TURING 图灵计算机科学丛书

信息检索导论

[美] Christopher D. Manning
Prabhakar Raghavan
[德] Hinrich Schütze
著

王斌 译

人民邮电出版社
北京

版 权 声 明

图书在版编目（CIP）数据

内 容 提 要

本书是一本讲授信息检索的经典教材。全书共 21 章，前八章详述了信息检索的基础知识，包括倒排索引、布尔检索及词项权重计算和评分算法等，后十三章介绍了一些高级话题，如基于语言建模的信息检索模型、基于机器学习的排序方法和 Web 搜索技术等。另外，本书还着重讨论了文本聚类技术这一信息检索中不可或缺的组成部分。全书语言流畅，由浅入深，一气呵成。

本书适合作为高等院校相关专业高年级本科生和研究生的课程教材，也可供信息检索领域的研究人员和专业人士参考。

图灵计算机科学丛书

信息检索导论

◆ 著 [美] Christopher D. Manning, Prabhakar Raghavan
[德] Hinrich Schütze
译 王斌
责任编辑 杨海玲
执行编辑 罗词亮 陈潇

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京****印刷有限公司印刷

◆ 开本：
印张：
字数： 2010 年*月第 1 版
印数：1-3 000 册 2010 年**
著作权合同登记号 图字：01-2009-7281 号
ISBN 978-

定价：**元

读者服务热线：（010）51095186 印装质量热线：（010）67129223
反盗版热线：（010）67171154

符号对照表

符　号	原　书	含　义
γ	90	γ编码
γ	237	$\gamma(d)$表示分类或者聚类函数：$\gamma(d)$是d所属的类或者簇
Γ	237	第13、14章中的有监督学习方法：$\Gamma(\lambda)$是从训练集λ上学到的分类函数γ
λ	370	特征值
$\vec{\mu}(.)$	269	类质心（在Rocchio分类中）或簇质心（在K-均值和质心聚类中）
Φ	105	训练样本
σ	374	奇异值
$\Theta(.)$	10	算法复杂度的紧上界
ω, ω_k	328	聚类结果中的一个簇
Ω	328	聚类结果或簇集合$\{\omega_1,\ldots, \omega_K\}$
$\text{argmax}_x f(x)$	164	使 函数f取最大值的x的值
$\text{argmin}_x f(x)$	164	使 函数f取最小值的x的值
c, c_j	237	分类中的一个类别
cf_t	82	词项t的文档集频率（该词项在整个文档集中出现的总次数）
$\amalg$	237	类别集合$\{c_1,\ldots,c_J\}$
C	248	取值为类别集合$\amalg$中元素的随机变量
$\boldsymbol{C}$	369	词项–文档矩阵
d	4	文档集D中的第d篇文档的索引号
d	65	一篇文档
$\vec{d}, \vec{q}$	163	文档向量及查询向量
D	326	所有文档的集合$\{d_1,\ldots,d_N\}$
D_c	269	类别c中的文档集
$\tilde{\lambda}$	237	第13～15章中的已标记文档集$\{<d_1,c_1>,\ldots,<d_N,c_N>\}$，即训练集
df_t	108	词项t的文档频率（文档集中出现t的文档数目）
H	91	熵
H_M	93	第M个调和数
$I(X;Y)$	252	随机变量X和Y的互信息
idf_t	108	词项t的逆文档频率
J	237	类别数目
k	267	集合中排名前k的元素，如kNN中的前k个邻居、检索文档的前k个结果以及词汇表V中选出的前k个特征
k	50	k个字符组成的序列
K	326	簇的个数
L_d	214	文档d的长度（以词条为单位计数）
L_a	242	测试文档或应用文档的长度（以词条为单位计数）
L_{ave}	64	文档的平均长度（以词条为单位计数）

（续）

符　号	原　书	含　义
M	4	词汇表大小（即$\|V\|$）
M_a	242	测试文档或应用文档的词汇量
M_{ave}	71	文档集中每篇文档的平均词汇量
M_d	218	文档d的模型
N	4	检索或训练文档集中的文档数目
N_c	240	类别c中的文档数目
$N(\omega)$	275	事件ω发生的次数
$O(.)$	10	算法复杂度的界
$O(.)$	203	事件的优势率
P	142	正确率
$P(.)$	202	概率
P	425	转移概率矩阵
q	55	查询
R	143	召回率
s_i	53	字符串
s_i	103	域评分布尔值
$\mathrm{sim}(d_1,d_2)$	111	文档d_1和d_2的相似度
T	40	文档集中所有词条的数目
T_{ct}	240	词t在c类文档中的出现次数
t	4	词汇表V中第t个词项的索引号
t	56	词汇表中的一个词项
$\mathrm{tf}_{t,d}$	107	词项t在文档d中的出现频率（即t在d中的出现次数）
U_t	246	表示词项t存在与否的随机变量，当t存在时，值为1，否则为0
V	190	文档中的所有词项$\{t_1,\ldots,t_M\}$组成的词汇表（也称为词典lexicon）
$\vec{v}(d)$	111	文档d经长度归一化后的文档向量
$\vec{V}(d)$	110	文档d未经长度归一化的文档向量
$\mathrm{wf}_{t,d}$	115	词项t在文档d中的权重
w	103	权重，比如域的权重或者词项的权重
$\vec{w}^T\vec{x}=b$	269	超平面方程：$\vec{w}$是超平面的法向量，w_i是$\vec{w}$的第i个分量
$\vec{x}$	204	基于词项表示的文档向量$\vec{x}=(x_1,\ldots,x_M)$，更一般地说，为文档的特征表示
X	246	取值为词汇表V中元素的随机变量（比如，某个文档位置k上的词）
□	237	文本分类中的文档空间
$\|A\|$	56	集合A的势：集合A中的元素个数
$\|S\|$	570	方阵S的行列式
$\|s_i\|$	53	s_i的长度（以字符计）
$\|\vec{x}\|$	128	向量$\vec{x}$的大小
$\|\vec{x}-\vec{y}\|$	121	向量$\vec{x}$、$\vec{y}$的欧氏距离，也即向量$(\vec{x}-\vec{y})$的大小

译　者　序

第一次见到这本书的电子版是在 2007 年的年底 ,当时北京大学的闫宏飞博士向我推荐了这本书。从网上下载书稿的电子版之后，我便迫不及待地在一周时间内通读了这本书。读完之后便萌发了翻译这本书的冲动，随后我就联系作者、联系剑桥大学出版社并通过朋友寻找获得授权的国内出版社。辗转数月之后，我被告知该书已经交由其他学者翻译，很快便可出版。听到这个消息，虽然我有些遗憾，但也算是心里的一块石头得以落地。所以，当去年 8 月人民邮电出版社突然联系并询问我是否有意翻译这本著作时，我心里的惊讶可想而知。当然，惊讶之余我毫不犹豫地接受了这份邀请，并从此开始了长达数月的翻译历程。

之所以愿意翻译这本书不仅仅是由于该书的作者都是学术界甚至业界鼎鼎大名的人物，更主要的是因为本书在内容和组织上都有独到之处。之前也有很多信息检索方面的教材，但是其中很多内容已经过时。信息检索是一门不断发展并和其他领域、技术不断融合的学科。这本书补充了一些近年来受到广泛关注的新内容。比如：基于语言建模的信息检索模型、基于机器学习的排序方法、检索结果的 Snippet 生成、聚类标签生成、XML 检索、搜索广告、网页作弊等等。除此之外，本书每章末尾的“ 参考文献及补充读物” 一节也给出了相关技术的最新进展。本书在内容上与传统教材的另一个显著不同之处是加大了文本分类/聚类技术的介绍篇幅，实际上这些技术已经成为当代信息检索不可分割的一部分。另一方面，本书在深度上超过了大部分传统教材。在介绍信息检索技术的同时，本书深入介绍了其背后所依赖的原理。因此，本书不仅可以用作信息检索领域的入门教材，还能满足对该领域进行深入研究的需要。另外，本书给出了很多实际当中的运行算法和实施细节，这些内容对于信息检索技术的实际应用有很好的参考价值。最后值得一提的是，本书在结构上也进行了巧妙构思。首先通过一个例子引出基本技术，然后通过基本技术的不断增强来介绍信息检索的其他技术。全书浑然一体，读起来也有一气呵成的感觉。

这么一本优秀的著作在给译者的翻译带来无穷动力的同时，无疑也给翻译带来了无形的压力。为了尽量保证每章译稿的质量并保持译文的前后一致性，整本书的初译工作全部由译者本人独立完成，在翻译过程中译者也阅读了大量相关的教材和论文，并前后进行了六次自我校对。在校对过程中，有很多学术界同仁也提出了很多宝贵的意见和建议。他们包括：中科院研究生院的朱廷劭教授、中科院自动化所的赵军研究员、中科院软件所的孙乐研究员、复旦大学的黄萱菁教授、江西师范大学的王明文教授、江西财经大学的刘德喜博士、北京大学的闫宏飞博士、何靖博士、清华大学的张敏博士、北京语言大学的徐燕博士等等。译者所在的中科院计算所信

息检索课题组及选修研究生院《现代信息检索》课程的部分学生也提出了大量修改建议，他们是：郎皓、李亚楠、顾智宇、李鹏、李锐、马宏远、张爱华、蒋在帆、沈沉、史亮、卫冰洁、崔雅超、赵琴琴、李恒训、袁平广、邱泳钦、李丹、鲁凯、徐飞、张帅、张启龙、廖凤、钟进文、朱亮、赵娟等等。对于他们无私的帮助，我表示由衷的感谢。感谢我所在的前瞻研究实验室主任李锦涛老师对我的翻译工作给予的支持和肯定。当然，本书的翻译工作得以顺利完成，特别要感谢人民邮电出版社众多工作人员特别是责任编辑杨海玲女士在各方面的支持和帮助。另一个需要感谢的是我的妻子，在前前后后近八个月当中，除上班时间完成自己的科研工作外，我几乎所有的业余时间都用在翻译和校对上，而她却默默地承担起两岁的儿子的所有抚育责任。

翻译的过程中，我还有幸与原文的第二作者 Prabhakar Raghavan 教授进行了当面交流，他对我的翻译工作给予了极大鼓励。在与原文作者的邮件交流中，我也澄清了一些理解上的误区，并修正了原书中的多处错误。

虽然得到了众人的帮助，自己也算认真努力，但由于本人专业水平、理解能力和写作功底都十分有限，加上时间上仍显仓促，最后的译稿中一定存在不少理解上的偏差，译文也会有许多生硬之处。希望读者能不吝提出修改的意见和建议，以便对现有译稿不断改进，直至为国内信息检索领域的读者真正造福为止。来信请联系 wbxjj2008@gmail.com，对译稿的修改结果也会及时公布在网站 http://ir.ict.ac.cn/~wangbin/iir-book/上。原书的初稿电子版、相关课件、勘误表、论坛等信息也可以从网站 http://nlp.stanford.edu/IR-book/ information-retrieval-book.html 下载。

译　者

2010 年 5 月 3 日于中关村

目　录

前　言

研究表明，直到 20 世纪 90 年代，大多数人还是首选通过别人而不是使用信息检索系统来获取信息。当然，那时候大多数人也往往通过旅行社来安排自己的行程。然而，在过去的十年中，信息检索效果的不断优化已经使 Web 搜索引擎的质量达到了一个新的水平，大多数用户在大部分情况下都对搜索的结果感到满意。Web 搜索引擎已经成为用户发现和获取信息的常规和首选渠道。以统计数据为证，2004 年美国 Pew 研究中心的一项因特网调查（Fallows 2004）结果表明，有 92%的因特网用户认为因特网是人们获取日常信息的良好渠道。令很多人惊讶的是，信息检索也从一个以学术研究为主的领域，摇身一变而成为人们赖以获取日常信息的工具背后的基础学科。本书主要介绍该学科的核心理论基础，既考虑研究生科研的需求，也兼顾了高年级本科生学习的需求。

但是，信息检索并非始于 Web。在应对信息存取的各种挑战的过程中，信息检索逐渐发展成为一门为各种形式的内容搜索提供原理性方法的学科。信息检索起初主要面向科学文献和馆藏记录，但是很快就扩展到其他形式的内容，特别是新闻记者、律师、医生等特定领域专业人士所需的信息内容。信息检索中的很多学术研究都围绕上述内容展开，而其实践方面则主要是为公司或政府部门提供非结构化信息的获取服务，这些领域的研究和实践构成了本书的主要内容。

然而，近年来信息检索革新的主要推动力却来自万维网，因为网络上聚集了数以千万计的网络用户发布的内容。如果这些内容不能及时被发现、标注和分析，并为有需求的人们提供相关的、全面的信息，那么它们的存在将毫无意义。到 20 世纪 90 年代末，很多人逐渐意识到，由于 Web 的规模呈指数级增长，继续给整个 Web 建立索引很快会变得毫无可能。但是，卓越的科学创新、一流的工程水平、日益低廉的计算机硬件价格及 Web 搜索商业化基础的壮大等一系列因素，促成了当今主流搜索引擎的产生与成长。这些搜索引擎一天之内能够完成对数十亿网页的数亿次搜索请求，并且每次搜索都能够在亚秒级时间内返回高质量的结果。

本书组织结构及课程设计

本书是我们在斯坦福大学和斯图加特大学所讲授的一系列课程的教学成果总结。这些课程持续的时间从四分之一学期、半学期到一学期不等，主要面向低年级计算机专业的研究生，也曾用于高年级计算机专业的本科生和法律、医学信息学、统计、语言学及其他工程学科背景的学生的教学。因此，本书主要的写作原则是提供一个学期的信息检索研究生课程，并尽量覆盖

信息检索的内容重点。另一个原则是尽量让每章的内容能在约 75～90 分钟内讲授完。

本书前八章介绍信息检索的基础知识，特别是搜索引擎的核心理论。这八章对于任何信息检索课程来说都是核心部分。第 1 章主要介绍倒排索引，并说明如何通过这种索引实现简单的布尔查询。第 2 章介绍索引之前的文档预处理过程，并讨论在不同的功能和速度要求下对倒排索引进行改进的方法。第 3 章主要介绍词典搜索的数据结构，并给出查询存在拼写错误或者与被搜索的文档中的词汇不能精确匹配时的处理方法。第 4 章主要介绍基于文本集合构建倒排索引的几个算法，并着重介绍具有高扩展性的分布式算法，这类算法适用于大规模文档集的索引构建。第 5 章介绍词典和倒排索引的压缩技术，这些技术对于实现大型搜索引擎的亚秒级查询响应十分关键。第 1 章～第 5 章中介绍的索引和查询仅针对布尔检索（Boolean retrieval），即一篇文档和查询要么匹配，要么不匹配。那么，如何度量查询和文档的匹配程度，或者说如何根据文档和查询的匹配情况对结果打分呢？对这个问题的回答构成了第 6、第 7 章词项权重计算和评分算法的主要内容。也就说是，给定查询，我们可以利用这两章介绍的技术，按照文档评分的结果次序输出结果列表。第 8 章主要介绍信息检索系统的评价技术，即根据检索系统返回结果的相关性对不同系统进行评价，从而可以在基准文档集和查询上对不同系统的性能进行比较。

在前八章的基础上，本书的第 9 章到第 21 章涵盖了信息检索的一些高级话题。第 9 章介绍了相关反馈和查询扩展技术，其目的在于增加相关文档返回的可能性。第 10 章介绍了采用 XML 和 HTML 等标记语言的结构化文档的检索，这其中我们将结构化文档的检索进行约简，并采用第 6 章所介绍的向量空间模型进行求解。第 11 章和第 12 章介绍基于概率论的信息检索模型。其中，第 11 章介绍传统的概率检索模型，它提供了一个相关度计算框架，在给定一系列查询词项时，能够计算一篇文档与查询相关的概率。这个概率显然可以用于文档的评分和排序。第 12 章给出了另一种方法，即对文档集中的每篇文档建立一个语言模型，然后在每个模型下估计查询生成的概率。这个概率也显然可以用于文档的评分和排序。

第 13 章到第 18 章介绍了信息检索中各种形式的机器学习和数值方法。第 13 章到第 15 章主要关注文档分类的问题，即在给定一系列文档及其归属类别的前提下，将新文档分配到某个或者某几个类别中去。第 13 章首先指出统计分类是一个成功的搜索引擎所必需的关键技术之一，接着介绍了朴素贝叶斯算法（该算法概念虽然简单，但是文本分类的效率很高），最后给出了文本分类的评价技术。第 14 章将第 6 章所讲述的向量空间模型应用于文本分类，介绍了几种基于向量空间模型的分类方法，主要包括 Rocchio 和 kNN（*k* nearest neighbor）两种分类算法。本章最后给出了用于分类方法选择的偏差–方差折中准则，而偏差–方差折中也是学习问题的一个重要特点。第 15 章介绍了支持向量机，这是目前公认的效果最好的文本分类算法。另外，本章还将分类问题和一些看上去与文本分类无关的问题（比如如何从给定的训练集合中推导出检索的评分函数）联系起来。

第 16～18 章主要介绍文档的聚类技术。第 16 章在概述信息检索中的一些重要聚类应用的基础上，主要介绍了两个扁平聚类算法：*K*–均值算法和 EM 算法。前者是一个效率很高并被广

泛应用的算法；后者虽然计算复杂度高一些，但是灵活性更好。第 17 章介绍信息检索对层次聚类（而非扁平聚类）的应用需求，并介绍了一些能产生层次簇结构的聚类算法。这一章还探讨了自动生成聚类标签的难题。第 18 章介绍了一些线性代数的方法，它们是对聚类方法的扩展，并且为线性代数方法在信息检索的应用提供了极具吸引力的前景，其中最具代表性的方法是隐性语义索引。

第 19 章到 21 章主要介绍 Web 搜索这个具体的应用。第 19 章概述了 Web 搜索所面临的基本挑战，并给出了 Web 信息检索中的一些普遍使用的技术。第 20 章介绍了一个基本网络采集器的体系结构和必要需求。最后，第 21 章讨论了链接分析在 Web 搜索中的作用，其中用到了线性代数和高级概率论中的方法。

本书并没有囊括信息检索的所有主题，因为有些主题超出了信息检索入门课程的范围。当然，感兴趣的读者可以参见如下参考书籍。

Cross-language IR(跨语言检索)：Grossman and Frieder 2004，第 4 章；Oard and Dorr 1996。

Image and multimedia IR (图像和多媒体检索)：Grossman and Frieder 2004，第 4 章；Baeza-Yates and Ribeiro-Neto 1999，第 6、11、12 章；del Bimbo 1999；Lew 2001，Smeulders 等人 2000。

Speech retrieval (语音检索)：Coden 等人 2002。

Music retrieval (音乐检索)：Downie 2006 及网站 http://www.ismir.net/。

User interfaces for IR (信息检索中的用户界面)：Baeza-Yates and Ribeiro-Neto 1999，第 10 章。

Parallel and peer-to-peer IR (并行和 p2p 检索)：Grossman and Frieder 2004，第 7 章；Baeza-Yates and Ribeiro-Neto 1999，第 9 章；Aberer 2001。

Digital libraries (数字图书馆)：Baeza-Yates and Ribeiro-Neto 1999，第 15 章；Lesk 2004。

Information science perspective (基于信息科学视角的信息检索)：Korfhage 1997；Meadow 等人 1999；Ingwersen and Järvelin 2005。

Logic-based approaches to IR (基于逻辑的信息检索)：van Rijsbergen 1989。

Natural language processing techniques (自然语言处理技术)：Manning and Schütze 1999；Jurafsky and Martin 2008；Lewis and Jones 1996。

预备知识

所有 21 章都需要数据结构和算法、线性代数以及概率论的基本知识。为方便读者和教师使用本书，各章更具体的要求如下。

第 1 章到第 5 章需要数据结构和算法的基本知识。第 6、7 章还另外需要线性代数的知识，包括向量和内积的基本概念。第 8 章到第 10 章不需要其他的预备知识。第 11 章需要概率论的基础知识，其中，11.1 节简单介绍了第 11 章到第 13 章所需要的基本概念。第 15 章假定读者熟

悉非线性优化的基本概念，当然如果读者对非线性优化没有深入的了解，在阅读上也不会有太多的问题。第 18 章需要线性代数的基本知识，包括矩阵的秩、特征向量等概念，18.1 节对这些概念有一个简单的介绍。第 21 章还需要了解特征值和特征向量的概念。

书中的标记符号

本书正文中用铅笔符号（✎）标记例子。高级或者难度较大的章节用剪刀符号（✂）标记，习题用问号（？）标记，习题中分别用[*]、[**]、[***]符号标识“容易”、“难度适中”和“难度大”的习题。

致谢

首先感谢剑桥大学出版社允许本书的电子样稿在网上公布，各种反馈促进了本书的写作过程。感谢 Lauren Cowles，她是名出色的编辑，在本书样式、组织、覆盖面等方面提出了非常好的建议。如果说这本书最终达到了我们的写作预期的话，那么很大程度上应归功于她。

我们对那些在写作过程中对样稿提出建议和错误纠正意见的人们表示衷心的感谢。他们是 Cheryl Aasheim、Josh Attenberg、Luc Bélanger、Tom Breuel、Daniel Burckhardt、Georg Buscher、Fazli Can、Dinquan Chen、Ernest Davis、Pedro Domingos、Rodrigo Panchiniak Fernandes、Paolo Ferragina、Norbert Fuhr、Vignesh Ganapathy、Elmer Garduno、Xiubo Geng、David Gondek、Sergio Govoni、Corinna Habets、Ben Handy、Donna Harman、Benjamin Haskell、Thomas Hühn、Deepak Jain、Ralf Jankowitsch、Dinakar Jayarajan、Vinay Kakade、Mei Kobayashi、Wessel Kraaij、Rick Lafleur、Florian Laws、Hang Li、David Mann、Ennio Masi、Frank McCown、Paul McNamee、Sven Meyer zu Eissen、Alexander Murzaku、Gonzalo Navarro、Scott Olsson、Daniel Paiva、Tao Qin、Megha Raghava、Ghulam Raza、Michal Rosen-Zvi、Klaus Rothenhäusler、Kenyu L. Runner、Alexander Salamanca、Grigory Sapunov、Tobias Scheffer、Nico Schlaefer、Evgeny Shadchnev、Ian Soboroff、Benno Stein、Marcin Sydow、Andrew Turner、Jason Utt、Huey Vo、Travis Wade、Mike Walsh、Changliang Wang、Renjing Wang 及 Thomas Zeume。

很多人主动或应我们的要求对每个章节提出了非常细致的意见。就这点，我们要特别感谢如下这些人：James Allan、Omar Alonso、Ismail Sengor Altingovde、Vo NgocAnh、Roi Blanco、Eric Breck、Eric Brown、Mark Carman、Carlos Castillo、Junghoo Cho、Aron Culotta、Doug Cutting、Meghana Deodhar、Susan Dumais、Johannes Fürnkranz、Andreas Heß、Djoerd Hiemstra、David Hull、Thorsten Joachims、Siddharth Jonathan J. B.、Jaap Kamps、Mounia Lalmas、Amy Langville、Nicholas Lester、Dave Lewis、Stephen Liu、Daniel Lowd、Yosi Mass、Jeff Michels、Alessandro Moschitti、

Amir Najmi、Marc Najork、Giorgio Maria Di Nunzio、Paul Ogilvie、Priyank Patel、Jan Pedersen、Kathryn Pedings、Vassilis Plachouras、Daniel Ramage、Stefan Riezler、Michael Schiehlen、Helmut Schmid、Falk Nicolas Scholer、Sabine Schulte im Walde、Fabrizio Sebastiani、Sarabjeet Singh、Alexander Strehl、John Tait、Shivakumar Vaithyanathan、Ellen Voorhees、Gerhard Weikum、Dawid Weiss、Yiming Yang、Yisong Yue、Jian Zhang 及 Justin Zobel。

最后，我们还要感谢书稿的审阅人员，他们为本书提出了大量高质量的建议。感谢他们为本书的内容和结构提出了重要建议，我们对他们表示深深的谢意，他们是：Pavel Berkhin、Stefan Büttcher、Jamie Callan、Byron Dom、Torsten Suel 及 Andrew Trotman。

第 13、14、15 章的部分初稿基于 Ray Mooney 慷慨提供的报告幻灯片。尽管后来的内容做了大量的修改，但是我们对 Ray Mooney 为这三章的贡献表示由衷的感谢，特别在各种分类器算法的时间复杂度分析方面，本书基本沿用了 Ray Mooney 的工作。

上述感谢的名单并不完整，我们仍然在不断整理来自各方面的反馈。当然，和其他作者一样，我们不一定留意到每一条建议，这点还请大家谅解。另外需要指出的是，本书的出版版本文责自负。

作者还要感谢斯坦福大学和斯图加特大学提供的优良的学术环境。在此环境中，大家可以自由交流思想，并通过授课来促进本书的写作和完善。C. D. Manning 感谢他的家人给他时间投入到本书的写作上，并希望明年能更多地利用周末的时间陪伴家人。P. Raghavan 感谢他的家人默默提供的支持，并感谢 Yahoo!公司提供了一个良好的写作环境。H. Schütze 要感谢他的父母、家人和朋友在他写作期间给予他的支持。

网站和联系方式

与本书（英文原版）配套的网站地址是http://informationretrieval.org。该网站不仅收录了多个相关资源的链接，还提供了每章的教学课件供大家下载使用。我们也欢迎大家将更多的反馈意见和修改建议发邮件至informationretrieval@yahoogroups.com。

第1章 布尔检索

术语信息检索（Information Retrieval，简称IR）的含义可以非常广。从钱包里抽出一张信用卡是为了看到卡号以便输入，这个过程就可以看成是信息检索的一种形式[①]。然而，学术意义上的信息检索定义为：

