档都由一系列的词条组成。标记可以是标题、作者和其他结构化元素。我们假设这些仅仅与信息检索系统的运行环境或其包含的文档有关。

本章中我们将考虑Web检索这一特定语境，这也许对你来说非常熟悉。假设这个特定语境能给我们提供一般语境所不能提供的文档特征。最重要的特征之一就是超链接提供的结构信息。这些页面之间的链接通常是图片或锚文本，它们提供给我们大量有价值的信息，包括每个页面的信息和它们之间的关系。

伴随这些特性而来的也有一些问题，主要与网页或网站的“质量”、“权威性”和“受欢迎程度”密切相关，涉及范围从仔细编辑的国际新闻网站的网页到高中生的个人主页。另外，很多网页实际上是**垃圾**（spam）网页——通过伪装以达到商业诈骗或其他目的的恶意网页。尽管大部分网站的主人都希望在主要的商用搜索引擎中有比较靠前的排名，而且会不惜一切代价来提高它们的排名，但是垃圾网页的创建者与商用搜索引擎的运营商的关系是对立的。这些网页的创建者会积极尝试修改用于排名的特征，修改的方法主要是通过在网页的内容和质量上制造一些假象。

其他问题则是因为Web的规模——数十亿的网页分散在数百万台主机上。为了对这些网页进行索引，**爬虫**（crawler）将它们从Web上抓取下来并放在本地由搜索引擎处理。由于多数网页每天或每小时都在变，因此Web的快照需要定时刷新。爬虫在抓取网页的时候同时也会检测重复或近似重复的页面，然后进行适当的处理。例如，在很多网站中都能找到用Java编程语言写的标准文件，但是响应例如〈“java”，“vector”，“class”〉这样的查询时，搜索引擎最好是只返回java官方网站java.sun.com。

另外需要考虑的问题是商业Web搜索引擎收到的查询的数量和种类，这直接反映了Web上信息的数量和种类。由于查询通常比较简短——一两个词的长度——这样搜索引擎就对用户输入的查询或用户要找的内容知之甚少。当用户输入查询〈“UPS”〉时，他有可能是要查快递包裹，也有可能是想买通用电源，或是参加普吉特湾大学（University of Puget Sound）的夜班学习。尽管这样的查询二义性对于所有的信息检索应用来说都是需要考虑的问题，但是这种现象在Web信息检索应用中尤为突出。

15.1 Web的结构

图15-1给出一个例子说明了Web结构最重要的特征。它展示了三个网站上的网页：分别为W、M和H。[⊖]站点W提供了一部普通的百科全书，包括有关莎士比亚（w0.html）以及他的两部剧作：《Hamlet》（w1.html）和《Macbeth》（w2.html）。站点H提供历史信息，包括莎士比亚的妻子（h0.html）和他的儿子（h1.html）。站点M提供影视信息，包括1971年由Roman Polanski导演的电影版《Macbeth》的网页（m0.html）。

⊖ 为简单起见，我们用单个字母的网站名和简化的页面名来代替它们全称。如用http://W/w0.html代替完整的URL：http://en.wikipedia.org/wiki/William_Shakespeare。

图中解释了这些站点和页面之间的链接结构。如网页 w0.html 的 HTML 锚记（anchor）

 Anne Hathaway

在页面 h0.html 和站点 H 之间建立起链接。与这个链接（"Anne Hathaway"）相关的锚文本则指示了目标页面的内容。

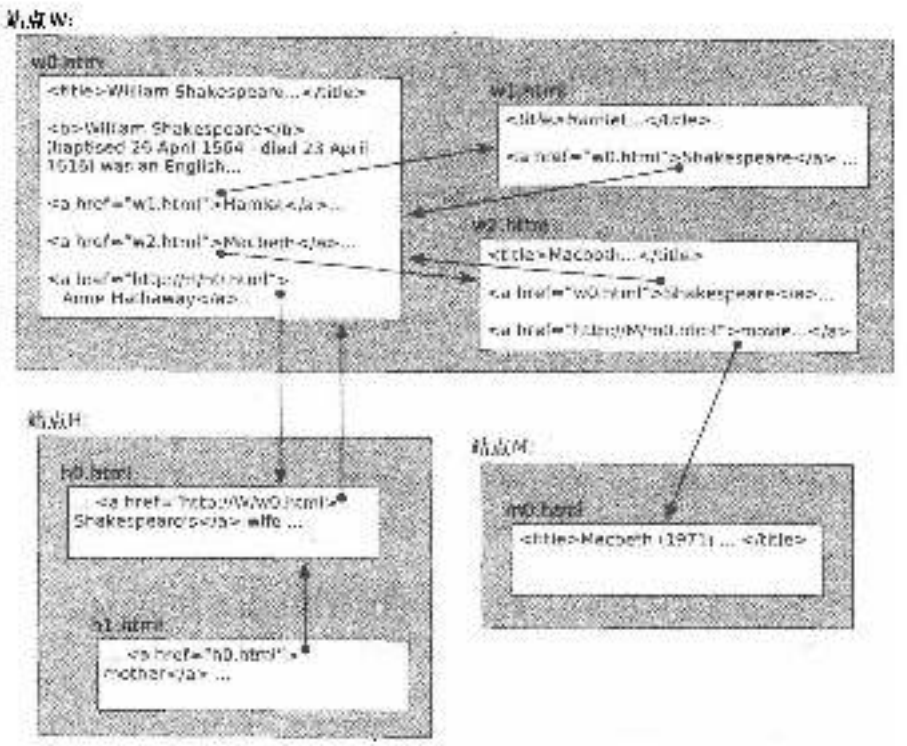


图 15-1 Web 结构。网页可相互链接（<a href=…>），并且可通过锚文本（例如："mother"）与各个链接关联起来

15.1.1 Web 图

这些用超链接表示的网页与站点之间的联系构成了所谓的 Web 图（Web graph）。如果将 Web 图看成一个纯粹的数学对象，每个网页就是图中的节点，每个超链接就是链接这些节点的有向边。图 15-2 所示的 Web 图对应于图 15-1。简单起见，我们简化网页的标记，如将 http: //W/w0.html 简写为 w_0。

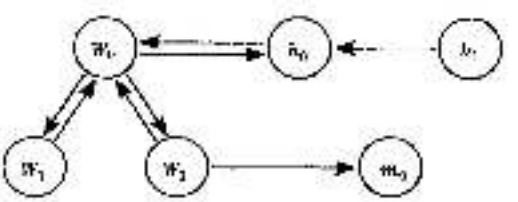


图 15-2 对应图 15-1 中的网页的 Web 图。每个链接对应图中的一条有向边

从实际角度考虑，谨记以下原则十分重要：当提到 Web 图的时候，并不单单是指数学方面的抽象概念——网页在站点上聚集，锚文本和图片也有可能与每个链接相关；链接可能指向页面内部某个标记的位置，也有可能指向整个页面。同样应该注意到 Web 具有高动态和流动的性质。全世界的网站都在不断地添加或删除页面和链接，因此只能捕捉整个 Web 图的一个粗略的近似。

有时我们仅对 Web 中的部分内容感兴趣，也许只是某一个网站或组织中的所有页面。

在这种情况下，捕捉这个站点或组织的 Web 图的精确快照比捕捉整个 Web 的精确快照要容易得多。但是，大多数网站都会持续地增长和变化，因此得定期重新捕捉抓取快照以维持快照的精确。

为后面章节用到的 Web 图给出一个正式的定义。设 Φ 为 Web 图中所有网页的集合，$N=|\Phi|$ 为网页数量，E 表示图中链接（或边）的数量。给定一个页面 $\alpha\in\Phi$，定义 out(α) 为从 α 页面指向其他页面的**出链接**（out-link）的数量（也称**出度**（out-degree））。同样地，定义 in(α) 为从其他网页指向页面 α 的**入链接**（in-link）的数量（也称**入度**（in-degree））。在图 15-2 所示的 Web 图中，out(w_0)=3，in(h_0)=2。如果 in(α)=0，则 α 称为**源**（source）；如果 out(α)=0，则 α 称为**汇点**（sink），定义 Γ 为汇点集。在图 15-2 所示的 Web 图中，页面 m_0 为汇点，页面 h_1 为源。

每个单独网页本身就有可能是高度复杂和结构化的对象。它们通常包含菜单、图片和广告。脚本用于第一次加载页面时生成网页内容，以及与用户交互时更新网页内容。一个网页也可能作为框架保存其他页面，或只是将浏览器重定向到另一个页面。这些重定向可以通过脚本或特殊的 HTTP 响应来实现，并且可能会被多次重复调用，因此大大增加了其复杂性。除了 HTML 页面，Web 中还包含 PDF、Word 文档以及其他格式的文档。

15.1.2 静态与动态网页

现在常常会听到页面有“静态”的和“动态”的。静态网页的 HTML 都是在用户请求之前已经生成好并存储在磁盘上，需要时就被传送给浏览器或 Web 爬虫。一个组织的主页就是一个典型的静态网页。动态网页是在请求出现后才生成的，页面内容部分是由请求的具体内容来决定的。一个搜索引擎的结果页面（search engine result page，SERP）就是一个典型的动态网页，由用户的查询决定页面的内容。

通常会认为静态页面比动态页面更容易抓取和索引，确实也有很多类型的动态页面不适合抓取和索引。例如，在线日历系统可供人和事件管理约会和时间表，根据日期给出动态生成的网页（对于过去和未来）。如果按照链接一路访问下去，可以访问任何一个日期的页面。虽然一个在线日历可以很灵活地制定从今往后 25 年的预约，但对于一个 Web 搜索引擎来说索引无穷多的空白月份也是不合适的。

虽然很多动态网页不应进行抓取和索引，但是还是有很多这样的网页是应该这样做的。例如零售网站的目录页，这通常是访问关系数据库当前商品和价格时动态产生的。随后生成的结果转化成 HTML 格式，并通过菜单和其他固定信息对其进行包装，最后呈现在浏览器中。为满足用户搜索产品的需求，搜索引擎必须对这些网页进行抓取并索引。

动态网页有时可以根据其 URL 的特征对其进行识别。例如，通过通用网关接口（CGI）访问的服务器可能包含 URL 的路径元素 cgi-bin。由微软的动态服务器页面（ASP）技术产生的动态网页的 URL 中带有扩展名.asp 或.aspx。遗憾的是，这些 URL 的特征并不总是显示出来的。理论上讲，所有网页都可能是静态的或是动态的，并且没有确定的方式来对其进行分辨。对于网页抓取和索引而言，重要的是网页的内容，而不是它们是静态还是动态的。

15.1.3 暗网

很多网页是所谓的“暗网”、“潜网”或“深网”的一部分。暗网（Hidden Web）包含很多没有链接指向的网页，受密码保护的网页以及只有查询数字图书馆或数据库时才看得到的网页。虽然这些网页包含有价值的内容，但是它们很难或不可能被 Web 爬虫找到。

内网（intranet）中的网页是一种特殊的暗网。内网中的网页只有该公司内部或相似实体内部才可以访问。用于索引内网的企业搜索引擎使用很多与普通 Web 搜索引擎一样的技术，但根据内部网的一些独有特征进行调整。

15.1.4 Web 的规模

即使排除所有暗网，也很难对 Web 的规模进行准确估计。添加或移除一个主机都在不同程度上改变了可访问页面的数量，改变的大小取决于主机内容的多少。有的主机包含数以百万的有价值网页；而有的主机包含数以百万没用或用处不大的内容的网页（见练习 15.14）。

然而，很多网页很少出现在搜索结果的前面。排除那些来自于搜索引擎但没什么用的网页。我们可以非正式地将那些值得被通用 Web 搜索引擎包括的网页称为**可索引 Web**（indexable Web）（Gulli 和 Signorini，2005）。这个可索引 Web 将会包含全部可对搜索结果产生实质性影响的网页。

如果假定包含在主流搜索引擎索引中的网页构成了可索引 Web 的一部分，那么可索引 Web 规模的下限就可由主流搜索引擎的联合覆盖率来决定。具体地说，如果集合 A_1，A_2，…表示每个主流搜索引擎索引的网页集，可索引 Web 规模的下限就是这些集合的并集$|UA_i|$。

遗憾的是，也很难精确计算这个集合。我们虽然可以检测给定的 URL 是否出现在索引中，但主流搜索引擎并不公布它们索引网页的列表，甚至不提供包含网页的数量。即使我们知道了每个搜索引擎包含的网页数量，这个并集的规模也小于所有网页的数量和，因为很多网页是重复的。

Bharat 和 Broder（1998）以及 Lawrence 和 Giles（1998）提出了一种估计主流搜索引擎联合覆盖率的方法。首先，产生一个 URL 的测试集。这个步骤可以通过对主流搜索引擎进行一系列的随机查询，然后从每一个返回结果集中随机选取一些 URL 来完成。我们假设这个测试集代表了搜索引擎索引页面的一个均匀抽样。其次，测试集中的每个 URL 再次经过每个搜索引擎的验证，然后记录包含它的搜索引擎。

给定两个搜索引擎 A 和 B，图 15-3 解释了它们的集合之间的关系。对我们的 URL 测试集进行采样，可以估计 $\Pr[A \cap B \mid A]$，即如果某个 URL 在 A 中，那么它同时也在交集中的概率：

$$\Pr[A \cap B \mid A] = \frac{\text{同时包含 } A \text{ 和 } B \text{ 的测试 URL 数}}{\text{包含 } A \text{ 的测试 URL 数}} \tag{15-1}$$

我们可用同样的方式估计 $\Pr[A \cap B \mid B]$。如果我们知道 A 的大小，则可以估计交集的大小为：

$$|A \cap B| = |A| \cdot \Pr[A \cap B \mid A] \tag{15-2}$$

以及 B 的大小为

$$|B| = \frac{|A \cap B|}{\Pr[A \cap B \mid B]} \tag{15-3}$$

因此，可以估计并集的大小为

$$|A \cup B| = |A| + |B| - |A \cap B| \tag{15-4}$$

如果 A 与 B 的大小已知，交集的大小就可以根据这两个集合的大小估计得到，并且由这些估计值的平均值可估计并集的大小：

$$|A \cup B| = |A| + |B| - \frac{1}{2}\left(|A| \cdot \Pr[A \cap B \mid A] - |B| \cdot \Pr[A \cap B \mid B]\right) \qquad (15\text{-}5)$$

这种方法可扩展到多引擎的情况。采用这种方法的变形，Bharat 和 Broder（1998）估计了 1997 年中期可索引 Web 的规模大概是 2.8 亿个网页。在大约同时期的研究中，Lawrence 和 Giles（1998）估计可索引 Web 规模大约是 3.2 亿个网页。大约一年以后，可索引 Web 规模就已经扩大到了 8 亿个网页（Lawrence 和 Giles，1999）。到 2005 年中期，可索引 Web 规模达到了 11.5 亿个网页（Gulli 和 Signorini，2005）。

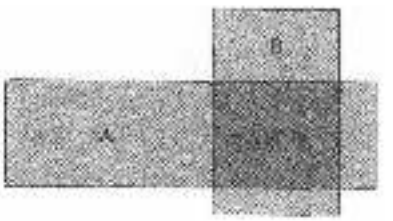


图 15-3 搜索引擎 A、B 间的重叠交集可用于估计可索引 Web 的规模

15.2 查询与用户

Web 查询通常非常短。Spink 和 Jansen（2004）对主流搜索引擎中查询日志的研究进行了总结。尽管各个研究得到的确切数据各不相同，但是它们一致显示出多数查询的长度通常只有 1 或 2 个词项，平均长度则在 2～3 个词项之间。这些查询的主题覆盖了人们的各种兴趣，如性、健康、商业以及某些主题娱乐。

或许 Web 查询如此简短并不奇怪，真实最近，Web 搜索引擎的查询处理策略仍然不鼓励长查询。正如 2.3 节中所述，这些处理策略通过对排名前列查询词项进行布尔合取来过滤结果。对于搜索结果中包含的某个网页，所有的查询词项必须以某种方式与之相关联，或是出现在页面内容中，或已现在指向该页面的锚文本中。因此，通过添加相关词项来增加查询长度可能会导致：只用户漏掉几个相关词，在过滤时就会滤掉一些相关网页。这种严格的过滤机制在近年来已经放宽很多了。例如，可以接受用同义词代替精确匹配了——可以用词项“howto”替代“FAQ”。尽管如此，在查询处理过程中过滤效率仍是一个问题，而且 Web 搜索引擎对长查询的处理效果不好。

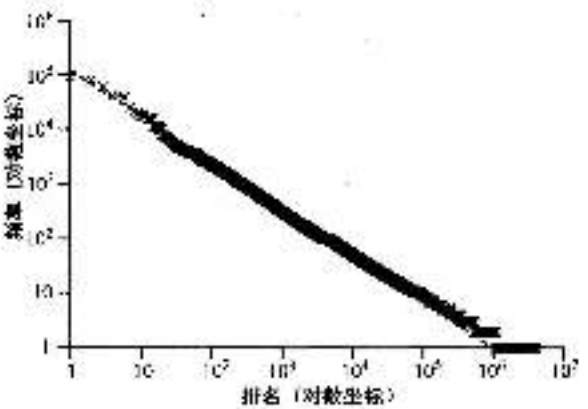


图 15-4 根据排名排序后的查询频率，数据来自一个商用搜索引擎的日志中的 1000 万条查询记录。图中虚线部分与 $\alpha = -0.85$ 时的齐夫定律相符

Web 查询的分布遵循齐夫定律（Zipf's Law）（见图 15-4）。在一个具有代表性的查询日志（包含 1000 万条查询记录）中，单个的最频繁查询占了总数的 1% 以上，而接近一半的查询都只出现过一次。在对搜索引擎的调试与评价中，都需要考虑齐夫定律的“长尾效应”。总而言之，不频繁查询和频繁查询都很重要。

15.2.1 用户意图

一些研究人员研究 Web 搜索引擎中出现的查询集合，力图从这些查询中挖掘出隐藏其中的用户意图，并提取其特征。Broder（2002）对使用 Altavista 搜索引擎的用户及查询日

志进行了研究，从而得出一种 Web 搜索引擎分类方法。他将 Web 查询分为三类，反映了用户的明显意图，如下：

- 进行**导航**（navigational）查询的意图是定位 Web 上的特定网页或站点。例如，如果一个用户想找到 CNN 新闻网的主页，他将会输入查询〈"CNN"〉。导航查询通常只有一个正确的结果。然而，这个正确的结果对不同用户而言会有所不同。一个美国的讲西班牙语的用户搜索 CNN 时可能希望找到用西班牙语显示的主页（www.cnn.com/espanol）；突尼斯（Tunisia）的用户可能想要阿拉伯版本的网页（arabic.cnn.com）。
- 进行**信息**（informational）查询的用户往往想要获知某个特定主题的相关知识，在假定知识来源可靠的前提下，用户对信息的来源并不感兴趣。这些主题构成了第 1 章中提到的测试集，并且在本书中前四个部分用来进行信息查询意图的评价。当用户输入查询〈"president"，"obama"〉时，她得到的结果可能来自于 CNN 新闻网、维基百科或其他网站。也许在找到所有她需要的信息前，她会浏览并阅读所有来自这些网站的信息。像这样的信息查询，其背后的需求也是因人而异的，范围可能更宽或更窄些。用户可能是要找政府政策的详细描述、个人简介，也可能只是奥巴马的生日。
- 进行**交易**（transactional）查询的用户在找到想要的结果以后可能会与网站进行交互。这种交互动作可能涉及很多活动，如在线游戏、采购物品、旅游预订，或下载图片、音乐、视频等。这类查询还包含那些搜索服务的查询，如地图、天气等，而不是要找提供这些服务的特定网站。

Rose 和 Levinson（2004）对 Broder 的工作进行了扩展，将三个分类细化成用户目标的不同层级。他们在层级的顶端保留了 Broder 的三个分类，但是将"交易查询"重新命名为"源查询"。他们将信息查询和交易查询分成了很多子类。例如，用户进行**直接信息**（directed informational）查询的目标是寻找某个特定问题（如"奥巴马的生日是什么时候?"）的答案，反之，进行**非直接信息**（undirected informational）查询的用户的目的只是想了解某个主题（如"给我介绍一下奥巴马总统。"）的相关信息。进一步地，直接信息查询可分为**开放式**（open）或**封闭式**（closed）两种，取决于问题的答案是开放式的还是特定的。

导航查询和信息查询的区别可以简单概括为：用户是否要搜索特定的网站？交易查询与其他两类查询的区别并不十分明显。我们可以假定查询〈"mapquest"〉是导航查询，用户想要找的网站是 www.mapquest.com，但同时也可认为用户想与该网站进行交互，以获得他想要的方位和地图。用户输入查询〈"travel"，"washington"〉时，可能要查找有关在华盛顿旅游的信息，同时也想在那里预订酒店，这就使得这一查询既是信息查询又是交易查询。如这些例子一样，有些查询可以看成是多类查询的结合，如导航/交易查询、信息/交易查询等。

根据 Broder（2002）的研究，导航查询占 Web 查询总数的 20%～25%，交易查询至少占 22%，剩余的就是信息查询。根据 Rose 和 Levinson（2004）的研究，12%～15% 的查询是导航查询，24%～27% 的查询是交易查询。最近 Jansen 等人（2007）的研究结果与这些数据稍有不同。他们的研究表明 80% 以上的查询是信息查询，剩下的由导航查询和交易查询平分。尽管如此，这些研究都表明这几类查询在 Web 查询中占有很大比例，因此 Web 搜索引擎必须准确识别用户意图中的不同点。

"导航查询"和"信息查询"是常见的术语。如果查询潜在的目的是一样的或者相似的，那么无论是哪个用户提交的查询，这种术语的用法都是可以理解的。但对于某些查询来说，

它们的类别因用户不同有很大的差异。因此我们强调这些查询类别只是基本描述了提交查询的用户的目的和意图，而不是查询本身内在的意义。例如，一个用户进行查询("UPS")，那么他可能想进行：

- 信息意图，想知道通用电源（universal power supply，UPS）的供电原理。
- 交易意图，想为其个人计算机购买便宜的通用电源。
- 交易/导航意图，想跟踪快递包裹。
- 导航/信息意图，想要了解普吉特海大学（University of Puget Sound）的课程信息。

尽管这个查询是非典型的，可归入多个查询类别，但（由除用户以外的人员）将任意给定查询分入一个类别，都不是一个有据推测。

15.2.2 点击曲线

导航查询和信息查询之间的区别可从用户行为中明显看出。Lee 等人（2005）研究了用户点击（clickthrough），并以此为推断用户查询意图的一个特征。点击是指在搜索引擎结果页面中点了某个搜索结果的行为。商用搜索引擎常常记录点击来衡量自己的性能（见 15.5.2 节）。

尽管几乎所有的点击都发生在搜索结果的前 10 位（Joachims 等人，2005；Agichtein 等人，2006b），但是点击模式还是根据查询的不同而不同。通过对提交相同查询的大量不同用户的点击进行研究，Lee 等人（2005）得出了信息查询和导航查询的典型点击分布。对导航查询而言，点击分布趋向单个结果；对信息查询而言，点击分布是平缓的。

Clarke 等人（2007）对商用搜索引擎的日志进行了分析，并提供了进一步的分析和样例。图 15-5 的曲线是由他们论文中的样例得到的。两图分别显示了对排名 2～10 的查询结果点击所占的百分比。每个点击表示不同用户第一次点击的搜索结果。图 15-5a 所示的是典型的导航查询的点击分布，显示了在网站 www.craigslist.org 上的高点击率，这个网站是一个经典的广告网站，也是设想的目标查询网站。图 15-5b 所示的是典型的信息查询的点击分布。对于这两种查询，点击数随着排名的增加而下降；信息查询中排名靠后的结果点击数相对较高。

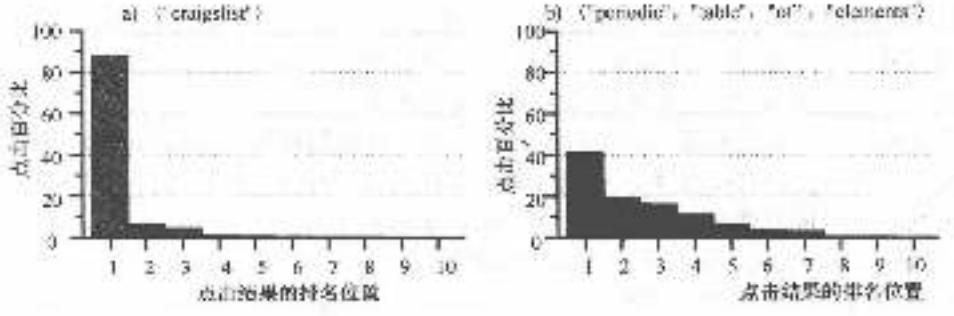


图 15-5 典型的导航查询（("craigslist")）和信息查询（("periodic", "table", "of", "elements")）的点击曲线

15.3 静态排名

Web 检索工作分两个阶段进行。第一阶段发生在索引过程中，对页面进行静态排名（static rank）（Richardson 等人，2006）。这种排名可以非正式地反映出网页的质量、权威

性或流行程度。从理论上讲，静态排名对应于网页的相关性先验概率——先验概率越大，网页静态排名越高。

静态排名独立于任何查询。在查询阶段，Web 检索的第二阶段在查询时才发生。在这一阶段中，静态排名将与独立于查询的特征结合起来，例如词项邻近度和词频等，这些特性都将用于产生**动态排名**（dynamic rank）。

索引时进行静态排名使得 Web 搜索引擎可以考虑使用一些在查询时不能考虑的特征。这些特征中最重要的是源自于**链接分析**（link analysis）技术的一些特征。这些技术用于从 Web 图的结构中提取信息。有助于静态排名的其他特征包括提取自网页内容和用户行为的特征（15.3.5 节）。

我们首先在 15.3.1 节介绍 PageRank 的基础知识作为对静态排名的引入，PageRank 是简单的但是最著名的链接分析技术。尽管该节介绍的算法一般被称为 PageRank，但其在 Web 搜索中的实用价值可能有限，因为该算法基于某些朴素假设，而忽略了 Web 结构。15.3.2 节和 15.3.3 节中将介绍并分析这些算法的扩展版本，使之更好地适应更复杂的 Web 结构。15.3.4 节将概要介绍其他链接分析技术，15.3.5 节将简略介绍适用于静态排名的其他特征。15.4 节中将介绍动态排名。