信息检索是从大规模非结构化数据（通常是文本）的集合（通常保存在计算机上）中找出满足用户信息需求的资料（通常是文档）的过程。

在此定义下，信息检索过去往往是某些特定人士才能从事的一项活动，这些人士包括图书馆馆员、律师助理和其他专业搜索人员等。然而，当今世界发生了巨大变化，当上亿的用户每天使用Web搜索引擎或者查找邮件[②]时，他们实际上都在从事信息检索活动。信息检索已经替代传统的数据库式搜索[③]而迅速成为信息访问的主要形式。

信息检索也能涵盖上述基本定义之外的数据类型和信息处理问题。术语“非结构化数据”（unstruc tured data）指的是那些没有清晰和明显语义结构的数据，而计算机不易处理这类数据。与之相对的是“结构化数据”（structured data），最典型的例子就是关系数据库，它们往往用来保存公司的产品清单和人事记录。当然，严格意义上的非结构化数据在实际中并不存在。比如，文本数据往往被认为是典型的非结构化数据，但是如果考虑文本中隐含的语言结构信息，那么它们也不能算是“非结构化数据”。退一步讲，即使承认这种语言结构属于无明显语义的结构，现实中的大部分文本仍然都有其他结构，如文本的标题、段落、脚注等，这些结构往往通过显式的标记来体现（如网页中的格式标签）。有时，我们也把网页这种具有格式标记的数据称为“半结构化数据”（semistructured data）。信息检索往往也支持这种半结构化数据的搜索，比如查找一篇标题含有 Java 同时正文含有 threading 的文档。

用户对文档的浏览、过滤或对返回的文档进行进一步处理也属于信息检索的研究范畴。在给定文档集的前提下，聚类（clustering）是一种基于文档的内容进行自动聚团的任务。这很像

---

① information retrieval 中的 retrieval 是返回、获取的意思，也就是说，information retrieval 广义上是获取信息的意思，这个例子可以被看成是获取信息的一种形式。基于此，国内也有学者将 information retrieval 翻译成信息获取。——译者注

② 现代的说法中，search 正逐渐代替 information retrieval。search 这个术语相当有歧义，但是在本书的某些上下文中，我们将之与 information retrieval 视作同义词使用。

③ 在数据库式搜索中，服务人员或者这样回答你：“对不起，只有你提供了订单号我才能查到你的订单信息。”

在书架上将一系列书按照它们所属的主题重新摆放的过程，也就是分类[①]。分类（classification）是一种根据给定的主题、固定的信息需求或者其他类别体系（比如按照读者的年龄段对内容进行分类），将每篇文档分到一个或者多个类别的任务。实现这个任务的通常做法是，先手工标注一些文档的类别，然后借助这些已标注文档来自动判断其他文档的类别。

信息检索也可以按照它们所处理数据的规模进行区分，比如可以将信息检索按照数据的规模分成 3 个主要级别。第一个级别是以 Web 搜索（web search）为代表的大规模级别，此时需要处理存储在数百万台计算机上的数十亿篇文档。这种规模下需要重点关注的问题包括：如何采集到这种规模的文档？如何在这种大规模数据量的情况下建立高效运行的系统？如何应对 Web 特性所带来的特殊问题（比如，怎样利用超链接信息，如何防止一些别有用心的人通过伪造网页内容来提高其网站在商业搜索引擎中的排名，等等）？这些问题我们将在第 19 到 21 章中详细介绍。第二个级别是小规模，可以看成是与第一种规模相对的另一极端情况，这种规模的一个代表性例子为个人信息检索（personal information retrieval）。近年来，很多操作系统中已经融合了信息检索的功能（如苹果的 MacOS X 操作系统中的 Spotlight 搜索及微软 Windows Vista 系统中的即时搜索等）。邮件程序中也往往不仅提供搜索功能，还提供分类功能，比如它们至少会提供一个垃圾邮件过滤器，有时还提供手动或者自动的方法来对邮件进行分类，以便组织各类邮件。这类信息检索中需要关注的问题主要包括：如何处理个人计算机上各种格式的文档？如何保证搜索系统的免维护？如何在启动搜索系统、处理信息和使用磁盘时保持简单且占用的系统资源足够少而不至于对用户的正常工作造成影响？等等。介于第一种大规模和第二种小规模之间的信息检索主要面对的是中等规模的数据，包括面向企业、机构和特定领域的搜索（domain-specific search），比如对公司内部文档、专利库或生物医学文献的搜索。这种情况下，文档往往存储在集中的文件系统中，由一台或者多台计算机提供搜索服务。本书提到的技术对上述 3 种数据规模的信息检索系统都具有参考价值，但是由于缺乏公开的资料，涉及第一种大规模并行分布式搜索的内容相对较少一些。而事实上，除了在少数 Web 搜索公司之外，软件开发者更多面对的是个人搜索和企业级搜索这两类场景。

本章一开始将给出一个简单的信息检索的例子，接着介绍词项–文档矩阵（1.1 节）和倒排索引数据结构的概念（1.2 节），然后我们介绍布尔检索模型及布尔查询的处理过程（1.3 节及 1.4 节）。

1.1　一个信息检索的例子

很多人都有*Shakespeare's Collected Works*（《莎士比亚全集》）这本大部头的书。假定你想知道其中的哪些剧本包含 Brutus 和 Caesar 但不包含Calpurnia[②]。一种办法就是从头到尾阅读这本

① 简单地说，聚类事先没有主题引导，而分类却事先定义了主题。关于聚类和分类的区别请参见第 13~17 章。——译者注

② Brutus、Caesar 和 Calpurnia 都是莎士比亚作品中人物的名字，其中，Brutus 即 Marcus Brutus(马可斯·布鲁图斯)，

全集，对每部剧本都留心它是否包含Brutus 和 Caesar且同时不包含Calpurnia。这种线性扫描就是一种最简单的计算机文档检索方式。这个过程通常称为 grepping，它来自于Unix下的一个文本扫描命令`grep`。在文本内进行grepping扫描可以很快，在使用现代计算机的情况下会更快，并且在扫描过程中还可以通过使用正则表达式来支持通配符查找。总之，在使用现代计算机的条件下，对一个规模不大的文档集(《莎士比亚全集》也就只有不到 100 万个单词)进行线性扫描非常简单，根本不需要做额外的处理。

但是，很多情况下只采用上述扫描方式是远远不够的，我们需要做更多的处理。这些情况如下所述。

1. 大规模文档集条件下的快速查找。在线数据量的增长并不低于计算机速度的增长，我们可能需要在几十亿到上万亿单词的数据规模下进行查找。

2. 有时我们需要更灵活的匹配方式。比如，在 `grep` 命令下不能支持诸如 Romans NEAR countrymen 之类的查询，这里的 NEAR 操作符的定义可能为“5 个词之内”或者“同一句子中”。

3. 需要对结果进行排序。很多情况下，用户希望在多个满足自己需求的文档中得到最佳答案。

此时，我们就不能再采用上面的线性扫描方式了。一种非线性扫描的方式是事先给文档建立索引（index）。我们仍然回到上述《莎士比亚全集》的例子，并通过这个例子来介绍布尔检索模型的基本知识。给定词表(《莎士比亚全集》中共使用约 32 000 个不同的词)，假定我们对每篇文档（这里指每部剧本）都事先记录它是否包含词表中的某个词，结果就会得到一个由布尔值构成的词项–文档关联矩阵（incidence matrix）（见图 1-1）。词项[①]（*term*）是索引的单位（将在 2.2 节中进一步讨论），它通常可以用词来表示，目前可以把词项当成词。当然，在信息检索的文献中一般都采用词项这个更正式的说法，而词项也不一定就是词，比如I–9、Hong Kong也可以作为词项，但是它们并不是词。再回到上述矩阵，根据从行还是列的角度来看，可以得到不同的向量[②]：从行来看，可以得到每个词项对应的文档向量，表示词项在哪些文档中出现或不出现；从列来看，可以得到每个文档对应的词项向量，表示文档中哪些词项出现或不出现。

他是晚期罗马共和国的一名元老院议员，组织并参与了对凯撒的谋杀。Caesar 即 Julius Caesar(尤利乌斯·凯撒)，也即凯撒大帝，罗马帝国的奠基者。Calpurnia 即 Calpurnia Pisonis(卡尔普尼亚·皮索尼斯)，凯撒的妻子。——译者注

① 本书中，表示索引单位的单词“term”都统一翻译成“词项”，当然它不一定是词。其他书籍中常用的翻译还包括“标引项”、“索引项”、“特征项”等等，有人甚至称之为“关键词”。——译者注

② 正式情况下，每个向量均采用列向量表示的方式，为使原来行表示的向量转化成列表示的向量，我们只须对原矩阵进行转置即可。这个问题中，为得到每个词项对应的向量，需要对矩阵进行转置。

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	…
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
…							

图 1-1　词项–文档关联矩阵，其中每行表示一个词，每列表示一个剧本。当词t在剧本d中存在时，矩阵元素（t，d）的值为 1，否则为 0[①]

为响应查询 Brutus AND Caesar AND NOT Calpurnia，我们分别取出 Brutus、Caesar 及 Calpumia 对应的行向量，并对 Calpumia 对应的向量求反，然后进行基于位的与操作，得到：

110100 AND 110111 AND 101111 = 100100

结果向量中的第 1 和第 4 个元素为 1，这表明该查询对应的剧本是 *Antony and Cleopatra* 和 *Hamlet*（参见图 1-2）。

Antony and Cleopatra, Act III, Scene ii
Agrippa [Aside to Domitius Enobarbus]:　Why, Enobarbus,
When Antony found Julius Caesar dead,
He cried almost to roaring; and he wept
When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii
Lord Polonius:　I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

图 1-2　查询 Brutus AND Caesar AND NOT Calpurnia 对应的结果

布尔检索模型接受布尔表达式查询，即通过 AND、OR 及 NOT 等逻辑操作符将词项连接起来的查询。在该模型下，每篇文档只被看成是一系列词的集合。

下面我们来考虑一个更真实的场景，同时也引出信息检索中的一些术语和概念。假定我们有 N=1 000 000 篇文档。所谓“文档”（document）指的是检索系统的检索对象，它们可以是一条条单独的记录或者是一本书的各章（进一步的讨论请参见 2.1.2 节）。所有的文档组成文档集（collection），有时也称为语料库（corpus）。假设每篇文档包含约 1 000 个词（相当于 2～3 页书），每个词的平均长度是 6 B（含空格和标点符号），那么我们就会得到一个大约 6 GB 大小的文档集。通常，在这些文档中大概会有 M=500 000 个不同的词项。这里给出的数字并没有什么特殊意义，实际中遇到的数字也许高出或者低出一个或多个数量级，我们只期望通过上述数字能使我们对问题的处理维度有一个感性认识。在 5.1 节，我们还要对上述数字之间的关系进行建模。

我们的目标是开发一个能处理ad hoc检索（ad hoc retrieval）任务（一种常见的信息检索任

① 本例中对应的莎士比亚作品名称分别为：《安东尼与克莉奥佩屈拉》(Antony and Cleopatra)、《尤利乌斯·凯撒》(Julius Caesar)、《暴风雨》(The Tempest)、《哈姆雷特》(Hamlet)、《奥赛罗》(Othello)及《麦克白》(Macbeth)。Antony 即马克·安东尼，古罗马的政治家和军事家。Cleopatra 即著名的埃及艳后。——译者注

务）的系统。在这个任务中，任一用户的信息需求通过一次性的、由用户提交的查询传递给系统，系统从文档集中返回与之相关的文档[①]。信息需求（information need）指的是用户想查找的信息主题，它和查询（query）并不是一回事，后者由用户提交给系统以代表其信息需求。如果一篇文档包含对用户需求有价值的信息则认为是相关的（relevant）。上面举的例子当中，信息需求采用多个特定的词语组合来表达，从这点来说，人工构造的痕迹明显。而通常来说，假如用户对pipeline leaks感兴趣，在检索时，不管是严格采用这些词还是采用反映这一概念的其他词（如pipeline rupture）来表达该需求，都能得到相关的结果。对检索系统的效果（effectiveness，搜索结果的质量）进行评价，通常需要知道某个查询返回结果的两方面的统计信息。

正确率（Precision）：返回的结果中真正和信息需求相关的文档所占的百分比。

召回率（Recall）：所有和信息需求真正相关的文档中被检索系统返回的百分比。

有关信息检索系统评价的内容我们将在第 8 章详细介绍。

回到刚才的例子，我们显然不能再采用原来的方式来建立和存储一个词项–文档矩阵。由于词项的个数是 50 万，而文档的篇数为 100 万，所以其对应的词项–文档矩阵大概有 5 000 亿（50 万×100 万）个取布尔值的元素，这远远大于一台计算机内存的容量。另外，我们不难发现，这个庞大的矩阵实际上具有高度的稀疏性，即大部分元素都是 0，而只有极少部分元素为 1。我们可以对上述例子做个粗略计算，由于每篇文档的平均长度是 1 000 个单词，所以 100 万篇文档在词项–文档矩阵中最多[②]对应 10 亿（1 000×1 000 000）个 1，也就是在词项–文档矩阵中至少有 99.8%（1–10 亿/5 000 亿）的元素为 0。很显然，只记录原始矩阵中 1 的位置的表示方法比词项–文档矩阵更好。

上述思路将引出信息检索中的第一个核心概念——倒排索引（inverted index）。从名称上看，倒排索引中“倒排”二字似乎有些多余，因为一般提到的索引都是从词项反向映射到文档的。然而，倒排索引（有时也称倒排文件[③]）已经成为信息检索中的一个标准术语。倒排索引的基本思想参见图 1-3。左部称为词项词典（dictionary，简称词典，有时也称为vocabulary或者lexicon[④]。本书中dictionary 指图 1-3 中的数据结构，而vocabulary则指词汇表）。每个词项都有一个记录出现该词项的所有文档的列表，该表中的每个元素记录的是词项在某文档中的一次出现信息（在后面的讨论中，该信息中往往还包括词项在文档中出现的位置），这个表中的每个元素通常称为倒排记录（posting）[⑤]。每个词项对应的整个表称为倒排记录表（posting list）或倒排表（inverted

① 信息检索中与 ad hoc 检索任务相对的另一种任务称为过滤（filtering）。在 ad hoc 任务中，信息需求动态变化，而文档集则相对静止。在过滤任务中，信息需求在一段时间内保持不变，而文档集则变化频繁。过滤任务的一个典型应用是信息订阅。有关过滤任务的进一步介绍，请参见第 13 章。——译者注

② 显然，在 1 000 个词都不重复的情况下，1 的数目最多。——译者注

③ 有些学者倾向于采用倒排文件这一名称，但是倒排索引构建及倒排索引压缩等说法已经根深蒂固，而通常不说倒排文件构建及倒排文件压缩。为了保持一致性，本书采用了倒排索引的说法。

④ 本书中提到的 dictionary 指图 1-3 所示的数据结构，而 vocabulary 则指词汇表。可以将前者理解成物理上的词典，而后者是逻辑上的词典。当然，有时我们并不严格区分。——译者注

⑤ 在一个无位置信息的倒排索引中，一条倒排记录往往只是文档的 ID 号，实际上该 ID 号必须和其对应的词项一起来说才有意义，也就是说，我们有时提到倒排记录的时候，会采用（词项，文档 ID）这种二元组的表示方法。

list)。所有词项的倒排记录表一起构成全体倒排记录表 (postings)。图 1-3 中的词典按照字母顺序进行排序，而倒排记录表则按照文档ID号进行排序。在 1.3 节我们将会了解这样做的原因，在后面的章节中我们也会考虑其他可能的做法 (参见 7.1.5 节)。

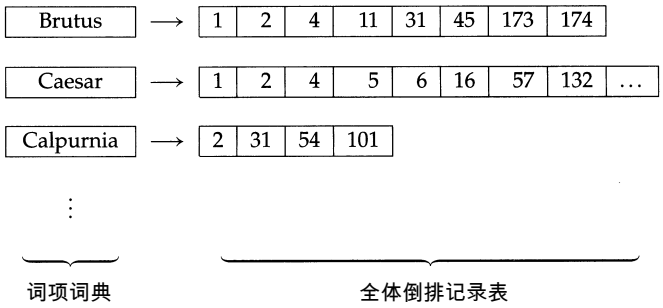


图 1-3　倒排索引的两个部分。词典部分往往放在内存中，而指针指向的每个倒排记录表则往往存放在磁盘上

1.2　构建倒排索引的初体验

为获得检索速度的提升，就必须要事先建立索引。建立索引的主要步骤如下。

(1) 收集需要建立索引的文档，如：

Friends, Romans, countrymen. | So let it be with Caesar…

(2) 将每篇文档转换成一个个词条 [①] (token) 的列表，这个过程通常称为词条化 (tokenization)，如：

Friends | Romans | countrymen | So…

(3) 进行语言学预处理，产生归一化的词条来作为词项，如：

Friends | roman | countrymen | So…

4. 对所有文档按照其中出现的词项来建立倒排索引，索引中包括一部词典和一个全体倒排记录表。

我们将在 2.2 节介绍第 1 ~ 3 步，在此之前，为了便于理解，我们可以把词条及归一化后的词条都看成是词。现在，假定前 3 步已经执行完毕，我们来讲述如何通过基于排序的索引构建方法 (sort-based indexing) 来建立一个基本的倒排索引。

给定一个文档集，我们假定每篇文档都有一个唯一的标识符即编号 (docID)。在索引构建过程中，我们可以给每篇新出现的文档赋一个连续的整数编号。在上述的前 3 步处理结束后，对每篇文档建立索引时的输入就是一个归一化的词条表，也可以看成二元组 (词项，文档 ID) 的一个列表 (参见图 1-4)。建立索引最核心的步骤是将这个列表按照词项的字母顺序进行排序，

① 这里我们把 token 译成词条，当然我们平时提到词条往往指某个词典或词汇表中的一个条目。在没有特别说明的情况下，本书中词条均指 token。——译者注

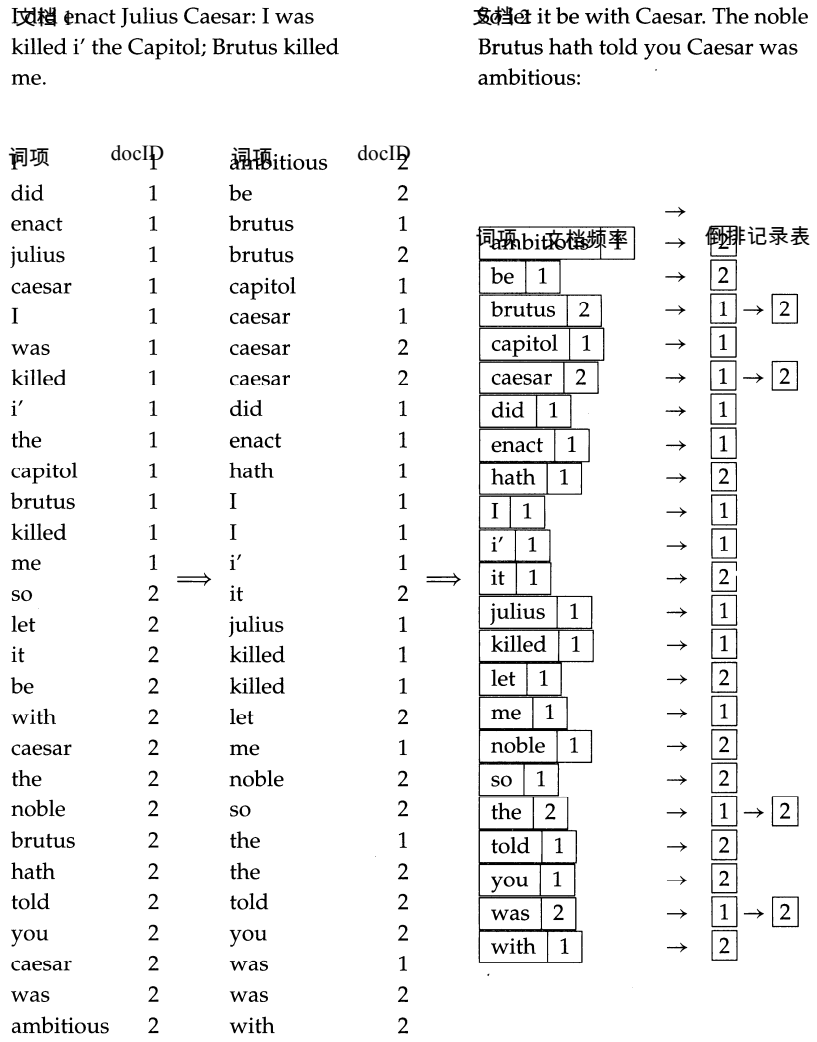


图 1-4 通过排序和合并建立倒排索引的过程。每篇文档的所有词项加上文档 ID（图左部）通过按照字母顺序排序（图中部）。然后，同一词项进行合并。最后，将词项和文档 ID 分开（图右部）。词项存储在词典中，每个词项有一个指针指向倒排记录表。词典中往往还会存储一些其他的概要信息，如这里所存储的每个词项的文档频率。这个信息可以用于提高查询执行时的时间效率，也会应用于后面要讨论的结果排序中的权重计算方法。每个倒排记录表存储了词项出现的文档列表，也可以存储一些其他信息，比如词项频率（term frequency，即词项在文档中出现的次数）和词项在文档中出现的位置

之后我们得到图 1-4 中部显示的结果，其中一个词项在同一文档中的多次出现会合并在一起[①]，最后整个结果分成词典和倒排记录表两部分，如图 1-4 右部所示。由于一个词项通常会在多篇

① Unix 用户将会发现这些步骤类似于在 Unix 中先使用 `sort` 命令然后再使用 `uniq` 命令。

文档中出现，上述组织数据的方法实际上也已经减少了索引的存储空间。词典中同样可以记录一些统计信息，比如出现某词项的文档的数目，即文档频率（document frequency），这里也就是每个倒排记录表的长度，该信息对于一个基本的布尔搜索引擎来说并不是必需的，但是它可以在处理查询时提高搜索的效率，因此它在后面介绍的排序检索模型中被广泛使用。倒排记录表会按照docID进行排序，这为高效的查询处理提供了重要基础。在ad hoc文本检索中，倒排索引是其他结构无法抗衡的高效索引结构。

在最终得到的倒排索引中，词典和倒排记录表都有存储开销。前者往往放在内存中，而后者由于规模大得多，通常放在磁盘上。因此，两部分的大小都非常重要。在第 5 章中，我们将介绍如何对每个部分进行存储优化来提高访问效率。现在我们考虑对每个倒排记录表应该采用的数据结构。由于有些词在很多文档中出现，而另外一些词出现的文档数目却很少，所以，如果采用定长数组的方式将会浪费很多空间。对于内存中的一个倒排记录表，可以采用两种好的存储方法：一个是单链表，另一个是变长数组。单链表（singly linked list）便于文档的插入和更新（比如，对更新的网页进行重新采集），因此通过增加指针的方式可以很自然地扩展到更高级的索引策略（如 2.3 节讨论的跳表（skip list））。而变长数组（variable length array）的存储方式一方面可以节省指针消耗的空间，另一方面由于采用连续的内存存储，可以充分利用现代计算机的缓存（cache）技术来提高访问速度。额外的指针在实际中可以编码成偏移地址融入到表中。如果索引更新不是很频繁的话，变长数组的存储方式在空间上更紧凑，遍历也更快。另外，我们也可以采用一种混合的方式，即采用定长数组的链表方式。当倒排记录表存在磁盘上的时候，它们被连续存放并且没有显式的指针（参见图 1-3），这样可以在把倒排记录表读入内存时，将该倒排记录表的大小及磁盘寻道的次数降到最小。

? 习题 **1-1** [*]　画出下列文档集所对应的倒排索引（参考图 1-3 中的例子）。

文档 1　new home sales top forecasts

文档 2　home sales rise in july

文档 3　increase in home sales in july

文档 4　july new home sales rise

习题 **1-2** [*]　考虑如下几篇文档：

文档 1　breakthrough drug for schizophrenia

文档 2　new schizophrenia drug

文档 3　new approach for treatment of schizophrenia

文档 4　new hopes for schizophrenia patients

a. 画出文档集对应的词项–文档矩阵；

b. 画出该文档集的倒排索引（参考图 1-3 中的例子）。

习题 **1-3** [*]　对于习题 1-2 中的文档集，如果给定如下查询，那么返回的结果是什么？

a. schizophrenia AND drug

b. for AND NOT (drug OR approach)

1.3 布尔查询的处理

如何使用倒排索引和基本布尔检索模型来处理一个查询呢？我们先以一个简单“与”查询（simple conjunctive query）为例：

Brutus AND Calpurnia (1-1)

我们可以使用图 1-3 所示的倒排索引进行如下操作：

(1) 在词典中定位 Brutus；

(2) 返回其倒排记录表；

(3) 在词典中定位 Calpurnia；

(4) 返回其倒排记录表；

(5) 对两个倒排记录表求交集，如图 1-5 所示。

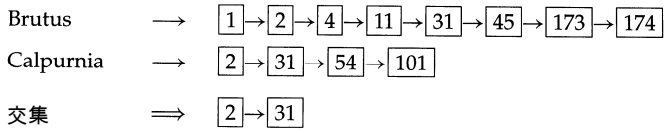


图 1-5 对图 1-3 中 Brutus 和 Calpurnia 所对应的倒排记录表进行交集计算的示意图

在这里，交集（intersection）操作非常关键，这是因为我们必须快速将倒排记录表求交集以尽快找到哪些文档同时包括两个词项。该操作有时称为合并（merge），这里合并的概念有点违背我们的直觉，特别是我们在使用合并算法（merge algorithm）这个术语时，往往指的是通过移动每个列表中的指针而将多个有序列表进行迭加[①]。而我们这里所说的合并操作指的是执行逻辑“与”操作。