15.3.1 基本 PageRank

PageRank 是在 20 世纪 90 年代中期，由斯坦福大学计算机学院研究生 Larry Page 和 Sergey Brin 共同提出的。这个算法是他们的 Backrub 搜索引擎的核心算法，并在 Google 中迅速成熟起来。

PageRank 算法想象一个用户在 Web 上是随机浏览的。在用户的浏览器中，网络上所有网页都是可访问的。接下来用户就可以：

1）点击当前页上的一个链接。

2）通过键入网址，随机跳转到一个独立网页并对其进行访问。

在任何一步中用户点击一个链接的固定概率设为 δ。那么，进行直接跳转的概率就是 $1-\delta$。δ 的合理取值范围是 0.75～0.90，在研究文献中最经常使用的是 0.85。简单起见，在实验和例子中我们取 $\delta=3/4$，特殊说明除外。

Web 图中的汇点可迫使跳转动作发生：当我们浏览到一个汇点时，我们一定总是选择跳转。假设我们的浏览速度非常快而且可以不停歇地持续很长一段时间，那么页面 α 的 PageRank 值 $r(\alpha)$就表示该页面被用户浏览的相对频率。

概率 δ 有时是指**重启概率**（restart probability）或**阻尼系数**（damping factor），因为它降低了用户点击一个链接的概率。从统计学角度看，与不允许跳转的等价算法相比，采用阻尼系数来允许进行随机跳转可提高 PageRank 算法的稳定性（见 15.3.3 节）。非正式地说，就是 Web 图中小的变化不会对 PageRank 造成很大的影响。

$r(\alpha)$的值可采用链接到它的网页的 PageRank 值和 Web 图中汇点的 PageRank 值表示，公式如下：

$$r(\alpha)=\delta\cdot\left(\sum_{\beta\to\alpha}\frac{r(\beta)}{\text{out}(\beta)}+\sum_{\gamma\in\Gamma}\frac{r(\gamma)}{N}\right)+(1-\delta)\cdot\sum_{\alpha\in\Phi}\frac{r(\alpha)}{N} \tag{15-6}$$

简单起见，由于选择是任意的，我们假设：

$$\sum_{\alpha\in\Phi}r(\alpha)=N \tag{15-7}$$

适当简化该公式，可得：

$$r(\alpha)=\delta\cdot\left(\sum_{\beta\to\alpha}\frac{r(\beta)}{\text{out}(\beta)}+\sum_{\gamma\in\Gamma}\frac{r(\gamma)}{N}\right)+(1-\delta) \tag{15-8}$$

因为$\sum_{\alpha\in\Phi}r(\alpha)/N=1$，随机浏览到页面 α 的概率是 $r(\alpha)/N$。

接下来，我们对公式（15-8）进行详细分析。首先，（1−δ）的值反映了随机跳转到页面 α 的贡献度。分析其他来自链接和汇点的对 α 页面的 PageRank 值的贡献就要复杂多了。从链接角度看，可能顺着页面 β 的链接可以到达页面 α。因为页面 β 上可能有多个指向其他网页的链接，所以网页对网页的 PageRank 值的贡献就依赖它的出度 out(β)。最后，汇点对网页 α 的 PageRank 值的贡献就显得较小了。这是因为在 Web 图中汇点 r 可能指向图中的任何一个网页（共 N 个）——包括它自己——那么汇点对 PageRank 值做出的贡献为 $r(\gamma)/N$。

对图 15-2 的 Web 图应用公式（15-8）可得出以下公式：

$$r(w_0)=\delta\cdot\left(r(w_1)+\frac{r(w_2)}{2}+r(h_0)+\frac{r(m_0)}{6}\right)+(1-\delta)$$

$$r(w_1)=\delta\cdot\left(\frac{r(w_0)}{3}+\frac{r(m_0)}{6}\right)+(1-\delta)$$

$$r(w_2)=\delta\cdot\left(\frac{r(w_0)}{3}+\frac{r(m_0)}{6}\right)+(1-\delta)$$

$$r(h_0)=\delta\cdot\left(\frac{r(w_0)}{3}+r(h_1)+\frac{r(m_0)}{6}\right)+(1-\delta)$$

$$r(h_1)=\delta\cdot\left(\frac{r(m_0)}{6}\right)+(1-\delta)$$

$$r(m_0)=\delta\cdot\left(\frac{r(w_2)}{2}+\frac{r(m_0)}{6}\right)+(1-\delta)$$

令 $\delta=3/4$，简化可得：

$$r(w_0)=\frac{3r(w_1)}{4}+\frac{3r(w_2)}{8}+\frac{3r(h_0)}{4}+\frac{r(m_0)}{8}+\frac{1}{4}$$

$$r(w_1)=\frac{r(w_0)}{4}+\frac{r(m_0)}{8}+\frac{1}{4}$$

$$r(w_2)=\frac{r(w_0)}{4}+\frac{r(m_0)}{8}+\frac{1}{4}$$

$$r(h_0)=\frac{r(w_0)}{4}+\frac{3r(h_1)}{4}+\frac{r(m_0)}{8}+\frac{1}{4}$$

$$r(h_1)=\frac{r(m_0)}{8}+\frac{1}{4}$$

$$r(m_0)=\frac{3r(w_2)}{8}+\frac{r(m_0)}{8}+\frac{1}{4}$$

为计算 PageRank，我们需解得线性系统公式中的 6 个变量 $r(w_0)$、$r(w_1)$、$r(w_2)$、$r(h_0)$、$r(h_1)$和 $r(m_0)$的值。有很多算法可以用来求解线性系统的数值解，上面介绍的就是其中一种。这种适用求 PageRank 值的方法其实是**不动点迭代**（fixed-point iteration）的一种形式。

不动点迭代是解决公式、线性以及其他系统问题的常规技术。使用时，每个变量用一个关于其他变量和自己的函数来表达。PageRank 公式已经写成了这种形式，公式左边是各个单独出现的网页的 PageRank 值，而公式右边是这些 PageRank 值的函数。

算法开始时先给各个变量一个初始化估计。然后将这些值代入公式右边，从而每个变量得到一个新的近似值。重复上述操作，不断带入当前值以获得新的近似值。如果得出的值收敛，即每次迭代产生相同的结果，那么就得到公式系统的解。

这种方法为每个页面 α 产生了一系列的近似值 $r(\alpha)$：$r^{(0)}(\alpha)$、$r^{(1)}(\alpha)$、$r^{(2)}(\alpha)\cdots$如果设定每个网页的初始值为 $r^{(0)}(\alpha)=1$，我们可以得到公式（15-7）所需的：

$$\sum_{\alpha\in\Phi} r^{(0)}(\alpha)=N \tag{15-9}$$

根据公式（15-8），使用以下公式从现有的近似值计算出新的近似值：

$$r^{(n+1)}(\alpha)=\delta\cdot\left(\sum_{\beta\to\alpha}\frac{r^{(n)}(\beta)}{\text{out}(\beta)}+\sum_{\gamma\in\Gamma}\frac{r^{(n)}(\gamma)}{N}\right)+(1-\delta) \tag{15-10}$$

对图 15-2 所示的 Web 图计算 PageRank 值，迭代以下公式：

$$r^{(n+1)}(w_0)=\frac{3r^{(n)}(w_1)}{4}+\frac{3r^{(n)}(w_2)}{8}+\frac{3r^{(n)}(h_0)}{4}+\frac{r^{(n)}(m_0)}{8}+\frac{1}{4}$$

$$r^{(n+1)}(w_1)=\frac{r^{(n)}(w_0)}{4}+\frac{r^{(n)}(m_0)}{8}+\frac{1}{4}$$

$$r^{(n+1)}(w_2)=\frac{r^{(n)}(w_0)}{4}+\frac{r^{(n)}(m_0)}{8}+\frac{1}{4}$$

$$r^{(n+1)}(h_0)=\frac{r^{(n)}(w_0)}{4}+\frac{3r^{(n)}(h_1)}{4}+\frac{r^{(n)}(m_0)}{8}+\frac{1}{4}$$

$$r^{(n+1)}(h_1)=\frac{r^{(n)}(m_0)}{8}+\frac{1}{4}$$

$$r^{(n+1)}(m_0)=\frac{3r^{(n)}(w_2)}{8}+\frac{r^{(n)}(m_0)}{8}+\frac{1}{4}$$

表 15-1 给出了这个公式系统对 PageRank 值的逼近过程。第 18 次迭代后估计值精确到小数点后 3 位。

表 15-1 迭代计算图 15-1 中的 Web 图的 PageRank 值

n	$r^{(n)}(w_0)$	$r^{(n)}(w_1)$	$r^{(n)}(w_2)$	$r^{(n)}(h_0)$	$r^{(n)}(h_1)$	$r^{(n)}(m_0)$
0	1.000	1.000	1.000	1.000	1.000	1.000
1	2.250	0.625	0.625	1.375	0.375	0.750
2	2.078	0.906	0.906	1.188	0.344	0.578
3	2.232	0.842	0.842	1.100	0.322	0.662
4	2.104	0.891	0.891	1.133	0.333	0.648
5	2.183	0.857	0.857	1.107	0.331	0.665
6	2.128	0.879	0.879	1.127	0.333	0.655
7	2.166	0.864	0.864	1.114	0.332	0.661
8	2.140	0.874	0.874	1.123	0.333	0.657
9	2.158	0.867	0.867	1.116	0.332	0.660
10	2.145	0.872	0.872	1.121	0.332	0.658
11	2.154	0.868	0.868	1.118	0. 332	0.659

（续）

n	$r^{(n)}(w_0)$	$r^{(n)}(w_1)$	$r^{(n)}(w_2)$	$r^{(n)}(h_0)$	$r^{(n)}(h_1)$	$r^{(n)}(m_0)$
12	2.148	0.871	0.871	1.120	0.332	0.658
13	2.152	0.869	0.869	1.119	0.332	0.659
14	2.149	0.870	0.870	1.120	0.332	0.658
15	2.151	0.870	0.870	1.119	0.332	0.659
16	2.150	0.870	0.870	1.119	0.332	0.658
17	2.151	0.870	0.870	1.119	0.332	0.659
18	2.150	0.870	0.870	1.119	0.332	0.658
19	2.150	0.870	0.870	1.119	0.332	0.659
20	2.150	0.870	0.870	1.119	0.332	0.658
...	...	...	...	...	...	...

图 15-6 为计算 PageRank 的详细算法。算法假设 Web 图中的网页分别用 $1 \sim N$ 连续标记，用长度为 E 的数组来存放 Web 图中的链接，Link[i]. from 表示链接的来源，Link[i]. to 表示链接的目的地。数组 R 存放当前 PageRank 的近似值；数组 R'存放新的近似值。

```
1   for i ← 1 to N do
2     R[i] ← 1
3   loop
4     for i ← 1 to N do
5       R'[i] ← 1 − δ
6     for k ← 1 to E do
7       i ← link[k].from
8       j ← link[k].to
9       R'[j] ← R'[j] + δ·R[i]/out(i)
10    s ← N
11    for i ← 1 to N do
12      s ← s − R'[i]
13    for i ← 1 to N do
14      R[i] ← R'[i] + s/N
```

图 15-6 基本的 PageRank 算法。数组 link 用于存储 Web 图中的链接。第 3～14 行是一个无限循环。每次该循环结束后，数组 R 中存放的是所有页面 PageRank 值的最新估计。实际应用中，当多次迭代后的 PageRank 值小于某个阈值时，或进行了一定次数迭代后，循环将终止

第 1～2 行的循环设置 R 的初始近似值。主循环（第 3～14 行）每次迭代完成后数组 R 中存放的都是下一个近似值。从代码上看，这构成了一个无限循环。实际上，一旦近似值收敛或迭代一定次数后就应停止循环。我们没有给出确切的终止条件，是因为它取决于期望的准确度和计算的查准率。在大多数 Web 图中，几百次迭代就可得到可接受的值了。主循环每次迭代需要时间为 $O(N+E)$；整个算法的时间复杂度取决于主循环迭代的次数。

第 4～5 行通过存储跳转贡献值来初始化数组 R'。第 6～9 行依次考虑每个链接，求取源网页对目标网页的 PageRank 值的贡献。由于$\sum_{\alpha \in \Phi} r(\alpha)=N$，汇点集的贡献可忽略不计。相反地，在初始化 R'以及应用链接数以后，我们对 R'中的元素进行求和。求和结果与 N 的差值就是 $\sum_{\gamma \in \Gamma} r(\gamma)$，即 Web 图中汇点的贡献值。

作为第 6～9 行的一个替代，我们也可以迭代所有的节点。对于每个节点，我们累加所有指向它的节点的 PageRank 贡献值，直接完成公式（15-8）的计算。尽管这种替代对于小的 Web 图是合理的，但是目前的这个方法对链接进行迭代所使用的数据结构更简单，也更

节省内存空间，这对于大规模的 Web 图来说是很重要的考虑因素。

对链接进行迭代允许我们将链接存储在磁盘文件中，进一步降低了内存需求。不需要在数组上进行迭代，在主循环的每一次迭代中，第 6～9 行的循环都会从头到尾顺序读取这个链接文件一次。如果链接存放在磁盘文件中，算法就只需要 $O(N)$ 的内存空间来存储数组 R 和 R'。

为证明该算法可用于大规模的 Web 图，我们对英文版的维基百科网站应用该算法（见练习 1.9）。表 15-2 中列出了排名前 12 位的网页。毫不意外，由于处理的只是英文版的维基百科，所以排名靠前的主要是使用英文的国家。剩下的也都是经济发达的国家，并且和美国或其他使用英文的国家关系密切。在维基百科中日期链接通常也扮演着重要角色，这就是为什么年份越靠近现在排名越靠前。20 世纪最重大的事件“第二次世界大战”排名靠前也是意料之中的事。总而言之，这些排名反映了维基百科的作者所处的时代和文化，这是一个合理的结果。

表 15-2 对维基百科网应用基本的 PageRank 算法得到的排名前 12 的网页

网页：α	PageRank：$r(\alpha)$	概率：$r(\alpha)/N$
美国（United States）	10 509.50	0.004 438
英国（United Kingdom）	3 983.74	0.001 682
2006	3 781.65	0.001 597
英国（England）	3 421.03	0.001 445
法国（France）	3 340.53	0.001 411
2007	3 301.65	0.001 394
2005	3 290.57	0.001 389
德国（Germany）	3 218.33	0.001 359
加拿大（Canada）	3 090.20	0.001 305
2004	2 742.86	0.001 158
澳大利亚（Australia）	2 441.65	0.001 031
第二次世界大战（World War Ⅱ）	2 417.38	0.001 021

15.3.2 扩展的 PageRank

如果我们重新检视和考虑 PageRank 算法的初衷：随机浏览，那么基本的 PageRank 算法的局限性就变得明显了。现实中没有用户是以这种随机方式浏览 Web 的。即使我们将随机浏览视为是人为构造的——代表了 Web 用户的普遍行为——将随机选择链接和跳转看成均匀分布仍然是不现实的。

更现实一点讲，随机浏览的用户应该会偏向于某类链接或跳转。例如，相比于指向广告的链接，她更喜欢导航链接；相比于网页下面的链接，她更喜欢网页顶端的链接；相比于点击那些字体超小或隐藏字体的链接，她更喜欢点击正常字体的链接。当随机浏览用户进行随机跳转时，顶层的网页比深层嵌套的网页更受欢迎，持久存在的网页也比刚出现的网页更受欢迎，文字较多的网页比文字较少的网页更受欢迎。

很幸运，可以很容易地扩展 PageRank 算法来适应以上用户偏好。为适应用户的跳转倾向，可定义一个**传送向量**（teleport vector）或**跳转向量**（jump vector）J，长度为 N，其中元素 $J[i]$ 表示当随机浏览用户跳转到目标页面 i 的概率。由于 J 是一个概率向量，需要保证 $\sum_{i=1}^{N} J[i]=1$。J 的一些元素可取 0，但这可能会导致某些页面的 PageRank 值为 0。如

果 J 的所有元素的取值均为正值，即对所有 $1 \leqslant i \leqslant N$ 都有 $J[i]>0$，那么所有页面的 PageRank 值均为非 0（15.3.3 节）。

为了适应用户的点击链接的倾向，定义一个大小为 $N \times N$ 的**追随矩阵**（follow matrix）F，其中元素 $F[i,j]$表示随机浏览用户从页面 i 跳转到页面 j 的概率。如果页面不是汇点，那矩阵 F 中对应行的总和为 1，即 $\sum_{i=1}^{N} F[i,j] = 1$。如果页面 i 是汇点，则所在行的元素值的总和为 0。

F 是**稀疏的**(sparse)，N^2 个元素中至多有 E 个非零元素，每个元素对应一个链接。在实现扩展 PageRank 算法时，我们利用了 F 的这一特性。算法如图 15-7 所示。除了第 5、9 和 14 行进了泛化以使用跳转向量和追随矩阵以外，该算法与基本 PageRank 算法类似。这种泛化对算法的时间复杂度并没有影响，主循环每次迭代耗时仍为 $O(N+E)$。

由于 F 是稀疏的，F 中的元素可能与其对应的链接存储在一起。更具体地讲，我们可以给数组中的每个元素添加一个域，如下定义：

$$link[k].follow = F[link[k].from, link[k].to]$$

```
1    for i ← 1 to N do
2        R[i] ← J[i] · N
3    loop
4        for i ← 1 to N do
5            R'[i] ← (1 − δ) · J[i] · N
6        for k ← 1 to E do
7            i ← link[k].from
8            j ← link[k].to
9            R'[j] ← R'[j] + δ · R[i] · F[i, j]
10       s ← N
11       for i ← 1 to N do
12           s ← s − R'[i]
13       for i ← 1 to N do
14           R[i] ← R'[i] + s · J[i]
```

图 15-7 扩展的 PageRank 算法。跳转向量 J 的每个元素 $J[i]$表示用户随机跳转到页面 i 的概率。追随矩阵 F 中的每个元素 $F[i, j]$ 表示用户点击链接时从页面 i 跳转到页面 j 的概率

扩展的链接数组也可以存储在磁盘文件中，修改后的第 6～9 行在这个文件上进行迭代。通过将链接存储在磁盘上，该算法只需要 $O(N)$的 RAM 空间来存储数组 R 和 R'以及跳转向量 J。

用这个扩展算法来计算基本的 PageRank 值，我们构造一个跳转向量，其全部元素均为 $1/N$。对于追随矩阵，设页面 α 指向的所有页面的元素值均为 $1/\text{out}(\alpha)$，其余元素值均取 0。如果我们将图 15-2 中 Web 图的节点从 1～6 依次编号命名为 w_0、w_1、w_2、h_0、h_1 和 m_0，则可得基本 PageRank 的追随矩阵和跳转向量：

$$F = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad J = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix} \tag{15-11}$$

现在，假设我们已知一些外部信息：网站 W 中的内容都是高质量且经过仔细编纂的，

同时其他网站中的信息的相对质量是未知的。我们可以对追随矩阵和跳转向量进行调整来反映这个情况，假设随机浏览用户通过链接或跳转而访问 W 网站的概率要比访问其他网站高两倍，具体调整如下：

$$F=\begin{pmatrix} 0 & \frac{2}{5} & \frac{2}{5} & \frac{1}{5} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{2}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad J=\begin{pmatrix} \frac{2}{9} \\ \frac{2}{9} \\ \frac{2}{9} \\ \frac{1}{9} \\ \frac{1}{9} \\ \frac{1}{9} \end{pmatrix} \tag{15-12}$$

对一般 Web 检索而言，为反映外部知识，我们可能会对追随矩阵和跳转向量做很多次调整。但一般来说，不应做那些反映 Web 图本身属性的调整。例如，假如很多网页指向一个给定网页，我们可将该网页看成是“受欢迎程度”较高的网页，从而增大跳转向量中它的概率值。尽量不要进行这样的调整。这种类型的调整是不需要也不推荐的。当调整追随矩阵和跳转向量时应只关注外部信息。这是 PageRank 的本职工作：反映 Web 图的性质。

建立追随矩阵和跳转向量应考虑到很多外部信息。受篇幅限制，我们无法描述和分析所有可能的情况，只给出如下可能的情况。这个信息列表还不够详尽（见练习 15.6），而且在实际应用中并非会用到所有（或任一）建议。

- **页面内容与结构。**在网页抓取和索引过程中，搜索引擎会对将要出现在浏览器中的页面的结构和组织进行分析。该分析将根据网页布局和外观对链接赋予不同的概率值。例如，用户可能更喜欢点击在页面顶部或菜单栏里的链接。页面内容的诸多方面都可以反映出页面质量，使页面更易于成为跳转的目标，从而影响跳转向量的值。大块的可读文档可理解为一个正面因素，HTML 标签较多而文本较少则可能是一个负面因素。字体过小或不可读都会误导搜索引擎，使得当页面显示的时候用户看到的信息是错误的。
- **网站内容与结构。**用户往往更倾向于访问站点的顶层网页，而不喜欢访问深层嵌套的网页。并且，越长的网址用户越不容易记住，也不太会直接在浏览器地址栏中输入。网站中网页的数量也是确定跳转向量时应考虑的因素之一：一个网站不应该因为其包含的网页多就获得更高的跳转概率。从网站本身来看，用户更倾向于根据网站导航逐步访问站内链接，而不喜欢点击一个跳转到其他网站的链接。另一方面，一些教育站点和商业站点之间的链接会更受欢迎，因为这些链接的商业目的不是很强，更迎合大众口味。同时，较早成立的较为成熟的网站会比新网站更受欢迎。
- **显式判断。**雇用编辑人工确定（分类）那些高质量的网站，从而这些网站获得更高的跳转概率。这些编辑们有的是搜索服务本身的专家，也有一些是为如开放式目录管理系统（ODP——见 Open Directory Project[⊖]）这样的 Web 目录工作的志愿者。主流搜索引擎会让内部员工来管理他们自己的 Web 目录，将其作为整体服务的一部分。
- **隐式反馈。**点击 Web 搜索结果上的网页可以看成是对该网页质量进行了隐式反馈（见 15.5.2 节）。很多主流搜索服务都提供**工具条**（toolbar）。这些浏览器的扩展功能为补充基本搜索服务提供了支持。经用户许可，工具条可将与用户的访问习惯相关的

⊖ www.dmoz.org

信息反馈给搜索服务。用户访问的站点和网页可能会被记录下来，用来调整追随矩阵和跳转向量。许多搜索服务也提供免费电子邮箱账号。访问通过这些电子邮件服务发送的链接可以直接到达这些搜索服务，也反映了网站和网页的质量。当然，所有的隐式反馈都是在保护用户隐私的前提下进行的。

对跳转向量的调整也可用来计算 PageRank 的各种变形。为计算**个性化** PageRank（personalized PageRank），我们对个人用户感兴趣的网页赋予较高的跳转概率（Page 等人，1999）。这些页面可通过用户的浏览器书签或个人主页得到，也可以通过监测其一段时间内的浏览习惯来完成。**面向主题的** PageRank（topic-oriented PageRank）或**聚焦的** PageRank（focused PageRank）会给那些与如体育、商业等特定主题相关的网页较高的跳转概率（Haveliwala，2002）。这些面向主题的网页可能来自诸如开放式目录管理系统（ODP）这样的 Web 目录。

例如，我们从维基百科语料库中选取一个网页，然后专门针对这个网页，生成特定主题的 PageRank 值。为生成跳转向量，我们将 50% 的跳转概率赋予选定页面，剩余的 50% 平均分配给其他网页。为确保所有网页都得到一个非零的 PageRank 值，我们给跳转向量的每个元素都赋予一个非零值，但事实上就算我们为选定页面赋予 100% 的跳转概率，输出也不会改变。

表 15-3 所示为两个主题“William Shakespeare”和“Information Retrieval”的前 12 位网页。对于“Information Retrieval”这个主题，面向特定主题的 Page Rank 会为许多与此主题相关的页面赋予高分。在第 2 章和第 8 章中出现过的名字 Karen Spärck Jones 和 Gerard Salton；罗格斯大学（Rutgers University），伦敦城市大学（City University London）和格拉斯哥大学（University of Glasgow），它们都拥有杰出的信息检索研究小组；Google 和 SIGIR 的出现就更加不足为奇了。然而令人惊讶的是，主题“William Shakespeare”的 PageRank 结果与图 15-2 中的一般的 PageRank 结果非常接近。除了“English language”的网站和莎士比亚主页以外，其他网页都出现在了一般的 PageRank 结果的前 12 位。尽管我们进行的是特定主题的搜索，美国的网页还是在两个列表中都位居第二。

表 15-3 在维基百科中，为两个主题采用特定主题的 PageRank 算法后排名前 12 位的网页

William Shakespeare		Information Retrieval	
文章	PageRank	文章	PageRank
William Shakespeare	303 078.44	Information retrieval	305 677.03
United States	7 200.15	United States	8 831.25
England	5 357.85	Association for Computing Machinery	6 238.30
London	3 637.60	Google	5 510.16
United Kingdom	3 320.49	GNU General Public License	4 811.08
2007	3 185.71	World Wide Web	4 696.78
France	2 965.52	SIGIR	4 456.67
English language	2 714.88	Rutgers University	4 389.07
2006	2 702.72	Karen Spärck Jones	4 282.03
Germany	2 490.50	City University, London	4 274.76
2005	2 377.21	University of Glasgow	4 222.44
Canada	2 058.84	Gerard Salton	4 171.45

特定主题的 PageRank 和一般的 PageRank 都揭示了同一个网页集合中的概率分布。继续讨论刚刚的例子，通过比较这些分布我们可一个到更加明确的结果。在 9.4 节中我们定义了 kullback-Leibler 距离，或称为 KL 距离，即离散概率分布 f 和 g 之间的相对熵：

$$\sum_x f(x) \cdot \log \frac{f(x)}{g(x)} \tag{15-13}$$

这里我们定义 f 为特定主题的 PageRank，g 为一般的 PageRank，且对它们进行归一化以表示概率。根据每个网页 α 对 f 和 g 之间相对熵的**贡献**（contribution）对其重新进行排名：

$$f(\alpha) \cdot \log \frac{f(\alpha)}{g(\alpha)} \tag{15-14}$$

回顾如果我们用分布 g 取代正确的分布 f，那么相对熵就表示压缩每个符号平均需要额外增加的位的数量。公式（15-14）表明额外增加的位数由页面 α 决定。表 15-4 列出了使用公式（15-14）对特定主题的 PageRank 进行调整后所产生的影响。

表 15-4 根据每个网页对相对熵（特定主题的 PageRank 与一般的 PageRank）的贡献重新获得特定主题的 PageRank 的排名

William Shakespeare		Information Retrieval	
文章	相对熵	文章	相对熵
William Shakespeare	1.505 587	Information Retrieval	2.390 223
First Folio	0.007 246	SIGIR	0.027 820
Andrew Cecil Bradley	0.007 237	Karen Spärck Jones	0.026 368
King's Men (playing company)	0.005 955	C.J. van Rijsbergen	0.024 170
Twelfth Night , or What You Will	0.005 939	Gerard Salton	0.024 026
Lord Chamberlain' s Men	0.005 224	Text Retrieval Conference	0.023 260
Ben Jonson	0.005 095	Cross-language information retrieval	0.022 819
Stratford-upon-Avon	0.004 927	Relevance (information retrieval)	0.022 121
Richard Burbage	0.004 794	Assoc. for Computing Machinery	0.022 051
George Wilkins	0.004 746	Sphinx (search engine)	0.021 963
Henry Condell	0.004 712	Question answering	0.021 773
Shakespeare's reputation	0.004 710	Divergence from randomness model	0.021 620

现在，所有排名靠前的网页都是与主题相关的网页。Andrew Cecil Bradley 是斯坦福大学的一名教授，他以关于莎士比亚的戏剧和诗集的著作而闻名。Ben Jonson 和 George Wilkins 是跟莎士比亚同一时期的剧作家。Richard Burbage 和 Henry Condell 是 King's Men 戏剧公司出演莎士比亚作品的演员和剧组成员。在“Information Retrieval”一栏，Spärck Jones 和 Salton 以及 C. J.（Keith）van Rijsbergen 都是该领域的巨匠。Sphinx 是一个基于关系数据库系统的开源搜索引擎。其他的搜索结果应该是读者们耳熟能详的了。

15.3.3 PageRank 的性质

在阐述 PageRank 时我们忽略掉了几个很重要的考虑因素：如图 15-7 中的不动点迭代总是会收敛的吗？对某些 Web 图而言该算法会不会失效？如果算法收敛，那么收敛速度又如何？忽略初始估计值，该算法每次都会收敛到同一 PageRank 向量吗？

幸运的是，PageRank 算法具有保证运行正常的性质。为简单讨论这些性质，我们将追随矩阵与跳转向量结合起来得出一个**转移矩阵**（transition matrix）。首先对以下矩阵进行扩展以处理汇点，设 F' 是一个 $N \times N$ 的矩阵：

$$F'[i,j] = \begin{cases} J[j] & \text{若} i \text{为汇点} \\ F[i,j] & \text{否则} \end{cases} \tag{15-15}$$

接下来，令 J' 为一个 $N \times N$ 的矩阵，其中每个行元素均是一个跳转向量。最后，得到的转

移矩阵如下：

$$M=\delta\cdot F'+(1-\delta)\cdot J' \tag{15-16}$$

对于页面 i 和 j，$M[i,j]$ 表示不管是通过跳转还是点击链接，随机浏览用户从页面 i 转移到页面 j 的概率。例如，由公式（15-12）中的 F 和 J 得出的转移矩阵如下：

$$M=\begin{pmatrix} \frac{1}{18} & \frac{16}{45} & \frac{16}{45} & \frac{37}{180} & \frac{1}{18} & \frac{1}{18} \\ \frac{29}{36} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} \\ \frac{5}{9} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{11}{36} \\ \frac{7}{9} & \frac{1}{36} & \frac{1}{36} & \frac{1}{36} & \frac{1}{36} & \frac{1}{36} \\ \frac{1}{36} & \frac{1}{36} & \frac{1}{36} & \frac{7}{9} & \frac{1}{36} & \frac{1}{36} \\ \frac{7}{36} & \frac{7}{36} & \frac{7}{36} & \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix} \tag{15-17}$$

正如 1.3.4 节中所述，可以把随机矩阵 M 看做一个代表马尔可夫链转移矩阵的随机矩阵。每个页面对应一个状态；随机浏览用户开始时访问的初始位置设为初始状态。此外，PageRank 向量 R 具有如下性质：

$$M^{\mathrm{T}}R=R \tag{15-18}$$

可以把 R 看做矩阵 M^{T} 的特征向量[⊖]，特征值均为 1。给定一个 $n\times n$ 的矩阵 A，如果 $A\vec{x}=\lambda_{\vec{x}}$，则称 $\vec{x}$ 为 A 的**特征向量**（eigenvector），λ 为 A 的**特征值**（eigenvalue）。

对于马尔可夫链和特征向量的了解已经足够多了，接下来我们可以将其用于判断 PageRank 的性质。简单起见，暂时假设跳转向量中所有元素都是正的，但是当元素值为 0 时这些性质仍然成立。后面章节我们再考虑跳转向量有 0 个元素的情况。

潜在 PageRank 是指当随机浏览用户长时间浏览网站时，他在每个网页上所耗的时间收敛于一个固定值，这个值与用户起始访问地址是无关的。这个概念就反映了马尔可夫链的一个重要特性——**遍历性**（ergodicity）。如果两个性质都成立，说明该马尔可夫链是**遍历的**（ergodic），当 J 中元素值都为正时，则对于 M 也成立。第一个性质，**不可约性**（irreducibility），是指随机浏览用户可以在有限步数内从任意一个状态到达到其他任意状态。更确切地说，给定任意两个状态 i 和 j，若用户当前处于状态 i，存在一个正概率使得用户在 k 步以内到达状态 j，其中 $k\leqslant N$。如果跳转向量元素仅包含正数，则对于所有的状态 i 和 j 有 $M[i,j]>0$，且存在一个正概率使得用户从状态 i 经过 $k=1$ 步就能到达状态 j。

第二个性质，转移矩阵必须是**非周期**的（aperiodic），排除马尔可夫链中存在多值之间的状态循环的概率。状态 i 在周期 k 内是**周期的**（periodic），是指经过 k 的倍数步后能够重新回到本身。例如，若用户总是经过偶数步后返回到同一状态，那么这个状态的周期为 2。如果矩阵所有状态的周期均为 1，则称该矩阵是非周期的。因为对于所有状态 i 有 $M[i,j]>0$，所以用户可从一个状态跳到其他任一状态。因此，满足所有状态的周期均为 1 这一性质。

设 M^{T} 的特征向量为 $\vec{x}_1$，$\vec{x}_2$，…，$\vec{x}_N$，对应的特征值为 λ_1，…，λ_N。按惯例对这些特征向量进行排序，可得 $|\lambda_1|\geqslant|\lambda_2|\geqslant\cdots\geqslant|\lambda_N|$。由于 M^{T} 中所有元素值均为正，且行元素和为 1，由**佩龙-弗罗宾尼斯定理**（Perron-Frobenius theorem）可知 $\lambda_1=1$，所有其他特征值均满足 $|\lambda_i|<1$，且 $\vec{x}_1$ 中所有元素均为正数。因此，M^{T} 的**主特征向量**（principal eigenvector）是 PageRank **向量**（PageRank vector）。按惯例，特征向量需规范化到单位长度，对所有 i 有 $\|\vec{x}_i\|=1$，使 $\vec{x}_1=R/N$。

⊖ 如果你已经忘记了线性代数的一些知识，可以参考 15.3.4 节前面的部分。

现在我们用矩阵表示法重新阐述计算 PageRank 值的算法。设 $\vec{x}^{(0)}$ 为 PageRank 的初始估计（归一化使其长度为 1）。图 15-7 中所示的主循环每次进行迭代时，将当前估计值与矩阵 M^{T} 相乘，从而获得新的估计值。第一次迭代可得 $\vec{x}^{(1)}=M^{\mathrm{T}}\vec{x}^{(0)}$，第二次迭代可得 $\vec{x}^{(2)}=M^{\mathrm{T}}\vec{x}^{(1)}$，如此类推。第 n 次迭代后可获得估计值

$$\vec{x}^{(n)}=\left(M^{\mathrm{T}}\right)^{n}\vec{x}^{(0)} \tag{15-19}$$

因此，如果

$$\lim_{n\to\infty}\vec{x}^{(n)}=\vec{x}_1 \tag{15-20}$$

算法计算出 pageRanK 的值。矩阵 M 的遍历性保证了算法是收敛的。在马尔可夫链的术语中，$\vec{x}_1$ 称为 M 的**静态分布**（stationary distribution）。这个计算矩阵的主特征向量的算法称为**乘幂法**（power method）（Golub 和 Van Loan，1996）。

为确定收敛速率，将初始估计 $\vec{x}^{(0)}$ 表示为（未知的）特征向量的线性组合：

$$\vec{x}^{(0)}=\vec{x}_1+a_2\vec{x}_2+a_3\vec{x}_3+\cdots+a_N\vec{x}_N \tag{15-21}$$

在第一次迭代后，可得：

$$\begin{aligned}\vec{x}^{(1)}&=M^{\mathrm{T}}\vec{x}^{(0)}\\&=M^{\mathrm{T}}\left(\vec{x}_1+a_2\vec{x}_2+\cdots+a_N\vec{x}_N\right)\\&=M^{\mathrm{T}}\vec{x}_1+a_2M^{\mathrm{T}}\vec{x}_2+\cdots+a_NM^{\mathrm{T}}\vec{x}_N\\&=\vec{x}_1+a_2\lambda_2\vec{x}_2+\cdots+a_N\lambda_N\vec{x}_N\end{aligned}$$

在 n 次迭代后，因为λ_2 为位居λ_1 之后的第二大特征向量，可得：

$$\begin{aligned}\vec{x}^{(n)}&=\vec{x}_1+a_2\lambda_2^n\vec{x}_2+\cdots+a_N\lambda_N^n\vec{x}_N\\&\leqslant\vec{x}_1+\lambda_2^n\left(a_2\vec{x}_2+\cdots+a_N\vec{x}_N\right)\\&=\vec{x}_1+O(\lambda_2^n)\end{aligned}$$

由此可见，收敛速度取决于 M^T 的第二大特征向量的值。值越小，收敛速度越快。

对于 PageRank、Haveliwala 和 Kamvar（2003）证明了第二大特征向量值为 δ，即为阻尼系数[⊖]。如果 δ 的值不太接近于 1，则收敛速度在可接受范围内。更为重要的是，收敛速率不依赖于 Web 图本身的特征。如果对于一个 Web 图来说，算法收敛速度足够快，那么即使 Web 图随着时间不断变化，算法对于该图的收敛速度还是相当快的。

Haveliwala 和 Kamvar 同时还指出 PageRank 算法的稳定性也与 δ 有关。δ 的值越小，Web 图上的小变动对 PageRank 造成的影响越小。当然，如果 δ 值接近于 0，PageRank 就无法获得任何关于 Web 图的有用信息，因为几乎在每一步都会产生随机跳转。为保证 PageRank 算法有好的稳定性和收敛性，最好保持传统的值，取 $\delta=0.85$。

如此看来，虚拟访问者的随机跳转确定了 PageRank 的稳定性和收敛性。跳转向量保证了 M 的遍历性，从而也就保证了 PageRank 算法的收敛性。阻尼系数决定了算法的稳定性和收敛速度。而 Web 图的结构本身的作用并不是很大。

以上讨论均假设跳转向量所有元素值均为正数。如果假设不成立，跳转向量中包含值为 0 的元素，那么 M 就不一定是不可约的了。跳转向量中的 0 值元素说明随机浏览用户永远都不可能随机跳转到该页面。假设存在网页 α，从其他任何跳转概率非零的网页都不能跳转到该网页。那么就意味着访问者一旦离开页面 α，就不可能再通过其他页面跳转回来。

⊖ 更准确地说，$|\lambda_2|\leqslant\delta$，但对于任何现实的 Web 图均有$\lambda_2=\delta$。

如果假设访问次数为无穷多，那么跳转概率最终会趋向于 1。每次跳转之后到达网页 a 的概率都为 0。因此，不用经过精确计算，我们就可以知道网页 a 的 PageRank 值为 0。

设 Φ' 为跳转可达页面集合（图 15-8）。除 Φ' 以外的页面 PageRank 值均为 0，而且在计算 Φ' 中的网页的 PageRank 时，这些 PageRank 值为 0 的网页可忽略不计。设 M' 是 Φ' 的转移矩阵，J' 为其跳转向量。J' 可能仍含有值为 0 的元素，但是与这些零值元素相关的网页都可以通过非零值的网页跳转到达。因为每一步中都可能发生跳转，而 Φ' 中的任何页面都可能在每次跳转后到达，因此 M' 是不可约的。每一跳的概率值也有助于保证 M' 是非周期的（练习 15.5）。由于 M' 是不可约非周期的，因此它是遍历的且其 PageRank 是收敛的。稳定性和收敛性等其他性质在 M' 中也体现出来了。

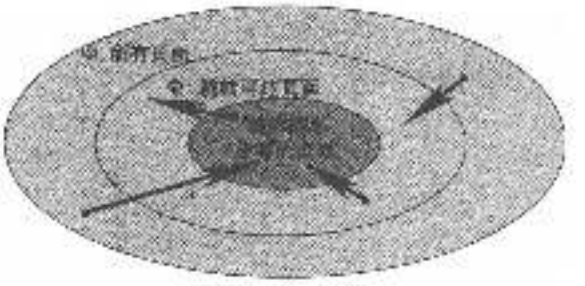


图 15-8 跳转可达页面

15.3.4 其他链接分析方法：HITS 和 SALSA

除 PageRank 以外，还有大量关于 Web 的链接分析技术。其中有两种方法最为著名，分别是 Kleinberg 的 HITS 算法（Kleinberg，1998，1999）与 Lempel 和 Moran（2000）提出的 SALSA 算法。HITS 算法是在大约与 PageRank 算法同一时期提出的。紧接着就产生了 SALSA 算法，它兼有 PageRank 和 HITS 两种算法的一些特性。

HITS 潜在的含义是考虑一个页面对应某个给定主题在 Web 中所扮演的角色。其中一种称为权威（authority）网页，即网页包含大量该主题的可靠信息。第二个是链接（hub）网页，即网页聚集了大量与该主题相关的链接。一个好的链接网页会指向多个权威网页；一个好的权威网页也会被大量链接网页引用。需要强调的是，这些都是理想模型，某个网页在某种情况下可能既是权威网页又是链接网页。

正如人们推测 PageRank 算法适用于整个 Web 一样，Kleinberg 预言 HITS 将会适用于小型 Web 图（小型网络——译者注）。这些 Web 图可能是由在某个搜索引擎上执行某个查询得到的前几百个网页及其相邻网页所组成的。基于这些检索到的页面，HITS 计算对于这个查询而言哪些是权威网页哪些是链接网页。在发明 HITS 之时，人们还不知道 Web 搜索引擎可以与链接分析技术一并使用（Marchiori，1997），在没有搜索引擎明确的支持下，HITS 提供了一种可以利用链接分析技术的好方法。

对于给定页面 a，HITS 计算两个值：权威值 $a(a)$ 和链接值 $h(a)$。一个页面的权威值由指向它的网页的链接值计算得到：

$$a(\alpha)=w_a\cdot\sum_{\beta\to\alpha}h(\beta) \tag{15-22}$$

一个页面的链接值由它指向的网页的权威值得到：

$$h(\alpha)=w_h\cdot\sum_{\alpha\to\beta}a(\beta) \tag{15-23}$$

下面简短讨论一下权重 w_a 和 w_h 的意义。指向页面自身的链接（自环（self-loop））以及站内链接可以忽略不计，因为这些链接起到的多半是单纯的导航作用，而并非权威-链接方面的作用。

我们可以将公式（15-22）和公式（15-23）写成矩阵/向量形式，设 Web 图中有 N 个页面，定义 $\vec{a}$ 为权威值，$\vec{h}$ 为链接值。[⊖]令 W 为 Web 图的**邻接矩阵**（adjacency matrix），当有链接从页面 i 指向页面 j 时，$W[i,j]=1$，否则 $W[i,j]=0$。例如，图 15-2 中的 Web 图（保留站内链接）的邻接矩阵如下：

$$W=\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{15-24}$$

采用矩阵/向量的形式，公式（15-22）和公式（15-23）可写成如下形式：

$$\vec{a}=w_a \cdot W^{\mathrm{T}}\vec{h} \quad \text{和} \quad \vec{h}=w_h \cdot W\vec{a}. \tag{15-25}$$

代入可得：

$$\vec{a}=w_a \cdot w_h \cdot W^{\mathrm{T}}W\vec{a} \quad \text{和} \quad \vec{h}=w_a \cdot w_h \cdot WW^{\mathrm{T}}\vec{h} \tag{15-26}$$

如果定义 $A=W^{\mathrm{T}}W$，$H=WW^{\mathrm{T}}$，且 $\lambda=1/(w_a \cdot w_h)$，可得：

$$\lambda\vec{a}=A\vec{a} \quad \text{且} \quad \lambda\vec{h}=H\vec{h} \tag{15-27}$$

由此可得，权威向量 $\vec{a}$ 为矩阵 A 的特征向量，链接向量 $\vec{h}$ 为矩阵 H 的特征向量。

矩阵 A 和 H 在文献计量学（Lempel 和 Moran，2000；Langville 和 Meyer，2005）中有一个十分有趣的解释：矩阵 A 是 Web 图的**共被引矩阵**（co-citation matrix），其中 $A[i,j]$ 代表同时指向 i 和 j 的页面数；矩阵 H 是 Web 图的**互指矩阵**（co-reference matrix）或**耦合矩阵**（coupling matrix），其中 $H[i,j]$ 表示同时被 i 和 j 指向的网页数量。矩阵 A 和 H 都是对称的，这是计算特征向量 $\vec{a}$ 和 $\vec{h}$ 时一个重要的特性。

正如 PageRank 算法一样，我们可以用幂形式的不动点迭代方法来计算 $\vec{a}$ 和 $\vec{h}$。令 $\vec{a}^{(0)}$ 和 $\vec{h}^{(0)}$ 分别为 $\vec{a}$ 和 $\vec{h}$ 的初始估计。第 n 次估计值 $\vec{a}^{(n)}$ 和 $\vec{h}^{(n)}$ 可用以下公式根据前面的估计得到：

$$\vec{a}^{(n)}=W^{\mathrm{T}}\vec{h}^{(n-1)}/\|W^{\mathrm{T}}\vec{h}^{(n-1)}\| \quad \text{和} \quad \vec{h}^{(n)}=W\vec{a}^{(n-1)}/\|W\vec{a}^{(n-1)}\| \tag{15-28}$$

正则化可保证每个估计都是单位长度，从而避免对特征向量的显式计算。如果

$$\lim_{n\to\infty}\vec{a}^{(n)}=\vec{a} \quad \text{且} \quad \lim_{n\to\infty}\vec{h}^{(n)}=\vec{h} \tag{15-29}$$

连续运用这些公式，将计算出 $\vec{a}$ 和 $\vec{h}$。矩阵 M 的遍历性保证了 PageRank 算法的收敛性。在 HITS 方法中，只要初始估计有一个方向与主特征向量的方向相同，A 和 H 的对称性就可以保证 HITS 方法是收敛的（Kleinberg，1999；Golub 和 Van Loan，1996）。任意元素值全为正数的单位向量都可满足这一要求，如：

$$\vec{a}^{(0)}=\vec{h}^{(0)}=\langle 1/\sqrt{N}, 1/\sqrt{N}, \cdots \rangle$$

但是很遗憾，该算法不能保证收敛到唯一值（Langville 和 Meyer，2005）。基于不同的初始估计，HITS 算法可能收敛到不同结果（见练习 15.9）。

SALSA，即**链接结构的随机分析法**（stochastic approach for link-structure analysis），把 PageRank 算法的随机浏览理论引入到 HITS 中（Lempel 和 Moran，2000）。SALSA 的提出受这样的

⊖ 如果你不想详细了解，可以直接跳到 15.3.5 节。

观察启发：一小组高度互联的网站（或“紧耦合网络”）中，通过对成员赋予高的链接值和权威值，会对链接值和权威值产生不成比例的影响。

SALSA 假设随机浏览用户会在链接和**“回到上一页”链接**（backlink）（即反向链接）中跳转。在奇数步时用户从当前页面随机选择一个出链接进行访问。在偶数步时，用户随机选择一个从外部页面指向当前页面的入链接，并对其进行反向访问，找到链接源。随着访问步数的增加，奇数步前在网页中的停留时间表示网页的链接值；偶数步前在网页中的停留时间则表示网页的权威值。

尽管这里没有进行具体描述，但是我们知道 SALSA 算法也可以表示成特征向量的形式。Lempel 和 Moran（2000）讨论了可简化 SALSA 算法执行的一些性质。Rafiei 和 Mendelzon（2000）打算把随机跳转理论引入到类 SALSA 算法中，引入的内容还包括在进行一次奇数或偶数步访问后，随机跳转的概率。

15.3.5 其他静态排名方法

静态排名方法为每个网页提供了独立于请求的得分。尽管静态排名计算中最核心的部分是链接分析，但是其他特征也有着不同的作用。

在 15.3.2 节中讨论链接分析方法的时候，我们提到**用户的隐式反馈**（implicit user feedback）为网页提供了一些重要特征。所谓“隐式”，就是指用户在忙于其他事情时，不经意地进行了反馈动作。例如，对 Web 搜索结果的点击反映出用户对该网站或网页的偏好。如果排名靠前的结果被用户跳过，那么意味着对这些结果进行了负面反馈（Joachims 等人，2005）。商用搜索引擎提供的工具条可以跟踪用户访问的网站，并将这些信息反馈给搜索服务（在用户允许的情况下）。一个网站的受欢迎程度可以用访问量来衡量，也可能通过点击频率和用户停留时间来衡量（Richardson 等人，2006）。

网页本身的内容也可能对其静态排名造成影响。Ivory 和 Hearst（2002）描述了网页评定专家是如何采用定量的方式来鉴定网页内容和结构的，以预测网页的质量评分。他们通过考虑网页内容的总量和复杂度，网页布局中图片元素的位置和形式，以及对字体和颜色的选择等因素来进行评定。

最后，也会考虑 URL 的内容和结构。简短的 URL 比冗长的 URL 更受用户喜爱，特别是对导航查询的用户而言（Upstill 等人，2003）。对于商业查询而言，域名为 com 的网站比域名为 edu 的网站更合适；对学术查询而言则刚好相反。

与 11.7 节中描述的类似，Richardson 等人（2006）采用机器学习计算静态排名。他们说明如果把基本 PageRank 与受欢迎程度、页面内容、URL 特征等因素一起综合考虑，能比只使用基本 PageRank 获得很大的提高。15.3.2 节中我们曾讨论过，在扩展的 PageRank 方法中怎样用这些特征去调整追随向量和跳转向量。在静态排名方法中如何更好地利用这些特征至今仍是值得探讨的话题。

15.4 动态排名

查询阶段搜索引擎根据那些独立于查询的特征——如词频和邻近度——将每个网页的静态排名结合在一起，为这个查询产生一个动态的排名结果。这种应用于商用搜索引擎的动态排名算法是属于第三部分介绍的理论，由于 Web 搜索引擎不同，这些算法的细节也会有很大差异。此外，由于搜索引擎服务的查询和用户数据十分庞大，这些算法也需要不断地发展。

尽管本书的范围不包括特定的 Web 搜索引擎所采用的动态排名算法，但是其中有两点还是值得注意的。第一点是锚文本，它代表了 Web 搜索中特别重要的一类排名特征。锚文本通常提供对所指向页面的描述，可以用来提高检索效果。第二点——新颖性——源于 Web 的规模和结构。尽管给定网站中的许多网页都与某个查询相关，但相比于从同一个网站中返回很多网页，可能分别从多个网站中返回一个或两个网页这一做法会更好，因此也能提供更多不同的信息。

15.4.1 锚文本

锚文本通常对其指向的网页提供标记或描述。表 15-5 中所列出的就是在维基百科全书网站内指向网页 en.wikipedia.org/wiki/William_Shakespeare 的锚文本，且按照字符串出现次数进行了排序。维基百科全书网站外也有网页指向该页面，但要通过对整个 Web 进行抓取才能获得那些锚文本。表中所示的是一些混杂的文本，包括拼写错误的和一些昵称（如“the Bard”）。总而言之，在 6889 个锚文本中有 71 个不同的字符串可链接到该页面。然而，这些锚文本中有 45%以上都使用“Shakespeare”一词，与“William Shakespeare”几乎是一样的。

表 15-5 指向维基百科全书网页 en.wikipedia.org/wiki/William_Shakespeare 的锚文本，按照出现次数的多少进行排列

#	锚文本	#	锚文本
3123	Shakespeare	2	Will
3008	William Shakespeare	2	Shakesperean
343	Shakespeare's	2	Shakespeare
210	Shakespearean	2	Shakespearean studies
58	William Shakespeare's	2	Shakespeare
52	Shakespearian	2	Bill Shakespeare...
10	W. Shakespeare	1	the Bard's
7	Shakespeare, William	1	lost play of Shakespeare
3	William Shakepeare	1	Shakespearesque
3	Shakesphere	1	Shakespearean theatre
3	Shakespeare's Theatre	1	Shakespearean plays
3	Bard		...
2	the Bard		...