求两个倒排记录表交集的一个简单有效的合并算法如图 1-6 所示。我们对每个有序列表都维护一个位置指针，并让两个指针同时在两个列表中后移。该算法对于倒排记录表集(即待合并的两个倒排记录表)的大小而言是线性的。每一步我们都比较两个位置指针所指向的文档 ID，如果两者一样，则将该 ID 输出到结果表中，然后同时将两个指针后移一位。如果两个文档 ID 不同，则将较小的 ID 所对应的指针后移。假设两个倒排记录表的大小分别是 x 和 y，那么上述求交集的过程需要 $O(x+y)$次操作。更正式的说法是，查询的时间复杂度为 $\Theta(N)$，其中 N 是文档

① 即相当于逻辑“或”或者求并集。——译者注

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

图 1-6　两个倒排记录表的合并算法

集合中文档的数目 ①。和线性扫描相比，这种索引方法并没有带来Θ意义上时间复杂度的提高，而最多只是一个常数级别的变化。但是，实际当中这个常数很大。如果要使用上述合并算法，那么倒排记录表必须按照全局的统一指标进行排序。通过文档ID的数值进行排序是一个简单的实现方法。

我们可以对上述合并算法进行扩展，使之能够用于处理更复杂的查询：

(Brutus OR Caesar) AND NOT Calpurnia　　(1-2)

查询优化（query optimization）指的是如何通过组织查询的处理过程来使处理工作量最小。对布尔查询进行优化要考虑的一个主要因素是倒排记录表的访问顺序。那么，哪种访问顺序具有最优性呢？考虑一个对 t 个词项进行“ 与” 操作的查询，例如：

Brutus AND Caesar AND Calpurnia　　(1-3)

对每个词项，我们必须取出其对应的倒排记录表，然后将它们合并。一个启发式的想法是，按照词项的文档频率（也就是倒排记录表的长度）从小到大依次进行处理，如果我们先合并两个最短的倒排记录表，那么所有中间结果的大小都不会超过最短的倒排记录表 ②，这样处理所需要的工作量很可能最少。因此，对于图 1-3 对应的倒排记录表，我们将按照如下顺序来处理查询（1-3）：

(Calpurnia AND Brutus) AND Caesar　　(1-4)

这里我们第一次用到了词项的文档频率，这也给出了在词典中保存文档频率的一个充分理由，即它可以在访问之前用于决定倒排记录表的访问次序。

考虑更一般的查询，如：

(madding OR crowd) AND (ignoble OR strife) AND (killed OR slain)　　(1-5)

如前所述，我们可以获得所有词项的文档频率，由此我们可以保守地估计出每个 OR 操作后的结果大小，然后按照结果从小到大的顺序执行 AND 操作。

① Θ( )用于表示算法复杂度的渐进紧密界，在非正式的情况下，有时会写成 $O(\ )$，但是这个概念的实际意思是渐进上界，而不一定是紧密的。（参见 Cormen 等人 1990。）

② 这是因为多个集合的交集元素个数肯定不大于其中任何一个集合的元素个数。

对于任意的布尔查询，我们必须计算并临时保存中间表达式的结果。但是，在很多情况下，不论是由于查询语言本身的性质所决定，还是仅仅由于这是用户所提交的最普遍的查询类型，查询往往是由纯“与”操作构成的。在这种情况下，不是将倒排记录表合并看成两个输入加一个不同输出的函数，而是将每个返回的倒排记录表和当前内存中的中间结果进行合并，这样做的效率更高而最初的中间结果中可以调入最小文档频率的词项所对应的倒排记录表。该算法如图 1-7 所示。因此，该合并算法是不对称的：中间结果表在内存中而需要与之合并的倒排记录表往往要从磁盘中读取。此外，中间结果表的长度至多和倒排记录表一样长，在很多情况下，它可能会短一个甚至多个数量级。当然，倒排记录表的合并仍然可以采用图 1-6 所示的算法来实现，但是在倒排记录表的长度相差很大的情况下，就可以使用一些策略来加速合并过程。对于中间结果表，合并算法可以就地对失效元素进行破坏性修改或者只添加标记。或者，通过在长倒排记录表中对中间结果表中的每个元素进行二分查找也可以实现合并。另外一种可能是将长倒排记录表用哈希方式存储，这样对中间结果表的每个元素，就可以通过常数时间而不是线性或者对数时间来实现查找。然而，上述策略很难融合到第 5 章所讨论的压缩后的倒排记录表中。另外，如果查询中的两个词项都是常见词时，那么还是有必要采用标准的倒排记录表合并方法。

```
INTERSECT(⟨t_1, ..., t_n⟩)
1  terms ← SORTBYINCREASINGFREQUENCY(⟨t_1, ..., t_n⟩)
2  result ← postings(first(terms))
3  terms ← rest(terms)
4  while terms ≠ NIL and result ≠ NIL
5  do result ← INTERSECT(result, postings(first(terms)))
6     terms ← rest(terms)
7  return result
```

图 1-7 在输入多个词项的与查询时对它们的倒排记录表进行合并的算法

? 习题 **1-4** [*] 对于如下查询，能否仍然在 $O(x+y)$次内完成？其中 x 和 y 分别是 Brutus 和 Caesar 所对应的倒排记录表长度。如果不能的话，那么我们能达到的时间复杂度是多少？

a. Brutus AND NOT Caesar

b. Brutus OR NOT Caesar

习题 **1-5** [*] 将倒排记录表合并算法推广到任意布尔查询表达式，其时间复杂度是多少？比如，对于查询

c. (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

我们能在线性时间内完成合并吗？这里的线性是针对什么来说的？我们还能对此加以改进吗？

习题 **1-6** [**] 假定我们使用分配律来改写有关 AND 和 OR 的查询表达式。

a. 通过分配律将习题 1-5 中的查询写成析取范式；

b. 改写之后的查询的处理过程比原始查询处理过程的效率高还是低？

c. 上述结果对任何查询通用还是依赖于文档集的内容和词本身？

习题 **1-7** [*] 请推荐如下查询的处理次序。

d. (tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

其中，每个词项对应的倒排记录表的长度分别如下：

词项	倒排记录表长度
eyes	213 312
kaleidoscope	87 009
marmalade	107 913
skies	271 658
tangerine	46 653
trees	316 812

习题 **1-8** [*]　对于查询

e. friends AND romans AND (NOT countrymen)

如何利用 countrymen 的文档频率来估计最佳的查询处理次序？特别地，提出一种在确定查询顺序时对逻辑非进行处理的方法。

习题 **1-9** [**]　对于逻辑与构成的查询，按照倒排记录表从小到大的处理次序是不是一定是最优的？如果是，请给出解释；如果不是，请给出反例。

习题 **1-10** [**]　对于查询 *x* OR *y*，按照图 1-6 的方式，给出一个合并算法。

习题 **1-11** [**]　如何处理查询 *x* AND NOT *y* ？为什么原始的处理方法非常耗时？给出一个针对该查询的高效合并算法。

1.4　对基本布尔操作的扩展及有序检索 ①

和布尔检索模型相对的是排序检索模型或有序检索模型(ranked retrieval model ,见 6.3 节)，后者不是通过具有精确语义的逻辑表达式来构建查询，而往往是采用一个或者多个词来构成自由文本查询（free text query）。有序检索系统要能够确定哪篇文档最能满足用户的需求。尽管有序检索模型在学术界已经提出并发展了几十年，也有很多优点，但是在上世纪 90 年代初(差不多万维网开始出现的时间)之前，在大型的商业信息提供商 30 年来所提供的检索系统中，基于布尔模型的检索系统还一直占主流地位，有时甚至是检索系统中唯一的检索模型。当然，这些检索系统中并不只包含我们迄今为止提到的基本的布尔运算(AND、OR 及 NOT)。严格的布尔运算所得到的无序结果集远远不能使用户满意，所以，这些系统中往往加入了更多的操作，如“词项邻近”（term proximity）操作。邻近操作符（proximity）用于指定查询中的两个词项应该在文档中互相靠近，靠近程度通常采用两者之间的词的个数或者是否同在某个结构单元（如句子或段落）中出现来衡量。

① 这里的扩展是指除基本布尔操作之外，系统还支持一些其他操作（如邻近操作）。这和其他教材中所提到的扩展布尔模型（如 p-norm 模型）并不是一回事，后者是对基本布尔模型进行改进，使之能够支持更多的功能（比如能对文档进行排序）。——译者注

例 1-1 Westlaw 商业布尔搜索系统。按付费用户的数目计，Westlaw（http://www.westlaw.com/）是目前世界上最大的法律搜索服务提供商，每天大约有 50 万个用户对数十太字节的文本数据进行数百万次的查询。该服务开始于 1975 年，直到 2005 年，该系统默认的检索方式仍然是布尔搜索（Westlaw 自己称这种搜索为 Terms and Connectors），尽管在 1992 年面向有序检索模型的自由文本查询已经添加到搜索系统中（Westlaw 称这种搜索为 Natural Language），但是使用布尔搜索的用户的比例仍然更高。以下是 Westlaw 系统所支持的一些查询的例子。

信息需求：与防止先前受雇于竞争对手的员工泄露商业机密相关的法律理论信息（Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company）

查询："trade secret" /s disclos! /s prevent /s employe!

信息需求：残疾人士能够进入工作场所的要求（Requirements for disabled people to be able to access a workplace）

查询：disab! /p access! /s work-site work-place (employment /3 place）

信息需求：主人对客人醉酒负责的案例（Cases about a host's responsibility for drunk guests）

查询：host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

---

我们可以注意到，上面的例子中不仅包含了非常精确语义的长查询，还使用了邻近操作符，这些在通常的 Web 搜索中都很少见。上面的几个查询的平均长度大概有 10 个单词。和一般 Web 搜索不同的是，上述查询中的空格表示逻辑"或"关系，&表示 AND，/s、/p 及/*k* 分别表示处于同一句子、段落和 *k* 个词之内。双引号表示的是短语查询（phrase search，即多个词连续出现，参见 2.4 节），感叹号（!）表示尾通配符查询（参见 3.2 节）。因此，liab!表示和所有以 liab 开始的词匹配。另外，work-site 表示可以和 worksite、work-site 及 work site 匹配（参见 2.2.1）。一个典型的专家查询常常会很小心地定义，并且会被逐步改进直到用户满意为止。

很多用户，特别是专业用户，更喜欢布尔查询模型。布尔查询表达上更精确：文档要么满足查询，要么不满足查询。这可以让用户对返回结果拥有更好的控制力和透明度。并且对某些领域的信息（如法律资料），在布尔检索内部也可以提供排序机制。比如，Westlaw 对返回结果按照时间排序，越新的结果排序越靠前，这也是一种非常有效的排序方式。2007 年，大多数法律图书馆员仍然推荐使用布尔搜索来获得高召回率的搜索结果，大部分法律用户也感觉这种方式更好控制。但是，这并不意味着对于专业搜索人员来说，布尔查询更加有效。实际上，一个基于 Westlaw 部分文档集的实验（Turtle，1994)表明，对于实验中的大部分信息需求，采用自由文本的查询能够比 Westlaw 参考馆员给出的布尔查询取得更好的结果。布尔搜索的一个普遍问题就是采用 AND 操作符产生的结果正确率虽高但是召回率偏低，而采用 OR 操作符召回率高

但是正确率低，很难或者说不可能找到一个令人满意的折衷方案。

本章中，我们讲述了基本倒排索引的结构和构建，整个索引包含词典和倒排记录表两部分。我们介绍了布尔检索模型，并考察了如何通过线性时间复杂度的合并方法和简单的查询优化方法来进行高效检索。在第 2 ~ 7 章中，我们会考虑更丰富的查询模型及各种用于高效查询处理的增强的索引结构。在这里，我们仅仅先提一下它们的要点。

(1) 我们要更好地确定词典中的词项表，提供一个能够容忍拼写错误以及当查询和文档中词语表达不一致时的检索方法。

(2) 对能够表示某概念的复合词或者短语（如“ operating system”）进行搜索是非常有用的。正如上面 Westlaw 的例子所示，有时我们希望能够执行诸如“ Gates NEAR Microsoft”这类的邻近查询，为处理这类查询，索引结构必须要改进。

(3) 布尔模型只记录词项存在或者不存在，但是我们往往需要累加各种证据来得到文档相关的可信度。比如，如果一个查询项在某文档中多次出现，我们会给该文档赋予一个权重，该权重会比仅仅出现一次查询项的文档要高。为了实现这个目的，我们需要引入词项频率(词项在文档中出现的次数)的概念。

(4) 布尔查询仅仅返回一个无序的文档集，但是我们往往需要对返回的结果排序（或排名），这就需要提供一个机制在给定查询的情况下对每个文档的好坏进行评分。

带着上述思路，我们在后面的章节中将会看到这些基本技术在非结构化数据 ad hoc 搜索中的应用。近年来，ad hoc 搜索已经占据了主导地位，不仅是 Web 搜索引擎，还包括大型电子商务网站的非结构化数据搜索。尽管主要的搜索引擎对自由文本查询的重视程度不尽相同，但是我们将会看到，实际上它们背后的大部分索引和查询技术都基本一致。另外，随着时间的推移，很多 Web 搜索引擎都或多或少增加了扩展布尔模型的一部分操作，其中短语查询非常流行，大部分搜索引擎都支持部分布尔操作。然而，尽管这些操作为搜索专家所喜欢，但是大部分用户却很少使用。因此，它们并不是提高 Web 搜索引擎性能的重点所在。

? 习题 **1-12** [*]　利用 Westlaw 系统的语法构造一个查询，通过它可以找到 professor、teacher 或 lecturer 中的任意一个词，并且该词和动词 explain 在一个句子中出现，其中 explain 以某种形式出现。

习题 **1-13** [*]　在一些商用搜索引擎上试用布尔查询，比如，选择一个词（如 burglar），然后将如下查询提交给搜索引擎

(i) burglar；(ii) burglar AND burglar；(iii) burglar OR burglar。

对照搜索引擎返回的总数和排名靠前的文档，这些结果是否满足布尔逻辑的意义？对于大多数搜索引擎来说，它们往往不满足。你明白这是为什么吗？如果采用其他词语，结论又如何？比如以下查询

(i) knight；(ii) conquer；(iii) knight OR conquer。

从前面两个查询的结果能够得到第三个查询结果的什么界？该界是否被观察到？

1.5 参考文献及补充读物

计算机信息检索开始于上世纪 40 年代（Cleverdon，1991；Liddy，2005），科学文献的大量涌现及计算机可用性的提高，引起了人们对自动文档检索的兴趣。其中，这些文献的大部分都以非完全正式的技术报告而不是传统的期刊论文形式出现。然而，当时的文档检索主要是基于作者、标题和关键词来进行，直到很久之后才出现了全文检索。

Bush（1945）的论文为这一新领域提供了源源不断的灵感：

> “考虑一个供个人使用的未来设备，它是一种机械化的个人档案和图书馆。由于它需要一个名字，所以我随便造了一个名字叫 memex。每个人可以将他所有的书、笔记或者和他人的通信内容存储在 memex 上，它是一个机械化设备，能够提供快速灵活的查阅功能。可以将它看成是对人记忆的补充。”

Information Retrieval 这个术语是 Calvin Mooers 于 1948 年到 1950 年间提出的（Mooers 1950）。

1958 年，很多报纸都对 IBM 在一次会议上展出的“自动索引”机器给予了极大关注（参见 Taube 和 Wooster，1958）。这个机器主要基于 H. P. Luhn 的工作而建成。很多商业组织很快转向布尔检索系统，但在早期，有关检索系统的各种不同的技术之间存在着激烈的争论。例如，Mooers 就曾对布尔模型的广泛使用表示过异议（1961）：

> “在考虑将数百万美元投资到各种检索硬件时，现在的很多人认为 George Boole（1847）的布尔代数是最合适的检索系统形式体系，这是个常见的谬误。这个错误的观点一直广泛地毫无批判地被人们所接受。”

逻辑与（AND）和逻辑或（OR）会导致检索结果在正确率和召回率这一对折中量上走向两个极端，但是很难达到合理均衡状态（Lee and Fox，1988）。

Witten 等人在 1999 年撰写的著作（Witten 等人 1999）中深入地比较了倒排索引和其他数据结构的时空消耗，这本书可以被视为标准的参考资料。一个更简洁的新的介绍参见 Zobel and Moffat（2006）。我们将在第 5 章进一步讨论一些方法。

Friedl 在 2006 年概述了正则表达式（regular expressions）搜索的实际使用。相关的计算机

科学基础知识参见（Hopcroft 等人 2000）。

第2章 词项词典及倒排记录表

首先回顾一下构建倒排索引的几个主要步骤：

(1) 收集待建索引的文档；

(2) 对这些文档中的文本进行词条化；

(3) 对第 2 步产生的词条进行语言学预处理，得到词项；

(4) 根据词项对所有文档建立索引。

本章首先定义文档的基本组成单位并介绍在文档中确定这些单位所含字符序列的方法（参见 2.1 节）。接着，我们将详细讨论在词条化和语言学预处理过程中所涉及到的主要的语言学相关问题，通过词条化和语言学预处理可以确定系统所用的词项词典（参见 2.2 节）。所谓词条化（tokenization）指的是将原始的字符流转换成一个个词条（token）的过程。而语言学预处理的主要目的在于建立词条的等价类，其中每个等价类对应一个词项，这些词项最终用于建立文档的索引。构建索引的过程主要在第 1 章和第 4 章介绍，本章暂不详述。本章最后讨论倒排记录表的具体实现问题。2.3 节考察了一个扩展的带跳表的倒排记录表数据结构，该结构能够支持查询的快速处理。2.4 节主要介绍适合于处理处理短语查询和邻近查询的索引结构，这些查询在支持扩展布尔操作的检索系统和 Web 搜索系统中的使用十分普遍。

2.1 文档分析及编码转换

2.1.1 字符序列的生成

作为索引构建过程的输入，数字文档一般由文件中或者Web服务器上的一系列字节组成。因此，文档处理的第一步往往是将这些字节序列转换成线性的字符序列。对于ASCII编码的纯英文文本来说，处理起来似乎并非难事。然而，实际中往往会遇到非常复杂的情况。比如字符序列可能采用各种单字节或者多字节编码方式（比如 Unicode中的UTF-8 编码），也可能采用不同国家、不同厂家的特定编码方式。因此，为了实现从字节序列到字符序列的转换，首先要正确地判断出文档的编码方式。该判断过程看成一个基于机器学习的分类问题[①]（我们将在第 13 章讨论）来处理，但在实际中往往通过启发式方法来实现，也可以利用文档的元信息或者直接由用户手工选择来确定。确定编码方式后，我们就可以将字节序列转换成字符序列，在此过程中还应该保存编码信息，因为该信息有时能帮助确定文档的语言种类。

① 分类器是一个将具有同种属性的对象归入一个或多个类别的函数，它往往通过机器学习方法（如概率方法）来实现，当然它也可以通过人工编写的规则来实现。

有时，我们必须从一些二进制文档（如 Microsoft 的.doc 文档或.zip 之类的压缩文档等等）中得到字符序列。这时，我们也必须先确定文档的格式，然后才能采用合适的编码转换方式还原出字符序列。即便是对纯文本文档，有时也需要进行额外的转换过程。比如，对于 XML 文档（参见 10.1 节），一些转义字符串就需要就转换成它原本代表的字符，如“ &”就要转换成“ &”。最后，如果文档中同时存在文本和非文本部分，当不需要对非文本部分进行处理时，就需要将文本部分单独抽取出来进行处理。比如，我们在处理 XML 文档时，可以不考虑其中的标签。而对于 ps 或者 PDF 文件，也可以采用上述处理方法。有关文本和非文本混合的问题，本书并不打算进行更深入的讨论，这里我们假设文档只由一系列的文本字符构成。由于用户只想和数据的本来面目打交道，所以商业产品往往都需要支持各种文档和编码格式。当然，在应用中用户常常把文档想成就是文本，甚至都不会意识到它们在存放时还有编码格式问题。编码格式问题往往通过购买格式及编码处理类软件库的许可证来解决。

将文本视为线性字符序列的看法在某些文字系统中并不成立，比如阿拉伯语就表现出某种二维和混序（mixed-order）的特性（分别参见图 2-1 和图 2-2）。然而，尽管一些复杂的文字系统存在自己的习惯，但是其背后的发音序列仍然可以表示出来，所以，实质上仍然存在某种线性结构。阿拉伯语的数字化表示也是如此（参见图 2-1）。

كِتابٌ ⇐ ك ِ ت ا ب ٌ
un b ā t i k
/kitābun/ *‘a book’*

图 2-1 一个现代标准阿拉伯语单词及其发音的例子[①]，阿拉伯文字的书写顺序是从右向左，并且在组词过程中字母会发生复杂的词形变化。本例当中，短元音（在这里，/i/和/u/）和最后的 /n/（词尾变化）在单词中的表示并不严格满足线性组合规律，实际上它们通过字母上方或下方的发音符号表示出来。尽管如此，这个代表性的例子还是明显地反映了符号标音的线性顺序。像本例一样的完整发声法（即在单词中保留全部的发音符号），通常只出现在《古兰经》和儿童启蒙类图书中。日常文字中并不标出发音符号（即短元音不会被表示出来，当然代表ā的字母仍会出现）或只标出部分发音符号。当然，在可能会出现歧义的地方，作者也会标出短元音。这些选项进一步增加了索引的复杂性

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.
← → ← → ← START
‘Algeria achieved its independence in 1962 after 132 years of French occupation.’

图 2-2 这种文字的概念(线性)次序未必和我们看到的显示次序相一致。在希伯来语和阿拉伯语等从右到

① 阿拉伯语是由 28 个辅音字母和 12 个发音符号组成的拼音文字。元音通过字母上方或下方的发音符号来表示，不过这些符号通常省略，只在《古兰经》或者初级启蒙书中出现。图 2-1 中第一行给出了一个 4 个字母(从右往左的第 1、3、4、5 个位置上的符号)和 2 个发音符号(从右往左的第 2、6 个位置上的符号)构成的阿拉伯语单词(左部)。组合成单词时，发音符号直接从上下和辅音字母组合，组合过程中还会有词形变化。图 2-1 中第二行是拉丁转写形式，而第三行是单词的国际音标和英文释义。——译者注

左书写的语言中，往往会参杂有从左到右书写的内容，例如数字和金额 ①。在采用现代Unicode 编码方式的情况下，文件中的字符存储次序和概念次序是一致的，而字符的逆序展示则由显示系统来完成，但上述做法可能并不适合旧编码文件的处理

2.1.2　文档单位的选择

下一步我们需要确定索引的文档单位（document unit）。至今为止，我们都假设在建立索引时每篇文档就是固定的索引单位。比如，我们将某个目录下的每个文件都看成一个文档。但是，很多情况下并非如此。传统的 Unix （mbox 格式）邮件系统将某个邮件目录下的一系列邮件都存放到单个文件中，但是我们希望将每封邮件都看成一个独立的文档。很多邮件还有附件，我们还希望将每封邮件和它的每个附件都分开看成不同的文档。如果邮件的附件是一个 zip 文件，我们想将附件解压缩并将其中的每个文件都看成一个文档。与此不同的是，网络上的各种软件（如 latex2html）有时把我们认为的单一文档（如一个 PowerPoint 文件或 Latex 文档）以幻灯片或者小节为单位切分成多个 HTML 页面，并将每个页面保存到独立的文件中。这种情况下，我们反而希望将多个文件合并成一个文档来构建索引。

对于长文档而言，一个更一般的说法是说此时存在一个“索引粒度”（indexing granularity）的问题。对于一个书库来说，将整本书作为索引单位通常是个糟糕的做法。这种情况下，如果搜索“Chinese toys”，那么很有可能返回这样一本书，它的第 1 章提到了“China”而最后一章提到了“toys”，这显然不是我们想要的结果。可取的做法是将每章或每段看成一个微型文档（mini-document）来建立索引。此时，匹配的结果可能与查询更相关，而由于匹配单位的粒度更小，用户也更容易在文档中找到相关的段落。但是谈到这里，另一个问题出现了，为什么不进一步采用更小粒度的单位呢？比如，我们可以将每个句子作为索引单位。很显然，这里存在着一个正确率和召回率之间的权衡问题：如果索引粒度太小，那么由于词项散布在多个细粒度文档中，我们就很可能错过那些重要的段落，也就是说此时正确率高而召回率低；反之，如果索引粒度太大，我们就很可能找到很多不相关的匹配结果，即正确率低而召回率高。

当然，索引粒度过大造成的问题也可以通过采用显式或隐式的邻近搜索方法来缓解（参见 2.4.2 节和 7.2.2 节），上面提到的结果性能的权衡问题我们将在第 8 章讨论。索引粒度问题（特别是在需要对文档同时进行多种粒度索引时）常见于 XML 检索中，我们将在第 10 章再讨论这一问题。一个信息检索系统应该能够提供不同粒度索引的选项，在具体实施时如果要达到合理选择的目的，就必须要对文档集、用户及其可能的需求和使用模式等信息有一个非常深刻的理解。为叙述方便，在后面的讨论当中我们都假设已经选择了合理的索引粒度，同时，如果需要的话，我们也能通过非常合适的文件划分或合并方法来组织索引粒度。

① 也就是说，这些语言中的文字显示次序是混杂的。但是概念次序是一般是从左往右(即从前往后)，在 Unicode 编码中，字符存储也是从前往后的，而其显示次序由其他部件来完成。——译者注

2.2 词项集合的确定

2.2.1 词条化

定义好文档单位之后，词条化是将给定的字符序列拆分成一系列子序列的过程，其中每个子序列称为一个词条（token）。当然，在这个过程中，可能会同时去掉一些特殊字符，如标点符号等。下面给出了一个词条化的例子：

输入：Friends, Romans, Countrymen, lend me your ears;
输出：Friends | Romans | Countrymen | lend | me | your | ears

在非严格的情况下，词条往往和词项或词通用。然而，有时我们需要对词条和词条类进行严格的区分。一个词条指的是在文档中出现的字符序列的一个实例，而一个词条类（type）指的是相同词条构成的集合。一个词项指的是在信息检索系统词典中所包含的某个可能经过归一化处理的词条类。词项集合和词条集合可以完全不同，比如可以采用某一个分类体系中的类别标签作为词项。当然，在实际的信息检索系统中，词项往往和词条密切相关。但是，词项未必就是原始的词条，实际上它往往要通过对原始词条进行归一化来得到（参见 2.2.3）[①]。举例来说，如果要对“to sleep perchance to dream”进行索引，那么会得到 5 个词条，但是此时只有 4 个词条类（出现的两个 to 的实例归成一类）。然而，如果索引时将to看成停用词去掉（参见 2.2.2 节）的话，那么最后就只有 3 个词项：sleep、perchance和dream。

词条化的主要任务就是确定哪些才是正确的词条。对于上例而言，该任务实现起来似乎非常简单，因为此时只须根据空格将字符串进行拆分并去掉标点符号即可。然而，上面的例子仅仅代表的是一种最简单的情况，真实情况并非如此，实际上即使对于单词之间存在空格的英文来说也存在很多难以处理的问题。比如，英文中的上撇号“’”既可以代表所有关系也可以代表缩写，应当在词条化过程中究竟应该如何对它进行处理？参考下面的例子：

Mr. O’Neill thinks that the boys’ stories about Chile’s capital aren’t amusing.

对其中的“O’Neill”来说，词条化的结果可能有如下几种形式可以选择，那么到底哪一种才正确？

① 在上述定义下，没有进行索引的词条（如停用词）不是词项。如果多个词条通过归一化过程归结在一起，那么它们将作为一个词项并采用归一化结果来索引。然而，由于在第 13 到第 18 章的有关分类和聚类的讨论中并不存在索引问题，所以我们在那里放宽了上述定义。具体地，我们放弃了词项一定要包含在索引词典中的要求。在那几章中，词项就是一个归一化后的词。

neill
oneill
o'neill
o' neill
o neill?

同样，对于“ aren't”，我们也要从以下几种形式中选出我们所要的结果：

aren't
arent
are n't
aren t?

一个非常简单的做法就是在既非字母也非数字的字符处进行拆分，此时尽管 o neill 看上去还可以勉强凑合，但 aren t 直观看上去就很不好。对于上面可能的各种拆分策略来说，最后的选择结果会决定哪些布尔查询会被匹配上、哪些不会被匹配上。给定查询 neill AND capital，上述五种拆分策略中有 3 种会被匹配上（即第 1、4、5 种情况）。而如果给定查询 o'neill AND capital，则在没有对查询进行任何预处理的情况下，上述策略中只有一种能匹配上。不管是输入布尔查询或者自由文本查询，人们总是希望对文档和查询进行同样的词条化处理，这往往通过采用相同的词条化工具来实现。这样做能够确保文本与查询中的同一字符串序列的处理结果相一致[①]。

词条化处理往往与语言本身有关，不同语言下的词条化处理并不相同。因此，处理时必须要知道文档的语言种类。由于大多数语言都有自己独特的特征模式（参见 2.5 节给出的参考文献），因此一种非常有效的语言种类识别（language identification）方法是利用短字符子序列作为特征来进行分类。

对大多数语言特别是一些特定领域的语言来说，往往有一些特定的词条需要被识别成词项，如编程语言“ C++”和“ C#”、“ B-52”之类的飞行器名字或者叫“ M*A*S*H”的电视秀节目等等，其中“ M*A*S*H”已经成为一个固定的流行语，我们已经发现一些诸如“ M*A*S*H-style hospitals”之类的用法。计算机技术中也引入了很多新的字符序列类型，在处理时也应该将它们看成是独立的词条。这些新的字符序列类型包括邮件地址（如 jblack@mail.yahoo.com）、URL（如 http://stuff.big.com/new/specials.html）、IP 地址（如 142.32.48.231）和包裹追踪号码（1Z9999W99845399981）等等。一种做法是不对包括货币量、数字、URL 等在内的词条进行索引，这是因为如果对这些词条进行索引则会显著扩大索引的词汇量。当然，这样做会对用户的

① 这个结论对于自由文本来说非常直观，而对于布尔查询来说，情况则更复杂一些。这样的词条化方法可能对一个查询词产生多个词项。此时可以通过将这些词项通过 AND 操作符连接或者将它们合成一个短语查询（参见 2.4 节）来进行处理。与此相反，如果用户输入的两个词项可能在文档处理词条化的过程中被归并成一个，那么这种情况处理起来则要困难的多。

搜索产生一些限制。比如，人们可能会在程序缺陷（bug）库中搜索错误发生的行号，但是经过上述处理之后的系统显然不能返回正确结果。当然，有些具有明确语义的对象，比如电子邮件的发送日期，往往会被当成文档的元数据进行独立索引（参见 6.1 节）。

在英文中，连字符“-”存在着多种不同用法，比如它可以用于分隔单词中的元音字母（如 co-education），也可以用于将多个名词连接成一个新的名称（如 Hewlett-Packard），还可以在审稿设备上显示多个词语的组合结果（如 the hold-him-back-and-drag-him-away maneuver）。很容易判定，第一个例子应该作为一个词条来索引（实际上，该词条往往被书写为 coeducation）。最后一个例子应该分开为一个个的单词来索引。而对于第二个例子，判断起来却并不如其他两个例子那么容易。因此，对连字符的处理可能会很复杂，它可以当成一个分类任务来处理，也可以通过一些启发式规则来处理，比如允许单词包含连字符，但此时可以规定只允许出现短前缀而不是长前缀。

即使根据空格进行拆分有时也会将概念上本应该看成单个词条的对象分开，比如一些名称（San Francisco，Los Angeles）、外来短语（au fait）或那些书写时可分可合的复合词（white space vs. whitespace）。其他的例子还包括电话号码[（800） 234-2333]、日期（Mar 11, 1983）等。如果在空格处拆分这些对象可能会导致很差的检索结果，比如，输入 York University（约克大学）时会返回包含 New York University（纽约大学）的文档。连字符和空格甚至会互相影响，比如，有关航空公司价格的广告中往往包括诸如 San Francisco-Los Angeles 之类的说法，如果仅仅简单地从空格处拆分将会得到不好的结果。在这些情况下，词条化问题往往也会和短语查询的处理相互影响（将在 2.4 节讨论），尤其是当我们输入 lowercase 或 lower-case 或 lower case 都想返回相同结果时影响更大。后两种输入情况可以通过在连字符处拆分并对短语进行索引的方式来解决，但是要处理第一种输入情况就必须知道 lowercase 有时可以写成两个词，并且系统也要基于这两个词来建立索引。在一些布尔检索系统（如 Westlaw 和 Lexis-Nexis，参见例 1-1）中实际广泛采用的一种很有效的策略是，鼓励用户在输入查询时尽可能地加上连字符，而不论是否真正存在连字符形式，系统都能产生一个包含三种形式（一个词、中间加连字符或者两个词的形式）的查询。因此，用户输入查询“over-eager”时，系统将以 over-eager OR “over eager” OR overeager 为查询表达式进行真正的处理。当然，这种策略依赖于对用户的培训，如果用户不是以连字符方式输入而是采用其他任一方式，那么系统就不会以上述查询表达式进行搜索。

每种新的语言还会遇到一些新问题。例如，法语中有对元音开头的单词前面的定冠词“the”进行简化的省略用法（例如 l’ensemble）。另外，在祈使句和问句中的后依附代词中也有连字符的一些用法（如 donne-moi “give me”）。由于人们很希望不管是提到 l’ensemble 还是 un ensemble 的文档都可以用 ensemble 来查询到，因此对前一个例子处理的处理与否会影响到相当一部分名词和形容词索引的正确性。对于其他语言来说，问题可能会更复杂。德语的复合名词（compound noun）中一般没有空格（比如：Computerlinguistik – “computational linguistics”，Lebensversicherungsgesells chaftsangestellter – “life insurance company employee”）。德语检索系统

的效果很大程度上取决于一个称为复合词拆分器（compound splitter）的模块，该模块往往通过将一个词拆分成词典中的多个词来实现。对于一些主要的东亚语言（如汉语、日语、韩语和泰语等）来说，由于词之间并不存在空格，所以问题更加严重。图 2-3 给出了一个汉语的例子。一个解决办法就是先进行分词（word segmentation）。分词的方法包括基于词典的最大匹配法（采用启发式规则来进行未定义词识别）和基于机器学习序列模型的方法（如隐马尔可夫模型或条件随机场模型）等，后者需要在手工切分好的语料上进行训练（参见 2.5 节的参考文献）。由于存在多种切分可能（见图 2-4），上述分词方法都有可能导致错误的切分结果，因此，永远不能保证只能得到一个完全一致的唯一切分结果。另一个解决方法则摒弃了基于词的索引策略而采用短字符序列的方法（如字符的 *k*-gram 方法）。这种方法并不关心词项是否会跨越词的边界。该方法之所以能够引起人们的兴趣主要有以下 3 个原因：第一，一个汉字更像是一个音节而不是字符，它往往具有语义信息；第二，大部分词都很短（最常见的汉语词长度是 2 个字）；第三，由于缺乏公认的分词标准，词的边界有时也很难确定。即使对英语来说，有些情况下词边界的判断也缺乏标准，很多也只是习惯用法而已，比如 notwithstanding 和 not to mention、into 和 on to 中的词边界标准就不一样。当然，对于英语而言，在正规教育引导下人们都会尽量在书写单词时采用一致的空格用法。

莎拉波娃现在居住在美国东南部的佛罗里达。今年 4 月 9 日，莎拉波娃在美国第一大城市纽约度过了 1 8 岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

图 2-3　一段采用中国大陆简体中文编码的未切分文本。词之间甚至句子之间都没有空格，句号后面显示出的“空格”只是由于句号占显示位置的左下角而导致的排版结果。第一个句子全部由中文字符构成，中间没有空格。第二个和第三个句子中还包含阿拉伯数字和标点符号，它们可以将中文字符分开

和尚

图 2-4　汉语切分中的歧义现象。“和尚”可以看成一个词，也可以分成为“和”与“尚”两个单字词

2.2.2　去除停用词

某些情况下，一些常见词在文档和用户需求进行匹配时价值并不大，需要彻底从词汇表中去除。这些词称为停用词（stop word）。一个常用的生成停用词表的方法就是将词项按照文档集频率（collection frequency，每个词项在文档集中出现的频率）从高到低排列，然后手工选择那些语义内容与文档主题关系不大的高频词作为停用词。停用词表中的每个词将在索引过程中被忽略。图 2-5 给出了一个停用词表的片段。使用停用词表可以大大减小系统所需要存储的倒排记录表的数目，具体的统计数字可以参见表 5-1。不对停用词建立索引一般情况下不会对系统造成太大的影响，比如搜索时采用 the 或 by 进行查询似乎没有什么意义。但是，对于短语查询来

说情况并非如此，比如短语查询 President of the United States 中包含两个停用词，但是它比查询 President AND “United States”更精确。如果忽略掉 to，那么 flights to London 的意义将会丢失。搜索 Vannevar Bush 的那篇经典文章 *As we may think* 时，如果将前 3 个单词都看作停用词，那么搜索将会很困难，因为系统只返回包含 think 的文章。更为严重的是，一些特定的查询类型会受到更大的影响。比如一些歌名或者著名的诗歌片段可能全部由常用的停用词组成（如 *To be or not to be*，*Let It Be*，*I don't want to be* 等）。

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

图 2-5 Reuters-RCV1 语料库中的 25 个停用词

在信息检索系统不断发展的历程中，有从大停用词表（200～300 个词）到小停用词表（7～12 个词）最后到不用停用词的趋势。Web 搜索引擎通常都不用停用词表。一些现代 IR 系统更关注如何利用语言的统计特性来更好地处理常见词问题。在 5.3 节中，我们将介绍如何采用好的压缩技术来降低常用词倒排记录表的存储开销。6.2.1 节会讨论常规的词项权重计算方法，此方法能将高频常用词对文档排序的影响降至极低。7.1.5 节会介绍在一个索引按照影响度大小排序的 IR 系统中，当权重变小时，会及早停止对倒排记录表的继续扫描，因此对一般查询而言，即使停用词的倒排记录表非常大，它们也不会增加很多额外的处理开销。因此，对于现代 IR 系统来说，不论是对于索引大小还是查询处理的时间而言，不去除停用词所增加的开销并没有那么大。

2.2.3 词项归一化

将文档和查询转换成一个个的词条之后，最简单的情况就是查询中的词条正好和文档中的词条相一致。然而在很多情况下，即使词条之间并不完全一致，但实际上人们希望它们之间能够进行匹配。比如查询 USA 时我们希望能够返回包含 U.S.A.的文档。

词条归一化（token normalization）就是将看起来不完全一致的多个词条归纳成一个等价类，以便在它们之间进行匹配的过程 ①。最常规的做法是隐式地建立等价类 ②，每类可以用其中的某个元素来命名。比如，在文档和查询中，都把词条anti-discriminatory 和 antidiscriminatory映射成词项antidiscriminatory, 这样对两个词中的任一个进行搜索，都会返回包含其中任一词的文档。

我们看到，上面使用了去掉连字符的映射规则来构建等价类。这种处理方法的优点在于：一方面，等价类的建立过程是隐式的，不需要事先计算出等价类的全部元素，在映射规则下输出相同结果的词项一起构成等价类集合；另一方面，仅仅构建“ 去除字符” 这种映射规则也比

① 这个过程也往往称为词项归一化（term normalization），但是这里我们将词项这个术语作为归一化过程的输出而不是输入结果。

② 这里构造的等价类并不像同义词词表一样被显式构造出来，而是以映射规则的方式隐式存在。在映射规则下映射成相同结果的词条属于同一等价类。——译者注

较容易。当然，由于等价类隐式存在，所以构建“增加字符”这种规则时并不容易。例如，很难知道会将 antidiscriminatory 转化为 anti-discri minatory。

另一种建立等价类的方法是维护多个非归一化词条之间的关联关系。该方法可以进一步扩展成同义词词表的手工构建，比如将car 和 automobile归成同义词（这个话题将在第 9 章进一步讨论）。这些词项之间的关系可以通过两种方式来实现。常用的方式是采用非归一化的词条进行索引，并为某个查询词项维护一张由多个词组成的查询扩展词表。当输入一个查询词项时，则根据扩展词表进行扩展并将扩展后得到的多个词所对应的倒排记录表合在一块。另一种方式是在索引构建时就对词进行扩展。比如，对于包含automobile的文档，我们同时也用car来索引（同样，包含car的文档也用automobile来索引）[①]。上述两种方式相对于隐式建立等价类的方法来说效率较低，这是因为它们需要存储和合并更多的倒排记录。第一种方式增加了一部查询扩展词典，因此在查询处理时需要更多的时间。第二种方式需要更多的存储空间来存储倒排记录。传统上认为，高存储空间的方法往往不太有利，但是这对于现代的存储开销来说问题并不大，独特的倒排记录表（distinct postings list）结构所带来的灵活性提升可能更具吸引力。

另一方面，由于两个关联词的扩展词表之间可以存在交集但不必完全相同，所以上述两种方式相对于隐式建立等价类的方法来说更具灵活性。这也意味着从不同关联词出发可以进行不对称的扩展。图 2-6 给出了一个例子。该例子中，如果用户输入 windows，那么我们希望返回包含 Windows 操作系统的文档。但是如果用户输入 window，虽然此时可以和小写的 windows 相匹配，但是不太可能会和 Windows 操作系统中的 Windows 相匹配。

查询词项	文档中应当匹配的词项
Windows	Windows
windows	Windows, windows, window
window	window, windows

图 2-6　能够对用户期望进行有效建模的不对称扩展的例子

隐式建立等价类或查询扩展的使用幅度仍然是个开放的问题。适度使用绝对没错，但是过度使用很容易会在无意间造成非预期的扩展结果。例如，通过删除U.S.A.中的句点可以把它转化成USA，由于在首字母省略用法中存在这种转换模式，所以上面的做法乍看上去非常合理。但是，如果输入查询C.A.T.，返回的很多包含cat的文档却肯定不是我们想要的结果[②]。

接下来我们将给出一些在实际当中会遇到的词条归一化问题及其对策。进行词条归一化处理之后再很多情况下会提高检索的效果，但有时也可能会损害检索的效果。实际上，读者也许

① 也就是说这段中的第一种方式保存一个同义词表，索引时并不考虑词项之间的同义关系，而只在查询处理时利用该表进行扩展。而第二种方式在索引建立时就使用了同义词表，同义词所在的文档也会被索引。此时，查询处理时并不需要再扩展。——译者注

② 在本章撰写之际（2005 年 8 月），Google 的搜索结果就是如此：输入 C.A.T.后 Google 返回的最靠前的结果就是一个关于 cat 的网站——猫爱好者之家 www.fanciers.com/。

对等价类的建立细节有所担心，但是事实证明，只要对查询和文档的处理保持一致，等价类处理的细节就不会对检索的结果造成很大的影响。

重音及变音符号问题。英语中变音符号的使用越来越少见，尽管如此，人们很可能希望 cliche 和 cliché 或者 naive 和 naïve 能匹配。这可以通过在词条归一化时去掉变音符号来实现。而在许多其他语言中，变音符号属于文字系统的常规部分，不同的变音符号表示不同的发音。有时候，不同单词之间的区别只是重音不同。比如，西班牙语中，peña 的意思是“悬崖”，而 pena 的意思却是“悲哀”。然而，关键并不是规范或者语言学问题，而是用户如何构造查询来查找包含这些词的文档。出于各种原因，实际上在很多情况下用户并不会输入变音符号，这些原因包括输入速度、用户的惰性、软件限制以及许多计算机系统中难以使用非 ASCII 文本所养成的其他日常习惯等等。

大小写转换问题。大小写转换（case-folding）问题的一个一般处理策略是将所有的字母都转换成小写。这种做法通常的效果不错，比如这样可以允许句首的 Automobile 和查询 automobile 匹配。对于 Web 搜索引擎来说，这种做法也很有好处，因为大多数用户输入 ferrari 时实际想找的是 Ferrari（法拉利）车。但是，这种全部进行小写的处理方法可能会将那些应该区分的词语等同化。很多专有名词由普通名词构成，因此，大小写不同则意义也不同，而且这种意义上的不同只能通过大小写来区分。这种专有名词包括公司名称（如 General Motors，The Associated Press）、政府组织（the Fed 和 fed）和人名（Bush、Black）等等。前面提到的利用首字母缩写词进行查询扩展的例子中实际上并不只包含缩写归一化的问题（C.A.T.→ CAT），同时还包括大小写的问题（CAT→ cat）。

对英语来说，大小写处理的另一种做法是只将部分词条转换成小写形式。最简单的启发式处理方法是，将句首词转换成小写形式，并将标题中大写或首字母大写的全部单词都转换成小写形式。这些首字母大写的单词通常都是普通词。句中首字母大写的单词通常仍然保留其原来的形式，而这种处理通常都是正确的。在大小写形式需要区分的情况下，这样做通常能够避免大小写转换处理。对于上述工作，也可以采用机器学习中的序列模型，基于多种特征来实现更精确的大小写决策。这也称为真实大小写处理（truecasing）。然而，通过这类方法获得正确的大小写形式可能对用户并没有什么实际帮助，这是因为他们往往忽略大小写而使用小写方式来输入查询。因此，实际上通常采用全部转换成小写的做法。

英语中的其他问题。英语中还存在一些独特的归一化做法。比如，用户希望将 ne'er 和 never、英式英语的拼写方式 colour 和美式英语的拼写方式 color 等同起来。日期、时间和其他类似的对象往往以多种形式出现，这给归一化造成了额外的负担。人们可能希望将 3/12/91 和 Mar. 12, 1991 统一起来。但是，要正确处理这个例子将会十分复杂，因为在美国，3/12/91 指的 1991 年 3 月 12 日（Mar. 12, 1991），而在欧洲，却指的是 1991 年 12 月 3 日（3 Dec. 1991）。

其他语言的问题。英语在万维网上占据了主要地位，据 Gerrand（2007）估计，大概有 60% 的网页是英语网页。但是，仍然有 40%的网页是其他语言。由于全世界只有不到 1/3 的网民和

不到 10%的人口主要说英语，预计其他语言在网页中的比例将会随时间推移而不断增长。目前已经能看到一些转变的迹象，Sifry（2007）的报告显示，只有约 1/3 的博客帖子使用英语。

其他语言在进行等价类构建时同样会遇到一些特别的问题，比如在法语中，the 对应的那个单词的具体形式并不仅仅取决于人的性别、后续名词的单复数，还与后续单词是否以 le、la、l’ 或 les 等元音开头有关。我们要将 the 的这些形式进行归一化处理。在德语中有个习惯，带变音符的元音字母可以表示成一个双元音连字。比如我们会将 Schütze 和 Schuetze 看成同一词。

如图 2-7 所示，日语也是一个著名的难以处理的文字系统。现代日语主要由多种字母混合而成，主要包括汉字、两种字音表（平假名和片假名）和一些西方字符（拉丁字母、阿拉伯数字和各种符号）。尽管在教育系统中，文字的选择上存在着很强的惯例和标准，但是在很多情况下同一个词在不同文字系统中可以写成不同的形式。例如，日语中的一个词可以写成片假名形式以示强调（某种程度上类似于斜体方式），有时也可以写成平假名形式或汉字形式。因此，一个成功的日语检索系统需要面向复杂的不同文字系统的等价类构建方法。实际上，由于输入上的方便性，终端用户通常会提交平假名形式的查询，这和西方用户往往输入小写形式的查询是一样的道理。

待索引的文档集可以同时包括来自不同语言的文档，单篇文档中也很可能同时出现不同语言的文本。比如，一封法语邮件中也许会引述英文书写的合同文件条款。一个非常普遍的做法是，先判定语言种类，然后采用该语言的词条化和归一化规则在预定的粒度（如整篇文档或者每个段落）水平上进行处理，但是，这种方式仍然不能正确地处理在简短引述过程中发生语言变化的情况（如上面提到的法语邮件的例子）。当文档集包含多个语言时，单个索引中可能要包含来自不同语言的词项。一种做法就是，在文档上运行一个语言识别器，然后将每个词项的语言种类标记在词项词典中。或者根本不保留这些标记，因为对于同一字符串序列来说基本不会同时在不同语言中都是词。

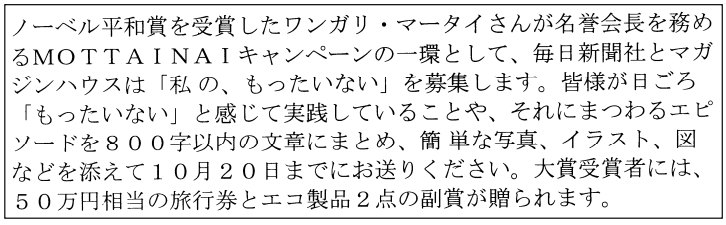

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるＭＯＴＴＡＩＮＡＩキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを８００字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて１０月２０日までにお送りください。大賞受賞者には、５０万円相当の旅行券とエコ製品２点の副賞が贈られます。

图 2-7　日语使用了多种混合的文字体系，并且同中文一样没有分词。上述文本主要由中文字符组成，并使用平假名音节作为屈折变化的词尾和功能词。上段文本中用拉丁字母书写的部分其实是个日语词汇，但却被 2004 年诺贝尔奖获得者 Wangari Maathai 作为一个环境保护运动的名称使用。他姓名的片假名写法位于第一行文本的中间。最后一行的前四个字符表示￥500 000（500 000 日元）

当处理外来词或者复合词，特别是外来的名称时，拼写可能不明确或者在不同的音译标准下会产生不同的拼写结果（如 Chebyshev 和 Tchebycheff、Beijing 和 Peking 等）。处理此类情况

的一种做法是，采用启发式方法来建立等价类，或者根据发音相同来进行词项扩展。后者最出名的算法是将在 3.4 节介绍的 Soundex 算法。

2.2.4 词干还原和词形归并

出于语法上的要求，文档中常常会使用词的不同形态，比如 organize、organizes 和 organizing。另外，语言中也存在大量意义相近的同源词，比如 democracy、democratic 和 democratization。在很多情况下，如果输入其中一个词能返回包含其同源词的文档，那么这样的搜索似乎非常有用。

词干[①]还原和词形归并的目的都是为了减少屈折变化的形式，并且有时会将派生词转化为基本形式。比如：

am, are, is ⇒ be

car, cars, car's, cars' ⇒ car

利用上述方式对文本进行映射处理，可以得到类似如下的结果：

The boy's cars are different colors ⇒

the boy car be differ color

然而，词干还原（stemming）和词形归并（lemmatization）这两个术语所代表的意义是不同的。前者通常指的是一个很粗略的去除单词两端词缀的启发式过程，并且希望大部分时间它都能达到这个正确目的，这个过程也常常包括去除派生词缀。而词形归并通常指利用词汇表和词形分析来去除屈折词缀，从而返回词的原形或词典中的词的过程，返回的结果称为词元（lemma）。假如给定词条 saw，词干还原过程可能仅返回 s，而词形归并过程将返回 see 或者 saw，当然具体返回哪个词取决于在当前上下文中 saw 到底是动词还是名词。这两个过程的区别还在于：词干还原在一般情况下会将多个派生相关词合并在一起，而词形归并通常只将同一词元的不同屈折形式进行合并。词干还原或词形归并往往通过在索引过程中增加插件程序的方式来实现，这类插件程序有很多，其中既有商业软件也有开源软件。

英文处理中最常用的词干还原算法是 Porter 算法（Porter 1980），它的高效性已反复被实践所证明。整个算法很长也很复杂，在这里我们就不详细介绍，下面只给出算法的核心思想。Porter 算法包括 5 个顺序执行的词约简步骤。在每个步骤中，都有选择规则的一些不同约定，比如从规则组中选择作用时词缀最长的那条规则。比如在第一步，规则选择方法如下：