在整个 Web 上指向一个页面的链接数量可能从一个到上百万个。当页面被多个链接指向的时候，锚文本通常是重复的，而且重复的锚文本通常是页面的精确描述（除了练习 15.12所述的情况）。拼写错误、昵称以及相似的锚文本都是很有用的，因为即使这些词汇不会出现在目标网页中，也可能会出现在用户输入的查询中。

作为排名特征的锚文本必须与指向目标页面关联。在建立 Web 索引以前，必须从每个页面中提取出锚文本，并将其写成如下形式的元组集合：

```
<target URL, anchor text>
```

根据 URL 对这些元组进行排序。然后每个 URL 的锚文本与 URL 代表的页面内容合并在一起，构成一个索引用的组合文档。

为便于检索，锚文本也可看做一个文档域，和 8.7 节中所述的标题域和其他域一样。当计算词项权重时，我们将像对待普通文本一样对待锚文本，否则锚文本的大量重复会影响词

频的计算（Hawking 等人，2004）。在网页的静态排名中，锚文本也在计算词项权重时起到一定作用（Robertson 等人，2004）。在静态排名靠前的网页中出现的锚文本可以赋予比在静态排名靠后的网页中出现的锚文本高的权重。

15.4.2 新颖性

由于 Web 中的信息量大且种类繁多，对于搜索引擎而言，给用户提供一个包含各种信息的搜索结果集合是至关重要的。以查询〈"UPS"〉为例，搜索引擎返回的结果最好是一个包含快递服务、电源以及大学等的混合结果。有几种简单的方法可以降低搜索结果的冗余，包括找出索引中重复的网页，在检索后进行过滤以限制来自同一个 Web 网站的网页数量。

即使 Web 中有很多重复的（或近似重复的）网页，但检索结果中通常应该只保留其中一个副本。Web 中出现重复网页的原因有很多。很多网站在 URL 失效时就会返回默认网页，因此这些网站中所有的失效 URL 就会造成这个默认页面的多个重复版本。特定类型的网页内容通常会出现在多个网站间。例如，一个新闻文章会同时出现在多个报纸 Web 网站中。在 Web 爬虫抓取的时候就应该对这些重复进行检测和标记（见 15.6.3 节），在查询阶段就使用静态排名来选出最优的副本。

一般来说，网站中的重复页面应该从索引项中删除。然而，当这些重复页面出现在不同站点中时，保留它们也是合理的。多数商用 Web 搜索引擎都将检索范围限制在指定的网站或网域中。例如，如果查询中包含词项 "site:wikipedia.org"，就把检索范围限制在了维基百科网内。保留在多个网站中重复出现的网页有助于搜索引擎返回正确的检索结果。

增加搜索结果多样性的另一种方法就是进行检索后过滤。动态排名方法出现后，多数商用搜索引擎都通过过滤检索结果的方式来降低冗余。例如，在 Google SOAP API[⊖]（现已不用）的技术文档中描述了一种简单的检索后过滤算法。在检索之后，可采用两种方法过滤检索结果：

1）如果几个结果有完全相同的标题或内容片断，则只保留其中之一。

2）只为一个 Web 网站保留两个最好的检索结果。

API 将第二种过滤方法称为 "主机密集"。十分有趣的是，第一种过滤方法为了使检索结果**看起来**（appearance）没那么冗余，有可能将那些实际上并不冗余的相关结果过滤掉了。

15.5 评价 Web 搜索

理论上在 Web 搜索中采用传统信息检索的评价方法（例如，P@10 和 MAP）是可行的。但是，Web 上可用资源的数量引起了一些问题。例如，一个信息查询可能会产生成百上千的相关文档。通常这个查询返回的前 10 个检索结果是相关的。这种情况下，传统的评价方法中的二元相关评价（"相关"/"不相关"）可能就不够了，不能得到有意义的评价；相比之下，分级相关评价可能更加合适。

如果分级相关评价的值是有效的，可以采用像 nDCG（12.5.1 节）这样的评价方法，利用这些值对系统进行评价（Richardson 等人，2006）。例如，Najork（2007）在 Web 搜索评价中使用 nDCG，对来自 Windows Live 搜索引擎的超过 28 000 个查询结果进行了评价。从中获得了大约 50 万个人工判定结果。这些判定都采用 6 级分级制度进行分级如下：精确、很好、较好、一般、差、极差。

⊖ code.google.com/apis/soapsearch/reference.html（2009 年 12 月 23 日访问）

Web 的特性也给评价带来了新的影响。正如 15.2 节中提到的，很多 Web 查询都是导航式的。对于这类查询而言，只有一个特定的网页是相关的。15.5.1 节中将介绍用于这类查询的评价方法。此外，商用 Web 搜索服务处理的查询和用户数量也为根据用户行为来进行相关性评判提供了契机，可用于扩展或替代人工评价。15.5.2 节将从评价角度出发，概要介绍用于理解点击及类似的用户行为的相关技术。

15.5.1 指定页面发现

指定页面发现（named page finding）任务是一种 Web 评价任务，该任务假设用户要检索某个她以前见过的或听说过的网页。用户输入一个描述网页内容的查询，然后希望这个网页能出现在搜索结果的第一位（或只返回这个页面）。例如，查询〈“Apollo”，“11”，“mission”〉描述有关第一次登月的美国宇航局（NASA）的历史网页。尽管其他网页也可能和此主题相关，但是只有这个网页被认为是正确的检索结果。2002～2004 年间的 TREC Web 专题以及 2005 年和 2006 年 TREC TB 专题都包括了指定页面发现任务（Hawking 和 Craswell，2001；Craswell 和 Hawking，2004；Clarke 等人，2005；Büttcher 等人，2006）。

指定页面发现背后的前提代表了导航查询的一个重要子集。尽管如此，很多导航查询（如〈“UPS”〉）更倾向于搜索某些具体的网站，而非具体的内容。为解决这类问题，指定页面发现的应用范围应该扩展到“主页发现”查询（Craswell 和 Hawking，2004）。

在 2006 年的 TB 专题检索中，参与者创建了 181 个主题。对于每个主题，创建者都从 GOV2 文档集中找出了对应的答案网页。每次提交都包含超过 1000 个搜索结果。在评价运行结果时，使用 Bernstein 和 Zobel（2005）的 DECO 算法将近似重复网页检测出来，该算法是 15.6.3 节中提到的近似重复网页检测算法的变形。所有与正确网页近似的网页也可看做是正确的。这里有 3 个指标可用于评价：

- MRR：第一个正确结果的**平均排名倒数**（mean reciprocal rank）。
- %前 10 名：正确结果在检索结果排名前 10 的查询占所有查询的百分比。
- %未找到：在检索结果排名前 1000 的网页中都没有出现正确结果的查询占所有查询的百分比。

对于给定主题，排名倒数是指正确网页第一次出现位置（排名）的倒数。如果排第 1，则排名倒数为 1；如果排第 5，则排名倒数就是 1/5。如果正确网页在检索结果列表中没有出现，则它的排名倒数可表示为：$1/\infty=0$。平均排名倒数就是所有主题检索结果排名倒数的平均值。

Büttcher 等人（2006）提供了全部的检索结果。从第三部分中选出几个信息检索方法，应用于指定页面发现，表 15-6 列出这些结果。为了评价这些任务，改用 BM25F 替代 BM25。为便于比较，表中还列出了该检索会议中最好的检索结果（Metzler 等人，2006）。该检索方法包含了大量的针对 Web 的技术，如链接分析（静态排名）和锚文本。第三部分中的标准检索方法和 Metzler 的检索方法的差异说明了这些技术对导航查询任务的重要性。

表 15-6 第三部分的信息检索方法用于指定页面发现的结果。为便于比较，其中包含了 TREC 2006（Metzler 等人，2006）中最好的检索结果

方法	MRR	%前 10	%未找到
BM25（Ch. 8）	0.348	50.8	16.0
BM25F（Ch. 8）	0.421	58.6	16.6

（续）

方法	MRR	前 10	未找到
LMD（Ch. 9）	0.298	48.1	15.6
DFR（Ch. 9）	0.306	46.9	19.9
Metzler 等人（2006）	0.512	69.6	23.8

15.5.2 用户隐式反馈

隐式反馈是指在用户与搜索引擎进行交互时附带获得的反馈。点击就是这种隐式反馈中重要且有效的一种。每当用户输入一个查询并点击其中一个检索结果的链接时，这个点击的记录就会通过浏览器传送给搜索引擎，搜索引擎将其写入日志，以便将来分析。

对给定查询，将很多用户的点击画成一条点击曲线（clickthrough curve），以反映用户对此查询的点击模式。图 15-5 给出一个例子，分别是导航查询和信息查询的典型点击曲线。在这两条曲线中，点击数都随着排名的降低而减少。如果将点击视为是部分用户的正向偏好，这些曲线的形状正如我们所期望的：排名高的结果更像是相关的，被点击的次数也越多。就算被下来的结果也具有同样的相关性，但我们还是希望排名越高的结果可以被点击的次数越多。如果用户顺序地查看结果，那么排名高的页面会比排名低的先被查看。这被称为用户行为中的信任偏见（trust bias）（Joachims 等人，2005）：用户期待搜索引擎先返回最好的检索结果，用户会先点击排名高的网页，即使其中不一定包含他想要的信息。

图 15-9 展示了第三种点击曲线，其来源与前两种曲线一样（Clarke 等人，2007）。该图中的曲线是针对信息/交易查询（"kids"，"online"，"games"）的点击而绘制的。图中出现了一定数量的**点击逆转**（clickthrough inversion），即有个别的网页获得的点击数要比排名紧跟其后的网站还要少（如，排名第 2 和第 3 的两个网页，以及排名第 7 和第 8 的两个网页）。

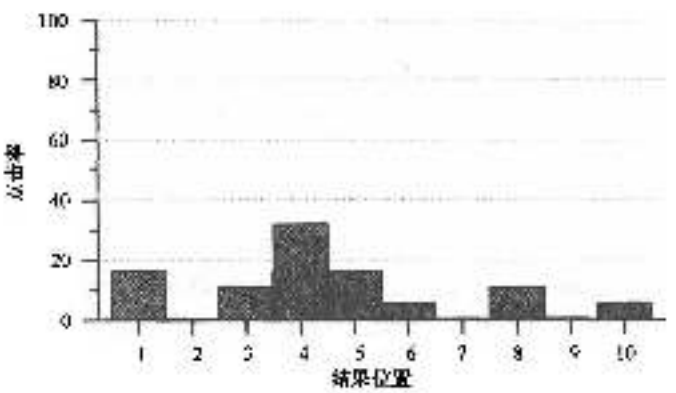


图 15-9 查询（"kids"，"online"，"games"）的点击曲线。点击逆转（例如，排名第 2 和第 3 的网页）表示搜索结果存在次优排名

点击逆转可能预示着存在着次优排名，即相关性低的文档比相关性高的文档排名靠前。然而，这种现象的出现也许是因为与相关性无关的因素。例如，排名高的结果的标题及片段内容或许没有准确地描述用户希望的网页（Clarke 等人，2007；Dupret 等人，2007）。如果在阅读完网页标题和片段之后，用户仍不明白为何该结果是相关的，那么用户就会忽略这个结果并转到其他结果。尽管如此，当标题和片段能够准确描述结果时，点击逆转就可以解释为用户偏好于排名靠后的结果（Joachims 和 Radlinski，2007）。

查询丢弃（abandonment）——输入查询后并不点击任何检索结果——表示用户对实质性的内容并不满意（Joachims 和 Radlinski，2007）。如果大量用户都丢弃给定的查询，这种行为就意味着排名高的检索结果全都是不相关的。某些情况下，用户添加、修改或更正了查

询词项后发生了查询丢弃。有时搜索服务可以过浏览器 cookies 或其他类似的机制来跟踪到这样的查询修改。[⊖]

通过检查大量用户的查询修改，就可以进行拼写错误纠正、扩展词项建议和识别缩写（Cucerzan 和 Brill，2004；Jones 等人，2006）。例如，用户输入查询〈“brittany”，“spears”〉，可能会被更正为〈“britney”，“spears”〉。[⊜]在查询修改后的点击意味着页面与原始查询相关。例如，Joachims 和 Radlinski（2007）指出用户经常输入查询〈“oed”〉，然后被修改为〈“oxford”，“english”，“dictionary”〉，之后用户会点击返回的搜索结果的第一项。

捕捉其他形式的用户隐式反馈需要用户的合作，这需要安装浏览器工具条或类似应用程序，然后显式地同意进行这种捕捉（Kellar 等人，2007）。页面**驻留时间**（dwell time）就是通过此种方法获得的隐式反馈之一。如果用户点击一个检索结果后又立即返回到搜索结果页面，则可能说明该结果是不相关的。

总而言之，这些隐式反馈及其他的反馈方式有助于 Web 搜索结果的质量取得实质性的提升，因为这些反馈便于我们进行评价（Agichtein 等人，2006b），同时也提供更多的排名特征（Agichtein 等人，2006a）。

15.6 Web 爬虫

网络爬虫的功能类似于一个用户在访问 Web，不过规模要大得多。就像用户跟着链接访问一个又一个的页面一样，爬虫依次下载页面，从中提取链接，然后跟随这些新链接继续下载。Web 爬虫技术每次发展时遇到的挑战都与网络规模和爬行速度有关，这是必然会发生的。

假设我们的目标是在一周时间内下载 80 亿个网页快照（商用搜索引擎规定的标准的较小的网页快照）。假设网页的平均大小为 64 KB，则我们必须以一个稳定的速率：

$$64\ \text{KB/page} \cdot 8\ \text{giga-pages/week} = 512\ \text{TB/week}$$
$$= 888\ \text{MB/second}$$

来下载数据为达到如此之大的下载速率，网络爬虫必须同时从多个网站下载页面，因为每个网站对每个单独的下载请求的响应时间就需要几秒钟。在下载页面的同时，爬虫还要监控自身的工作流程，以便避免重复下载页面以及对下载失败的页面进行重新下载。

爬虫还要注意保证其工作不会干扰网站的正常运行。同步下载需要在整个 Web 范围内进行，在访问独立网站时还要注意时间间隔，以避免该网站超载。同时网络爬虫需要尊重其访问的网站，按照网站的规定进行操作，避免抓取某些特定的页面的链接。

许多网页都定期进行更新。因此一段时间后爬虫需要重新抓取这些网页，否则搜索引擎的索引就会是过期的。为避免这一情况，我们也许每周都要对整个 Web 重新抓取，但是这种方法浪费了大量资源，而且也不符合 Web 搜索的实际需求。包含一些“受欢迎的”或“重要的”信息的网页可能会定期更新，因此需要更频繁的重新抓取，每小时或每天一次。其他很少更新或包含极少有价值信息的网页不需要频繁的重访问，每周一次或每个月一次就足够了。

为管理抓取和重抓取的顺序和频率，爬虫需要为可见 URL 维护一个优先级队列。优先级队列中 URL 的排名应该可以反映出网页的多种指标，包括它们的相关重要性和更新频率。

⊖ www.w3.org/Protocols/rfc2109/rfc2109

⊜ labs.google.com/britney.html（2009 年 12 月 23 日访问）

15.6.1 爬虫的组成

本节将通过在爬虫的爬取过程中，跟踪一个 URL（en.wikipedia.org/wiki/William_Shakespeare），来认识爬虫工作的每一个步骤和它的组成部分。跟踪一个 URL 可使我们的讲解更加简单易懂，但需要强调的是，为了保证 Web 搜索引擎抓取速度，这些步骤需要同时对大量的 URL 进行。

图 15-10 简单画出了爬虫的处理步骤和组成部分。当从优先级队列顶端移出一个 URL 时，抓取过程开始，当该 URL 与其他链接一同从某个被访问页面中又回到优先级队列时，抓取过程结束。尽管各个爬虫的具体实现细节不同，但是所有的爬虫都包含这些工作步骤，只是形式不同而已。例如，使用一个线程对一个 URL 执行所有的步骤，然后成千上万个线程并发工作；或者，URL 被分成不同批来处理，每批中的 URL 同时执行每一步，执行结束才能进行下一步。

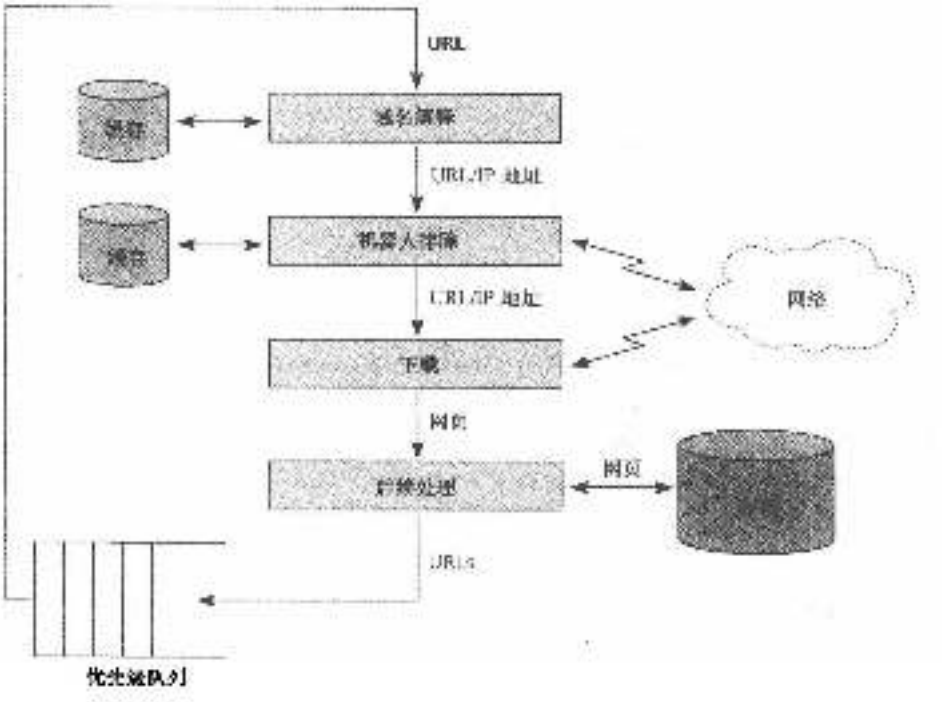


图 15-10 网络爬虫的组成

正如在支持的搜索引擎那样（14.1 节），大规模 Web 爬虫必须在多台机器上进行分布式处理以满足所需的下载速率。通过为每台机器分配 URL 子集可以达到分布式处理的目的，本质上讲就是每台机器负责一部分 Web。构造这些子集的依据可以是主机名、IP 地址或者其他因素（Chung 和 Clarke，2002），例如，我们可能指定某台机器负责来自某个特定主机的网页。优先级队列则成集中到一台机器上，以便于把 URL 分配给其他机器抓取，或者每台机器为其分配到的 URL 子集单独维护一个优先级队列。

1. 域名解析

对 URL 的处理开始于将其主机名翻译成 32 位的 IP 地址。对于主机 en.wikipedia.org，对应的 IP 地址为 208.80.152.2（撰写本书时的 IP）。这个地址将用于获取主机上的网页。

由于该项工作是为了浏览器及其他网络应用程序而完成的，因此这种解析是受域名系统（Domain Name System，DNS）影响的，这是一种标准的 Internet 服务，通过分布在 Internet 中的分布式层次化的服务器和缓存来实现。虽然 DNS 的速度对于多数应用来讲是可接受的，但却远远不能满足 Web 爬虫的要求，因为爬虫要求每秒钟获得上千个域名解析。为满足这一解析速度，爬虫必须有自己的解析缓存。该缓存需要管理主机名和 IP 地址之间的映射，并在它们过期时将其标记为失效或对其刷新。虽然定制 DNS 缓存只是爬虫的一个很小的组成部分，但对该缓存的需求可以反映出爬虫在高速工作时是会遇到问题的。

2. 机器人排除协议

获得 IP 地址后，爬虫还必须检查所访问的网页是否得到所属网站的许可。这种许可就是一个非官方（但已被大家所接受）的协议，即**机器人排除协议**（robots exclusion protocol）（非正式时也叫做“robots.txt 协议”）。对一个给定网站，这个访问许可协议可以通过在主机名后添加路径“/robots.txt”获得，并由此下载对应的页面。

在我们的例子中，URL 为 http://en.wikipedia.org/robots.txt。图 15-11 中展示了这个页面的一小部分内容。该网页由一系列的注释和操作指令构成。“User-agent”指令列出允许在这个网站内进行操作的爬虫的名字，其中“*”表明该指令适用于所有爬虫。例如，“Disallow”指令表示拒绝访问该前缀的所有网页。下载任何带有这样前缀的网页都是不允许的。图 15-11 中的示例表示拒绝 wget 爬虫的所有访问，并要求其他网络爬虫禁止访问随机文章链接和搜索链接，因为这两种链接都是动态生成内容的。有关机器人排除协议的相关细节请参见网址 Web Robots site[⊖]。

```
#
# robots.txt for http://www.wikipedia.org/ and friends
#
...

#
# Sorry, wget in its recursive mode is a frequent problem.
#
User-agent: wget
Disallow: /
...

#
# Friendly, low-speed bots are welcome viewing article pages, but not
# dynamically-generated pages please.
#
User-agent: *
Disallow: /wiki/Special:Random
Disallow: /wiki/Special:Search
...
```

图 15-11 摘自 robots.txt 的内容

Web 爬虫通常对其访问过的每个主机的 robots.txt 进行缓存，以避免对来自该主机的 URL 进行操作时造成重复下载。这些缓存信息会很快过期，可能几小时或几天。

当爬虫被拒绝访问时，URL 将被送回优先级队列中，并标记这个下载是不允许的。过段时间爬虫可能会重试这个 URL，看是否可以访问；或者，这个 URL 被永久标记为拒绝访问的，再也不会进行重试，并将其作为一个占位符维护起来，以防爬虫日后对其再访问。

3. 下载

确认下载是被允许的以后，爬虫通过 HTTP 协议对 Web 网站进行访问并下载网页。网页格式可能是 HTML（如图 8-1 所示）或其他格式，如 PDF。相关网页可能同时被访问和下载，或者将其预留给将来的抓取循环。例如，如果网页定义了一个框架集，那么集合中的所有网页就可能被同时下载。如果支持图片搜索服务，也会对图片进行下载。

在下载期间，爬虫也许还要解决重定向的问题，这表明网页的实际内容在其他地方。重定向会大大增加下载处理的复杂度，部分是因为重定向有多种实现方式。在 HTTP 协议层，

⊖ www.robotstxt.org

有多种 Web 服务器响应消息可以指示网页已被永久地或临时地移至其他位置了。重定向也可通过 HTML 网页中的标签来说明。最后，跟 HTML 页面一同加载的 JavaScript 也在执行时引起重定向。

例如，维基百科网站采用 HTTP 协议“301 Moved Permanently”来将 en.wikipedia.org/wiki/william_shakespeare 重定向到 en.wikipedia.org/wiki/William_Shakespeare。在维基百科中重定向到正确的拼写和大小写上也是很常见的。在进行网站的重新设计或重新构建时，其他网站也会利用 HTTP 重定向来使得旧的 URL 可以继续使用。

使用 JavaScript 而产生的重定向为 Web 爬虫带来的麻烦最大，因为爬虫要通过执行 JavaScript 后才能知道重定向的目标。此外，根据不同的因素（例如用户浏览器类型等），JavaScript 还有可能重定向到不同的网页。为考虑到全部的可能性，理论上要求爬虫在不同配置环境下不停地重复执行这些 JavaScript，直到找出所有的重定向目标为止。

考虑到可用资源有限，对于爬虫而言，为上百万个下载网页执行 JavaScript 是不太可能的。相反，爬虫可以尝试进行部分评价和其他启发式策略，这已足以在很多情况下确定重定向的目标。另一方面，如果爬虫忽略 JavaScript 重定向，那么剩下的有用信息就没多少了，甚至可能完全没有，这表明执行 JavaScript 的重定向是必须的。

4. 后续处理

下载完成后，爬虫将页面存档以便搜索引擎进行索引。在存档中可能还保留着这些页面的旧副本，以使爬虫估计这些网页的更新周期和变化的类型。如果下载失败，可能由于网站暂时无法访问，这时还是需要索引旧的副本。下载失败后，URL 可能会返回到优先级队列并在一段时间后重试。如果几天内的若干次下载重试都失败，那么该页面的旧副本也将被标记为过期，并将该页面从索引中移除。

此外，在存档过程中，为提取 URL，网页会被进行分析或**分解**（scrape）。正如浏览器一样，爬虫通过解析 HTML 来定位锚标签及其他包含链接的元素。索引要用到的锚文本和其他信息也同时被提取出来，然后进行存档以方便搜索引擎进行操作。在这次分析中，与下载页面重复（或近似重复）的页面也会被检测出来（见 15.6.3 节）。

在后续处理过程中，爬虫要遵守网站的要求，不索引某个页面或跟随某个链接。如果标签

```
<meta name = "robots" content = "noindex">
```

出现在页面头部（〈header〉——译者注），就表示搜索引擎不得索引该网页。如果在锚标签中出现“rel=nofollow”属性，就表示爬虫要忽略该链接。例如，在写阶段，下面的外部链接出现在维基百科里有关莎士比亚的网页中：

```
<a href = "http://www.opensourceshakespeare.org" rel = "nofollow">
    Open Source Shakespeare
</a>
```

只有在其他地方发现该页的链接时，网络爬虫才能对其进行抓取，而且这个链接不对排名产生任何影响。像博客或维基百科这样的网站都允许用户创建外部链接，而在用户创建这些链接时，网站会自动给这些外部链接添加“rel=nofollow”。这一策略可有效阻止创建不合适的链接，这种链接只以增加目标网页的 PageRank 值为目标。通过添加“rel=nofollow”，用户产生这样的垃圾链接是得不到什么好处的。

由于执行 JavaScript 是在下载阶段完成的，所以在后续处理阶段会造成问题。当执行该 JavaScript 后，它可能会以新的内容和链接完全重写该网页。如果该执行操作失败，爬虫可能会采取启发式的方法去提取 URL 和其他信息，但这不能保证提取成功。

5. 优先级队列

在后续处理中提取出来的URL会插入到优先队列中。如果队列中已有该URL，后续处理中抓取到的信息可能会改变它的存储位置。

实现这个优先级队列是一个难题。假设URL的平均长度为64字节，一个包含80亿个网页的优先级队列就需要将近0.5 TB的空间来存储URL（不考虑进行压缩）。这些存储需求，再加上每秒数以百万计的更新需求，就限制了可用于管理优先级队列的策略。在15.6.2节中将讨论这些管理策略，并忽略实现问题。然而，开发一个实用的爬虫时，这些实现问题是不能忽略的。

15.6.2 抓取顺序

假设我们第一次对Web中的内容进行抓取，目的是抓取并维护数十亿网页的快照。我们对Web中的网站和网页一无所知，这将留待我们对其进行发掘。一开始，我们可能先建立一个小的**种子集**（seed set），其中收录一些众所周知的主要门户网、零售网和新闻网的URL。可以用来自ODP[⊖]或其他Web目录的网页链接建立一个种子集。如果我们采用广度优先的原则，那我们将在抓取早期得到很多高质量的网页（Najork和Wiener，2001）。

随着抓取工作的不断进行，我们对Web的了解也逐渐增多。从某种程度上讲，特别是爬虫在为一个实际的搜索引擎服务时，重新访问以前的网页与爬取新的URL是一样重要的。否则搜索引擎的索引就会过期。更进一步讲，随着爬虫不断地扩展和更新Web的快照，抓取过程是**增量式的**（incrementally）。随着访问和重新访问网页，爬虫发现新的URL，并剔除不用的网页。