（2-1）

规则		示例	
SSES	→SS	caresses	→caress
IES	→I	ponies	→poni
SS	→SS	caress	→caress
S	→	cats	→cat

带格式的: 英语(美国)

① 在语言学中，在词干（stem）和词根（root）两个概念的差别上仍然存在一些争议。有语言学家认为两者一样，也有语言学家认为它们是不一样的。因此，也有些人将 stemming 翻译成词根还原。这里我们为避免语言学上的争议，采用词干还原的译法。——译者注

后续步骤的很多规则中都使用了词的测度（measure）的概念，即通过粗略检查音节的个数来判断词是否足够长，在足够长的情况下才将规则的匹配部分看成词缀而不是词干的一部分来进行处理。这是一种合理的做法。例如：有一条规则为

（$m > 1$）　EMENT →

在这条规则下，可以把repalcement转换成replac，但是此时不能将cement转换成c[①]。Porter词干还原工具的官方网址是http://www.tartarus.org/˜ martin/PorterStemmer/。

另外，还存在一些其他的词干还原工具，比如比 Porter 更早的基于单遍扫描的 Lovins 词干还原工具（Lovins 1968），以及较新的诸如 Paice/Husk（Paice 1990）之类的还原工具等。具体信息请参见：

http://www.cs.waikato.ac.nz/˜eibe/ stemmers/

http://www.comp.lancs.ac.uk/computing/research/stemming/

图 2-8 给出了对上述不同词干还原工具的一个非正式的比较结果。词干还原工具使用和具体语言相关的规则，但是与需要完整词汇表及词形分析的词形归并工具相比，所需要的知识要少一些。特定的领域也许还需要特定的词干还原规则。当然，词干的精确形式本身并不重要，重要的是能够得到等价类[②]。

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can iead to a pict of express that is mor biolog transp and access to interpret

图 2-8　不同词干还原方法应用在样例文本上的结果比较

我们可以选择词形归并工具（lemmatizer）而不是词干还原工具来处理相关问题。词形归并工具是一种自然语言处理工具，它能够通过进行详尽的词形分析来精确地得到每个词的词元。进行详尽的词形分析只能为检索带来极其有限的帮助，不会太多，因为总体而言，上面两种归一化方法往往并不能显著改善英文检索的性能。尽管进行归一化对于部分查询来说有帮助，但是同时也会降低其他很多查询的效果。词干还原能够提高召回率但是会降低正确率。下面将给出一个例子，该例子使用 Porter 词干还原工具来进行归一化，通过这个例子，我们来看看其中

① 即 $m > 1$ 表示除去 EMENT 之外的单词部分的长度要大于 1。——译者注

② 这里的意思是词干还原得到的具体结果形式并不重要，重要的是它表明哪些词或者词项属于同一等价类。——译者注

带格式的: 书面挪威语(挪威)

带格式的: 书面挪威语(挪威)

带格式的: 书面挪威语(挪威)

带格式的: 书面挪威语(挪威)

究竟出了什么问题。利用 Porter 词干还原工具会将下列所有单词：

operate operating operates operation operative operatives operational

都转换为 oper。然而，operate 在大多数形式下的都是普通动词，因此，使用 Porter 词干还原工具会降低如下查询的正确率：

operational AND research
operating AND system
operative AND dentistry

对于上面的情况，即使采用词形归并工具仍然不能完全解决问题，这是因为在特定的搭配中使用了特定的屈折变化形式，比如一个包含 operate 和 system 的句子对于查询 operating AND system 来说并不是一个好的结果。要从词项归一化中获得更好效果取决于语用分析而不只是词形分析。

对于形态更加丰富的语言（如西班牙语、德语和芬兰语）来说，情况可能又不一样。欧洲 CLEF 评测的多次结果反复表明，词干还原工具（对于德语来说还包括复合词分割工具）的使用能够大大改善检索的结果，具体情况可以参见 2.5 节介绍的参考文献。

习题 **2-1** [*] 请判断如下说法是否正确。

a. 在布尔检索系统中，进行词干还原从不降低正确率。
b. 在布尔检索系统中，进行词干还原从不降低召回率。
c. 词干还原会增加词项词典的大小。
d. 词干还原应该在构建索引时调用，而不应在查询处理时调用。

习题 **2-2** [*] 请给出如下单词的归一化形式（归一化形式也可以是词本身）。

a. 'Cos
b. Shi'ite
c. cont'd
d. Hawai'i
e. O'Rourke

习题 **2-3** [*] 如下词经过 Porter 词干还原工具处理后会输出同样的结果，你认为哪对（几对）词不应该输出同样的结果？为什么？

a. abandon/abandonment
b. absorbency/absorbent
c. marketing/markets
d. university/universe
e. volume/volumes

习题 **2-4** [*] 对于（2-1）中所示 Porter 工具中使用的规则集：

a. 为什么要引入一条自己到自己的规则：SS→SS?
b. 采用该规则集进行处理，请给出如下词的词干还原结果。

circus canaries boss

c. 要对 pony 进行正确的词干还原处理，应该增加一条什么规则？

d. 对 ponies 和 pony 进行词干还原处理的结果看上去可能很怪异，这会对检索的结果造成负面影响吗？为什么？

2.3　基于跳表的倒排记录表快速合并算法

本章的剩余部分将讨论倒排记录表数据结构的其他扩展形式，以及如何提高倒排记录表的使用效率。回想一下在 1.3 节所讨论的倒排记录表的基本合并算法：同时在两个表中遍历，并且最后算法的时间复杂度为记录表大小的线性函数。假定两个表的大小分别是 m 和 n，那么合并过程有 $O(m+n)$ 次操作。很自然的一个问题就是我们能否做得更好？也就是说，能否在亚线性时间内完成合并过程？下面我们将看到，如果索引变化不太频繁的话那么答案是肯定的。

一种实现的方法是采用跳表（skip list），在构建索引的同时在倒排记录表上建立跳表（如图 2-9 所示）。跳表指针能够提供捷径来跳过那些不可能出现在检索结果中的记录项。构建跳表的两个主要问题是：在什么位置设置跳表指针？如何利用跳表指针进行倒排记录表的快速合并？

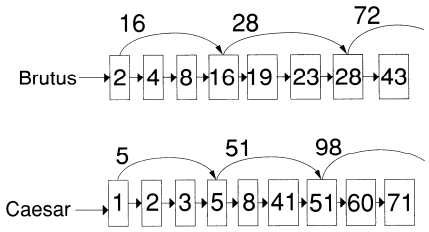


图 2-9　带有跳表指针的倒排记录表。当跳表指针目标项仍然小于另一个表的当前比较项时可以采用跳表指针直接跳转

我们以图 2-9 为例来先考虑快速合并的问题。假定我们在两个表中遍历一直到发现共同的记录 8 为止，将 8 放入结果表中之后我们继续移动两个表的指针。假定第一个表的指针移到 16 处，而第二个表的指针移到 41 处，两者中较小项为 16。这时候我们并不继续移动上面的表指针，而是检查跳表指针的目标项，此时为 28，仍然比 41 要小，因此此时可以直接把上表的表指针移到 28 处，这样就跳过了 19 和 23 两项。基于跳表的倒排记录表合并算法有很多变形，它们的主要不同可能在于跳表检查的时机不一样。图 2-10 给出了一种合并算法的例子。跳表指针只在原始倒排记录表上可用，对于复杂查询产生的中间结果，调用 `hasskip(p)` 函数则永远只会返回 `false`。最后，需要注意的是，跳表指针只对 AND 类型的查询有用，而对 OR 类型的查询不起作用。

我们再考察另一个问题，即在什么位置上放置跳表指针？这里存在一个指针个数和比较次数之间的折中问题。跳表指针越多意味着跳跃的步长越短，那么在合并过程中跳跃的可能性也更大，但同时这也意味着需要更多的指针比较次数和更多的存储空间。跳表指针越少意味着更

少的指针比较次数，但同时也意味着更长的跳跃步长，也就是说意味着更少的跳跃机会。放置跳表指针位置的一个简单的启发式策略是，在每个 $\sqrt{P}$ 处均匀放置跳表指针，其中 P 是倒排记录表的长度。这个策略在实际中效果不错，但是仍然有提高的余地，因为它并没有考虑查询词项的任何分布细节。

```
IntersectWithSkips(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then Add(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8               then if hasSkip(p1) and (docID(skip(p1)) ⩽ docID(p2))
 9                       then while hasSkip(p1) and (docID(skip(p1)) ⩽ docID(p2))
10                               do p1 ← skip(p1)
11                       else p1 ← next(p1)
12               else if hasSkip(p2) and (docID(skip(p2)) ⩽ docID(p1))
13                       then while hasSkip(p2) and (docID(skip(p2)) ⩽ docID(p1))
14                               do p2 ← skip(p2)
15                       else p2 ← next(p2)
16  return answer
```

图 2-10 基于跳表指针的倒排记录表合并算法

如果索引相对固定的话，建立有效的跳表指针则比较容易。但是如果倒排记录表由于经常更新而发生变化，那么跳表指针的建立就比较困难。恶意的删除策略可能会使跳表完全失效。

对于系统构建者来说，选择倒排索引的最优编码方法永远是个不断变化的“游戏”。这是因为，编码的效果严重依赖于后台的多种计算机技术及其相对速度和大小。在传统计算机中，CPU 速度较慢，因此，具有高压缩率的技术并不是最优的。而现代计算机的 CPU 速度很快，但是磁盘访问速度仍然很慢，因此如何减少磁盘上倒排记录表的存储空间便成了重点。然而，对于一个将所有部件都放到内存运行的搜索引擎来说，情况又会大不一样。4.1 节将会介绍硬件参数对索引构建的影响，而第 5 章将讨论索引大小给系统速度带来的影响。

? 习题 **2-5** [*] 为什么跳表方式并不适用于或查询 x OR y?

习题 **2-6** [*] 对于两个词组成的查询，其中一个词（项）的倒排记录表包含下面 16 个文档 ID：

[4,6,10,12,14,16,18,20,22,32,47,81,120,122,157,180]

而另一个词（项）对应的倒排记录表仅仅包含一个文档 ID：

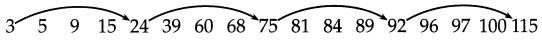
[47]

请分别采用如下两种策略进行倒排记录表合并并计算所需要的比较次数，同时简要地说明计算的正确性。

a. 使用标准的倒排记录表。

b. 使用倒排记录表+跳表的方式，跳表指针设在 $\sqrt{P}$ 处。

习题 **2-7** [*]　考虑利用如下带有跳表指针的倒排记录表



和一个中间结果表（如下所示，不存在跳表指针）进行合并操作。

3　5　89　95　97　99　100　101

采用图 2-10 所示的倒排记录表合并算法，请问：

a. 跳表指针实际跳转的次数是多少（也就是说，指针 p_1 的下一步将跳到 `skip(p1)`）？

b. 当两个表进行合并时，倒排记录之间的比较次数是多少？

c. 如果不使用跳表指针，那么倒排记录之间的比较次数是多少？

2.4　含位置信息的倒排记录表及短语查询

很多复杂的或技术性的概念、机构名和产品名等都是由多个词语组成的复合词或短语。用户希望能够将类似 Stanford University 的查询中的两个词看成一个整体，从而一篇含有句子 The inventor Stanford Ovshinsky never went to university 的文档不会与该查询匹配。大部分搜索引擎都提供了双引号语法（如“ Stanford University”）来支持短语查询，这种语法很容易理解并被用户成功使用。大概有 10%的 Web 查询是短语查询，有更多的查询虽然输入时没有加双引号，但实际上是隐式的短语查询（如人名）。要支持短语查询，只列出词项所在的文档列表的倒排记录表已不足以满足要求。本节当中，将考虑两种支持短语查询的方式及它们的交叉使用。搜索引擎要不仅支持而且能够高效实现短语查询。一个相关但是截然不同的概念是基于词项邻近关系的权重设置（term proximity weighting），在该设置下一个包含多个词项并且词项出现的位置较接近的文档会被优先返回。这种技术将在 7.2.2 节讨论。

2.4.1　二元词索引

处理短语查询的一个办法就是将文档中每个接续词对看成一个短语。例如，文本 Friends, Romans, Countrymen 会产生如下的二元接续词对（biword）：

friends romans

romans countrymen

这种方法将每个接续词对看成词项，这样马上就能处理两个词构成的短语查询，更长的查询可以分成多个短查询来处理。比如，按照上面的方法可以将查询 stanford university palo alto 分成如下的布尔查询：

“stanford university” AND “university palo” AND “palo alto”

可以期望该查询在实际中效果会不错，但是偶尔也会有错误的返回例子。对于该布尔查询返回的文档，我们并不知道其是否真正包含最原始的四词短语。

在所有可能的查询中，用名词和名词短语来表述用户所查询的概念具有相当特殊的地位。

但是相关的名词往往被各种虚词分开，比如短语the abolition of slavery或者renegotiation of the constitution。这种情况下，可以采用如下方法来建立二元词索引：首先对文本进行词条化然后进行词性标注[①]，这样就可以把每个词项归成名词（N，也包括专有名词）、虚词（X，冠词和介词）和其他词。然后将形式为NX*N非词项序列看成一个扩展的二元词。每个这样的扩展二元词对应一个词项。例如：

renegotiation of the constitution
N X X N

要利用这样的扩展二元词索引处理查询，也需要将查询分析成 N 和 X，然后将查询划分成扩展的二元词，最后在索引中进行查找。

当将长查询分析成布尔查询时，上述算法在直观上并不总是最好。利用上述算法，可以将查询 cost overruns on a power plant 分析成 “cost overruns” AND “overruns power” AND “power plant”，实际上忽略中间的那个二元词所形成的查询的效果会更好。如果使用更精确的词性模式来定义扩展二元词可能会取得更好的结果。

二元词索引的概念可以扩展到更长的词序列，如果索引中包含变长的词序列，通常就称为短语索引（phrase index）。实际上，利用二元词索引来处理单个词的查询不太方便（必须要扫描整个词汇表来发现包含该查询词的二元词），因此同时还需要有基于单个词的索引。尽管总有可能得到错误的匹配结果，但是在长度为 3 或者更长的索引短语上发生匹配错误的可能性实际上却很小。然而在另一方面，存储更长的短语很可能会大大增加词汇表的大小。穷尽所有长度超过 2 的短语并维护其索引绝对是一件令人生畏的事情，即使只穷尽所有的二元词也会大大增加词汇表的大小。然而，本节结束之前，我们将讨论在一个复合索引机制下如何有效地使用部分短语索引。

2.4.2 位置信息索引

很显然，基于上面谈到的原因，二元词索引并非标准的解决方案。实际中更常用的一种方式是采用所谓的位置信息索引（positional index，简称位置索引）。在这种索引中，对每个词项，以如下方式存储倒排记录（见图 2-11）。

文档 ID：〈位置 1，位置 2，…〉。

① 词性标注工具将词标记为名词、动词等，或者在实际中，常常可以标记成更细的类型如复数专有名词。现在有很多相当精确的词性标注工具（平均每种类型可以达到 96%的精确率），这些工具通常都是基于手工标注的文本训练的机器学习方法。参见 Manning and Schütze（1999, 第 10 章）。

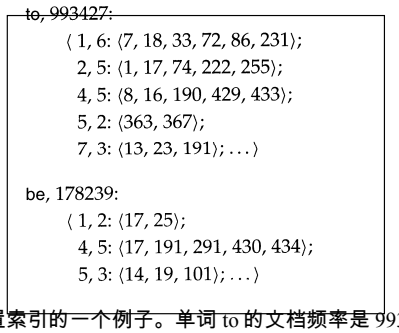


图 2-11　位置索引的一个例子。单词 to 的文档频率是 993 427，在文档 1 中出现 6 次，位置分别是 7、18、33 等等

我们在第 6 章将会讲到，在上述每个记录中同时也保存了文档中的词项频率信息。

为处理短语查询，仍然需要访问各个词项的倒排记录表。像以往一样，这里可以采用最小文档频率优先的策略，从而可以限制后续合并的候选词项的数目。在合并操作中，同样可以采用前面提到的各种技术来实现，但是这里不只是简单地判断两个词项是否出现在同一文档中，而且还需要检查它们出现的位置关系和查询短语的一致性。这就需要计算出词之间的偏移距离。

例 **2-1**　短语查询请求的应答处理。假定to和be的倒排记录表如 2-11 所示，某个查询为to be or not to be。需要访问如下词的倒排记录表：to、be、or和not。考虑to和be的倒排记录表的合并：首先，查找同时包含这两个词项的文档；然后，检查表中的位置看看是否有某个be的前面一个词条位置 ①上正好出现to。图 2-11 中给出的倒排记录表中，可能存在的一个匹配为：

to: 〈. . . ; 4: 〈. . . ,429,433〉 ; . . . 〉

be: 〈. . . ; 4: 〈 . . ,430,434〉 ; . . . 〉

同样，类似的方法可以用于 *k* 词近邻搜索（参见例 1-1）当中：

employment /3 place

这里，/*k* 意味着“从左边或右边相距在 *k* 个词之内”。很显然，位置索引能够用于邻近搜索，而二元词索引则不能。图 2-12 给出了一个用于 *k* 词近邻搜索的算法，习题 2-12 将对它进行深入的讨论。

① 这里给出的位置信息的例子（包括图 2-11 和例 2-1 中的 429、433、430 和 434）并不是我们通常所理解的词项在文档中的偏移位置，而是文档中词条的序号。——译者注

```
POSITIONAL INTERSECT(p1, p2, k)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then l ← ⟨ ⟩
 5             pp1 ← positions(p1)
 6             pp2 ← positions(p2)
 7             while pp1 ≠ NIL
 8             do while pp2 ≠ NIL
 9                do if |pos(pp1) − pos(pp2)| ⩽ k
10                      then ADD(l, pos(pp2))
11                      else if pos(pp2) > pos(pp1)
12                              then break
13                   pp2 ← next(pp2)
14                while l ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
15                do DELETE(l[0])
16                for each ps ∈ l
17                do ADD(answer, ⟨docID(p1), pos(pp1), ps⟩)
18                pp1 ← next(pp1)
19             p1 ← next(p1)
20             p2 ← next(p2)
21        else if docID(p1) < docID(p2)
22                then p1 ← next(p1)
23                else p2 ← next(p2)
24  return answer
```

图 2-12 邻近搜索中两个倒排记录表 p_1 和 p_2 的合并算法，算法寻找两个词项在 k 个词之内出现的情形，返回一个三元组<文档 ID，词项在 p_1 中的位置，词项在 p_2 中的位置>的列表

位置索引的大小。采用位置索引会大大增加倒排记录表的存储空间，即使对位置值或偏移值采用 5.3 节的压缩方法也无济于事。实际上，采用位置索引会加深倒排记录表合并操作的渐进复杂性，这是因为需要检查的项的个数不再受限于文档数目而是文档集中出现的所有的词条的个数 T。也就是说，布尔查询的复杂度为$\Theta(T)$而不是$\Theta(N)$。然而，由于用户往往期望能够进行短语搜索和邻近搜索，所以实际中的大部分应用并没有其他选择而不得不采用这种做法。

我们回头再看看采用位置索引的空间消耗。位置索引需要对词项的每次出现保留一个条目，因此索引的大小取决于文档的平均长度。网页的平均长度往往不到 1 000 个词项，而诸如证管会（SEC）股票文件、书籍甚至一些英雄史诗等文档的长度则很容易就能达到 100 000 个词项。考虑在平均 1 000 个词项中每个词项的频率都是 1，最后的结果就是：大文件条件下会造成索引存储空间大小的两个数量级规模的增长，如下所示。

文档规模	预计的倒排记录数	位置索引中预计的条目数
1 000	1	1
100 000	1	100

尽管确切的数字要视文档及其语言的具体类型而定，但是利用一些粗略的经验法则可以预期位置索引大概是非位置索引大小的 2～4 倍，而压缩后的位置索引大概为原始未压缩文档文本（去除标记信息）的 1/3～1/2。一个具体文档集的相关数字参见表 5-1 和表 5-6。

2.4.3　混合索引机制

二元词索引和位置索引这两种策略可以进行有效的合并。假如用户通常只查询特定的短语，如Michael Jackson，那么基于位置索引的倒排记录表合并方式效率很低。一个混合策略是：对某些查询使用短语索引或只使用二元词索引，而对其他短语查询则采用位置索引。短语索引所收录的那些较好的查询可以根据用户最近的访问行为日志统计得到，也就是说，它们往往是那些高频常见的查询。当然，这并不是唯一的准则。处理开销最大的短语查询往往是这样一些短语，它们中的每个词都非常常见，但是组合起来却相对很少见[①]。将查询Britney Spears 加入短语索引可能仅仅对该查询提供一个大概 3 倍的加速效果，这是因为很多提到其中一个单词的文档都是相关文档。而如果将The Who加入短语索引那么会对这个查询有 1 000 的加速效果。因此，实现中更期望将后者加入到短语索引中，尽管相对前者，其出现的频率较低（也就是说这些短语都是非常见查询）。

Williams 等人 (2004) 评估了一个更复杂的混合索引机制，其中除了包含上面两种形式的索引外，还在它们之间引入了一个部分后续词索引（next word index），即对每个词项，有个后续词索引记录了它在文档中的下一个词项。论文的结论是，虽然比仅仅使用位置索引增加了 26% 的空间，但是面对典型的 Web 短语混合查询，其完成时间大概是只使用位置索引的 1/4。

? 习题 **2-8** [*]　假设采用二元词索引，请给出查询 New York University 的一个返回文档的例子，它实际上与查询不相关。

习题 **2-9** [*]　下面给出的是一个位置索引的一部分，格式为：

词项: 文档 1:（位置 1, 位置 2, …）; 文档 2:（位置 1, 位置 2, …）;
angels: 2:（36,174,252,651）; 4:（12,22,102,432）; 7:（17）;
fools: 2:（1,17,74,222）; 4:（8,78,108,458）; 7:（3,13,23,193）;
fear: 2:（87,704,722,901）; 4:（13,43,113,433）; 7:（18,328,528）;
in: 2:（3,37,76,444,851）; 4:（10,20,110,470,500）; 7:（5,15,25,195）;
rush: 2:（2,66,194,321,702）; 4:（9,69,149,429,569）; 7:（4,14,404）;
to: 2:（47,86,234,999）; 4:（14,24,774,944）; 7:（199,319,599,709）;
tread: 2:（57,94,333）; 4:（15,35,155）; 7:（20,320）;
where: 2:（67,124,393,1001）; 4:（11,41,101,421,431）; 7:（16,36,736）;

那么哪些文档和以下的查询匹配？其中引号内的每个表达式都是一个短语查询。

a. “fools rush in”；

b. “fools rush in” AND “angels fear to tread”。

习题 **2-10** [*]　考虑如下位置索引的片段，格式为：

词: 文档:（位置, 位置, . . .）; 文档:（位置, . . .）
. . .

① 也就是说，虽然这些短语出现得不够频繁，但是通过短语索引后的处理效率会有更大的提高，所以也常常将这类短语加入短语索引中。——译者注

Gates: 1: （3）; 2: （6）; 3: （2,17）; 4: （1）;
IBM: 4: （3）; 7: （14）;
Microsoft: 1: （1）; 2: （1,21）; 3: （3）; 5: （16,22,51）;
/*k* 操作符的意思是，当查询为词 1 /*k* 词 2 时表示查找时词 1 和词 2 必须在 *k* 个词内出现（匹配时左右两边计算都行），其中参数 *k* 是一个正整数。若 *k* = 1，则意味着语 1 和词 2 相邻。

a. 给出满足查询 Gates /2 Microsoft 的所有文档；

b. 将 *k* 值划分成多个集合，使得同一集合内的 *k* 值对于查询 Gates /*k* Microsoft 返回同样的文档子集，而不同集合中的 *k* 值则返回不同的文档子集。

习题 **2-11** [**] 给定某篇文档，考虑位置索引的一般合并过程，为了确定文档中满足/*k* 子句的位置所在（通常来说，每个词项可能在单篇文档中的多个位置上出现），一开始每个词项都有一个指针对应其出现位置，然后该指针不断后移，在每个位置上都会判断是否满足/*k* 子句的要求。任一指针的每次移动都算成一步。假定两个词项在文档当中的总出现次数是 *L*，请问采用位置索引进行合并并在结果中输出所有位置信息的复杂度是多少（用 *O* 表示）？

习题 **2-12** [**] 考虑倒排记录表基本合并算法（图 1-6）的修改算法（图 2-12），后者能够处理邻近查询。一个处理该类查询的朴素算法的复杂度可能为 $O(PL_{\max}^2)$，其中 *P* 是所有倒排记录表的长度之和（实际上是词项的文档频率之和），L_{max} 是最长的文档的长度（以词条为单位）。

a. 分析该算法，说明它为什么满足上述复杂度。

b. 该算法的复杂度是多少？为什么？

c. 对于某些查询和数据分布，是否存在其他的更有效的算法？其复杂度是多少？

习题 **2-13** [**] 假定需要通过倒排记录表合并过程来简单确定满足 /*k* 子句的文档列表，而不需要像图 2-12 一样输出位置信息。简单起见，假定 $k \geq 2$。设 *L* 是两个词项在文档集出现的次数之和（也就是它们的文档集频率之和）。以下哪些说法是正确的？

a. 合并过程能在 *L* 的线性步内完成且与 *k* 无关，合并过程中每个指针只会向右移动；

b. 合并过程能在 *L* 的线性步内完成且与 *k* 无关，但是合并过程中指针可能不只是单向移动（比如，有时可以往回移动）；

c. 某些情况下合并过程可能需要 *kL* 步。

习题 **2-14** [**] IR 系统中如何同时使用位置索引和停用词表？潜在问题是什么？如何解决？

2.5 参考文献及补充读物

东亚语言处理中基于字的处理方法的集中讨论参见Lunde（1998）。尽管有些中文检索系统先分词然后再基于词建立索引，但二元字索引也许是中文文本最常规的索引方法。由于语言和文字系统的差异，分词通常是日语处理的常用步骤（Luk和Kwok，2002；Kishida 等人，2005）。在未分词文本上的基于字的*k*-gram索引结构和 3.2.2 节所讨论的并不一样：后者中每个*k*-gram指向的是倒排索引词典中的每个词条 ①，而前者则是直接指向倒排记录表。对于中文分词的进一

① 即建立索引的索引。——译者注

步讨论，可以参考如下论文（Sproat 等，1996；Sproat和Emerson，2003；Tseng 等人，2005；Gao 等人，2005）。

Lita 等人（2003）提出了一种真实大小写处理（truecasing）的方法。有关词语形态处理的自然语言方法参见（Sproat，1992；Beesley 和 Karttunen，2003）。语言类型的识别可能最早在密码学领域被引入，比如，Konheim（1981）给出了一个基于字符的 *k*-gram 语言识别方法。尽管随着数字文本的迅速传播，其他通过查找特定的有区别性的功能词和字母组合的语言识别方法也在被使用，然而基于字符的 *k*-gram 方法被很多人证实非常成功（Beesley，1998；Cavnar 和 Trenkle，1994；Dunning，1994）。书面语言的识别也因此被认为是一个非常容易解决的问题，但是口语的识别仍然非常困难，有兴趣的读者可以参看 Hughes 等人（2006）的一篇近期的综述。

有关词干还原的正面及负面作用的实验和讨论参见如下工作：Salton（1989）、Harman（1991）、Krovetz（1995）及 Hull（1996）。Hollink 等人（2004）给出了在 8 种欧洲语言上采用与语言相关的方法的详细结果。以 MAP（平均准确率）的提高百分比来计算，去掉读音符号能够比基准方法提高 23%（对芬兰语、法语和瑞典语特别有效），词干还原对芬兰语和西班牙语非常有效（对前者提高了 30%，对后者提高了 10%），但是对于包括英语在内的其他绝大多数语言，词干还原带来的提高仅仅在 0 到 5%之间，采用词形归并的效果则更差。复合词分割能够给瑞典语带来 25%的提高，给德语带来 15%的提高，但是对于荷兰语来说，提高值只有 4%。不采用依赖于语言种类的方法，而采用 *k*-gram 字符索引的方法（推荐中文使用）通常也能得到很好的结果。不采用词索引，而是采用词内字符 4-gram 索引的方法能够分别为芬兰语、瑞典语、德语带来 37%、27%、20%的提高，同时对于其他语言，如荷兰语、西班牙语和英语来说，效果都有所提高。Tomlinson（2003）也得出了类似的结论。Bar-Ilan 和 Gutman（2005）声称，他们在 2003 年的研究当中发现，主要的商业搜索引擎缺乏合适的语言相关的处理技术，比如，利用 www.google.fr 网站查找 l' électricité 不会把冠词 *l'* 分离出来，而是仅仅返回精确包含该字符串的网页。

有关跳表在 IR 中使用的经典文章参见 Moffat 和 Zobel（1996）。相关的扩展技术在 Boldi 和 Vigna（2005）中有所讨论。讨论有关 IR 算法方面的主要论文是 Pugh（1990），它利用多级跳表指针获得 $O(\log P)$ 的访问效率（和利用树结构的期望效率一样），实现时复杂度更低。实际上，使用跳表指针的效果依赖于各种系统的参数。Moffat 和 Zobel（1996）报告说“与”查询可以通过跳表获得 5 倍的速度提升，但是 Bahle 等人（2002, 第 217 页）则报告说，在现代 CPU 下，由于跳表增加了倒排记录表的大小，因此利用跳表会降低搜索的速度（也就是说，此时磁盘 I/O 对性能起决定性作用）。与之相对的是，Strohman 和 Croft（2007）的工作则再次表明，在一个为大内存、多核 CPU 做优化的系统架构设计中，通过跳表能够获得更好的性能。

Johnson 等人（2006）报告说，在 2002 年的 2 个 Web 搜索日志中，大概有 11.7%的查询包含短语查询，而 Kammenhuber 等人（2006）基于另一份数据集的报告结果只有 3%。Silverstein 等人（1999）指出很多显式上不包含短语操作符的查询实际上也是短语查询。

第3章 词典及容错式检索

在前两章中，我们介绍了倒排索引及其相关技术。在这些技术的支持下，我们能够处理布尔查询及邻近查询。本章，我们将介绍对查询中存在拼写错误或存在不同拼写形式具有鲁棒性的拼写校正技术。3.1 节主要介绍支持词典快速查找的多个数据结构。3.2 节主要介绍通配符查询（wildcard query），比如输入查询*a*e*i*o*u*，将返回包含上述 5 个元音字母并满足其先后次序要求的词项所在的文档。其中，通配符“ *”表示任意字符串或者空串。用户向搜索引擎提交此类查询时，可能是因为对输入词项的拼写形式没有绝对把握，或者是想找包含查询词以及其他一系列变体形式的文档。例如，输入查询 automat* 会返回包含词项 automatic、automation 及 automated 的所有文档。

3.3 节将转向介绍另外一种非精确的查询形式，即拼写上存在错误的查询。用户可能无意中会出现查询的拼写错误，或者是用户在检索查询词（如人名 Herman）时，并不清楚其在文档集中的正确拼写形式。本章详细介绍了一系列查询拼写的自动校正技术，其中包括针对每个查询词的独立校正技术，以及针对整个查询串的整体校正技术。最后，3.4 节考察了一种在词汇表中查找与查询词发音近似的词项的方法。由于用户提交查询时可能并不清楚文档集中专有名称的具体拼写形式，因此该方法对于诸如 Herman 的查询尤其有效。

由于本章中介绍了倒排索引结构的多种不同的版本，因此有时我们将第 1 章、第 2 章中所定义的倒排索引称为普通倒排索引（standard inverted index）。在普通倒排索引中，词汇表中的每个一词项都会对应一个由文档集中的多篇文档组成的倒排记录表。实际上，本章介绍的倒排索引对普通倒排索引中的词典部分再进行了一层索引，通过本章的倒排索引结构可以找到词项，然后通过普通倒排索引最终定位到文档。

3.1 词典搜索的数据结构

假设给定倒排索引及查询，那么我们的首要任务是确定每个查询词项是否在词汇表中，如果在，则返回该词项对应的倒排记录表的指针。词汇表的查找操作往往采用一种称为词典（dictionary）的经典数据结构，并且主要有两大类解决方案：哈希表方式和搜索树方式。在数据结构相关的文献中，词汇表中的每个条目（这里是词项）常常称为关键字或键（key）。选择具体解决方案（不论是哈希表方式和搜索树方式）时，通常要考虑如下问题：

(1) 关键字的数目有多少？

(2) 关键字的数目是固定的还是经常变化？在变化的情况下，是只插入新关键字，还是同时要删除某些旧关键字？

(3) 不同关键字的相对访问频率如何？

哈希表方式已在某些搜索引擎中用于词典查找。这种方式下，每个词项通过哈希函数映射成一个整数，映射函数的目标空间需要足够大，以减少哈希结果冲突的可能性。当然，这种方式很难避免冲突的发生，此时需要精心维护一个辅助结构来解决冲突问题[①]。查询时，对于每个查询项分别进行哈希操作，并解决存在的冲突，最后返回每个查询词项对应的倒排记录表的指针。由于查询词项稍有变化后哈希函数会生成截然不同的结果，哈希表方式难以处理查询词项存在轻微变形的情况（如单词resume的重音符和非重音符版本）。特别地，哈希表方式很难处理 3.2 节提到的前缀式查询，如查找以automat开始的词项所在的文档。最后需要指出的是，在词汇表不断增长的环境（如Web）中，为当前需求所设计的哈希函数可能会在几年内很快失效。

搜索树方式能够解决上面提到的大部分问题。比如，它可以支持以automat开始的前缀式查询。最出名的搜索树莫过于二叉树（binary tree）[②]，其每个内部节点都有两个子节点。在二叉树中搜索词项要从根节点开始，而每个内部节点（包括根节点）都代表一个二值测试，测试的结果用于确定下一步应该搜索的子树。图 3-1 给出了一个基于二叉树表示的词典的例子。二叉树的平衡性是实现高效搜索（此时比较次数为$O(\log M)$）的关键，也就是说任何节点两棵子树下的词项数目要么相等要么相差 1。这里的首要问题就是，如何在加入或删除词项时保持树的平衡性。而为了做到这一点，在增删词项时往往需要对树节点进行重新的平衡化处理。

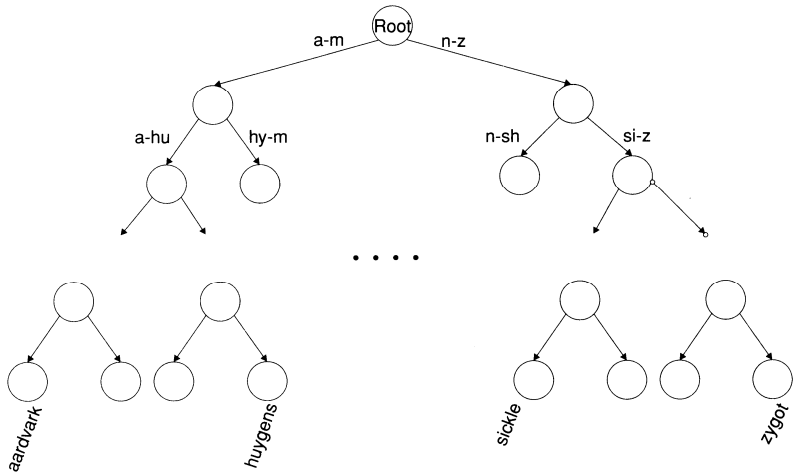


图 3-1 一棵二叉搜索树。图中，根节点将所有的词项按照首字母划分成左右两个部分，左边是从字母 a 到 m 开头的词项，右边

① 有一种称为完美哈希函数的方法能够避免冲突的发生，但是这类方法不论是在实现上还是在计算上都非常复杂。

② 这里的二叉树实际上是二叉排序树(Binary Sort Tree)，也称二叉查找树或二叉搜索树，即词项是有序的。当二叉树满足平衡性时，称为平衡二叉树(Balanced Binary Tree)，也称 AVL 树。——译者注

为其余词项

为了减轻重新平衡化处理的开销，有一种方法允许内部节点的子树数目在某个固定区间内变化。词典搜索中普遍使用的 B-树就是这类方法的一个实例。它每个内部节点的子节点数目在区间[*a*,*b*]内取值，其中 *a* 和 *b* 都是某个合适的正整数。图 3-2 给出了一棵 B-树的例子，其中 *a*=2，*b*=4。同图 3-1 中的二叉树一样，B-树上的每个内部节点也代表一个用于确定字符序列范围的测试条件。B-树可以看成是将二叉树的多层“压平”到一层而得到的树结构，在内存空间不足以存下全部词典而必须要将部分词典常驻磁盘时，这样做非常有效，因为在这种情况下“压平”可以在将词典调入内存时实现后续二值测试的预读取。此时，B-树中的整数 *a* 和 *b* 取决于磁盘块的大小。有关搜索树和 B-树的更多内容请参考 3.5 节给出的参考资料，这里不再介绍。

需要指出的是，和哈希表方式不同，搜索树要求树中的字符集有预定义的排序方式，比如，英语中 26 个字母常常按照从 A 到 Z 的顺序排列，而包括汉语在内的一些亚洲语言可能有多种排序方法。当然，现在所有的语言（包括汉语和日语在内）在系统中都有一个常规的排序方法。

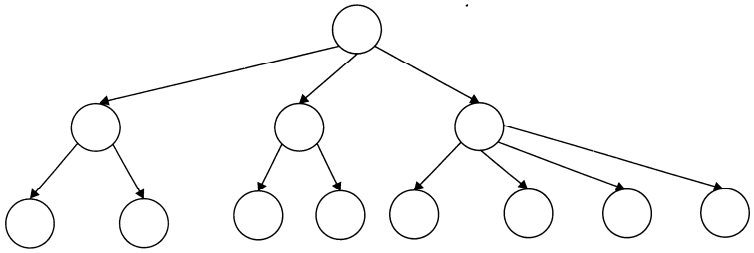


图 3-2　一棵 B-树的例子，该树中每个内部节点有 2～4 个子节点

3.2　通配符查询

通配符查询往往适用于如下任何一种场景：

(1) 用户对查询的拼写不太确定（比如，如果不能确定是 Sydney 还是 Sidney，就采用通配符查询 S*dney）；

(2) 用户知道某个查询词项可能有不同的拼写版本，并且要把包含这些版本的文档都找出来（比如 color 和 colour）；

(3) 用户查找某个查询词项的所有变形，这些变形可能还做了词干还原，但是用户并不知道搜索引擎是否进行了词干还原（比如 judicial 和 judiciary，可采用通配符查询 judicia*）；

(4) 用户不确定一个外来词或者短语的正确拼写形式（如查询 Universit* Stuttgart）。

因为通配符*在查询字符串末尾仅出现一次，所以一个诸如 mon* 的查询称为尾通配符查询（trailing wildcard query）。基于搜索树的词典结构对于处理尾通配符查询来说非常方便，比如对于查询 mon*，我们可以依次按照字符 m、o、n 从上到下遍历搜索树，直到能列举词典中所有以 mon 开头的词项集合 *W* 时为止。最后，在普通倒排索引中进行|*W*|次查找后取出 *W* 中所有词

项所对应的文档。

上述做法中存在的一个明显的问题就是，如果通配符不出现在查询尾部应该如何处理？在处理这种更一般的情况之前，首先对尾通配符查询做一个小小的推广。考虑首通配符查询（leading wildcard query）或者说*mon形式的查询。此时可以引入词典的反向B-树（reverse B-tree）结构，在该结构中，原来B-树中的每个从根到叶子路径所代表的词项全部反过来写。因此，词项lemon在反向B–树中的路径就是root-n-o-m-e-l。所以对反向B-树遍历后可以返回所有包含同一后缀的词项。

于是，同时使用B-树和反向B-树，我们可以处理更一般的单通配符查询，比如se*mon。具体处理时，可以通过B-树来返回所有前缀为se且后缀非空的词项子集W，再通过反向B-树来返回所有后缀为mon且前缀非空的词项子集R。然后，对W和R求交集$W \cap R$，之后对交集进行扫描，当前缀和后缀相同时去掉它们对应的词项[①]（比如，查询ba*ba对应的集合R和W都包含词项ba，这种情况下应该过滤掉ba），得到结果集合。最后，通过普通倒排索引来获取结果集合中所有词项对应的文档。这样，我们就可以通过同时引入B-树和逆B-树的方式来处理只包含单个*号的查询。

3.2.1 一般的通配符查询

本节中考察两种一般通配符查询的处理方法，它们都采用了同一种策略，即将给定的通配符查询 q_w 表示成布尔查询 Q，随后在特定的倒排索引上进行处理。此时，Q 对应的词项集合覆盖了 q_w 对应的词项集合。然后，逐一判断 Q 对应的词项集合中的每个元素，看是否满足 q_w 的需求，并将那些不满足其要求的词项剔除。最后，我们得到 q_w 对应的词项，再去查普通倒排索引即可返回相关文档。

轮排索引

我们要介绍的第一种专门用于一般通配符查询的索引是轮排索引（permuterm index）[②]，它是倒排索引的一种特殊形式。首先，我们在字符集中引入一个新的符号$，用于标识词项结束。因此，词项hello在这里表示成扩展的词项hello$。然后，构建一个轮排索引，其中对扩展词项的每个旋转结果都构造一个指针来指向原始词项。图 3-3 给出了词项hello对应的轮排索引的例子。

① 也就是说，某个词项既是前缀也是后缀。——译者注

② permuterm 是 permuted term 的缩写。——译者注

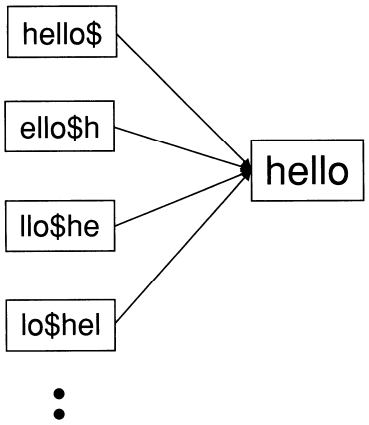


图 3-3　轮排索引的一部分

我们将词项旋转后得到的集合称为轮排词汇表（permuterm vocabulary）。那么，如何利用轮排索引来处理通配符查询呢？考虑通配符查询 m*n，这里的关键是将查询进行旋转让*号出现在字符串末尾，即得到 n$m*。下一步，在轮排索引中查找该字符串（可通过搜索树方式查找），实际上等价于查找某些词项（如 man 和 moron）的旋转结果。

既然我们可以使用轮排索引查找到匹配通配符的原始词典中的词项，那么我们就可以在普通倒排索引中查找这些词项，从而检索出匹配的文档。这样，我们就能够处理所有包含单个*号的通配符查询。但是，如果查询中存在多个通配符（如 fi*mo*er），那么我们应该如何处理？在这种情况下，首先返回轮排索引中 er$fi*对应的词项集合，然后通过穷举法检查该集合中的每个元素，过滤掉其中不包含 mo 的词项。上例中，最终会选出 fishmonger，而过滤掉 filibuster。最后，再利用剩下的词项去查普通倒排索引，从而得到最后的结果。轮排索引的一个最大缺点是其词典会变得非常大，因为它保存了每个词项的所有旋转结果。

我们注意到，B-树和轮排索引之间存在着密切的关联。实际上，有人建议上述轮排索引结构可以看成是一棵轮排 B-树。但是，为了突出轮排索引和 B-树的不同，这里使用了传统的术语。给定前缀时，利用轮排索引能够选择不同的旋转结果。

3.2.2　支持通配符查询的 *k*-gram 索引

尽管前面介绍的轮排索引结构尽管非常简单，但是由于要支持词项旋转，所以它会引起空间的急剧增长。对于一部英语词典来说，这种增长可能达到 10 倍左右。为此，下面介绍另外一种称为 *k*-gram 索引的通配符查询处理技术。3.3.4 节中我们还会用到 *k*-gram 索引。一个 *k*-gram 代表由 *k* 个字符组成的序列。对于词项 castle 来说，cas、ast、stl 都是 3-gram。我们用一个特殊的字符$来标识词项的开始或者结束，因此对于 castle 来说，所有的 3-gram 包括$ca、cas、ast、stl、tle 及 le$。

在*k*-gram索引结构中，其词典由词汇表中所有词项的所有*k*-gram形式构成，而每个倒排记

录表则由包含该*k*-gram的词项组成[①]。比如，3-gram etr可能会指向包括metric和retrieval在内的这些词项。图 3-4 给出了一个例子。

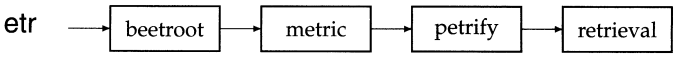


图 3-4 3-gram 索引的一个倒排记录表的例子。这里给出的 3-gram 是 etr，包含该 3-gram 的词项按照词典序进行了排序

我们通过一个例子来说明如何利用上述索引结构来处理通配符查询。考虑查询 re*ve，我们的目标是返回所有前缀是 re 且后缀是 ve 的词项。为此，我们构造布尔查询 $re AND ve$，这个查询可以在 3-gram 索引中进行查找处理，返回诸如 relive、remove 及 retrieve 的词项，然后我们可以在普通倒排索引中查找这些返回的词项，从而得到与查询匹配的文档。

然而，使用*k*-gram索引时往往还需要进行进一步的处理。考虑 3-gram索引结构的情况，对于查询red*，按照上面的处理步骤我们就会将原始查询转换为布尔查询$re AND red，这时可能会返回诸如retired的词项，因为它同时包含$re和red，但这个结果显然并不满足原始的查询red*，也就是说采用*k*-gram索引会导致非预期的结果[②]。

为了解决这个问题，我们引入一个称为后过滤（postfiltering）的步骤，即利用原始的查询red*对上述布尔查询产生的结果进行逐一过滤。过滤时只需要做简单的字符串匹配操作即可。和前面一样，我们会在普通倒排索引中查找上述过滤得到的结果词项，从而得到最后的文档集合。

从上面的介绍可知，一个通配符查询往往会返回多个词项，而每个词项再根据普通倒排索引返回其所在的文档。搜索引擎还可以通过布尔操作符支持多个通配符查询的组合，比如 re*d AND fe*ri。那么，该查询的语义是什么呢？由于每个通配符查询都会变成多个词项的或查询，所以上述查询最后变成多个或查询的与，即寻找包含同时匹配上 re*d 和 fe*ri 的词项的文档。

即使没有通配符查询的布尔组合，单个通配符查询的处理也是非常耗时的，除了最后要在普通倒排索引中查找之外，还要在特定索引（如轮排索引或 *k*-gram 索引）中进行查找、在结果中进行过滤等操作。搜索引擎可以支持这些丰富的功能，但是搜索引擎通常将这些功能隐藏在一个大部分用户从不访问的界面（如“高级搜索”界面）下。如果把这些功能暴露在一般搜索界面上，用户常常会受鼓励而使用这些功能，即便在他们不是特别需要的时候（比如，输入以a*开始的查询的前缀），这样就会大大增加搜索引擎的负担。

习题 **3-1** 在轮排索引中，轮排词汇表中的每个条目都指向其原始词汇表中能够生成它的源词项。那么，对于轮排词汇表的一个条目而言，其倒排记录表中会出现多少个源词项？

习题 **3-2** 写出由词项 mama 生成的轮排索引词汇表中的条目。

习题 **3-3** 在一个支持通配符查询的轮排索引中，如果想查找 s*ng，那么需要查找哪个或哪些键？

① *k*-gram 索引结构也是一种倒排索引结构，它也包括词典和倒排记录表两部分，这里的词典就是 *k*-gram 表，而倒排记录表保存的是所有包含该 *k*-gram 的词项。当然，在普通文档倒排索引中，词典中的词项指向包含该词项的所有文档所组成的倒排记录表。——译者注

② 也就说，这种做法只能保证结果的召回率，而并不能保证正确率。为提高正确率，需要进行后续过滤操作。——译者注

习题 **3-4**　回到图 3-4，在图标题说明文字中提出倒排记录表中的词项均按照词典次序排序，为什么这里的排序非常有用？

习题 **3-5**　再次考虑 3.2.1 节中的查询 fi*mo*er，如果采用 2-gram 索引的话，那么对应该查询应该会产生什么样的布尔查询？你能否举一个词项的例子，使该词匹配 3.2.1 节的轮排索引查询，但是并不满足刚才产生的布尔查询？

习题 **3-6**　考虑通配符查询 mon*h，如果只利用 2-gram 的与进行搜索的话，那么请给出一个错误的匹配例句。

3.3　拼写校正

本节主要关注查询的拼写校正问题。例如，用户输入 carot 时，实际上可能想返回包含词项 carrot 的文档。Google 的报告（www.google.com/jobs/ britney.html）指出，当用户输入 britian spears、britney's spears、brandy spears 或者 prittany spears 时，实际上搜索引擎都会当成是 britney spears 的错误拼写输入来处理。我们将会考察解决该问题的两个步骤：第一步基于编辑距离（edit distance），第二步基于 k-gram 重合度（*k*-gram overlap）。在介绍详细算法之前，我们首先简单介绍一下搜索引擎是如何实现拼写校正并使其成为用户体验的一部分的。

3.3.1　拼写校正的实现

对于大多数拼写校正[①]（spelling correction）算法而言，存在以下两个基本的原则。

(1) 对于一个拼写错误的查询，在其可能的正确拼写中，选择距离“最近”的一个。这就要求在查询之间有距离或者邻近度的概念。这些邻近度的度量方法将在 3.3.3 节介绍。

(2) 当两个正确拼写查询邻近度相等（或相近）时，选择更常见的那个。例如，grunt 和 grant 都是查询 grnt 的可能的正确拼写。算法将会从它们之中选择更常见的那个作为最后的拼写结果。最简单的情况下，“更常见”可以通过统计各词项在文档集中出现的次数来获得。因此，如果 grunt 在文档集中比 grant 出现得更多，则选择 grunt 作为校正结果。在很多搜索引擎中使用了另一种“更常见”的概念，其基本思路是，使用所有其他用户输入的查询中出现最频繁的拼写形式作为最后的选择。也就是说，如果用户输入 grunt 作为查询的次数相比 grant 更多的话，那么用户输入 grnt 更有可能是想要查询 grunt。

3.3.3 节一开始，将介绍两个查询之间的邻近度概念及其高效的计算方法。拼写校正算法基于这些邻近度计算，其功能主要通过以下几种方式来返回给用户。

(1) 输入查询 carot，系统往往在返回包含 carot 的文档的同时，也返回包含 carot 多种可能拼写校正结果（如 carrot 和 tarot）的文档。

(2) 当 carot 不在词典中时，采用第 1 种做法。

(3) 当原始查询返回的文档结果数目少于预定值（比如少于 5 篇文档）时，采用第 1 种做法。

① 还可以翻译成拼写纠正、拼写检查、拼写校对、拼写纠错等等。——译者注

(4) 当原始查询返回的文档数目少于预定值时，搜索界面中会给用户提供一个拼写建议（spelling suggestion），建议中会包含拼写校正之后的结果。因此，这实际上相当于搜索引擎和用户进行交互：“你是在找 carrot 吗”？

3.3.2 拼写校正的方法

这里我们主要关注两种拼写校正的方法：一种是词项独立（isolated-term）的校正，另一种是上下文敏感（context-sensitive）的校正。在词项独立的校正方法中，不管查询中包含多少个查询词项，其每次只考虑一个词项的校正，也就说在校正时词项之间是相互独立的。上面给出的 carot 的例子就属于这一类做法。利用这种词项独立校正的方法，很难检测到查询 flew form Heathrow 中实际上包含一个错误的词项 form（正确的形式应该是 from），这是因为在校正时每个词项之间是相互独立的。