爬虫的活动也取决于它的**刷新策略**（refresh policy）（Olston和Pandey，2008；Pandey和Olston，2008；Cho和Garcia-Molina，2000，2003；Wolf等人，2002；Edwards等人，2001）。该策略包含两个方面：（1）网页变化的频率和性质；（2）相比于抓取新的URL所带来的影响，这些网页的变化对搜索结果所造成的影响。最简单的刷新策略就是定期对全部网页进行重新访问，每X星期进行一次，同时按照广度优先策略继续爬取新的URL。尽管这种策略适用于不频繁更新的网页，但是对那些更新频繁且影响较大的网页（如www.cnn.com），应该进行更频繁的重访问。此外，网络爬虫应该优先抓取那些可能对搜索结果影响较大的新的URL。例如，通过处理已有查询日志和分析用户行为（如点击数据）来估计这种可能。

Web在持续改变中。Ntoulas等人（2004）在一年内跟踪了154个网站的变化，并得出每周新页面出现率约为8%，新链接的出现率为25%。删除率也很高。一年以后，仅有20%的初始网页仍在沿用。然而，一旦建立，多数网页在删除前的变化是很小的。即使过了一年，不到一半的网页在删除前改变不超过5%，这是使用TF-IDF进行评价基准得到的结果。当这些网页确实发生改变时，Cho和Garcia-Molina（2003）证明了这些网页的变化频率可用泊松分布来描述。因此，如果我们知道一个网页的历史，就可以预言它将来的变化频率。

即使一个网页频繁地被更新，重访率还是取决于网页性质的变化。Olston和Pandey（2008）发现网页中的不同部分的更新特点也不同，而且在对该页面建立刷新策略时就应考虑这些不同特点。网页的某些部分可能保持静态，而其他部分则显示出**波动行为**（churning behavior）。例如，网页中的广告在每次访问时都会变化，或网页中的一些内容是每天变化的，如“今日名言”，其他部分则保持不变。如博客或论坛这些网站会出现**滚动行为**（scrolling behavior），新的

⊖ www.dmoz.org

内容出现后会把旧的内容推向页面的后端。有些页面，如新闻网的主页，可能每小时都会滚动播出一些新闻内容。

重访率和未访问的 URL 排名都要由其对搜索结果的影响来决定。Pandey 和 Olston (2008) 将这种影响定义为：网页出现或即将出现在搜索结果中排名高的位置的次数。他们的研究意味着，当选择访问一个新的 URL 时，应该优先访问那些能提高某个或多个查询的检索结果的 URL。类似的，在确定重访率时我们也应该考虑这种影响。对新闻网的访问可以每小时一次甚至更频繁，因为好的新闻网站都是经常更新的而且影响较大。一个搜索关于爆炸性新闻的用户想获得最新的消息。另一方面，对于那种“今日名言”类的网页，如果其他网页保持不变，且返回结果只依赖于静态内容，那么就不需要对该网页每天都进行一次访问。

影响和静态排名相关。静态排名高的网页通常也是访问频率高的网页。但是，影响不等于静态排名 (Pandey 和 Olston，2008)。例如，一个网页的静态排名高是因为在同一个网站中指向它的网页具有比它更高的静态排名。但是，如果在搜索结果中它总是比同一网站的其他页面排名要低，那么这个页面的更新或改变也不造成什么影响。另一方面，一个网页因为主题相对难懂而导致静态排名较低，只有很少数的用户对其感兴趣。但如果缺少了该网页，对这些用户的搜索结果造成的影响是很大的。

15.6.3 重复与近似重复

大约有 30%～40% 的网页是与其他网页完全重复的，约 2% 的网页是近似重复的 (Henzinger，2006)。这些数据反映了搜索引擎的一个主要问题：由于保留不必要的重复网页增加了存储代价和处理开销。更重要的是，重复和近似重复的网页会影响新颖性，致使搜索结果不尽如人意。尽管主机集群或类似检索后过滤这些方法可以改善这个问题，但仍然损失了性能代价，而且近似重复的页面仍有可能会逃过过滤。

检测完全重复的网页相对来说比较简单。整个网页（包括标签和脚本）都可用哈希函数来表达。得到的整数哈希值可以拿来同其他网页的哈希值进行比较。MD5 算法 (Rivest，1992) 提供了一种检测完全重复的网页的可能的哈希函数。给定一个字符串，MD5 为其生成一个 128 位的“信息摘要”。MD5 算法通常用来验证数据传输以后文件的完整性，提供一种确定整个文件是否被正确传输的简单方法。由于其信息摘要长度为 128 位，所以发生数据冲突的可能性是很小的（练习 15.11 所述情况例外）。

通过计算哈希值足够检测出一般情况下造成的完全重复的网页，尽管只有每个字节都相同的情况下才能测出重复。网页在多个网站中都有镜像，如 Java 文档，就可以用这种方法识别。许多网站会为每个无效 URL 返回同样的“Not Found”页面。有些 URL 会包含用户或会话 id，无论它的值是什么，表述的内容都是一样的。理想情况下，在抓取阶段就应检测出完全重复。一旦检测到镜像网页，那它所指向的其他镜像网页就可以忽略了。

在用户看来，即使两个网页不是每个字节都相同，但用户也会认为它们是重复的，这是因为这两个网页包含了一些本质上相同的内容。为检测这些近似重复的网页，网页可被规范化为一个词条数据流，像我们在索引时所做的一样，将其内容简化成一个标准形式。这种规范化通常包括删除标签和脚本。此外我们会删除标点、大写单词和多余的空格。例如，图 8-1 中所示的 HTM 文档可被规范化为：

> william shakespeare wikipedia the free encyclopedia william shakespeare william shakespeare baptised 26 April 1564 died 23 April 1616 was an english poet and playwright he is widely regarded as the...

在规范化后，可采用一个哈希函数对剩余内容的重复网页进行检测。

但是，近似重复的网页通常会添加一点内容或做微小的变化，就不能通过整个页面的哈希值检测出来了。菜单、标题和其他模板可能会不同。当一个网站从另外一个网站复制内容时，通常会对副本的内容进行编辑。检测近似重复需要我们通过比较字符串来比较文档。例如，一篇新闻专线中的文章，当它也出现在其他网站时，它们之间会包含大量重复的子字符串（新闻故事），但是这些网页的其他部分则有很大差异。

衡量一个网页与另外一个网页的相似程度，可以通过比较它们共有的子字符串来实现。假设一条新闻出现在了两个网站中；规范化后，第一个网站中的页面大小为 24 KB，第二个网站中的页面大小为 32 KB。如果新闻本身的大小为 16 KB，则可根据重复内容计算出如下两个网页的相似度：

$$\frac{16\ \text{KB}}{24\ \text{KB}+32\ \text{KB}-16\ \text{KB}}=40\%$$

因此，这两个页面的内容相似度为 40%。

Broder 等人（1997）提出了一种进行相似度计算的方法，而且该方法可用于检测 Web 中近似重复的网页。在他们的方法中，将从规范化页面和比较页面中提取出来的子字符串定义为 shingle，并通过比较这些 shingle 的重叠程度来衡量。下面我们简单介绍一下该方法；更详细的介绍参见 Broder 等人（1997）发表的论文及其他文章（Henzinger，2006；Bernstein 和 Zobel，2005；Charikar，2002）。

提取自某个网页的长度为 w 的 shingle（w-shingle），由出现在该文档中所有长度为 w 的子字符串的词条所组成。例如，考虑以下出自《Hamlet》的三行文字：

#1：To be，or not to be：that is the question

#2：To sleep：perchance to dream：ay，there's the rub

#3：To be or not to be，ay there's the point

前两行出自本书一直使用的标准版本的莎士比亚文集。最后一行可能出自该戏剧的一个“盗版”版本——出自某个小演员的记忆。[⊖]

规范化后，每行得到如下的 2-shingle，其中每一个 shingle 都是按字母排序的，且已去除冗余：

#1：be or，be that，is the，not to，or not，that is，the question，to be

#2：ay there，dream ay，perchance to，s the，sleep perchance，the rub，there s，to dream，to sleep

#3：ay there，be ay，be or，not to，or not，s the，the point，there s，to be

对于这个例子，我们令 $w=2$。对于 Web 网页，$w=11$ 可能是更合适的选择（Broder 等人，1997）。

给定两个 shingle 集合 A 和 B，它们的**相似性**（resemblance）取决于它们共有的 shingle 数量：

$$\frac{|A\cap B|}{|A\cup B|} \tag{15-30}$$

相似性的取值范围在 0～1 之间，表示集合 A 与 B 所包含内容的重复程度。为简化相似性的计算，我们计算每个 shingle 的哈希值；对于这个目的来说，通常 64 位的哈希值已经足

⊖ internetshakespeare.uvic.ca/Library/SLT/literature/texts+1.html（2009 年 12 月 23 日访问）

够了（Henzinger，2006）。虽然 64 位哈希值的范围很小，而 Web 似乎包含很多不同的 shingle，导致计算得出相同的哈希值，但除非这些页面包含重复的内容，否则它们不可能包含多个匹配的 shingle。为达到示例中的目标，我们任意假设一个 8 位的哈希值，给定如下 shingle 集：

#1：43，14，109，204，26，108，154，172

#2：132，251，223，16，201，118，93，197，217

#3：132，110，43，204，26，16，207，93，172

网页中可能包含大量的 shingle。减少 shingle 数量的方法之一就是删除那些模 m 后不为 0 的 shingle，其中 $m=25$ 适用于 Web 数据（Broder 等人，1997）。另一种方法就是保留 s 个值最小的 shingle。通过对比剩余的 shingle，可以估计出相似性。对本例来说，我们采用了第一种方法，其中 $m=2$，最后得出偶数值的 shingle 如下：

#1：14，204，26，108，154，172

#2：132，16，118

#3：132，110，204，26，16，172

在这个例子中 shingle 集合之间进行逐对比较是容易的。然而，对数以十亿计的 Web 文档而言，逐对比较是不可行的。取而代之，我们构建以下元组：

〈shingle 值，文档 id〉

然后根据 shingle 值对这些元组进行排序（本质上是建立倒排索引）。根据本例可构建如下元组：

〈14，1〉，〈16，2〉，〈16，3〉，〈26，1〉，〈26，3〉，〈108，1〉，〈110，3〉，〈118，2〉，〈132，2〉，〈132，3〉，〈154，1〉，〈172，1〉，〈172，3〉，〈204，1〉，〈204，3〉

下一步就是合并 shingle 值相同的元组，并将其整理成如下形式：

〈id1，id2〉，

其中，每一对元组都表示两个文档共用一个 shingle。构建这些元组时，将 shingle 值较小的文档排在前面。现在，可以忽略 shingle 的真实值，因为每个元组中的文档都有相同的 shingle 值。本例中可得如下几对：

〈2，3〉，〈1，3〉，〈2，3〉，〈1，3〉，〈1，3〉

排序并统计，写成如下三元组的形式：

〈id1，id2，count〉

对于本例，这些三元组是：

〈1，3，3〉，〈2，3，2〉

从这些三元组中可估计出 #1 和 #3 之间的相似度为

$$\frac{3}{6+6-3}=\frac{1}{3}$$

#2 和 #3 之间的相似度为

$$\frac{2}{3+6-2}=\frac{2}{7}$$

#1 和 #2 之间的相似度为 0。

15.7　总结

尽管本章较长且包含了大量的内容，但都是在重复强调以下三个主要问题：规模、结构和用户。

- Web 中包含大量的关于各种主题的且使用不同语言的素材，而且它们还在不断地增长和变化。Web 的庞大规模暗示着许多查询的结果可能是大量的相关网页。在这种情况下，新颖性和质量等因素就成了排名中要考虑的重要因素。对于不同用户而言，同样的查询可能有不同的含义。为数以百万计的网站维护精确的、恰当的索引需要大量的计划和资源。
- HTML 标签和链接的结构为排名提供了重要特征。链接分析方法已被看做是分析页面相关质量的一种扩展方法了。在网页标题和锚文本中出现的词项的权重可作为调整网页状态的一个指标。如果没有链接，Web 的抓取工作几乎是不可能的。
- 用户的行为驱动了 Web 检索的进步。搜索引擎必须注意用户查询的理解范围和目的。信息查询比导航查询需要更多不同的结果，且对搜索引擎的评价也需要根据它们在这两种查询类型上的性能进行。可分析点击和搜索引擎日志中的其他用户隐式反馈，以评价并提高搜索引擎的性能。

15.8 延伸阅读

Web 搜索引擎是一个神秘而复杂的工具，它由很多无私奉献的工程师和开发者团队对其不断进行调整和提高。本章中我们仅仅介绍了 Web 搜索中该技术的一小部分知识。

现在 Web 搜索和 Web 广告的商业重要性已得到了足够的重视，许多大型搜索引擎运营商和搜索引擎优化（SEO）公司快速成长起来。这些 SEO 建议网站所有者如何通过合法的或（偶尔）非法的手段在主流搜索引擎中获得更高的排名。这些组织中的工作者经常在博客中记录他们的工作，出现在这些博客中的技术珍闻都是有趣的且值得我们阅读的。一个好的切入点是“搜索引擎圆桌会议”（Search Engine Roundtable[㊀]）。Matt Cutts（Google 的 Web 垃圾信息小组的组长）的个人博客是另外一个不错的切入点。[㊁]

20 世纪 90 年代早期，就在 Web 刚刚兴起不久，出现了第一个 Web 搜索引擎。到 20 世纪 90 年代中期，商用搜索引擎诞生了，如 Excite、Lycos、Altavista 和 Yahoo!，它们每天要处理数以百万计的查询。这些早期搜索引擎的历史可以参见搜索引擎观察（Search Engine Watch），该网站跟踪了自 1996 年以来的搜索引擎在商业方面的技术发展情况。[㊂]根据在第三部分提到的基于内容的特征，这些早期的搜索引擎使用到一些简单的基于链接特征的技术——例如，用网页入度来反映它的受欢迎程度（Marchiori，1997）。

15.8.1 链接分析

20 世纪 90 年代末期，静态排名和链接分析的重要性得到了广泛认可。1997 年，Marchiori 提出了链接分析技术，这项技术就是 PageRank 的前身。同年，Carriere 和 Kazman（1997）给出了一种分析 Web 网站间的链接关系的工具。

1998 年 4 月 15 日，在澳大利亚布里斯班（Brisbane）举行的第 7 次世界万维网会议（WWW）上，Brin 和 Page 展示的有关基本 PageRank 算法和早期 google 搜索引擎结构的论文至今仍堪称经典（Brin 和 Page，1998）。在发表这篇论文后不久，他们和他们的同事基于这项工作继续努力，描述了个性化 PageRank 和 Web 数据挖掘的基本算法（Brin 等人，1998；

㊀ www.seroundtable.com

㊁ www.mattcutts.com

㊂ searchenginewatch.com/showPage.html? page=3071951（2009 年 12 月 23 日访问）

Page 等人，1999）。当 Brin 和 Page 在构建 PageRank 算法（和 Google 搜索引擎）的时候，Kleinberg 在独立开发 HITS 算法，并在 1998 年 1 月举办的第 9 届离散算法年会（Annual Symposium on Discrete Algorithm）上公开了这个算法（Kleinberg，1998，1999）。在 Kleinberg 与他人合著的一篇论文中，他将 HITS 算法进行了扩展应用到了锚文本，该论文与 Brin 和 Page 的论文在同一天同一个会议中发表了（Chakrabarti 等人，1998）。Brin、Page 和 Kleinberg 等人的工作给链接分析技术的研究带来了一场革命。

Bharat 和 Henzinger（1998）将 HITS 和内容分析相结合。他们同时还认识到紧耦合网络给 HITS 算法带来的问题，并提出了这些问题的解决方法，后来被 Lempel 和 Moran（2000）发展成 SALSA 算法。根据 SALSA 为某个已知网页（它的“声誉”）确定其主题的思路，Rafiei 和 Mendelzon（2000）对 PageRank 进行了扩展。Davidson（2000）指出某些链接应比其他的链接分配更低的权重，并且训练了一个分类器将与商业相关的链接和单独有价值的链接分开。Cohn 和 Chang（2000）对 HITS 矩阵进行**主成分分析**（Principal Component Analysis，PCA）以从中提取出多个特征向量。还有很多学者提出了用于计算个性化和特定主题的 PageRank 的有效的方法（Haveliwala，2002；Jeh 和 Widom，2003；Chakrabarti，2007）。

Ng 等人（2001a，b）比较了 HITS 和 PageRank 算法的稳定性，证明了 PageRank 算法中跳转向量的作用，并建议 HITS 算法也使用跳转向量。Borodin 等人（2001）对 HITS 和 SALSA 进行了理论分析并为进一步优化提供了建议。Richardson 和 Domingos（2002）提出了一种独立于查询的 PageRank 算法，其中随机浏览用户更倾向于跟随与查询相关的链接对页面进行访问。Kamvar 等人（2003）提出了一种加快 PageRank 计算的方法。

最近，Langville 和 Meyer（2005，2006）对 PageRank 和 HITS 中的数学理论给出了一份详细并通俗易懂的研究报告。Bianchini 等人（2005）进一步研究了 PageRank 的特性。Craswell 等人（2005）讨论将扩展 BM25F 与静态排名结合在一起。Baeza-Yates 等人（2006）总结了跳转向量所起的作用，并用其他的阻尼函数将其取代，围绕 PageRank 构建了一个算法族。Najork 等人（2007）比较了 HITS 和基本 PageRank 的检索效果，在进行比较时，分别单独列出这两个算法的效果以及它们与 BM25F 结合后得到的效果，以说明基本 PageRank 算法的局限性。在 Najork（2007）的相关研究中，他比较了 HITS 算法和 SALSA 算法的检索效果，发现当做为静态排名特征时，SALSA 算法优于 HITS 算法。

Gyöngyi 等人（2004）提出了一种链接分析技术，名为**信任排名**（TrustRank），该分析方法用于检测 Web 垃圾信息。由 Gyöngyi 和 Garcia-Molina（2005）对 Web 垃圾信息问题进行了概括和讨论。除链接分析技术以外，面向内容的技术（如第 10 章中提到的垃圾邮件过滤）也可以用来解决 Web 垃圾信息问题。AIRWeb[⊖] 工作室发起了一个论坛，专门为检测 Web 垃圾信息提供测试和评价，并把 Web 垃圾信息检测看做 Web 中广告信息检索的一个扩展主题。

Golub 和 Van Loan（1996）是矩阵数值方法计算方面的权威。他们用了两个长章节来讨论特征值问题的解决方法，其中包括对乘幂法的深入讨论。

15.8.2 锚文本

在最早期的 Web 搜索引擎中，锚文本作为排名特征而存在（Brin 和 Page，1998）。Craswell

⊖ airweb.cse.lehigh.edu

等人（2001）将锚文本同内容特征进行对比后证明了锚文本的重要性。Robertson 等人（2004）描述了将锚文本结合进概率检索模型。Hawking 等人（2004）提出了一种减小锚文本权重的方法，该方法在锚文本大量重复时对词频进行调整。

15.8.3 隐式反馈

Joachims 和 Radlinski（2007）对解析隐式反馈的方法进行了概述；Kelly 和 Teevan（2003）对早期的研究做了文献综述。Dupret 等人（2007）、Liu 等人（2007）以及 Carterette 和 Jones（2007）都讨论了将点击用于 Web 检索评价中。Agichtein 等人（2006b）通过将多浏览和点击特征结合起来得到相关判断。Qiu 和 Cho（2006）提出一种通过点击得出用户兴趣的方法，由此允许个性化检索结果反映出用户的兴趣。

15.8.4 Web 爬虫

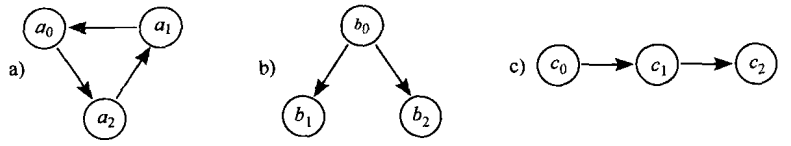
Web 爬虫的基本架构是由 Heydon 和 Najork（1999）给出的。Olston 和 Najork（2010）对该领域进行了一次最新的深入研究。

对日益增加的网络爬虫进行刷新策略的优化这一问题已被多个小组研究过（Edwards 等人，2001；Wolf 等人，2002；Cho 和 Garcia-Molina，2003；Olston 和 Pandey，2008）。Dasgupta 等人（2007）研究了如何均衡访问新的 URL 和重新访问以前爬取的网页。Pandey 和 Olston（2008）研究了新页面对检索结果造成的影响。Chakrabarti 等人（1999）描述了致力于抓取与某个主题相关的网页的定向爬虫。关于 Web 爬虫中优先队列的高效实现的内容很少发布，但是 Yi 等人（2003）的工作为这项研究提供了一个良好的开端。

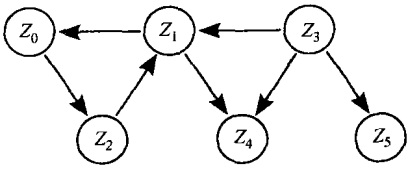
Broder 等人（1997）将 shingle 作为一种检测近似重复网页的方法引入了该领域。Henzinger（2006）通过实验将该方法与 Charikar（2002）的方法进行比较，并提出了一种胜于这两种方法的联合算法。Bernstein 和 Zobel（2005）用 shingle 的一个版本评价了近似重复网页对检索效果的影响。

15.9 练习

练习 15.1 计算下列 Web 图的基本 PageRank。假设 $\delta=3/4$。



练习 15.2 计算下列 Web 图的基本 PageRank。假设 $\delta=0.85$。



练习 15.3 利用公式（15-9）和公式（15-10），证明对于所有 $n\geqslant 0$ 有：

$$\sum_{\alpha\in\Phi} r^{(n)}(\alpha)=N$$

练习 15.4 根据公式（15-12）中的追随矩阵和跳转向量，计算其扩展 PageRank。假设 $\delta=0.85$。

练习 15.5 证明转移矩阵 M'对于任何 Web 图都是非周期的。

练习 15.6 除 15.3.2 节的信息来源以外，请给出其他信息来源，使其能用于建立追随矩阵和跳转向量。

练习 15.7 根据公式（15-24）中的邻接矩阵 W，计算其共被引矩阵 $A=W^{\mathrm{T}}W$ 和耦合矩阵 $H=WW^{\mathrm{T}}$。

练习 15.8 根据公式（15-24）中的邻接矩阵 W，计算其权威向量 $\vec{a}$ 和链接向量 $\vec{h}$。

练习 15.9 Langville 和 Meyer（2005）举出下面的例子来阐明 HITS 算法的收敛性。计算以下邻接矩阵的权威向量 $\vec{a}$，设初始估计为 $\vec{a}^{(0)}=\langle 1/2,\ 1/2,\ 1/2,\ 1/2\rangle^{\mathrm{T}}$。

$$W=\begin{pmatrix}0&0&0&0\\1&0&0&0\\1&0&0&0\\0&1&1&0\end{pmatrix} \tag{15-31}$$

初始估计为 $\vec{a}^{(0)}=\langle 1/\sqrt{3},\ 1/3,\ 1/3,\ 2/3\rangle^{\mathrm{T}}$ 时，重新计算权威向量 $\vec{a}$。

练习 15.10 查看知名网站的 robots.txt 文件，有禁止访问的页面或爬虫吗？为什么？

练习 15.11 我们知道 MD5 算法是不安全的。给定某个字符串的 MD5 值，可以构造出另外一个字符串，且它们的 MD5 值相等。恶意网站会如何利用这一弱点对 Web 搜索引擎进行攻击？查找相关资料并提出可行的解决方案。

练习 15.12 检索词项“Google bombing”。列出搜索引擎对该查询可能的处理方式。

练习 15.13（项目练习） Bharat 和 Broder（1998）提出了一项技术，后来 Gulli 和 Signorini（2005）对其进行了更新，采用这项技术估计可索引 Web 的规模。

练习 15.14（项目练习） 构建一个“爬虫陷阱”，即产生一个动态环境使一个 Web 网站看起来比它实际要大得多（数十亿的网页）。该陷阱需要产生随机页面，包含的随机文本（练习 1.13）中有随机指向陷阱中其他随机页面的链接。陷阱中的 URL 应带有随机数产生器的种子，这样访问页面时才能得到相同的内容。

注意：在运行的网站中部署陷阱会对该网站的检索结果造成不良影响。请慎行。如有必要，请申请许可。如果你部署了陷阱，最好增加一个 robots.txt 入口，引导正常爬虫远离该陷阱。

15.10 参考文献

Agichtein, E., Brill, E., and Dumais, S. (2006a). Improving Web search ranking by incorporating user behavior information. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 19–26. Seattle, Washington.

Agichtein, E., Brill, E., Dumais, S., and Ragno, R. (2006b). Learning user interaction models for predicting Web search result preferences. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–10. Seattle, Washington.

Baeza-Yates, R., Boldi, P., and Castillo, C. (2006). Generalizing PageRank: Damping functions for link-based ranking algorithms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 308–315. Seattle, Washington.

Bernstein, Y., and Zobel, J. (2005). Redundant documents and search effectiveness. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pages 736–743. Bremen, Germany.

Bharat, K., and Broder, A. (1998). A technique for measuring the relative size and overlap of public Web search engines. In *Proceedings of the 7th International World Wide Web Conference*, pages 379–388. Brisbane, Australia.

Bharat, K., and Henzinger, M. R. (1998). Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 104–111. Melbourne, Australia.

Bianchini, M., Gori, M., and Scarselli, F. (2005). Inside PageRank. *ACM Transactions on Internet Technology*, 5(1):92–128.

Borodin, A., Roberts, G. O., Rosenthal, J. S., and Tsaparas, P. (2001). Finding authorities and hubs from link structures on the World Wide Web. In *Proceedings of the 10th International World Wide Web Conference*, pages 415–429. Hong Kong, China.

Brin, S., Motwani, R., Page, L., and Winograd, T. (1998). What can you do with a Web in your pocket? *Data Engineering Bulletin*, 21(2):37–47.

Brin, S., and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th International World Wide Web Conference*, pages 107–117. Brisbane, Australia.