首先，我们介绍两种词项独立的校正方法：编辑距离方法及 *k*-gram 重合度方法。然后，我们介绍基于上下文敏感的校正方法。

3.3.3 编辑距离

给定两个字符串s_1及s_2，两者的编辑距离（edit distance）定义为将s_1转换成s_2的最小编辑操作（edit operation）数。通常，这些编辑操作包括：(i) 将一个字符插入字符串；(ii) 从字符串中删除一个字符；(iii) 将字符串中的一个字符替换成另外一个字符。对于这些操作，编辑距离有时也称为Levenshtein[①]距离（Levenshtein distance）。例如，cat和dog的编辑距离是 3。实际上，编辑距离的概念可以进一步推广，比如允许不同的编辑操作具有不同的权重。例如，将字符*s*替换成字符*p*的权重会比将*s*换成*a*的权重大（这是因为在键盘上*a*离*s*更近，因此花费的代价更小）。采用这样的权重定义方法(即根据字符之间替换的可能性计算权重)在实践中非常有效(参见 3.4 节的语音相似度计算问题)。然而，下面我们只考虑所有的编辑操作具有等权重的情况。

众所周知，可以在 $O(|s_1| \times |s_2|)$ 的时间复杂度下计算两个字符串 s_1 和 s_2 的（带权）编辑距离，其中，$|s_i|$（i=1,2）表示字符串 s_i 的长度。其主要解决思路是采用动态规划算法（如图 3-5 所示），其中 s_1 和 s_2 以字符数组的方式进行存放。整数矩阵 m 的行数和列数分别是两个字符串的长度，算法在运行中不断填写矩阵元素。算法运行结束后，矩阵第 i 行第 j 列的元素保存的是 s_1 的前 i 个字符构成的字符串和 s_2 前 j 个字符构成的字符串的编辑距离。在图 3-5 中的第 8～10 行描述了动态规划的核心过程，其中求最小值的 3 个参数分别对应在 s_1 中替换一个字符、在 s_1 中插入一个字符和在 s_2 中插入一个字符的操作。

① 1965 年提出编辑距离概念的俄罗斯科学家的名字。——译者注

```
EditDistance(s1, s2)
 1  int m[|s1|, |s2|] = 0
 2  for i ← 1 to |s1|
 3  do m[i, 0] = i
 4  for j ← 1 to |s2|
 5  do m[0, j] = j
 6  for i ← 1 to |s1|
 7  do for j ← 1 to |s2|
 8     do m[i, j] = min{m[i − 1, j − 1] + if (s1[i] = s2[j]) then 0 else 1fi,
 9                      m[i − 1, j] + 1,
10                      m[i, j − 1] + 1}
11  return m[|s1|, |s2|]
```

图 3-5　两个字符串 s_1 和 s_2 的编辑距离的算法

图 3-6 给出了一个采用图 3-5 中算法计算编辑距离的例子。矩阵元素[*i*,*j*]表示成 4 个元素组成的 2×2 单元，其中右下角的元素是其他 3 个元素中的最小值，其他 3 个元素分别是 m[*i*-1,*j*-1]+0 或 1（这取决于 $s_1[i]$是否等于 $s_2[j]$）、m[*i*-1,*j*]+1 和 m[*i*,*j*-1]+1，这正好对应图 3-5 中算法中的动态规划的核心过程。包含斜体数字的单元给出了计算编辑距离的路径。

		f		a		s		t	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
c	**1**	*1*	*2*	**2**	3	**3**	4	**4**	5
	1	*2*	*1*	**2**	**2**	**3**	**3**	**4**	**4**
a	**2**	**2**	**2**	*1*	*3*	*3*	*4*	4	5
	2	3	**2**	*3*	*1*	*2*	*2*	**3**	**3**
t	**3**	**3**	**3**	3	**2**	*2*	*3*	*2*	*4*
	3	4	**3**	4	**2**	*3*	*2*	*3*	*2*
s	**4**	**4**	**4**	4	**3**	**2**	3	*3*	*3*
	4	5	**4**	5	**3**	4	**2**	*3*	*3*

图 3-6　编辑距离计算的例子。矩阵元素[*i*,*j*]包含一个 2×2 单元，根据单元中 3 个数字的最小值可以得到第 4 个数字。包含斜体数字的单元确定了编辑距离计算的路径

然而，拼写校正问题不止需要计算出编辑距离：给定字符串集合 V（对应词项词汇表）和查询字符串 q，我们要从 V 中寻找和 q 具有最小编辑距离的字符串。我们可以把这个问题看成是解码问题，其中编码词（对应 V 中的字符串）已经预先定义好。最直接的方法是，计算 q 到 V 中每个字符串的编辑距离，然后选择其中编辑距离最小的字符串。很显然，这种穷举的搜索方法开销很大。因此，实际当中使用了多种启发式方法来提高查找的效率。

最简单的启发式方法是将搜索限制在与查询词具有相同首字母的字符串上，也就是说我们期望查询的拼写错误不会出现在第一个字符上。上述启发式方法的一个更复杂的做法是利用轮排索引的某种版本，其中忽略了词尾的$符号。考虑查询字符串$q$的所有旋转结果集合，对集合中每个旋转$r$，通过遍历B-树来访问轮排索引（即轮排索引的词汇表查找可以通过B-树来实现），

返回旋转结果中以r开头的词项。例如，如果q为mase，那么我们考虑其旋转r = sema，这时我们会返回诸如semantic和semaphore的词，而这些词和mase之间的编辑距离并不小。遗憾的是，采用这种做法我们会错过包括mare 和 mane在内的一些可能更相关的词[①]。为解决这个问题，可以对旋转的使用机制进行如下修改：对每个旋转结果，在遍历B-树之前我们忽略其长度为 ℓ 的后缀，这样做可以保证返回词项集合R中的每个词项和q之间都包含一段较长的公共子串。ℓ 的值取决于q的长度。当然，我们也可以将它设置成常数（比如 2）。

3.3.4 拼写校正中的 *k*-gram 索引

为了进一步限制计算编辑距离后得到的词汇表大小，以下介绍如何通过使用 3.2.2 节中所介绍的 *k*-gram 索引来辅助返回与查询 q 具有较小编辑距离的词项。一旦返回这些词项之后，利用 *k*-gram 索引，就能从中找出与 q 具有最小编辑距离的词。

实际上，我们利用 *k*-gram 索引来查找与查询具有很多公共 *k*-gram 的词项。只要对“具有很多公共 *k*-gram”进行合理定义，我们认为上述查找实际上是对查询字符串 q 中 *k*-gram 的倒排记录表进行单遍扫描的过程。

图 3-7 给出的是查询 bord 的 2-gram 索引的一个片段，其中包含 3 个 2-gram 组成的倒排记录表。假定我们想返回包含上面 3 个 2-gram 中的至少 2 个词项，对倒排记录表的单遍扫描（和第 1 章类似）会返回满足该条件的所有词项，本例当中，这些词项包括 aboard、boardroom 及 border。

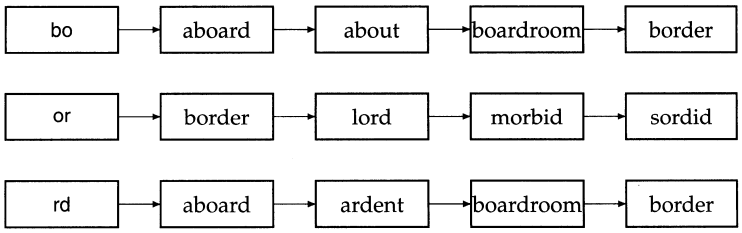


图 3-7 和查询 bord 存在至少两个 2-gram 的匹配过程

这种通过线性扫描并立即合并倒排记录表的做法十分简单，只需要待匹配词项包含查询 q 中某个固定数目的 *k*-gram 即可。但是这种做法有一个缺点，比如 boardroom 这种不可能是 bord 的正确拼写形式的词也会被返回。所以，我们需要计算词汇表中词项与查询 q 之间更精细的重合度计算方法。比如采用雅可比系数（Jaccard coefficient）就可以对先前的线性扫描合并方法进行修正。这里，雅可比系数的计算公式为$|A\cap B|/|A\cup B|$，其中 A、B 分别是查询 q、词汇表词项中的 *k*-gram 集合。扫描过程中，一旦扫描到当前的词项 t，就快速计算出 q 和 t 的雅可比系数，然后继续扫描下一个词项。如果雅可比系数超过预定的阈值，则将 t 输出。否则，我们往下继

① 这是因为 mase 的旋转结果为 mase、emas、sema 和 asem，任何一种旋转结果都不可能返回 mare 和 mane。而要做到这点可以通过忽略上述旋转结果的一段后缀来实现，如忽略 emas 的最后一个字母后再进行查找。——译者注

续扫描。上述过程表明，在计算雅可比系数时，我们需要知道 q 和 t 的 k-gram 集合。

由于是对q中的所有k-gram扫描倒排记录表，因此q中的k-gram集合是已知的，于是问题就变成如何求解t中的所有k-gram。理论上，我们可以穷举出t中所有的k-gram。但是这种做法不仅慢而且在实际中也难以实现。在很多情况下，倒排记录本身很可能并不包含完整的t字符串而往往是t的某种编码形式。也就是说，往往并不知道t的k-gram集合。然而，一个重要的发现使得这种情况下q和t雅可比系数的计算成为可能。这个发现就是，在计算雅可比系数的时候，我们只需要知道t的长度即可[①]。为了说明这一点，我们重新回到图 3-7 中的例子，对于查询q = board 扫描倒排记录表直到t = boardroom[②]。这时我们知道有 2 个 2-gram已经匹配上了。如果倒排记录表中已经预先记录了boardroom所包含的 2-gram的数目（这里是 8），那么所有计算雅可比系数的信息都已知，计算公式为：2/（8+3-2）。其中，分子为倒排记录的命中数（值为 2，分别来自bo和rd），分母为bord及boardroom中的 2-gram之和减去倒排记录表的命中数。

需要指出的是，也可以把雅可比系数替换成其他能够进行快速计算的衡量指标。那么，如何在拼写校正中使用这些度量方式呢？一个经验性的方法是，首先使用 k-gram 索引返回可能是 q 的潜在正确拼写形式的词项集合，然后计算该集合中的每个元素和 q 之间的编辑距离并选择具有较小距离的那些词项。

3.3.5　上下文敏感的拼写校正

独立的词项拼写校正方法在面对诸如 flew form Heathrow 中的输入错误时无能为力，因为这 3 个词单独看来拼写都没有错误。当输入这类查询时，搜索引擎可能会发现返回的文档非常少，随后也许会提供正确的查询建议 flew from Heathrow。这种功能的一种简单的实现方法就是，即使每个单词拼写都是对的，仍然要对每个单词找到可能的拼写正确词（采用 3.3.4 节中使用的方法），然后尝试对短语中的每个词进行替换。比如对于上面 flew form Heathrow 的例子，我们可能会返回如下短语 fled from Heathrow 和 flew fore Heathrow。对每个替换后的短语，搜索引擎进行查找并确定最后的返回数目。

如果单独的查询有多种可能的正确拼写形式，那么上述方法中穷举过程的开销会非常大，最后我们就会面对非常多的拼写组合。有一些启发式方法可以减小可能的拼写结果空间。上述例子中，当我们对 flew 及 from 进行扩展时，我们只保留文档集合或查询日志中的高频组合结果。比如，我们很可能会保留 flew from 而不是 fled fore 或 flea form，这是因为 fled fore 很可能比 flew from 出现的次数少。接下来，我们仅仅根据高频双词（如 flew from）来获得 Heathrow 的可能的正确拼写。计算双词频率的时候可以使用文档集，也可以使用用户的查询日志。

---

① 这个道理很简单，雅可比系数的分母等于两个集合的长度减去公共集合的长度。公共集合已知，则就可以进行计算。——译者注

② 从上下文看，此处的 boardroom 应该是图 3-7 中第三行而不是第一行的 boardroom。——译者注

习题 **3-7** 如果 $|s_i|$ 表示字符串 s_i 的长度，请证明 s_1 和 s_2 的编辑距离不可能超过 $\max\{|s_1|, |s_2|\}$。

习题 **3-8** 计算 paris 和 alice 之间的编辑距离，给出类似于图 3-5 中的算法结果，其中的 5×5 矩阵包含每个前缀子串之间的计算结果。

习题 **3-9** 写出在扫描 *k*-gram 索引倒排记录表过程中在线计算雅可比系数的伪代码（参见 3.3.4 节的讨论）。

习题 **3-10** 计算查询 bord 和图 3-7 中的每个包含 2-gram or 的词项之间的雅可比系数。

习题 **3-11** 考虑四词查询 catched in the rye，假定根据独立的词项拼写校正方法，每个词都有 5 个可选的正确拼写形式。那么，如果不对空间进行缩减的话，需要考虑多少可能的短语拼写形式（提示：同时要考虑原始查询本身，也就是每个词项有 6 种变化可能）？

习题 **3-12** 对于习题 3-11 中查询串的每个前缀——catched、catched in 及 catched in the，我们都会有一系列可能的前缀用于替换。假定我们只保留前十个高频替换前缀，频度采用其在文档集合中的出现次数计算，那么前十个之外的前缀以及对它们的进一步扩展形式都会被剔除，也就是说，如果 batched in 没有出现在文档集最常见的 10 个二词查询中，那么其后续的扩展形式都会被剔除。我们的问题是，每一步当中有多少可能的替换前缀被剔除？

习题 **3-13** 如果只保留 catched in 的最常见的 10 个替换前缀，能否保证 catched in the 的最常见的 10 个替换前缀结果中的一个会必然出现？

3.4 基于发音的校正技术

最后一项用户容错检索技术与发音校正有关，即拼写错误的原因在于用户输入了一个和目标词项发音相似的查询。该方法尤其适用于人名的查找，其基本的思路是：对每个词项，进行一个语音哈希操作，发音相似的词项都被映射为同一值。该思路最早源于国际警察部门的工作，它们自 20 世纪初开始就在全球范围内寻找与犯罪嫌疑人发音相似的人名，而不管这些人名在不同国家的发音是否有所不同。上述思路主要用于专有名词的语音校正。

这一类通过语音哈希的方法通常统称为 soundex 算法（soundex algorithm）。其中，一个原始的 soundex 算法的构建方法如下。需要指出的是，该基本算法存在很多变形。

(1) 将所有的词项转变为四字符的简化形式。基于这些简化形式建立原始词项的倒排索引，该索引被称为 soundex 索引。

(2) 对查询词项进行同样的操作。

(3) 当对查询进行 soundex 匹配时，就在 soundex 索引中搜索。

上述算法不同变形之间的主要区别在于将词项转变为四字符简化形式的方法不同。一个普遍使用的转换方法会得到四字符的编码结果，其中第 1 个字符是字母而其他 3 个字符是 0～9 之间的数字。转换方法如下。

(1) 保留词项的首字母。

(2) 将后续所有的 A、E、I、O、U、H、W 及 Y 等字母转换为 0。

(3) 其他字母的转换规则如下：

将 B、F、P 和 V 转换为 1；

将 C、G、J、K、Q、S、X 和 Z 转换为 2；

将 D 和 T 转换为 3；

将 L 转换为 4；

将 M 和 N 转换为 5；

将 R 转换为 6。

(4) 将连续出现的两个同一字符转换为一个字符直至再没有这种现象出现。

(5) 在结果字符串中剔除 0，并在结果字符串尾部补足 0，然后返回前四个字符，该字符由 1 个字母加上 3 个数字组成。

利用上述转换方法，就可以将 Herman 转换为 H655。给定一个查询（比如 herman），可以计算其 soundex 编码然后从 soundex 索引中返回具有相同编码的词项，最后再在普通倒排索引中得到最后结果。

上述算法依赖于如下观察结果：(1) 在名称转录时，元音是可以互换的；(2) 发音相似的辅音字母归成同一类。这就会导致相关的名称通常有相同的 soundex 编码结果。尽管上述规则在很多场合上都适用，特别是欧洲语言，但是上述规则仍然依赖于文字系统。比如，汉语中人名可以用韦氏拼音（Wade-Giles）或汉语拼音法表达。尽管 soundex 能够在两种拼法不同的某些情况下发生作用，比如将韦氏拼音中的 hs 和汉语拼音中的 x 都转换为 2，但是在很多其他情况下都会失败，比如：韦氏拼音中的 j 和汉语拼音中的 r 会映射出不同的结果。

? 习题 **3-14**　找出两个拼写不一致但 soundex 编码一致的专有名词。

习题 **3-15**　找出两个发音相似但 soundex 编码不一致的专有名词。

3.5　参考文献及补充读物

Knuth（1997）是一本详尽介绍搜索树相关信息的读物，包括 B-树系列和它们在词典查找中的使用方法。

Garfield（1976）给出了一个最早的关于轮排索引的完整介绍。Ferragina 和 Venturini（2007）给出了索引中空间“爆炸”问题的一种解决方案。

Damerau（1964）是最早的有关拼写校正形式化处理的文献之一。编辑距离的概念归功于 Levenshtein（1965），而图 3-5 中的算法则归功于 Wagner 和 Fischer（1974）。Peterson（1980）及 Kukich（1992）发展了基于编辑距离计算方法的多种变形，并且对 Zobel 和 Dart（1995）的方法进行了详细的经验性研究，研究结果表明，*k*-gram 索引对于寻找候选匹配项非常有效，但是需要和一个更精细的方法（比如编辑距离方法）相结合来判断最可能的错误拼写。Gusfield

（1997）是一本有关字符串算法（如编辑距离计算）的经典读物。

将概率模型（“噪声信道”模型）用于拼写校正的开创性工作参见 Kernighan 等人（1990），后来在 Brill 和 Moore（2000）及 Toutanova 和 Moore （2002）等工作中得到了继续发展。这些模型中，拼写错误的查询可以看成正确查询的满足某种概率的变形结果。它们与第 12 章的语言模型具有相似的数学基础，而且同样提供了一种方法来加入语音相似度、键盘的相近度及实际中的拼写输入错误等因素。很多人都认为这些方法是目前最好的方法。Cucerzan 和 Brill（2004）基于搜索引擎日志中查询重构的记录，构建了一个拼写校正模型。

Soundex 算法归功于 Margaret K. Odell、Robert C. Russelli （来自美国 1918 年和 1922 年的专利），这里介绍的版本主要参考了 Bourne 和 Ford （1961）。Zobel 和 Dart （1996） 对多种语音匹配算法进行了评估，结果发现 soundex 的某个变种算法对于一般的拼写错误处理性能很差，但是一些其他的基于词发音的语音相似度计算方法却取得了很好的效果。

第4章 索引构建

本章我们主要关注如何建立倒排索引，我们将这个过程称为索引构建（index construction或 indexing），而将构建索引的程序或计算机称为索引器（indexer）。索引构建算法的设计受硬件的配置所制约，因此本章首先将介绍与索引构建相关的计算机硬件的基本知识。然后，4.2 节将介绍一种面向静态文档集的高效单机索引算法——基于块的排序索引构建算法，它可以看作是第 1 章基于排序的基本索引算法的一个更具扩展性的版本。4.3 节将介绍内存式单遍扫描索引构建算法，和 4.2 节的算法相比，由于它并不将词汇表都加载到内存中，因此更具扩展性。对于像 Web 一样的大规模的文档集合，就要考虑如何在成百上千台计算机构成的计算机集群上进行分布式索引构建。因此，4.4 节我们将介绍分布式索引构建。另外，很多文档集会动态变化，这种情况下我们要考虑动态索引的构建，以便将文档集的变化即时反映到索引中。4.5 节将介绍动态索引构建。最后，在 4.6 节，我们讨论在索引构建中可能会遇到的一些复杂情况，如安全性和排序式检索中的索引问题。

索引构建和其他章节中的许多话题都息息相关。索引器需要原始文本，但是文档可能会采用各种编码格式（见第 2 章）。索引器对中间文件和最后的索引文件进行压缩或者解压缩（见第 5 章）。在Web搜索中，文档往往并不来自于本地，而必须要通过网络采集才能得到（参见第 20 章）。在企业搜索中，很多文档内嵌在各种内容管理系统、邮件系统或者数据库中。4.7 节我们将给出这样的一些例子。尽管大多数这类应用都能够通过http方式访问，但是采用原生API（Native API）[①]往往会更高效。需要提醒的是，即使将原始文本输送给索引过程这个看上去很简单的子系统的构建，其本身也很有可能是个具有挑战性的问题。

4.1 硬件基础

构建信息检索系统时，很多决策都依赖于系统所运行的硬件环境。因此，本章将首先简单介绍计算机硬件的性能特点，2007 年的典型计算机硬件性能参数如表 4-1 所示。

表4-1 一个2007年度的典型计算机系统的参数

符 号	含 义	值

① Native API 系指以二进制方式直接开放的应用程序开发接口 (Application Programming Interface)，可以直接由 C/C++ 来调用访问使用。——译者注

s	平均寻道时间	5ms = 5×10^{-3}s
b	每个字节的传输时间	0.02ms = 2×10^{-8}s
	处理器时钟频率	10^9HZ
p	底层操作时间(如：单词的比较或者交换)	0.01ms= 10^{-8}s

（续）

符　号	含　义	值
	内存大小	几个GB
	磁盘大小	1 TB或更大

注：寻道时间指的是将磁头移到新位置所花的时间。每个字节所需的传输时间指的是磁头就位以后从磁盘到内存的传输速率。

与 IR 系统的设计相关的硬件基本性能参数如下。

- 访问内存数据比访问磁盘数据快得多，只需要几个时钟周期（大概 5×10^{-9} s）便可以访问内存中的一个字节，与此形成鲜明对照的是，从磁盘传输一个字节所需要的时间则长得多（大概 2×10^{-8} s）。因此，我们会尽可能地把数据放在内存中，特别是那些访问频繁的数据。这种将频繁访问的磁盘数据放入内存的技术称为缓存技术（caching）。
- 进行磁盘读写时，磁头移到数据所在的磁道需要一段时间，该时间称为寻道时间，对典型的磁盘来说平均在 5 ms 左右。寻道期间并不进行数据的传输。于是，为使数据传输率最大，连续读取的数据块也应该在磁盘上连续存放。举例来说，基于表 4-1 中的数据的话，大概只需要 0.2 秒钟就可以将一个连续存放的 10MB 数据块从磁盘传输到内存，但是如果上述数据存放在 100 个非连续的块中，那么，需要移动 100 次磁头，因此总时间可能会需要 $0.2 + 100 \times (5 \times 10^{-3}) = 0.7$ s。
- 操作系统往往以数据块为单位进行读写。因此，从磁盘读取一个字节和读取一个数据块所耗费的时间可能一样多。数据块的大小通常为 8 KB、16 KB、32 KB 或 64 KB。我们将内存中保存读写块的那块区域称为缓冲区（buffer）。
- 数据从磁盘传输到内存是由系统总线而不是处理器来实现的，这意味着在磁盘 I/0 时处理器仍然可以处理数据。我们可以利用这一点来加速数据的传输过程，比如将数据进行压缩然后再存储在磁盘上。假定采用一种高效的解压缩算法的话，那么读磁盘压缩数据再解压所花的时间往往会比直接读取未压缩数据的时间要少。
- IR 系统的服务器往往有数 GB 甚至数十 GB 的内存，其可用的磁盘空间大小一般比内存大小要高几个数量级。

4.2 基于块的排序索引方法

建立不包含位置信息的索引的基本步骤如图 1-4 所示。首先，我们扫描一遍文档集合得到

所有的词项–文档 ID 对。然后，我们以词项为主键、文档 ID 为次键进行排序。最后，将每个词项的文档 ID 组织成倒排记录表，并计算诸如词项频率或者文档频率的统计量。对于小规模文档集来说，上述过程均可在内存中完成。本章我们主要讨论在大规模文档集条件下需要引入二级存储介质时的索引方法。

为使索引构建过程效率更高，我们将词项用其 ID 来代替（不是像在图 1-4 每个词项采用字符串来表示），每个词项的 ID 是唯一的序列编号。我们可以在处理文档集之余将词项映射成其 ID。或者在一种两遍扫描的方法中，第一遍扫描得到词汇表，第二遍扫描才构建倒排索引。本章介绍的索引构建方法均采用单遍扫描方式。4.7 节给出了多遍扫描的参考文献，它们在某些应用中往往更可取，比如，当磁盘空间非常少的情况下应优先考虑采用多遍扫描方法。

本章中我们采用 Reuters-RCV1 语料作为样例文档集，该语料库包含大概 1GB 的文本数据，由自 1996 年 8 月 20 日到 1997 年 8 月 19 日一年间的路透社新闻组成，大概有 800 000 篇文档。图 4-1 给出了一篇典型的文档。需要指出的是，在本书中我们忽略其中的图片等多媒体信息，而只关注其中的文本信息。Reuters-RCV1 覆盖了多个国际性的话题（包括政治、贸易、体育及科学等等）。该语料库的相关统计数字参见表 4-2。

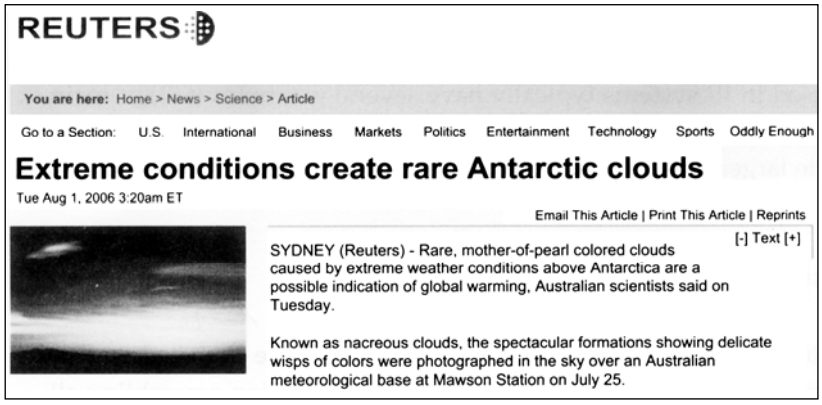


图 4-1　路透社的一篇新闻文档

表4-2　Reuters-RCV1语料的统计数据，其中所有值都进行了舍入处理

符号	含义	值
N	文档总数	800 000
L_{ave}	每篇文档的平均词条数目	200
M	词项总数	400 000
	每个词条的平均字节数（含空格和标点符号）	6
	每个词条的平均字节数（不含空格和标点符号）	4.5
	每个词项的平均字节数	7.5
T	词条总数	100 000 000