Broder, A. (2002). A taxonomy of Web search. *ACM SIGIR Forum*, 36(2):3–10.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997). Syntactic clustering of the Web. In *Proceedings of the 6th International World Wide Web Conference*, pages 1157–1166. Santa Clara, California.

Büttcher, S., Clarke, C. L. A., and Soboroff, I. (2006). The TREC 2006 Terabyte Track. In *Proceedings of the 15th Text REtrieval Conference*. Gaithersburg, Maryland.

Carrière, J., and Kazman, R. (1997). WebQuery: Searching and visualizing the Web through connectivity. In *Proceedings of the 6th International World Wide Web Conference*, pages 1257–1267.

Carterette, B., and Jones, R. (2007). Evaluating search engines by modeling the relationship between relevance and clicks. In *Proceedings of the 21st Annual Conference on Neural Information Processing Systems*. Vancouver, Canada.

Chakrabarti, S. (2007). Dynamic personalized PageRank in entity-relation graphs. In *Proceedings of the 16th International World Wide Web Conference*. Banff, Canada.

Chakrabarti, S., Dom, B., Raghavan, P., Rajagopalan, S., Gibson, D., and Kleinberg, J. (1998). Automatic resource list compilation by analyzing hyperlink structure and associated text. In *Proceedings of the 7th International World Wide Web Conference*. Brisbane, Australia.

Chakrabarti, S., van den Burg, M., and Dom, B. (1999). Focused crawling: A new approach to topic-specific Web resource discovery. In *Proceedings of the 8th International World Wide Web Conference*, pages 545–562. Toronto, Canada.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388. Montreal, Canada.

Cho, J., and Garcia-Molina, H. (2000). The evolution of the Web and implications for an incremental crawler. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 200–209.

Cho, J., and Garcia-Molina, H. (2003). Effective page refresh policies for Web crawlers. *ACM Transactions on Database Systems*, 28(4):390–426.

Chung, C., and Clarke, C. L. A. (2002). Topic-oriented collaborative crawling. In *Proceedings of the 11th International Conference on Information and Knowledge Management*, pages 34–42. McLean, Virginia.

Clarke, C. L. A., Agichtein, E., Dumais, S., and White, R. W. (2007). The influence of caption features on clickthrough patterns in Web search. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 135–142. Amsterdam, The Netherlands.

Clarke, C. L. A., Scholer, F., and Soboroff, I. (2005). The TREC 2005 Terabyte Track. In *Proceedings of the 14th Text REtrieval Conference*. Gaithersburg, Maryland.

Cohn, D., and Chang, H. (2000). Learning to probabilistically identify authoritative documents. In *Proceedings of the 17th International Conference on Machine Learning*, pages 167–174.

Craswell, N., and Hawking, D. (2004). Overview of the TREC 2004 Web Track. In *Proceedings of the 13th Text REtrieval Conference*. Gaithersburg, Maryland.

Craswell, N., Hawking, D., and Robertson, S. (2001). Effective site finding using link anchor information. In *Proceedings of the 24th Annual International ACM SIGIR Conference on*

Research and Development in Information Retrieval, pages 250–257. New Orleans, Louisiana.

Craswell, N., Robertson, S., Zaragoza, H., and Taylor, M. (2005). Relevance weighting for query independent evidence. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 416–423. Salvador, Brazil.

Cucerzan, S., and Brill, E. (2004). Spelling correction as an iterative process that exploits the collective knowledge of Web users. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 293–300.

Dasgupta, A., Ghosh, A., Kumar, R., Olston, C., Pandey, S., and Tomkins, A. (2007). The discoverability of the Web. In *Proceedings of the 16th International World Wide Web Conference.* Banff Canada

Davidson, B. D. (2000). Recognizing nepotistic links on the Web. In *Proceedings of the AAAI-2000 Workshop on Artificial Intelligence for Web Search*, pages 23–28.

Dupret, G., Murdock, V., and Piwowarski, B. (2007). Web search engine evaluation using clickthrough data and a user model. In *Proceedings of the 16th International World Wide Web Conference Workshop on Query Log Analysis: Social and Technological Challenges.* Banff, Canada.

Edwards, J., McCurley, K., and Tomlin, J. (2001). An adaptive model for optimizing performance of an incremental Web crawler. In *Proceedings of the 10th International World Wide Web Conference*, pages 106–113. Hong Kong, China.

Golub, G. H., and Van Loan, C. F. (1996). *Matrix Computations* (3rd ed.). Baltimore, Maryland: Johns Hopkins University Press.

Gulli, A., and Signorini, A. (2005). The indexable Web is more than 11.5 billion pages. In *Proceedings of the 14th International World Wide Web Conference.* Chiba, Japan.

Gyöngyi, Z., and Garcia-Molina, H. (2005). Spam: It's not just for inboxes anymore. *Computer*, 38(10):28–34.

Gyöngyi, Z., Garcia-Molina, H., and Pedersen, J. (2004). Combating Web spam with TrustRank. In *Proceedings of the 30th International Conference on Very Large Databases*, pages 576–584.

Haveliwala, T., and Kamvar, S. (2003). *The Second Eigenvalue of the Google Matrix.* Technical Report 2003-20. Stanford University.

Haveliwala, T. H. (2002). Topic-sensitive PageRank. In *Proceedings of the 11th International World Wide Web Conference.* Honolulu, Hawaii.

Hawking, D., and Craswell, N. (2001). Overview of the TREC-2001 Web Track. In *Proceedings of the 10th Text REtrieval Conference.* Gaithersburg, Maryland.

Hawking, D., Upstill, T., and Craswell, N. (2004). Toward better weighting of anchors. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 512–513. Sheffield, England.

Henzinger, M. (2006). Finding near-duplicate Web pages: A large-scale evaluation of algorithms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and development in Information Retrieval*, pages 284–291. Seattle, Washington.

Heydon, A., and Najork, M. (1999). Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229.

Ivory, M. Y., and Hearst, M. A. (2002). Statistical profiles of highly-rated Web sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 367–374. Minneapolis, Minnesota.

Jansen, B. J., Booth, D., and Spink, A. (2007). Determining the user intent of Web search engine queries. In *Proceedings of the 16th International World Wide Web Conference*, pages 1149–1150. Banff, Canada.

Jeh, G., and Widom, J. (2003). Scaling personalized Web search. In *Proceedings of the 12th International World Wide Web Conference*, pages 271–279. Budapest, Hungary.

Joachims, T., Granka, L., Pan, B., Hembrooke, H., and Gay, G. (2005). Accurately interpreting

clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 154–161. Salvador, Brazil.

Joachims, T., and Radlinski, F. (2007). Search engines that learn from implicit feedback. *IEEE Computer*, 40(8):34–40.

Jones, R., Rey, B., Madani, O., and Greiner, W. (2006). Generating query substitutions. In *Proceedings of the 15th International World Wide Web Conference*, pages 387–396. Edinburgh, Scotland.

Kamvar, S. D., Haveliwala, T. H., Manning, C. D., and Golub, G. H. (2003). Extrapolation methods for accelerating PageRank computations. In *Proceedings of the 12th International World Wide Web Conference*, pages 261–270. Budapest, Hungary.

Kellar, M., Watters, C., and Shepherd, M. (2007). A field study characterizing web-based information-seeking tasks. *Journal of the American Society for Information Science and Technology*, 58(7):999–1018.

Kelly, D., and Teevan, J. (2003). Implicit feedback for inferring user preference: A bibliography. *ACM SIGIR Forum*, 37(2):18–28.

Kleinberg, J. M. (1998). Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677. San Francisco, California.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

Langville, A. N., and Meyer, C. D. (2005). A survey of eigenvector methods of Web information retrieval. *SIAM Review*, 47(1):135–161.

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

Lawrence, S., and Giles, C. L. (1998). Searching the World Wide Web. *Science*, 280:98–100.

Lawrence, S., and Giles, C. L. (1999). Accessibility of information on the Web. *Nature*, 400:107–109.

Lee, U., Liu, Z., and Cho, J. (2005). Automatic identification of user goals in Web search. In *Proceedings of the 14th International World Wide Web Conference*, pages 391–400. Chiba, Japan.

Lempel, R., and Moran, S. (2000). The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Computer Networks*, 33(1-6):387–401.

Liu, Y., Fu, Y., Zhang, M., Ma, S., and Ru, L. (2007). Automatic search engine performance evaluation with click-through data analysis. In *Proceedings of the 16th International World Wide Web Conference Workshop on Query Log Analysis: Social and Technological Challenges*, pages 1133–1134. Banff, Canada.

Marchiori, M. (1997). The quest for correct information on the Web: Hyper search engines. In *Proceedings of the 6th International World Wide Web Conference*. Santa Clara, California.

Metzler, D., Strohman, T.; and Croft, W. (2006). Indri TREC notebook 2006: Lessons learned from three Terabyte Tracks. In *Proceedings of the 15th Text REtrieval Conference*. Gaithersburg, Maryland.

Najork, M., and Wiener, J. L. (2001). Breadth-first search crawling yields high-quality pages. In *Proceedings of the 10th International World Wide Web Conference*. Hong Kong, China.

Najork, M. A. (2007). Comparing the effectiveness of HITS and SALSA. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management*, pages 157–164. Lisbon, Portugal.

Najork, M. A., Zaragoza, H., and Taylor, M. J. (2007). HITS on the Web: How does it compare? In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 471–478. Amsterdam, The Netherlands.

Ng, A. Y., Zheng, A. X., and Jordan, M. I. (2001a). Link analysis, eigenvectors and stability. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages

903–910. Seattle, Washington.

Ng, A. Y., Zheng, A. X., and Jordan, M. I. (2001b). Stable algorithms for link analysis. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 258–266. New Orleans, Louisiana.

Ntoulas, A., Cho, J., and Olston, C. (2004). What's new on the Web?: The evolution of the web from a search engine perspective. In *Proceedings of the 13th International World Wide Web Conference*, pages 1–12.

Olston, C., and Najork, M. (2010). Web crawling. *Foundations and Trends in Information Retrieval.*

Olston, C., and Pandey, S. (2008). Recrawl scheduling based on information longevity. In *Proceedings of the 17th International World Wide Web Conference*, pages 437–446. Beijing, China.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Stanford InfoLab.

Pandey, S., and Olston, C. (2008). Crawl ordering by search impact. In *Proceedings of the 1st ACM International Conference on Web Search and Data Mining.* Palo Alto, California.

Qiu, F., and Cho, J. (2006). Automatic identification of user interest for personalized search. In *Proceedings of the 15th International World Wide Web Conference*, pages 727–736. Edinburgh, Scotland.

Rafiei, D., and Mendelzon, A. O. (2000). What is this page known for? Computing Web page reputations. In *Proceedings of the 9th International World Wide Web Conference*, pages 823–835. Amsterdam, The Netherlands.

Richardson, M., and Domingos, P. (2002). The intelligent surfer: Probabilistic combination of link and content information in PageRank. In *Advances in Neural Information Processing Systems 14*, pages 1441–1448.

Richardson, M., Prakash, A., and Brill, E. (2006). Beyond PageRank: Machine learning for static ranking. In *Proceedings of the 15th International World Wide Web Conference*, pages 707–715. Edinburgh, Scotland.

Rivest, R. (1992). *The MD5 Message-Digest Algorithm.* Technical Report 1321. Internet RFC.

Robertson, S., Zaragoza, H., and Taylor, M. (2004). Simple BM25 extension to multiple weighted fields. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 42–49. Washington, D.C.

Rose, D. E., and Levinson, D. (2004). Understanding user goals in web search. In *Proceedings of 13th International World Wide Web Conference*, pages 13–19. New York.

Spink, A., and Jansen, B. J. (2004). A study of Web search trends. *Webology*, 1(2).

Upstill, T., Craswell, N., and Hawking, D. (2003). Query-independent evidence in home page finding. *ACM Transactions on Information Systems*, 21(3):286–313.

Wolf, J. L., Squillante, M. S., Yu, P. S., Sethuraman, J., and Ozsen, L. (2002). Optimal crawling strategies for Web search engines. In *Proceedings of the 11th International World Wide Web Conference*, pages 136–147. Honolulu, Hawaii.

Yi, K., Yu, H., Yang, J., Xia, G., and Chen, Y. (2003). Efficient maintenance of materialized top-k views. In *Proceedings of the 19th International Conference on Data Engineering*, pages 189–200.

第 16 章

Information Retrieval: Implementing and Evaluating Search Engines

XML 检索

以 XML 表示的文档给信息检索系统提供了这样一个机会：在必要的时候系统可以充分利用文档的结构，返回单独的文档的某个部分作为结果。为响应用户查询，XML 检索系统返回的结果可能是段落、章节、文章、参考文献条目或其他元素的混合体。当文档集包含很长的文档时，如手册或书籍，这种方法就特别有用，可以直接将用户引到这些文档最相关的部分中。

本章将结束贯穿全书的关于文档结构的一系列思想的讨论。我们从第 1 章开始，通过引入包括《Macbeth》在内的莎士比亚戏剧集（图 1-2），对 XML 进行了简单概述。这个文档集在第 2 章中仍然是一个运行示例，并在此后的章节中也偶尔会出现。2.1.3 节中介绍了一种通过倒排索引支持轻量级结构的方法，并在 5.2 节中进行了扩展和规范化。在 8.7 节中，我们将结构与概率模型结合，根据结构元素出现的位置，对词项赋予权重。例如在标题或摘要中出现的词项权重要比出现在附录或脚注中的词项权重大。第 15 章中我们探讨了 Web 检索中锚文本和链接结构的作用。

本章将对 XML 和 XML 信息检索进行全面的介绍。16.1 节中将讨论前面忽略的或未提到的那些 XML 的基本知识，尤其是 XML 文档的递归结构和标签属性。16.2 节将从数据库的角度对 XML 检索进行进一步的讨论。这一节还引入了 XPath 查询语言，可用于检索满足某种特别标准的 XML 元素集合。XPath 在 XML 检索中的作用就如布尔代数在基本文档检索中的作用一样，允许我们选择元素的一个子集用于进一步处理，例如排名。我们还讨论了 XPath 的一个变形——NEXI 查询语言，专门用于解决与 XML 检索有关的问题。另外，本章简要介绍了 XQuery 语言，该语言为查询和操作 XML 文档提供了先进的方法。

本章介绍的多种语言，包括 XPath、XQuery 以及 XML 本身，都是万维网协会的（W3C）标准。W3C 是一个国际标准组织，发布了很多关于 Web 技术的官方标准，可以通过它的 Web 网站[⊖]访问到。

本章第 3～5 节分别对应本书的第二、三和四部分：16.3 节考察索引和查询处理方法；16.4 节讨论排名；16.5 节介绍评价方法。对索引和查询处理的讨论是基于 5.2 节中介绍的轻量级结构进行的。

对排名检索和评价的讨论都在 XML 检索评价倡议（INEX[⊖]）的内容上进行。INEX 自 2002 年就开始提供类似 TREC 的实验平台，针对 XML 编码的文档集，大家展示各自的信息检索技术，并对其进行评价。每年约有 100 个来自工业和学术界的小组参加 INEX。2006 年之前，INEX 采用发表在 IEEE 计算机协会中的杂志和期刊上的文章作为检索文档集。自 2006 年起，INEX 开始采用英文版维基百科全书中的文章的 XML 形式的文档集。NEXI 语言的诞生部分功劳是属于 INEX 的。

XML 检索是一个与传统数据库检索紧密相关的广泛课题，而且对该领域的基本构成也有很多不同观点。虽然主要的数据库供应商现在已经将支持 XML 的功能融入他们的产品当

⊖ www. w3. org

⊖ www. inex. otago. ac. nz

中，但是 XML 在信息检索中的作用还没有得到重视。总而言之，与前述章节相比，本章更倾向于作介绍性的概述，而不是深入研究。

16.1　XML 的本质

图 16-1 给出了一篇 XML 形式的期刊文章的部分内容。该图给出了一些前面章节没有提及的 XML 的几个方面。文档开始于一个 **XML 声明**（declaration），表示下面的内容遵从 XML 1.0 标准且内容使用 UTF-8 编码：

```
<?xml version = "1.0" encoding = "UTF-8"?>
```

每个元素都以形为〈name〉的开始标签作为开始，以形为〈/name〉的结束标签作为结束。XML 的开始标签中可能包含一个或多个属性，形如：

attribute=value

图中文章的标签包含了提供参考文献信息的属性：

〈article journal="IEEE TKDE" volume="9" number="2" year="1997"〉

```
<?xml version="1.0" encoding="UTF-8"?>
<article journal="IEEE TKDE" volume="9" number="2" year="1997">
   <!-- Foundational paper on inverted index compression -->
   <header>
      <title>Text Compression for Dynamic Document Databases</title>
      <author>Alistair Moffat</author>
      <author>Justin Zobel</author>
      <author>Neil Sharman</author>
      <abstract><p>For compression of text databases...</p>...</abstract>
   </header>
   <body>
      <section number="1">
         <title>Introduction</title>
         <p>Modern document databases contain vast quatities of text...</p>
         <p>There are good reasons to compress the text...</p>
         ...
      </section>
      <section number="2">
         <title>Reducing memory requirements</title>
         <p>In this section we assume a static text collection...</p>...
         <section number="1">
            <title>Method A</title>
            <p>The first method considered for choosing which words...</p>...
         </section>
      </section>
   ...
   </body>
</article>
```

图 16-1　采用 XML 编码的一篇期刊文章

章节标签中包含了指定其章节号的属性：

```
<section number = "2">
```

属性值要用单引号或双引号括起。标签也可用〈name/〉的形式。该形式（图中未给出）也称为**空元素标签**（empty-element tag）。空元素标签本质上与一个开始标签后紧跟一个结束标签（〈name〉〈/name〉）的效果是等价的。空元素标签没有内容但是可以有属性。

XML 文档也可能包含注释，以字符串“〈！——”开始并以字符串“——〉”结束。图中的注释给出了该论文的一个简要评论。用应用软件或工具创建或更新的 XML 文档通常会包含表明软件的名称和版本号的注释。注释既不是内容也不是结构。对于解析和处理的 XML 的软件而言，不要求这些注释是可见的，而且为了索引和检索，将这些注释保留在文档中可能也是不合适的。

XML 元素必须是嵌套的。不允许存在重叠交错的元素。与之不同的是，5.2 节中提到的区域代数是允许重叠的。实际上，HTML 也一样，至少在有限的范围内是允许重叠的。例如下面的 HTML 片段：

```
<p> Simple <b>example··· <p>
<p> ···illustrating <em>overlapping</b> tags in </em> HTML. </p>
```

尽管从技术上讲，这个片段不是一个合法的 HTML，但它能很好地说明上述问题。大部分浏览器都能正确处理这个片段，把这个片段渲染成

Simple **example...**

...illustrating *overlaping* *tags in* HTML.

当文档结构不是严格层次化的时候，XML 元素必须是嵌套的这一要求使其难以捕捉这样的文档结构（Hockey，2004）。例如，图 1-2 中摘录的《Macbeth》文档片段，第 6 行和第 8 行的台词对应多个讲演者。图 1-3 中的 XML 编码为保持层次结构而将这几行分离开，从而忽略了诗歌本身的结构（有争议说这样做更有意义）。下面的编码更好地反映了第 8 行的诗歌结构：

```
<SPEECH>
    <SPEAKER>First Witch</SPEAKER>
        <LINE>Where the place?
</SPEECH>
<SPEECH>
    <SPEAKER>Second Witch</SPEAKER>
                        Upon the heath. </LINE>
</SPEECH>
```

遗憾的是，上述编码并非结构良好的 XML。

尽管 XML 不支持重叠元素，但它支持递归结构。正如图 16-1 所示，章节中会包含章节，所包含章节中也包含了其他章节，如此类推。这种递归结构使我们能够根据需要构建子章节以及子章节的子章节，且不需要预先定义嵌套的最大深度。

XML 标准定义了**格式化的文档**（well-formed document）这一概念。除了其他要求外，格式化的文档中的标签、属性以及注释还必须遵循上述的格式规则。

XML 的一个重要特性就是能够提供**结构性元数据**（structural metadata）来约束格式化 XML 文档的内容。这些约束可能会要求某些特定的元素中需要包含其他特定的元素。例如，所有的期刊文章都必须要包含一个头部和一个主体。可以为每个元素指定一个允许的属性集，有些属性是必须的，而有些是可选的。例如，要求一篇论文使用某些属性来指明所发表期刊和发表年份，同时允许使用附加属性指明该论文的卷号和刊号。进一步的约束可以在属性值中设定。例如，章节号必须是自然数。当已为 XML 文档提供结构性元数据时，检索系统可以利用它来优化查询处理过程。

为 XML 指定结构性元数据有两个标准。第一个标准叫做**文档类型定义**（document type definition）或 DTD，它是两个标准当中比较旧的，也是比较简单的标准。它主要关注 XML 文档的结构性组织——元素必须怎样进行嵌套以及都拥有哪些属性。第二个标准叫做 XML **模式**（XML Schema），它允许为元素和属性定义复杂类型。DTDs 源自旧的 SGML 标准，

这是 XML 的基础。XML 模式是一个新标准，主要是为了解决 DTDs 的局限性。任何用 DTD 表示的约束都可以用 XML 模式表示，但反之并不成立。

16.1.1　文档类型定义

图 16-2 中的 DTD 与图 16-1 中的期刊文章是一致的。为了便于讨论，图中所示的 DTD 只包括元素和属性。事实上，期刊文章的 DTD 需要包含引用、脚注、图形、表格、标题、目录、公式、附录以及类似的结构。

```
<?xml version="1.0"?>
<!ELEMENT article (header body)>
<!ATTLIST article journal CDATA #REQUIRED>
<!ATTLIST article volume  CDATA #IMPLIED>
<!ATTLIST article number  CDATA #IMPLIED>
<!ATTLIST article year    CDATA #REQUIRED>
<!ELEMENT header (title, author+, abstract?)>
<!ELEMENT body (section+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT section (title, p*, section*)>
<!ATTLIST section number CDATA #REQUIRED>
<!ELEMENT abstract (p)>
<!ELEMENT p (#PCDATA)>
```

图 16-2　期刊文章的一个（部分）DTD

DTD 以一个声明开始，表示其采用了 XML 1.0 标准。尽管 DTD 看起来神秘，但它多少还是采用上下文无关的语法来描述了元素该如何嵌套以及每个元素可以包含什么属性。每个 ELEMENT 声明都定义了一个语法规则。如下面的声明：

```
<!ELEMENT article(header,body)>
```

表明一个文章元素包含两个子元素：头部和主体。继而头部由一个标题、一个或多个作者以及一个可选的摘要组成：

```
<!ELEMENT header(title,author+,abstract?)>
```

紧跟在元素名后面的问号（“?”）表示该元素是可选的。紧跟在元素名后面的加号（“+”）表示该元素会出现一次或多次。紧跟在元素名后面的星号（“*”）表示该元素会出现 0 次或多次。

DTD 中有可能混淆的地方就是 **ELEMENT** 声明中并没有指定各个元素的顺序。在头部中，标题、作者和摘要出现的顺序可以是任意的。用户普遍接受的顺序是头部先出现摘要，然后是作者，接下来是标题，最后还有另外两名作者。由于继承自 SGML，在 XML 的 DTD 中指定元素出现顺序是不可能的。

标题的声明表示它只包含文本内容：

```
<!ELEMENT title(#PCDATA)>
```

符号“**#PCDATA**”表示“已解析的字符数据”，其本质上就是一个普通文本。属性在 **ATTLIST** 声明中定义。下面的声明表示章节有一个必选属性提供章节号：

```
<!ATTLIST section number CDATA #REQUIRED>
```

类型为“CDATA”的属性表示它由普通文本组成。我们可能希望其属性值取自然数，但 DTD 无法表达这种类型的限制。

为什么在元素声明中用“#PCDATA”表示普通文本，而在属性声明中却用“#CDATA”表示普通文本呢？答案就在解析 XML 的细节中，不过这超出了本书的讨论范围。尽管 DTD 作为上下文无关的语法的基本概念看起来相对简单，但对其进行深入表述和讨论将需要足足一章内容。

为了将 DTD 和文档联系起来，文档中必须包含一个**文档类型声明**（document type declaration）。尽管 DTD 可以直接出现在类型声明中，但是为了方便最好将 DTD 放在一个单独的文件中，并通过它的文件名来访问它。这样多个文档就可以共享一个 DTD 了。例如，图 16-2 中的 DTD 可以存储在一个名为“article. dtd”的文件中。下面的类型声明可加入到

图 16-1 所示文档的第二行中，紧跟在 XML 声明之后：

```
<!DOCTYPE article SYSTEM "article.dtd">
```

16.1.2 XML 模式

XML 模式为元素和属性提供了非常强大的类型机制。我们只简单介绍一下 XML 模式；其全面的描述和讨论需要足足一本书（van der Vlist，2002）。XML 模式的核心是大量内置的**基本类型**（primitive type），包括整型、浮点型数字、日期、时间、时间段和 URL。**派生类型**（derived type）在这些基本类型上添加额外限制。例如，我们可能会限制表示期刊名的字符串的最短长度。**复合类型**（complex type）通过结合基本类型、派生类型以及其他复合类型来为元素提供类型。如文章及其头部中的元素顺序可以用复合类型指定，将其文件卷号属性值和期号属性值进行进一步约束为自然数，将其年份属性值约束为年份。

16.2 路径、树和 FLWOR

由于 XML 元素必须嵌套，所以可以把它看成一棵树。图 16-3 给出了图 16-1 中文档的树形结构。树形结构所表示的只是图 16-1 中给出的部分文档；其余部分为清晰和简单起见已经省略了。尽管图中并未包含这个文档的文本（如它的 #PCDATA），但文本也被视为一个叶节点的集合，附着在包含它们的元素上。其属性也可以一样被视为附着在对应元素上的叶节点。

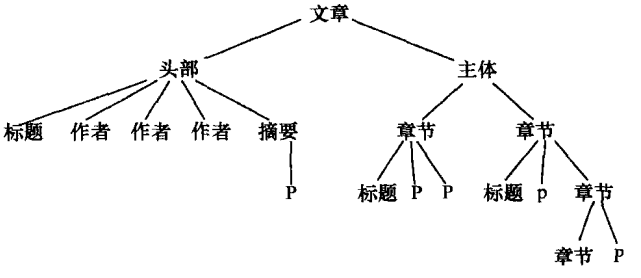


图 16-3 图 16-1 中 XML 文档的树形结构

或许因为 XML 文档很自然地被看成一棵树，所以 XML 检索也经常被看做是树匹配或路径匹配问题。这就是包括 XPath、NEXI 和 XQuery 在内的 XML 查询语言所采用的观点。

16.2.1 XPath

XPath 为指定元素集和 XML 文档中的其他节点提供了标准的符号。1999 年 11 月 W3C 最终敲定并发布了 XPath 的初始版本（版本 1.0）。[⊖]2007 年 1 月发布了第二版的 XPath（版本 2.0），同时也发布了第一版本的 XQuery 标准。[⊜]可是急切地想要从 XPath2.0 中构造一个严格的 XQuery 子集所导致的结果是，XPath 2.0 并不兼容 XPath 1.0。幸运的是，这些不兼容多数存在于底层数据模型的细节中，与本章所给出的简单例子并无关系。

XPath 给出了指定文档元素集合和其他节点的表达方式，称为**路径表达式**（path expression）。最简单的路径表达式就是用斜线来分隔元素名的列表。如以下路径表达式：

```
/article/body/section
```

指定了文章主体中所有一级章节的集合。类似的，路径表达式

```
/article/body/section/section
```

指定了所有二级章节（子章节）的集合。路径表达式

⊖ www.w3.org/TR/xpath

⊜ www.w3.org/TR/xpath20

```
/article/body/section/title
```

指定了所有的一级章节的标题。这个符号专门使用了与用于指定 URL 路径组成和 UNIX 操作系统中文档路径类似的表达方式。

除路径表达式以外，XPath 也支持各种算术、逻辑和字符串表达式，并且还提供大量的内置函数。这些支持使得 XPath 可以和其他语言联合使用，为那些语言提供一般化的表达方式。这样的语言包括 XQuery 和样式表语言 XSL（另一种 W3C 标准语言）。XPath 1.0 还提供了一种用于归并元素集合的运算符，XPath 2.0 将它扩展为支持其他集合的运算符。

路径表达式中可以包含谓词，用来指定属性值和元素位置。如以下路径表达式

```
/article/body/section[2]
```

表示了文档序中的第二个章节。路径表达式

```
/article/body/section[@number = 2]
```

表示属性值为 2 的一级章节（和前面的例子相同的那个元素）。谓词也可以用来指定元素或属性的内容。路径表达式

```
/article/body/section[contains(./title, "memory requirements")]
```

表示章节标题中包含字符串“memory requirements”，其中“. /title”表示当前节点。在路径中也可以采用 descendant-or-self 符号来“跳过”元素，该符号用双斜线（“//”）表示。例如，路径表达式

```
/article//section
```

表示一篇文章中的全部章节（忽略嵌套或深度）。路径表达式

```
//section/title
```

表示所有章节的标题。

上述例子没有涉及路径表达式的复杂性。除了上述特性，XPath 提供了一些奇怪的附加符号来作为在祖先节点、兄弟节点和子孙节点间的导航。

16.2.2 NEXI

XPath 不支持排名检索。正如 2.2.3 节中提到的布尔代数和 5.2 节中提到的区域代数一样，XPath 有完全匹配的语义。即一个路径表达式指定的节点集合与表达式本身指定的标准是完全匹配的。特别是 XPath 表达式的 contains 函数实现了字符串的完全匹配。如路径表达式

```
//article//section[contains(.,"compression codes")]
```

要求字符串“compression codes”逐字逐句地出现在对应的章节中。Xpath 没有为与“compression codes”有关的有序段落集提供任何指定方式。

在研究 INEX 时，Trotman 和 Sigurbjörnsson（2004）提出了 Xpath 的一个变形，称为 NEXI（Narrowed Extended XPath I）来打破这些限制。NEXI 将 XPath 所包含的路径表达式压缩成一个子集，这样就对排名检索提供了扩展支持。

由于元素的确切内容在信息检索应用中可能没那么重要，所以 NEXI 只为这些路径提供 descendant-or-self 符号（“//”）。为指定排名检索，NEXI 将 contains 函数替换为 about。有关“compression codes”的章节标题的排名列表应该用下面的路径表达式

```
//article//section[about(., "compression codes")]
```

NEXI 为路径表达式既提供了严格的解释也提供了宽松的解释。在严格解释时，上述表达式的结果集中将只包含文章的章节。在宽松解释时，结构信息将作为关于用户需求的提示来处理：路径表达式表明用户偏好文章的章节，但是系统可以返回其他元素，如段落。当对元素进行排名时，系统需要平衡好元素的内容和由用户提供的结构性提示，以决定总体相关性。

NEXI 允许用布尔操作符“and”和“or”来连接路径元素。例如，路径表达式

```
//article[about(.//p,"compression codes") and about(.//author,moffat)]
```

请求返回一个由 Moffat 写的文章的有序列表，且该文章的段落是关于“compression codes”的。NEXI 只对词组使用引号，因此该表达式中并没有对“Moffat”使用引号。NEXI 也允许在路径表达式中采用通配符，即用星号“*”表达的符号。如路径表达式

```
//section//*[about(.,"compression codes")]
```

请求返回一个章节中包含关于“compression codes”元素的有序列表。

尽管 NEXI 不是 W3C 的官方标准，它在 XML 信息检索领域中还是得到了广泛的认同。NEXI 在该领域外相对来讲还是鲜为人知的，随着 XML 信息检索的广泛应用该情况有可能会得到改善。

16.2.3 XQuery

本节将粗略地介绍一下 XQuery 语言。XQuery（也称 XML XQuery）本质上是为方便地操作和转换 XML 文档而对 XPath 进行的扩展。尤其是，XQuery 允许表达式动态地构建 XML，使得表达式输出本身就是 XML。XQuery 表达式对节点和值的有序序列进行操作。序列中每个节点和值都必须有与之相对应的类型，该类型由 XML 模式提供。

XQuery 的核心是 **FLWOR 表达式**（FLWOR expression）（发音和“flower”一样），它是 for、let、where、order by 和 return 的首字母缩写。这些缩写就暗示了 FLWOR 表达式各个部分和它的操作。FLWOR 表达式为序列的创建、过滤、排序以及迭代提供了一种机制。与 SQL 中的 select 语句相类似，FLWOR 表达式可计算多个 XML 文档的连接。

为了说明 FLWOR 表达式，我们从 XQuery 标准[⊖]中引出一个例子。假设有一个采用 XML 编码的戏剧的参考文献如下：

```
<bib>
    <play>
        <author>Shakespeare</author>
        <title>Hamlet</title>
        <year>1599</year>
    </play>
    <play>
        <author>Beckett</author>
        <title>Waiting For Godot</title>
        <year>1949</year>
    </play>
    <play>
        <author>Shakespeare</author>
        <title>Macbeth</title>
        <year>1603<year>
    </play>
    <play>
        <author>Stoppard</author>
        <title>Rosencrantz and Guildenstern Are Dead</title>
        <year>1966</year>
    </play>
</bib>
```

⊖ www.w3.org/TR/xquery

图 16-4 中的 FLWOR 表达式通过创建一个剧作家的列表来识别这个数据，每一个剧作家名字与他的作品列表一起出现。该例子由两个嵌套表达式组成。外部表达式构建了一个作者列表；内部表达式为每个作者构建了一个作品列表。对上述参考文献（与变量 $bib 绑定）执行这个表达式以后会产生以下输出：

```
<works>
    <playwright>
        <name>Beckett</name>
        <plays>
            <title>Waiting For Godot</title>
        </plays>
</playwright>
<playwright>
    <name>Shakespeare</name>
    <plays>
        <title>Hamlet</title>
        <title>Macbeth</title>
    </plays>
</playwright>
<playwright>
    <name>Stoppard</name>
    <plays>
        <title>Rosencrantz and Guildenstern Are Dead</title>
    </plays>
  </playwright>
</work>
```

由于受到数据库和编程语言理论研究很大的影响，XQuery 拥有复杂的形式语义和静态类型。它的学习曲线是陡峭的。令人意外的是，在类型系统付出如此大的努力并使用如此复杂的形式语义后，静态类型居然是可选的类型。如何进行强制操作类型是可以自由选择的：或者在查询前静态地进行转换，或者在执行过程中动态地抛出类型异常。混合采用以上两种方式也是可以的。

```
<works>
{
   for $a in fn:distinct-values($bib/play/author)
   order by $a
   return
      <playwright>
        <name> {$a} </name>
        <plays>
        {
           for $p in $bib/play[author = $a]
           order by $p/title
           return $p/title
        }
        </plays>
      </playwright>
}
</works>
```

图 16-4　XQuery 的一个 FLWOR 表达式

16.3　索引和查询处理

XML 的索引和查询处理引出了两个突出问题，这两个问题采用 5.2 节中的区域代数是很难解决的。第一，元素可能是递归嵌套的。如下面表达式的结果

```
//section/section
```

涵盖了包含在另外一个章节中的所有章节，且忽略嵌套或深度。尽管解决办法可以表示成一系列的间隔，一些章节嵌套着其他章节，但是它不可以表示成一个 GC-list。XPath 查询处理策略必须恰当地解决这一递归问题。第二，XPath 表达式可以直接地指定内容之间的关系。如下表达式的结果

```
/article/body/section/p
```

只涵盖了直接包含在所有一级章节中的段落，不包含那些在子章节和子章节的子章节中的段落。另一方面，与区域代数不同，XPath 对重叠的结构没有要求。由于元素一定是嵌套的，所以我们可以将文档看做树。

为适应递归结构和直接包含的情况，我们对倒排表 ADT 进行扩展，因此每个位置信息列表中的元素就变成了一个包含 3 个值的三元组：起始位置、结束位置和在树中的深度：

(start, end, depth)

与区域代数不同，元素的起始和结束标签没有单独的位置信息列表，但它们共享一个联合的位置信息列表。对于一个元素而言，三元组中的起始位置表示起始标签的位置，结束位置表示结束标签的位置。对其他词条而言，如词，起始位置和结束位置是相同的，以表示词条的位置。这些扩展位置信息列表中的三元组根据起始位置进行排序。

例如，回到图 2-1 中《Macbeth》的倒排索引，该戏剧的扩展位置信息列表变为

{(3, 40508, 1), (40511, 75580, 1), …, (1234602, 1271504, 1)}

这个位置信息列表的深度值始终为 1，因为戏剧只以最顶端元素的形式出现，而且一个位置信息列表中所有三元组的深度值不需要完全相同。例如，对白的扩展位置信息列表为

{(312, 470, 4), (471, 486, 4), …, (506542, 506878, 3), …, (1271273, 1271498, 4)}

演讲通常在深度为 4 的位置出现，嵌套在戏剧表演的一个场景中，但也可以出现在深度为 3 的位置，作为戏剧的开场。“witch” 的位置信息列表为

{(1598, 1598, 6), (27555, 27555, 6), …, (432149, 432149, 4), …, (1245276, 1245276, 6)}

女巫可能在深度为 6 的位置以讲话者的身份出现，或者在深度为 4 的位置出现在人物名单中。为实现这种扩展功能，我们将倒排列表 ADT 的四个方法修改成如下形式：

first(*term*)	返回给定 *term* 的第一个元组
last(*term*)	返回给定 *term* 的最后一个元组
next(*term*, *current*)	返回 *current* 位置的下一个元组
prev(*term*, *current*)	返回 *current* 位置的上一个元组

例如，

first(“⟨PLAY⟩”) = (3, 40508, 1)

last(“⟨SPEECH⟩”) = (1271273, 1271498, 4)

next(“⟨SPEECH⟩”, 400) = (471, 486, 4)

prev(“witch”, 20000) = (1598, 1598, 6)

这个经修改的倒排索引可以把从结构性元数据中得到的知识联合起来，有效地解决 XPath 表达式的问题。例如，图 16-5 中的代码解决了类似于图 16-1 的路径表达式

```
//section//section
```

在期刊文章上的问题。从图 16-2 所示的 DTD 中我们可以看出递归的章节，因此路径表达式可以得到不为空的结果集。比起明确地找出所有包含有其他章节的章节，我们只需要注意那些是结果集一部分的章节，且不要出现在最顶层。因此，程序对一级章节逐个进行处理，并报告包含在其中的所有章节，不需要考虑其深度。

```
1  (u, v, d) ← first(“⟨section⟩”)
2  (u′, v′, d′) ← next(“⟨section⟩”, u)
3  while u < ∞ do
4     if v′ < v then
5        report the interval [u′,v′]
6        (u′, v′, d′) ← next(“⟨section⟩”, u′)
7     else
8        (u, v, d) ← (u′, v′, d′)
```

图 16-5 解决路径表达式//section//section 的代码

第 1 行将第 1 章节存储在三元组（u，v，d）中。该章节一定是一级章节，因为它的

起始位置的值最小。第 2 行将第一个候选章节（可能是结果集中的一个元素）存储到三元组（u'，v'，d'）中。第 3～8 行的循环逐个检查候选章节，判断它是否包含于当前的一级章节中。如果是，在第 5 行中进行报告。如果不是，它将在第 8 行成为新的一级章节。报告一个章节可以是将其呈现给用户，或者更多情况下是将其保存以供进一步的处理。在计算结果集时，不需要用到深度值。

图 16-6 中的代码说明了深度值的用法，即求得了下面的路径表达式的结果集

/article/body/section/p.

从 DTD 中可以看出，章节仅仅包含在文章主体中，使得在评价路径表达式时不必要明确区分是文章还是文章主题。为了满足路径表达式的要求，在结果集中的段落一定要直接包含于一级章节中。通过 DTD，我们可看出这样的章节必须处在深度为 4 的位置。以第一个一级章节（第 1 行）作为开始，代码遍历了一级章节，报告了其包含的所有深度为 4 的段落（第 2～8 行）。从一个一级章节转到另一个一级章节时，第 8 行调用的方法通过在当前一级章节结束以后再进行索引，跳过了深度较浅的一级章节。第 4～7 行遍历了包含于一级章节中的全部段落，只有遇到深度为 4 的段落时才进行报告（第 5～7 行）。

```
1  (u, v, d) ← first("⟨section⟩")
2  while u < ∞ do
3      (u', v', d') ← next("⟨p⟩", u)
4      while v' < v do
5          if d' = 4 then
6              report the interval [u',v']
7          (u', v', d') ← next("⟨p⟩", u')
8      (u, v, d) ← next("⟨section⟩", v)
```

图 16-6　解决路径表达式/article/body/section/p 的代码

16.4　排名检索

XML 的排名检索根据查询支持的类型可视为两个不同的问题。第一个，正如标准的文档检索是词项向量排名一样，可以将 XML 信息检索看做把查询表达为词项向量的形式来排名元素。例如，给定一个词项向量〈“text”，“compression”〉，信息检索系统返回一个根据这些元素与这条查询的相关性概率排名的有序列表。

第二种观点就是可以把查询表示为使用如 NEXI 这样的语言得到的路径表达式。元素根据内容和结构进行排名。查询指定的结构既可以看成一个严格的过滤器，也可以不严格地看做一种提示。当把它解释为过滤器时，只有与路径表达式指明的结构完全匹配的元素才会出现在结果列表中。当不严格地看待它时，信息检索系统必须权衡结构需求和内容需求之间的潜在竞争关系。考虑下面的路径表达式

/article/body/section[about(./title,"memory requirements")]

当严格地看待它时，信息检索系统要求只能返回一级章节，这些章节按照它们的标题与“memory requirements”这一主题的相关性进行排序。当不严格地看待它时，相比于其他元素，信息检索系统可能会更偏向于章节，同时对出现在标题中的查询词项赋予更高的权重值，但是允许它返回子章节、子章节的子章节、段落和其他元素。

在 INEX 术语中，按照词项向量对元素进行排名的问题称为**纯内容任务**（content-only task），或 CO 任务。根据路径表达式对元素进行过滤和排名的问题称为**内容-结构任务**（content-and-structure task），或 CAS 任务。可以把 CO 任务中的查询看做是 CAS 任务查询的一个特例。为 CO 任务给定一个词项向量〈$t_1, t_2, \cdots, t_n$〉，我们可将其看做与下面的 CAS 查询是等价的

//*[about(.,"$t_1\ t_2\ t_n$")]

尽管大量关于 CAS 查询的排名方法都已经在研究文献中提出过了，但是该问题没有被很好地理解，并且这些方法中没有一个可以明确地胜过其他方法。解决 CAS 查询问题的一个简

单而又合理的方法就是通过两步检索操作来区别对待结构和内容。第一步，把路径表达式用作过滤器，把结果限制为元素的一个子集。在第一步中，忽略路径表达式的 about 函数，所有元素与任意主题的相关性都是一样的。在第二步中，利用出现在 about 函数中的词项构造出一个查询向量。第一步中选出的元素将根据这个词项向量进行排序。例如，给定路径表达式

```
/article/body/section[about(./title, "memory requirements")]
```

第一步中会创建一个一级章节的集合，第二步会根据词项向量〈“memory”，“requirements”〉对它们进行排名。在 about 函数（./title）中的结构约束可以通过排名函数起作用，也可以不起作用。类似地，对于路径表达式

```
//article[about(.//p, "compression codes")and about(.//author,Moffat)]
```

该方法会根据词项向量〈“compression”，“codes”，“moffat”〉来对文章进行排名。尽管这种简单的方法可能会忽略掉 CAS 查询的某些方面，如前面例子中分别对作者和段落指定约束，但是没有其他方法比该方法具有更好的性能了。该问题还有待更进一步的研究。

16.4 节的剩余部分将重点讲解 CO 任务。正如 CAS 查询一样，很多关于 CO 查询的排名方法都在文献中提出过，以及在 INEX 中测试过。尽管很多方法都有自己的价值，但是没有一种方法可以完全胜过以下的简单方法：采用如 BM25 等标准排名公式对独立元素进行排名。该方法把每个元素看做一个独立的文档。这样就可对这个元素集采用文档排名的方法。

当将文档排名技术应用到元素集合中时，马上就出现了一些问题。首先，一个文本片段可能属于几个不同的元素。例如，图 16-1 所示的期刊文章中包含的文本片段可能构成段落、子章节、主体和文章本身的一部分。若以检索为目的将这些元素都看成一个独立的“文档”，这种重复可能就会改变词项和文档的统计值。其次，这些元素的嵌套（在 INEX 术语中称为“重叠”）会导致检索结果的过分冗余。如果一个段落是高度相关的，那么包含它的子章节、章节、主体和文章有可能也具有很高的相关性，但是在结果列表中将这些元素一一列出可能也没什么好处。最后，并不是所有的元素类型都适合用作检索结果。有些元素，如段落和章节，呈现给用户是合理的，但是其他元素可能就不合适了。例如对用户而言，一个章节的标题可能远没有包含这个章节的章节重要。

16.4.1 排名元素

为了进行排名，如果我们把 XML 元素看做独立的文档，这些 XML 元素组成一个集合，那么某个词项的出现次数可能会出现在几个不同的元素中。例如，图 16-1 所示文档的第 2.1 节的第一段中出现的词项“method”（“The first method considered…”）。该词项的出现可看做是段落、子章节、章节、主体和文章的一部分。如果把这些元素都看成独立的文档，则该词项在“集合”中出现了 5 次。

当在 XML 信息检索中使用标准相关性排名技术时，会单独地计算每个元素的词频统计信息（Mass 和 Mandelbrod，2004）。对于给定元素 e，计算词频值 $f_{t,e}$：词项 t 在元素 e 中出现的次数。尽管相同词项会出现在多个元素中，为计算词频，可以将词项在每个元素中的出现都视为是不同的出现。

大部分排名技术需要进行全局统计信息，如逆文档频率需要将集合作为一个整体来统计。计算这种全局统计信息的过程中不能将每个元素看做独立的文档来考虑。在计算逆文档频率时，不能假设给定词项会重叠出现在全部包含它的元素中，因为包含该词项的元素数量只依赖于原始 XML 的结构安排（Vittaut 等人，2004；Kekäläinen 等人，2004）。我们选择

一个元素的子集，使每个词项只出现在这个子集的一个元素中。例如，该子集可能由全部文章元素构成。逆文档频率和其他全局统计也可能通过这个子集来计算得到。

采用该方法来计算词频和逆文档频率时，我们可以采用标准排名公式来对元素进行排名。表 16-1 显示了第 162 个 INEX 主题（“文本和索引压缩算法”）关于由 IEEE 杂志和期刊文章组成的 INEX 集合的前几个元素。该例子中原始 INEX 符号已经过校正，以使其与图 16-2中的 DTD 保持一致。第一列列出了 BM25 评分；第二列列出了文档标识符；第三列列出了这些文档中的元素。所有元素都出自 3 个文档。文档/tk/1997/k0302.xml 为图 16-1 中所示的期刊文章。

表 16-1 IEEE 杂志和期刊文章的 INEX 集合中针对查询〈“text”，“index”，“compression”，“algorithms”〉得到的排名靠前的元素。通过把每篇文章看做一个文档来计算 IDF 值

BM25 评分	文档标识符	元素
32.000 923	/co/2000/ry037.xml	/article[1]/body[1]
31.861 366	/co/2000/ry037.xml	/article[1]
31.083 460	/co/2000/ry037.xml	/article[1]/body[1]/section[2]
30.174 324	/co/2000/ry037.xml	/article[1]/body[1]/section[5]
29.420 393	/tk/1997/k0302.xml	/article[1]
29.250 019	/tk/1997/k0302.xml	/article[1]/body[1]/section[1]
29.118 382	/tk/1997/k0302.xml	/article[1]/body[1]
29.075 621	/co/2000/ry037.xml	/article[1]/body[1]/section[3]
28.417 294	/tk/1997/k0302.xml	/article[1]/body[1]/section[6]
28.106 693	/tp/2000/i0385.xml	/article[1]
27.761 749	/co/2000/ry037.xml	/article[1]/body[1]/section[7]
27.686 905	/tk/1997/k0302.xml	/article[1]/body[1]/section[3]
27.584 927	/tp/2000/i0385.xml	/article[1]/body[1]
27.273 247	/co/2000/ry037.xml	/article[1]/body[1]/section[4]
27.186 977	/tp/2000/i0385.xml	/article[1]/body[1]/section[1]
27.072 521	/tk/1997/k0302.xml	/article[1]/body[1]/section[3]/section[1]
26.992 224	/co/2000/ry037.xml	/article[1]/body[1]/section[5]/section[1]
...	...	...

16.4.2 重叠元素

正如表 16-1 所示，对 XML 元素集直接采用标准相关性排名技术会得到一个列表，其中高排名完全受结构相关元素的影响。得分较高的章节一般包含一些得分较高的段落，或是被包含在得分较高的文章中。如果这些元素每个都作为一个独立的结果呈现给用户，那么用户将浪费大量时间在查看和排除多余的内容上。

一种可能的解决方法就是只报告那些在树的指定路径上评分最高的元素，并移除排名靠后的包含该元素或包含于该元素的元素。表 16-2 显示了在表 16-1 的有序列表中采用该方法后得到的结果。除了表中的三个元素外，其他元素全部被淘汰了。对于剩下的元素，两个是完整的文章，一个是文章的主体。更大的元素，如文章，常常用于 INEX 集合。由于这个原因，INEX 集合中的很多文章都很专业并且长度都不太长，通常少于 1 万个词。结果就是：与主题相关的完整文章通常排名靠前。

表 16-2 移去表 16-1 中的重叠元素后得到的排名靠前的元素

BM25 评分	文档标识符	元素
32.000 923	/co/2000/ry037.xml	/article[1]/body[1]
29.420 393	/tk/1997/k0302.xml	/article[1]
28.106 693	/tp/2000/i0385.xml	/article[1]
...	...	...

另外，对给定路径只报告一个结果会使得 XML 信息检索的一些优点无法发挥。例如，一个外部元素可能包含大量信息，而这些信息不会出现在内部元素中，但是内部元素可能会专注于查询主题并且提供关键概念的一个简短概述。在这种情况下，对高排名元素，可以报告它包含的元素，或者包含于它的元素。甚至当整本书都与之相关时，用户依然想得到最重要的段落，来引导其阅读并节约时间（Fuhr 和 Groß johann，2001；Clarke，2005）。

16.4.3 可检索元素

尽管 XML 信息检索系统也可以检索任何元素，但是很多元素不适合作为检索结果。这通常就是元素包含很少的文本的情况（Kamps 等人，2004）。例如，仅包含查询词项的章节的标题在排名算法中会获得很高的评分，但是单单这个标题对用户是没什么用处的，用户更想看到实际章节的内容。其他元素反映了文档的物理结构而不是逻辑结构，如字体变化，这对用户来说几乎也是没有什么用处的。一个高效的 XML 信息检索系统必须只返回那些包含足够有价值的内容并且能作为一个对象独立存在的元素（Pehcevski 等人，2004；Mass 和 Mandelbrod，2003）。标准文档的组成部分，如段落、章节、子章节和摘要，通常满足这些要求；标题、斜体的短语以及相似的元素通常不满足这一要求。

16.5 评价

XML 信息检索系统的评价是 12 章中的有效性指标的扩展。在扩展这些指标时，我们必须考虑前面章节中排名方法所考虑的同样的问题，特别是有关重叠的问题。

16.5.1 测试集

迄今为止，INEX 只关注大规模 XML 信息检索的评价方法。截至 2006 年，INEX 上的主要检索集由 12 000 多篇文章组成，它们取自 1995 年到 2002 年间的 IEEE 计算机协会的杂志和期刊。图 16-1 显示了一个取自该集合的典型文档的例子（尽管 INEX 集合中采用的标记差别不大）。集合总大小约为 500 MB。在 2006 年，收集维基百科全书的文章得到了一个更大的集合。这个新集合由超过 600 000 篇英文文章组成，其总大小超过 4 GB。Denoyer 和 Gallinari（2006）描述了该集合的创建和组织情况。近几年，为进行一系列特定的 INEX 实验建立起了第三个集合。该集合包含 50 000 多册已过版权保护期的书籍，其总大小超过 50 GB（Wu 等人，2008）。

在 INEX 上的主题创建和评判一直以来就归功于所有参与者的共同努力。每个参与小组创建 2 或 3 个主题，然后 INEX 组织者对这些主题进行收集和审查。完成主题设定以后，组织者将它们分派给各个小组，这些小组会把 XML 元素的有序列表返回给组织者。评判池由各个小组提交的运行实例组成。使用组织者提供的评判界面，参与小组用评判池对他们的结果进行评判。基于这些评判结果，组织者计算有效性指标，这些指标将成为官方结果。

16.5.2 有效性指标

重叠会导致出现大量的检索评价问题，自创建 INEX 以来，组织者和参与者就在全力克服这些问题（Kazai 和 Lalmas，2006）。尽管已经取得了很显著的进步，但这些问题还没有得到完全解决。另外，由于 XML 信息检索的目标是找到那些完全覆盖一个主题但尽可能少包含无关信息的元素，因此对独立元素进行简单的“相关”与“不相关”判断是不够的。能够发展出一些在满足 XML 信息检索的需求同时保持判断过程的简便性的指标，这方面已经做出了多种尝试。

其中一种方法是：INEX 组织者根据两个方面判断相关性（Piwowarski 和 Lalmas，2004）。每个元素都按照这两个方面分别进行判断。第一个方面是**详尽性**（exhaustivity），反映元素涵盖主题的程度。第二个方面是**特异度**（specificity），反映元素对某个主题的专注程度。两个指标都采用四分级制。因此，a（3，3）元素具有高详尽性和高特异度，a（1，3)元素具有低详尽性和高特异度，a（0，0）元素是不相关的。

人们开始努力使已有的有效性指标中也采用这两种指标。其中一个尝试是：早期的 INEX 会议中采用了 2.3 节中描述的平均查准率均值（MAP）的一个版本作为指标，该版本的 MAP 做了如下调整：通过使用各种**量化函数**（quantization function）根据详尽性和特异度的值为不同元素赋予不同的权重值。如，**严格量化**（strict quantization）函数给（3，3）元素赋予的权重值为 1，其余元素全为 0。这个变形本质上就是标准 MAP 值，将（3，3）元素看做“相关”，把其他元素看做“不相关”。其他量化函数设计成对元素进行部分评分，这是由于元素缺乏详尽性和/或特异度而被列为“近似忽略”的元素。**泛化量化**（generalized quantization）函数和**特异度导向泛化**（specificity-oriented generalization，sog）函数都按照“元素的相关度”来给元素评分（Kazai 等人，2004），第二个函数更偏向于特异度。

遗憾的是，该版本的 MAP 并未处理重叠现象。特别是，即使（3，3）元素包含具有很高得分的元素，泛化量化函数和 sog 量化函数也还是给予评分。为解决这一问题，Kazai 等人（2004）以及 Kazai 和 Lalmas（2006）开发出了一种称为 XCG 的 XML **累计增益**（XML cumulated gain）的指标，它扩展了 12.5.1 节中的 nDCG 这一指标。XCG 指标把有序列表的累计增益与理想增益向量进行对比。通过消除重叠元素并且只留下给定路径上的最优元素，利用相关性判断构造出这个理想增益向量。因此，XCG 指标偏向于那些能避免重叠元素的检索运行实例。

最近，INEX 会议的参与者通过标记文章中的相关部分来进行相关性判断。当评价元素的有序列表时，可以通过计算检索到的标记文本部分占全部标记文本的比率来计算查全率，通过计算检索到的标记文本占全部检索到的文本的比率来得到查准率。

16.6 延伸阅读

XML 是一个重要的 Web 标准，其重要性已超出了信息检索的范围。大量并不断增多的应用均使用 XML 进行数据存储和数据交换。例如，XML 是 Microsoft Office 2007 和 OpenOffice 办公套件的主要文档格式。大多数主流数据库的供应商现在都通过对传统关系数据库功能进行扩展来实现对 XML 的支持。可以从各个层次认识 XML，从基本层次（Tittel 和 Dykes，2005）到专业层次（Evjen 等人，2007）。

在 20 世纪 90 年代后期，数据库研发人员开发并评价了大量关于 XML 查询语言的方案，其中最著名的有：Stanford 大学的 Lorel 语言（Abiteboul 等人，1997），来自 INRIA

的 YATL 语言（Cluet 等人，2000），还有 Quilt 语言（Chamberlin 等人，2000）。几年之间该项研究被整合为 XQuery 语言的初级标准，尽管 1.0 版本的 XQuery 语言到 2007 年才发布。[㊀]Melton 和 Buxton（2006）为 XPath、XQuery 和其他查询 XML 的方法提供了一个可读并非常详细的解释。

XQuery 1.0 标准不具备更新 XML 文档的功能。这些功能都是在 XQuery Update Facility 1.0 后加上去的，并且在 2008 年 8 月由 W3C 发布。[㊁]最近 W3C 为“纯文本”扩展到 XQuery 和 XPath 中提供了一些指导性意见。[㊂]这些意见包括支持关键词、布尔搜索和对元素排名提供有限支持。

需要专门索引来有效地解决路径表达式的问题。Al-Khalifa 等人（2002）描述了一种**结构化索引**（structural index），用于有效地匹配查询和文档树。Zhang 等人（2001）提出一些方法来解决传统关系数据库系统中内容的路径表达式子集的问题。Bruno 等人（2002）对该项工作做了实质性的扩展，提出了一种算法来支持小型查询树（称为小枝）与一棵表示 XML 文档集的大很多的树的匹配。他们称这个算法为**整枝连接**（holistic twig join），表示该方法是通过采用倒排列表来解决路径表达式问题的基本方法。在 16.3 节中提出的为倒排列表扩展的 ADT 就基于他们的数据结构的。

后来的那些解决路径表达式问题的研究都是基于整枝连接算法的。Jiang 等人（2003）提出了一种提高整枝连接算法效率的方法。Kaushik 等人（2004）将整枝连接算法和结构化连接算法相结合以便更好地利用可用的结构化信息。Lu 等人（2005）提出了另外一些用于整枝连接处理的提高和扩展的算法。Gottlob 等人（2005）在树匹配之外进行了一些探索，深入探索 XPath 查询的语义细节问题。Zhang 等人（2006）提出了一种基于特征的文档子结构索引，该方法用于为路径表达式快速找出候选结果集。

在数据库领域，排名检索通常被称为“top-k”检索。Theobald 等人（2008）就对 XML 和其他半结构化数据，包括支持 XML 纯文本扩展，进行 top-k 检索，提出了一些索引结构和查询处理方法。Trotman（2004）就对 XML 和其他结构性文档进行检索排名提出了其他解决方法。

在 INEX 组织者和参与者的共同努力下，XML 信息检索的研究得到了飞速发展。[㊃]现在 INEX 上的项目已从基本检索任务扩大到探索如实体排名、问答、数据挖掘、段落检索和链接分析等的主题了。近期 INEX 的一系列工作主要是 Fuhr 等人（2007，2008）最新的研究工作。

其他的 XML 信息检索研究都是解决更广泛的问题的，主要是权衡内容和结构之间的平衡。Amer-Yahia 等人（2005）描述并评价了一些用于 XML 信息检索的高效的查询处理方法，该方法考虑了异类集合上内容和结构的平衡。Kamps 等人（2006）验证了如何最好地表达跟内容和结构都有关的查询。Lehtonen（2006）研究了有关异类集合上进行 XML 信息检索的问题。Chu-Carroll 等人（2006）提出了一种在 XML 集合上进行语义搜索的基于 XML 的查询语言。Amer-Yahia 和 Lalmas（2006）对截至 2006 年的 XML 信息检索研究进行了一个简短的调查。

关于 XML 信息检索的评价方法的研究工作仍在进行。Kazai 等人（2004）针对 INEX 检索评价中的重叠问题给出了一个详细的阐述，并提出了大量潜在的评价方法。关于详尽性

㊀ www.w3.org/TR/xquery

㊁ www.w3.org/TR/xquery-update-10

㊂ www.w3.org/TR/xpath-full-text-10/

㊃ www.inex.otago.ac.nz

和特异度的更进一步信息可从 Piwowarski 和 Lalmas（2004）那里得到，他们对这些概念的基本原理进行了详尽的阐述。Kazai 和 Lalmas（2006）对 nDCG 评价方法应用于 XML 信息检索评价中的问题进行了精确的分析。Piwowarski 等人（2008）对 XML 信息检索评价进行了深入的讨论，从对 INEX 的学习中总结了大量的经验教训。Ali 等人（2008）为 XML 信息检索的评价提出了一个框架，他们认为当考虑重叠所产生的影响时，应考虑到用户可能会浏览历史记录。

理想的 XML 信息检索测试集的单一性应小于两个主要的 INEX 集合：IEEE 集合和维基百科集合。对这两个集合来说，一个 DTD 就能描述一个集合所包含的所有文章。这些集合中包含文章的大小都差不多。每篇文章都小到足以在很短时间内将其全部阅读完毕。理想的 XML 信息检索测试集应该包含大量不同的文档，这些文档的大小不同，且具有不同的 DTD。这样的集合应该包含书籍和其他较长的文档以及短一些的文章。在将来或许会有满足这些条件的 XML 书籍集（Wu 等人，2008；Koolen 等人，2009）。

16.7 练习

练习 16.1 为图 1-3 中的莎士比亚戏剧的结构创建一个与其相符的 DTD。

练习 16.2 通过修改图 16-6 中的第 5～7 行的代码跳过段落所包含的章节，从而可以在求解路径表达式 /article/body/section/p 时不考虑深度。请写出该修改算法。你的版本比图 16-6 中的版本更高效吗？

练习 16.3 请写出解决路径表达式/article/body/section/section 的一个算法，假设 DTD 如图 16-2 所示，倒排索引 ADT 如 16.3 节所述。

练习 16.4 请写出解决路径表达式/article/body/section [2] 的一个算法，假设 DTD 如图 16-2 所示，倒排索引 ADT 如 16.3 节所述。

16.8 参考文献

Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. (1997). The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88.

Al-Khalifa, S., Jagadish, H. V., Patel, J. M., Wu, Y., Koudas, N., and Srivastava, D. (2002). Structural joins: A primitive for efficient XML query pattern matching. In *Proceedings of the 18th IEEE International Conference on Data Engineering*, pages 141–152.

Ali, M. S., Consens, M. P., Kazai, G., and Lalmas, M. (2008). Structural relevance: A common basis for the evaluation of structured document retrieval. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 1153–1162. Napa, California.

Amer-Yahia, S., Koudas, N., Marian, A., Srivastava, D., and Toman, D. (2005). Structure and content scoring for XML. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 361–372. Trondheim, Norway.

Amer-Yahia, S., and Lalmas, M. (2006). XML search: Languages, INEX and scoring. *SIGMOD Record*, 35(4):16–23.

Bruno, N., Koudas, N., and Srivastava, D. (2002). Holistic twig joins: Optimal XML pattern matching. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 310–321. Madison, Wisconsin.

Chamberlin, D., Robie, J., and Florescu, D. (2000). Quilt: An XML query language for heterogeneous data sources. In *Proceedings of WebDB 2000 Conference*, pages 53–62.

Chu-Carroll, J., Prager, J., Czuba, K., Ferrucci, D., and Duboue, P. (2006). Semantic search via XML fragments: A high-precision approach to IR. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 445–452. Seattle, Washington.

Clarke, C. L. A. (2005). Controlling overlap in content-oriented XML retrieval. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 314–321. Salvador, Brazil.

Cluet, S., Siméoni, J., and De Voluceau, D. (2000). YATL: A functional and declarative language for XML. Bell Labs, Murray Hill, New Jersey.

Denoyer, L., and Gallinari, P. (2006). The Wikipedia XML corpus. *ACM SIGIR Forum*, 40(1):64–69.

Evjen, B., Sharkey, K., Thangarathinam, T., Kay, M., Vernet, A., and Ferguson, S. (2007). *Professional XML (Programmer to Programmer)*. Indianapolis, Indiana: Wiley.

Fuhr, N., and Großjohann, K. (2001). XIRQL: A query language for information retrieval in XML documents. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 172–180. New Orleans, Louisiana.

Fuhr, N., Kamps, J., Lalmas, M., and Trotman, A., editors (2008). *Focused Access to XML Documents: Proceedings of the 6th International Workshop of the Initiative for the Evaluation of XML Retrieval*, volume 4862 of *Lecture Notes in Computer Science*. Berlin, Germany. Springer.

Fuhr, N., Lalmas, M., Malik, S., and Szlávik, Z., editors (2005). *Advances in XML Retrieval: Proceedings of the 3rd International Workshop of the Initiative for the Evaluation of XML Retrieval*, volume 3493 of *Lecture Notes in Computer Science*. Berlin, Germany. Springer.

Fuhr, N., Lalmas, M., and Trotman, A., editors (2007). *Proceedings of the 5th International Workshop of the Initiative for the Evaluation of XML Retrieval*, volume 4518 of *Lecture Notes in Computer Science*.

Gottlob, G., Koch, C., and Pichler, R. (2005). Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems*, 30(2):444–491.

Hockey, S. (2004). The reality of electronic editions. In Modiano, R., Searle, L., and Shillingsburg, P. L., editors, *Voice, Text, Hypertext: Emerging Practices in Textual Studies*, pages 361–377. Seattle, Washington: University of Washington Press.

Jiang, H., Wang, W., Lu, H., and Yu, J. X. (2003). Holistic twig joins on indexed XML documents. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 273–284. Berlin, Germany.

Kamps, J., de Rijke, M., and Sigurbjörnsson, B. (2004). Length normalization in XML retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 80–87. Sheffield, England.

Kamps, J., Marx, M., de Rijke, M., and Sigurbjörnsson, B. (2006). Articulating information needs in XML query languages. *ACM Transactions on Information Systems*, 24(4):407–436.

Kaushik, R., Krishnamurthy, R., Naughton, J. F., and Ramakrishnan, R. (2004). On the integration of structure indexes and inverted lists. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 779–790. Paris, France.

Kazai, G., and Lalmas, M. (2006). eXtended cumulated gain measures for the evaluation of content-oriented XML retrieval. *ACM Transactions on Information Systems*, 24(4):503–542.

Kazai, G., Lalmas, M., and de Vries, A. P. (2004). The overlap problem in content-oriented XML retrieval evaluation. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 72–79. Sheffield, England.

Kekäläinen, J., Junkkari, M., Arvola, P., and Aalto, T. (2004). TRIX 2004 — Struggling with the overlap. In *Proceedings of INEX 2004*, pages 127–139. Dagstuhl, Germany. Published in LNCS 3493, see Fuhr et al. (2005).

Koolen, M., Kazai, G., and Craswell, N. (2009). Wikipedia pages as entry points for book search. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining*.

Lehtonen, M. (2006). Preparing heterogeneous XML for full-text search. *ACM Transactions on Information Systems*, 24(4):455–474.

Lu, J., Ling, T. W., Chan, C. Y., and Chen, T. (2005). From region encoding to extended Dewey: On efficient processing of XML twig pattern matching. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 193–204. Trondheim, Norway.

Mass, Y., and Mandelbrod, M. (2003). Retrieving the most relevant XML components. In

Advances in XML Retrieval: Proceedings of the 3rd International Workshop of the Initiative for the Evaluation of XML Retrieval, number 3493 in Lecture Notes in Computer Science, pages 53–58. Berlin, Germany: Springer.

Mass, Y., and Mandelbrod, M. (2004). Component ranking and automatic query refinement for XML retrieval. In *Proceedings of INEX 2004*, pages 53–58. Dagstuhl, Germany. Published in LNCS 3493, see Fuhr et al. (2005).

Melton, J., and Buxton, S. (2006). *Querying XML*. San Francisco, California: Morgan Kaufmann.

Pehcevski, J., Thom, J. A., and Vercoustre, A. (2004). Hybrid XML retrieval re-visited. In *Proceedings of INEX 2004*, pages 153–167. Dagstuhl, Germany. Published in LNCS 3493, see Fuhr et al. (2005).

Piwowarski, B., and Lalmas, M. (2004). Providing consistent and exhaustive relevance assessments for XML retrieval evaluation. In *Proceedings of the 13th ACM Conference on Information and Knowledge Management*, pages 361–370. Washington, D.C.

Piwowarski, B., Trotman, A., and Lalmas, M. (2008). Sound and complete relevance assessment for XML retrieval. *ACM Transactions on Information Systems*, 27(1):1–37.

Theobald, M., Bast, H., Majumdar, D., Schenkel, R., and Weikum, G. (2008). Topx: Efficient and versatile top-k query processing for semistructured data. *The VLDB Journal*, 17(1):81–115.

Tittel, E., and Dykes, L. (2005). *XML for Dummies* (4th ed.). New York: Wiley.

Trotman, A. (2004). Searching structured documents. *Information Processing & Management*, 40(4):619–632.

Trotman, A., and Sigurbjörnsson, B. (2004). Narrowed Extended XPath I (NEXI). In *Proceedings of INEX 2004*. Dagstuhl, Germany. Published in LNCS 3493, see Fuhr et al. (2005).

van der Vlist, E. (2002). *XML Schema: The W3C's Object-Oriented Descriptions for XML*. Sebastopol, California: O'Reilly.

Vittaut, J., Piwowarski, B., and Gallinari, P. (2004). An algebra for structured queries in Bayesian networks. In *Proceedings of INEX 2004*, pages 100–112. Dagstuhl, Germany. Published in LNCS 3493, see Fuhr et al. (2005).

Wu, H., Kazai, G., and Taylor, M. (2008). Book search experiments: Investigating IR methods for the indexing and retrieval of books. In *Proceedings of the 30th European Conference on Information Retrieval Research*, pages 234–245.

Zhang, C., Naughton, J., DeWitt, D., Luo, Q., and Lohman, G. (2001). On supporting containment queries in relational database management systems. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 425–436. Santa Barbara, California.

Zhang, N., Özsu, M. T., Ilyas, I. F., and Aboulnaga, A. (2006). FIX: Feature-based indexing technique for XML documents. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 259–270. Seoul, South Korea.

第六部分 附 录

附录 A

Information Retrieval: Implementing and Evaluating Search Engines

计算机性能

本书第二部分中的章节列举了大量与倒排索引相关的技术性能数据。为了使这些数据有合适的说明，我们给出了计算机性能方面一些重要因素的简要概述，并总结了本书中用于实验的计算机系统的关键性能指标（见表 A-1）。

表 A-1 用于本书实验的计算机系统的关键性能指标。系统有两个硬盘驱动器，采用 RAID-0（硬盘分割（striping））方式排列。第二部分中的索引/检索实验中的所有磁盘操作都是基于 RAID-0 的

CPU	
型号	1×AMD Opteron 154，2.8 GHz
数据缓存	64 KB (L1)，1024 KB (L2)
TLB 缓存	40entries (L1)，512entries (L2)
执行流水线	12 步
磁盘	
盘大小	2×465.8 GB
平均旋转延迟	4.2 ms (7200 rpm)
平均寻道延迟	8.5 ms
平均随机访问延迟	12.8 ms (≈3600 万个 CPU 周期)
顺序读/写吞吐量（单个磁盘）	43.3 MB/s
顺序读/写吞吐量（RAID-0）	87.4 MB/s
内存	
总大小	8048 MB
随机存取延迟	75 ns (≈270 个 CPU 周期)
顺序读/写吞吐量	3700 MB/s

信息检索系统，如大部分数据库系统一样，花费大量时间将数据从计算机的一部分转移到另一部分。它们的性能主要受两个因素的影响：数据访问局部性和流水线执行。

数据局部性在这里是指顺序访问与随机访问。顺序数据访问模式的一个例子就是第 5.1.2 节中提到的析取查询中的 term-at-a-time 的查询处理策略。随机数据访问的一个例子就是第 4.2 节中的有序词典，其中查询操作是通过二分查找实现的。无论我们将索引存储在内存还是磁盘，性能通常都会提高，我们可以通过重排数据来便于顺序访问。

流水线执行是现代微处理器的工作特点。它允许 CPU 在处理一条指令 I_n 的同时处理另一条指令 I_{n+1}。为了实现这一点，CPU 需要在完成 I_n 之前知道 I_{n+1} 的地址。如果 I_n 是分支指令（即“IF”语句），这就比较难处理了。

A.1 磁盘上的顺序访问和随机访问

对硬盘驱动器而言，在读出数据之前必须让驱动器的读写头指向数据所在的位置，因此顺序访问和随机访问的区别是十分明显的。这个定位操作包含两个阶段：第一阶段先将读写头定位到存储该数据的磁道上（**寻道延迟**（seek latency））；第二阶段等待磁盘转到正确位置，使我们要读的数据恰好在读写头所指位置（**旋转延迟**（rotational latency））。

平均旋转延迟取决于硬盘驱动器的转动速度。如果磁盘每分钟转动 7200 转（rpm），旋转一周用时 1/120 s≈8.3 ms。为读取磁盘任意位置上的数据，需要平均等待磁盘转动半周的时间——约为 4.2 ms。

索引操作的平均寻道延迟取决于索引的规模。较小的索引占用较少的磁道数，因此连续读操作之间磁头需要更少的移动距离。例如，尽管我们实验中用到的硬盘平均随机访问延迟约为 12.8ms（寻道延迟：8.6 ms；旋转延迟：4.2 ms），但是在 GOV2 文档集上进行的词典交错实验中平均查找延迟仅为 11.4 ms（表 4-4）——因为索引仅占用了磁盘 13%的空间。[⊖]

A.2 RAM 中的顺序访问和随机访问

对于内存来说，顺序访问和随机访问的差别并不十分明显，并且实际上我们通常把主存视为随机访问内存（random access memory），这意味着即使存在差别也是可以忽略的。然而，事实并非如此。

使用缓存可以加快 CPU 和主存之间的数据传输。在这里介绍两种最相关的缓存：

- **数据缓存**（data cache）。该缓存位于 CPU 和主存之间，存放最近使用过的数据的副本，以**缓存行**（cache lines）的形式进行排列，每行通常存放 64 字节的数据。缓存行代表了计算机主存和缓存通信的粒度。如果一个程序从 RAM 中读一个字节数据，整个缓存行都要被加载到缓存中，这样在后续操作中，同一行的其他字节的处理速度就会提高。数据缓存通常不是单个缓存，而是一个层级结构的缓存，如层级-1（L1），层级-2（L2），以此类推。L1 缓存位于最顶层，在所有层次中最快，但同时也最小。
- **后备转换缓冲器**（TLB）。实际上所有现代微处理器都支持虚拟内存，也就是程序访问的数据地址（虚拟地址，virtual address）不同于它在主存中的实际存储地址（物理地址，physical address）。虚拟地址和物理地址之间的转换是通过**页表**（page table）实现的；转换的粒度就是一个**页面**（page），大小通常为 4 KB 或 8 KB。

一种虚拟内存的简单实现是，每次内存访问需要两个物理操作：一个用于页表（将虚拟地址转换为物理地址），另一个用于实际的数据访问。为避免导致性能下降，处理器采用一类特殊的缓存，就是 TLB，它保存最近访问的页表的入口地址的副本。

不论是数据丢失还是 TLB 丢失，缓存丢失的代价是相当昂贵的，而且很容易耗费大量的 CPU 周期。在最坏情况下，一次内存访问能同时导致数据缓存丢失和 TLB 丢失。对于我们实验中所使用的计算机而言，这样的双重丢失会导致约为 75 ns 的时间损耗，足够顺序读出大约 300 字节的数据了。

值得指出的是，很多随机访问索引操作会导致数据/TLB 双重缓存丢失。这就是为什么第 6.4 节中提出的分组技术不仅能降低搜索引擎词典的规模，同时还能提高查找性能的原因

⊖ 然而，请注意，文件系统（ext3）并不能严格保证索引在磁盘上是连续的。很有可能索引不是严格连续的，但如果能保证索引是连续的，可以实现更低的延迟。

(见表 6-12)，因为它降低了在二分查找阶段每次词典查找的随机内存访问的次数。

A.3 流水线执行和分支预测

自 20 世纪 80 年代中期开始，流水线执行就成了 CPU 设计的标准技术，当时 Intel 公司推出了 i386 系列微处理器。其基本思想就是，一条机器指令不会同时占用 CPU 的各个部分。因此，通过流水安排执行的各个阶段，可以提高利用率和总体性能。

考虑最简单的三阶段流水线：

1) 取指令。从主存或缓存中加载指令到 CPU。

2) 解码。将指令的内存形式翻译成 CPU 可以执行的微指令。

3) 执行。处理器执行指令并更新受影响的 CPU 寄存器和内存位置。

如果运行一个没有分支指令的线性程序，那么流水线的 3 个阶段可一直保持工作状态。然而，如果现在流水线中的一个指令，如 I_n，是一个分支指令，那么处理器就不知道接下来的指令 I_{n+1}和 I_{n+2}的地址了，这样就不能在执行完 I_n 之前对它们进行加载了。这就产生了流水线**停顿**（stalled）。

现代处理器通过使用**分支预测**（branch prediction）技术来解决这一问题，该方法基于过去的分支行为尝试预测即将调用的是给定分支指令中的 IF 分支还是 ELSE 分支。基于这一预测，它们将接下来可能执行的指令加载到流水线中。一旦预测失败，丢弃对指令 I_{n+1} 和 I_{n+2}进行的工作，并将另外的指令加载到 CPU 中，这将导致流水线部分空转。

预测准确率通常可高达 90% 以上，但是对特定操作类型的预测准确率可能会低很多。大家可能还记得第 6.3.1 节中的 γ 编码。这种编码的一种简单译码方法是通过检查压缩位序列来处理每个码字的一元部分，每次处理 1 位，直至处理到第一个 $\overline{1}$ 位为止，大致如以下流程：

```
1  bitPos ← 0
2  for i ← 1 to n do
3      k ← 1
4      while bitSequence[bitPos] = 0 do
5          bitPos ← bitPos + 1
6          k ← k + 1
7      取接下面k位来对当前码字进行译码
8      bitPos ← bitPos + k
```

第 2 行的条件跳转是没有问题的，因为处理器能迅速认识到不应该跳出循环（假设 n 足够大）。然而，要预测第 4 行的分支就比较复杂了。考虑一个已编码位置信息序列，其中 γ 编码的一元部分为 1（即位序列 $\overline{1}$）代表位置信息的一半，2（即位序列$\overline{01}$）表示另一半位置信息。然后 CPU 大约会用 1/3 的时间在第 4 行和第 5 行之间运行，用 2/3 的时间执行第 8 行。最优的策略是始终进行分支预测（即假设 *bitSequence*[*bitPos*]为 $\overline{1}$）。它导致的预测失效率为 33%。

对于前面介绍的简单 3 阶段流水线而言，预测失效率为 33% 看起来还是不错的。然而，现代微处理器的执行流水线由十几或更多的阶段组成，重复的预测失效会大大地降低处理器的执行效率。表驱动解码减少了分支预测失效的次数，因此逐位解码程序更偏向采用这种方法。