注：未舍入之前的数据分别为：总共有 806 791 篇文档，其中平均每篇文档有 222 个词条，所有的不同词项个数为

391 523 个，算上空格和标点平均每个词条为 6.04 B，不考虑空格和标点则平均每个词条为 4.5 B，每个词项平均为 7.5 B，语料库中总共有 96 969 056 个词条。这些数值与表 5-1 的第 3 行（即“大小写转换”那行）对应。

Reuters-RCV1 语料约有 1 亿个词条，每个占 4 B，因此存储所有的词项 ID–文档 ID 对需要 0.8GB 存储空间。目前典型的语料规模往往比 Reuters-RCV1 高一到两个数量级，读者可以很容易就能想到，即使对大型计算机来说，将所有词项 ID–文档 ID 对放在内存进行排序都是一件十分困难的事情。如果在索引构建过程中生成的临时文件只占可用内存的一小部分，那么使用第 5 章将要讨论的压缩技术可能可以使得上述问题得到缓解。然而，对于很多大型的语料库来说，即使经过压缩后的倒排记录表也不可能全部加载到内存中。

由于内存不足，我们必须使用基于磁盘的外部排序算法（external sorting algorithm）。为了达到可以接受的速度，对该算法的核心要求是：在排序时尽量减少磁盘随机寻道的次数，如 4.1 节所述，磁盘顺序读取速度会比随机寻道速度快得多。BSBI（blocked sort-based indexing algorithm，基于块的排序索引算法）是一种解决的办法：第 1 步，将文档集分割成几个大小相等的部分；第 2 步，将每个部分的词项 ID–文档 ID 对排序；第 3 步，将中间产生的临时排序结果存放到磁盘中；第 4 步，将所有的中间文件合并成最终的索引。

该算法将文档解析成词项 ID–文档 ID 对，并在内存中一直处理，直到累积至放满一个固定大小的块空间（参见图 4-2 的 ParseNextBlock 函数）为止。我们选择合适的块大小，使之其能方便加载到内存并允许在内存中快速排序。排序后的块转换成倒排索引格式后写入磁盘，建立倒排索引的过程包括两步：第一步是对词项 ID–文档 ID 对进行排序；第二步是将具有同一词项 ID 的所有文档 ID 放到倒排记录表中，其中每条倒排记录都只有一个文档 ID。然后将基于块的倒排索引写到磁盘上。将该算法应用于 Reuters-RCV1 语料库上，并假定内存每次能加载 1 000 万个词项 ID–文档 ID 对，那么算法最后会产生 10 个块，每个块都是文档集的倒排索引的一部分。

```
BSBIndexConstruction()
1  n ← 0
2  while (all documents have not been processed)
3  do n ← n + 1
4     block ← ParseNextBlock()
5     BSBI-Invert(block)
6     WriteBlockToDisk(block, f_n)
7  MergeBlocks(f_1, ..., f_n; f_merged)
```

图 4-2 基于块的排序索引算法。该算法将每个块的倒排索引存入文件 $f_1, \ldots, f_n$ 中，最后合并成文件 f_{merged}

上述算法实现的最后一步是：将 10 个块索引同时合并成一个索引文件。图 4-3 给出了将两个块进行合并的例子，其中 d_i 表示文档集中第 i 篇文档。合并时，同时打开所有块对应的文件，内存中维护了为 10 个块准备的读缓冲区和一个为最终合并索引准备的写缓冲区。每次迭代中，利用优先级队列（即堆结构）或者类似的数据结构选择最小的未处理词项 ID 进行处理。读入该

词项的倒排记录表并合并，合并结果写回磁盘中。需要时，再次从文件中读入数据到每个读缓冲区。

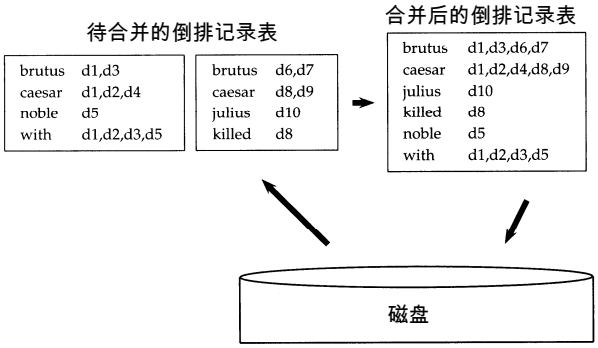


图 4-3　基于块的排序索引方法中的合并过程。两个块（图中“待合并的倒排记录表”所示）从磁盘读入内存，然后在内存中进行合并（图中“合并后的倒排记录表”所示），最后写回磁盘。为了方便理解，这里用了词项本身而不是其 ID

下面讨论 BSBI 算法的复杂度。由于该算法最主要的时间消耗在排序上，因此其时间复杂度为 $\Theta(T \log T)$，其中 T 是所需要排序的项数目的上界（即词项 ID–文档 ID 对的个数）。然而实际的索引构建时间往往取决于文档分析（PARSENEXTBLOCK）和最后合并（MERGEBLOCKS）的时间。习题 4-6 中将会计算 Reuters-RCV1 语料库索引构建的总时间，包括上面提到的几个步骤以及每个块的索引建立和磁盘写入等过程。

需要指出的是，在个人计算机标配都已经达到 1GB 或者多 GB 内存的年代，Reuters-RCV1 语料库并不是一个很大的语料库。采用合适的压缩技术（参见第 5 章），就能利用一台非顶级配置的服务器在内存中为它建立倒排索引。当然，本章只是以 Reuters-RCV1 语料库为例来介绍有关的索引构建技术，由于实际中的很多语料库远比这个语料库要大，所以这些技术都是必要的。

? 习题 **4-1**　如果需要 $T \log_2 T$ 次比较（T 是词项 ID–文档 ID 对的数目），每次比较都有两次磁盘寻道过程。假定使用磁盘而不是内存进行存储，并且不采用优化的排序算法（也就是说不使用前面提到的外部排序算法），那么对于 Reuters-RCV1 构建索引需要多长时间？计算时假定采用表 4-1 中的系统参数。

习题 **4-2**　[*] 在基于块的排序索引算法中，如何快速构建词典来避免对数据的额外扫描？

4.3　内存式单遍扫描索引构建方法

基于块的排序索引算法具有很好的可扩展性，但是需要一种将词项映射成其 ID 的数据结构。对于大规模的文档集来说，该数据结构会很大以致在内存中难以存放。一种更具扩展性的算法称为 SPIMI（single-pass in-memory indexing，内存式单遍扫描索引算法）。SPIMI 使用词项

而不是其 ID，它将每个块的词典写入磁盘，对于下一个块则重新采用新的词典。只要硬盘空间足够大，SPIMI 就能够索引任何大小的文档集。

SPIMI 算法的流程如图 4-4 所示，其中省略了文档分析以及将文档转换成词项–文档 ID 流（算法中称为 token_stream）的过程。反复调用 SPIMI-INVERT 函数直到将全部的文档集处理完为止。

```
SPIMI-INVERT(token_stream)
 1  output_file = NEWFILE()
 2  dictionary = NEWHASH()
 3  while (free memory available)
 4  do token ← next(token_stream)
 5     if term(token) ∉ dictionary
 6        then postings_list = ADDTODICTIONARY(dictionary, term(token))
 7        else postings_list = GETPOSTINGSLIST(dictionary, term(token))
 8     if full(postings_list)
 9        then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
10     ADDTOPOSTINGSLIST(postings_list, docID(token))
11  sorted_terms ← SORTTERMS(dictionary)
12  WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13  return output_file
```

图 4-4 SPIMI 算法中块倒排索引的生成

算法逐一处理（程序第 4 行）每个词项–文档 ID 对。如果词项是第一次出现，那么将之加入词典（最好通过哈希表来实现），同时建立一个新的倒排记录表（程序第 6 行）；如果该词项不是第一次出现则直接返回其倒排记录表（程序第 7 行）。

BSBI 和 SPIMI 的一个区别在于，后者直接在倒排记录表中增加一项（程序第 10 行）。和那种一开始就整理出所有的词项 ID–文档 ID 并对它们进行排序的做法（这正好是 BSBI 中的做法）不同，这里每个倒排记录表是动态增长的（也就是说，倒排记录表的大小会不断调整），同时立即就可以实现全体倒排记录表的收集。这样做有两个好处：第一，由于不需要排序操作，因此处理的速度更快；第二，由于保留了倒排记录表对词项的归属关系，因此能够节省内存，词项的 ID 也不需要保存。这样，每次单独的 SPIMI-INVERT 调用能够处理的块大小可以非常大，整个倒排索引的构建过程也因此会非常高效。由于事先并不知道每个词项的倒排记录表大小，算法一开始会分配一个较小的倒排记录表空间，每次当该空间放满的时候，就会申请加倍的空间（程序的第 8～9 行）。这意味着一些空间被浪费，也正好抵消了不用保存词项 ID 所省下的空间。然而，总体来说，在 SPIMI 中对块建立索引所需的内存空间仍然比 BSBI 少。当内存耗尽时，包括词典和倒排记录表的块索引将被写到磁盘上（程序第 12 行）。在这之前，为使倒排记录表按照词典顺序排序来加快最后的合并过程，要对词项进行排序操作（程序第 11 行）。如果每个块的倒排记录表没有事先排好序，那么合并过程很难通过一个简单的逐块扫描算法来实现。每次对 SPIMI-Inverter 的调用都会写一个块到磁盘，这和 BSBI 一样。SPIMI 最后一步就是将多个块合并成最后的倒排索引（对应图 4-2 的第 7 行，图 4-4 中没有）。

除对每个块建立新词典及去除高代价的排序操作之外，SPIMI 有一个重要的第三方组件：压缩。如果使用压缩技术，那么不论是倒排记录表还是词项都可以在磁盘上进行压缩存储。由于压

缩一方面能使算法处理更大的块，另一方面能够使得原来的每个块所需要的磁盘空间更少，所以压缩技术能够进一步提高算法的效率。对这方面感兴趣的读者可以参考相关文献（参见 4.7 节）。

SPIMI 算法的时间复杂度是 $\Theta(T)$，这是因为它不需要对词项–文档 ID 对进行排序操作，所有操作最多和文档集大小成线性关系。

4.4　分布式索引构建方法

实际当中，文档集通常都很大，在单台计算机上很难高效地构建索引。特别是对于万维网来说，情况更是如此。对Web构建规模合理的索引常常需要大规模的计算机集群[①]。因此，Web搜索引擎通常使用分布式索引构建（distributed indexing）算法来构建索引，其索引结果也是分布式的，它往往按照词项或文档进行分割后分布在多台计算机上。本节主要介绍基于词项分割的分布式索引的构建方法。对于大部分大型搜索引擎来说，它们更倾向于采用基于文档分割的索引（当然这种索引也很容易从基于词项分割的索引生成）。有关这部分内容的详细介绍参见 20.3 节。

本节介绍的分布式索引构建方法是 MapReduce 的一个应用。MapReduce 是一个通用的分布式计算架构，它面向大规模计算机集群而设计。集群的关键是，利用价格低廉的日用计算机（称为节点，node）来解决大型的计算问题，这些计算机都采用通用的标准部件（处理器、内存和磁盘），而不是象超级计算机那样采用专用硬件。尽管在这样的一个计算机集群当中包含成百上千台计算机，但每台计算机都有可能在任意时刻失效。因此，要保障分布式索引构建过程的鲁棒性，就必须把整个任务分成易分配的子任务块，并在节点失效时能够重新分配。集群中的主控节点（master node）负责处理任务在工作节点上的分配和重分配。

MapReduce 中的 Map 和 Reduce 过程将计算任务划分成子任务块，以便每个工作节点在短时间内快速处理。图 4-5 给出了 MapReduce 的具体步骤，而图 4-6 给出了一个包含两篇文档的文档集的例子。首先，输入数据（这里是网页集合）被分割成 n 个数据片（split），数据片大小的选择一定要保证任务的均匀、高效分布，同时每个数据片不能太大，数据片的数目也不能太多。在分布式索引中，数据片的大小采用 16MB 或者 64MB，这些选择的效果通常不错。各个数据片并不预先分配给各台计算机，而是在运行过程中由主控节点分配：一旦一台计算机完成了某个数据片的处理任务，主控节点就会分配下一个数据片给它进行处理。如果某台机器宕机或者由于硬件问题导致机器变慢，它上面运行的任务数据片就可以重新分配给其他计算机。

① 这里的 cluster 指的是一组紧耦合共同工作的计算机。这和第 16～18 章中的 cluster 不是一回事，在那里 cluster 是一组语义相似的文档。

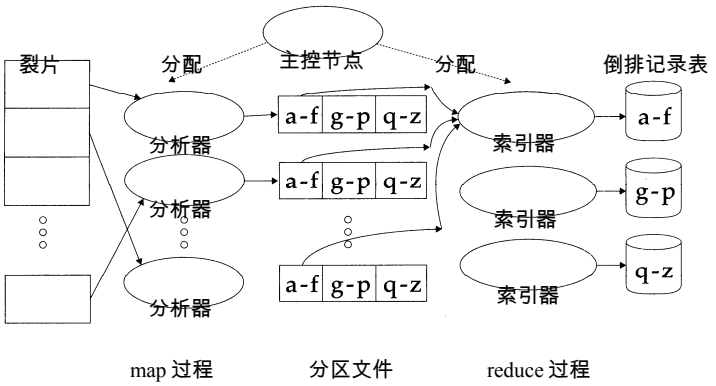


图 4-5 一个使用 MapReduce 进行分布式索引构建的例子。来自 Dean and Ghemawat（2004）

一般来说，MapReduce 会通过键–值对（key-value pair）的转换处理，将一个大型的计算问题转化成较小的子问题。在索引构建中，键–值对的形式就是（词项 ID, 文档 ID）。在分布式索引构建过程中，从词项到其 ID 的映射同样要分布式进行，因此分布式的索引构建方法要比单机上的索引构建方法复杂得多。一种简单的解决方法就是维护一张高频词到其 ID 的映射表（这张表可以预先算好）并将它复制到所有节点计算机上，而对低频词则直接使用词项本身而不是其 ID。下面的讨论中我们不再考虑这个问题，只假设所有节点都共享了一份一致的词项到其 ID 的映射表。

MapReduce 的 Map 过程将输入的数据片映射成键–值对。这个映射过程对应于 BSBI 和 SPIMI 算法中的分析任务，因此也将执行 Map 过程的机器称为分析器（parser）。每个分析器将输出结果存在本地的中间文件（也称为分区文件，segment file）中（图 4-5 中以 a~f | g~p | q~z 来表示）。

在Reduce阶段，我们想将同一键（词项ID）的所有值（文档ID）集中存储，以便快速读取和处理。实现时，将所有的键按照词项区间划分成*j*个段，并将属于每个段的键–值对写入各自分区文件即可。图 4-5 中，所有的词项按照首字母来分成 3 段：a ~ f、g ~ p及q ~ z[①]。词项的分割方法由运行索引系统的用户来定义（参见习题 4-10）。每个分析器各自写相应的分区文件，每个分区文件对应一个词项区间，因此，在整个系统中，每个词项区间会对应*r*个分区文件，其中，*r*是分析器的个数。举例来说，图 4-5 中对于a ~ f分区，有 3 个a ~ f分区文件，它们分别对应 3 个分析器。

给定一个键（词项 ID），将所有的值（文档 ID）汇总并组织成倒排表的过程由 Reduce 阶段的倒排器（inverter）来完成。主控节点将每个词项分区分配给一个不同的倒排器，并在倒排器失效或者变慢的时候将在其上处理的词项分区进行重新分配。每个词项分区（对应 *r* 个分区文件，其中每个文件存放在一个分析器上）由一个倒排器来完成。这里我们假定多个分区文件

① 为了方便说明，这里用了非常简单的键分区方法。通常来说，键分区并不一定要使得词项或者词项 ID 连续。——译者注

的大小适合在单机上处理（参见习题 4-9）。最后，每个键对应的所有值要进行排序并写到最终的排序倒排记录表（图中以“倒排记录表”来表示）中。需要指出的是，图 4-6 中每个倒排记录当中还包括词项频率，而本章其他节中的倒排记录仅仅只是文档 ID。针对 a ~ f 分区处理的数据流如图 4-5 所示。到这里为止，整个倒排索引的构建才宣告完成。

Map和Reduce函数的构架	
Map: 输入	→ list(k, v)
Reduce: (k,list(v))	→ 输出
索引构建中上述构架的实例化	
Map: Web文档集	→ list(词项ID, 文档ID)
Reduce: (<文档ID_1, list(docID)>, <文档ID_2, list(docID)>, …)	→ (倒排记录表1, 倒排记录表2，…)
索引构建的一个例子	
Map: d_2: C died. d_1: C came, C c'ed.	→ (<C,d_2><died,d_2>,<C,d_1><came,d_1><C,d_1>,<c'ed,d_1>)
Reduce: (<C,(d_2,d_1,d_1)>,<died, (d_2)>,<came, (d_1)>,<c'ed,(d_1)>)	→ (<C,(d_1:2,d_2:1)>,<died, (d_2:1)>,<came,(d_1:1)>, <c'ed,(d_1:1)>)

图 4-6　MapReduce 中的 Map 和 Reduce 函数。通常 Map 函数会产生一个键–值表。对于同一个键，在 Reduce 阶段会汇总到一张表中。在后续阶段，该表会被进一步处理。图中给出了两个函数的实例化及在索引构建中使用的一个例子。由于 Map 阶段往往在一个分布式环境中进行处理，所以在该例子中，词项 ID–文档 ID 对并不需要一开始就正确排序。为了方便理解，例子中直接给出了词项本身而不是其 ID。另外，Casesar 简写成 C，conquered 简写成 c'ed

分析器和倒排器并不一定是不同的机器。主控节点发现空闲的机器后会给它分配新的任务。同一台机器在 Map 阶段中可以作为分析器，而在 Reduce 阶段也可以作为倒排器。另外，索引构建的同时，机器上往往也在同时运行其他任务，所以在做分析器和倒排器之外，一台机器也可能运行采集程序或者其他不相关的任务。

为了尽量减少在倒排器对数据进行 Reduce 之前的写时间，每个分析器都将其分区文件写到本地磁盘。在 Reduce 阶段，主控节点会通知倒排器与之相关的分区文件的位置（例如，词项 a ~ f 分区对应的 r 个分区文件）。在每个分析器上，由于与某个特定倒排器相关的数据已经被分析器写入一个单独的分区文件中，所以每个分区文件仅需要一次顺序读取过程。这种设置方法可以使索引时所需的网络通信开销最小。

图 4-6 给出了MapReduce的通用构架。由于输入和输出通常都是键-值对列表本身，所以多个MapReduce任务能够串行执行[①]。实际上，这正是 2004 年Google的索引系统的设计方案。本节所介绍的索引过程仅仅包含该Google索引系统的 1/5 ~ 1/10 的MapReduce操作。另一个MapReduce操作则将刚才创建的按照词项分割的索引转换成按照文档分割的索引。

MapReduce 为分布式环境下的索引构建提供了一个鲁棒的、概念简洁的实现框架。通过提供半自动的方法将索引构建分割成多个子任务，那么就可以在给定足够规模的计算机集群的情况下，将索引构建扩展到任何规模的文档集上。

① 这里的意思是说一个 MapReduce 过程的输出结果可以作为另一个 MapReduce 过程的输入。——译者注

? 习题 **4-3** 对于 $n = 15$ 个数据片，$r = 10$ 个分区文件，$j = 3$ 个词项分区，假定使用的集群的机器的参数如表 4-1 所示，那么在 MapReduce 构架下对 Reuters-RCV1 语料进行分布式索引需要多长时间？

4.5 动态索引构建方法

迄今为止，我们都假设文档集是静态的，这对于很少甚至永远不会改变的文档集（如《圣经》或莎士比亚的著作）来说没有任何问题。然而，大部分文档集会随文档的增加、删除或更新而不断改变。这也意味着需要将新的词项加入词典，并对已有词项的倒排记录表进行更新。

最简单的索引更新办法就是周期性地对文档集从头开始进行索引重构。如果随时间的推移文档更新的次数不是很多，并且能够接受对新文档检索的一定延迟，再加上如果有足够的资源能够支持在建立新索引的同时让旧索引继续工作，那么周期性索引重构不失为一种较好的选择。

如果要求能够及时检索到新文档，那么一种解决方法是同时保持两个索引：一个是大的主索引，另一个是小的用于存储新文档信息的辅助索引（auxiliary index），后者保存在内存中。检索时可以同时遍历两个索引并将结果合并。文档的删除记录在一个无效位向量（invalidation bit vector）中，在返回结果之前可以利用它过滤掉已删除文档。某篇文档的更新通过先删除后重新插入来实现。

每当辅助索引变得很大时，就将它合并到主索引中。合并操作的开销取决于索引在文件系统中的存储方式。如果将每个词项的倒排记录表都单独存成一个文件，那么要合并主索引和辅助索引，只需要将辅助索引的倒排记录表扩展到主索引对应的倒排记录表即可完成。上述机制中，保留辅助索引的原因在于可以减少随时间推移所需要的磁盘寻道次数。单独更新每篇文档最多需要 M_{ave} 次磁盘寻道，其中 M_{ave} 是文档集中平均每篇文档的词汇表大小。而采用辅助索引的话，在合并辅助索引和主索引时，只需要将额外的负载放到磁盘上。

遗憾的是，由于绝大多数文件系统不能对大数目的文件进行高效处理，因此将每个倒排记录表存成一个文件这种方式实际是不可行的。另一种简单的方法是将索引存到一个大文件中，也就说将所有倒排记录表存到一起。现实当中，我们往往在上述两种极端机制之间取一个折衷方案（参见 4.7 节）。为讨论方便起见，下面我们只选择将索引存成一个大文件的方式。

在这种机制下，对每个倒排记录我们都会处理 $\lfloor T/n \rfloor$ 次，这是因为 $\lfloor T/n \rfloor$ 次合并中的每一次都会处理倒排记录，其中 n 是辅助索引的大小，T 是所有倒排记录的数目。因此，总的时间复杂度[①]是 $\Theta(T^2/n)$。

通过引入 $\log_2(T/n)$ 个索引 I_0、I_1、I_2…，其中每个索引的大小分别是 $2^0 \times n$、$2^1 \times n$、$2^2 \times n$…，

① 这里我们忽略了词项的表示，而只考虑文档 ID。计算时间复杂度时，可以将倒排记录表简单地看成是文档 ID 的列表。

可以进一步降低上述时间复杂度。整个倒排记录表在上述索引序列中进行向上过滤（percolate up）操作[①]，每一层的倒排记录仅仅只会处理一次。这种机制称为对数合并（logarithmic merging，参考图 4-7）。同以往一样，内存中的辅助索引（我们称之为Z_0）最多能容纳n个倒排记录。当达到上限n时，Z_0中的$2^0 \times n$个倒排记录会移到一个建立在磁盘上的新索引I_0中。当Z_0下一次放满时，它会和I_0合并，并建立一个大小为$2^1 \times n$的索引Z_1。Z_1可以以I_1的方式存储（如果I_1不存在）或者和I_1合并成Z_2（如果I_1已存在）。上述过程可以持续下去。对搜索请求进行处理时，我们会利用内存的索引Z_0和现有的磁盘上的有效索引I_i进行处理，并将结果合并。看到这里，对二项式堆结构[②]比较熟悉的读者会发现它与刚才提到的对数合并下的倒排索引结构之间具有很高的相似性。

```
LMERGEADDTOKEN(indexes, Z0, token)
 1  Z0 ← MERGE(Z0, {token})
 2  if |Z0| = n
 3    then for i ← 0 to ∞
 4         do if Ii ∈ indexes
 5               then Zi+1 ← MERGE(Ii, Zi)
 6                    (Zi+1 is a temporary index on disk.)
 7                    indexes ← indexes − {Ii}
 8               else Ii ← Zi    (Zi becomes the permanent index Ii.)
 9                    indexes ← indexes ∪ {Ii}
10                    BREAK
11         Z0 ← ∅

LOGARITHMICMERGE()
1  Z0 ← ∅    (Z0 is the in-memory index.)
2  indexes ← ∅
3  while true
4  do LMERGEADDTOKEN(indexes, Z0, GETNEXTTOKEN())
```

图 4-7　对数合并示意图，其中每个词条（词项 ID, 文档 ID）一开始通过 LMERGEADDTOKEN 函数加入到内存索引 Z_0 中。LOGARITHMICMERGE 对 Z_0 和索引初始化

由于每个倒排记录在 $\log_2(T/n)$ 层的每一层中都只处理一次，因此整个索引构建的时间是 $\Theta(T\log_2(T/n))$。当然，在获得索引构建效率提升的同时，我们也以降低查询处理的效率为代价。这是因为，现在我们要合并的不止是主索引和辅助索引这两个索引的搜索结果，而是 $\log_2(T/n)$ 个索引的结果。同辅助索引机制一样，这种机制偶尔也需要进行大规模索引合并操作，这会降低在合并时的搜索效率。当然，这种情况的发生频率较低，而且平均而言，合并中的索引规模也较小。

同时拥有多个索引会增加全局统计信息维护的复杂性。例如，它会影响 3.3 节提到的拼写

① 向上过滤，有时称为上滤，指的是堆数据结构中节点向上移动的过程。——译者注

② 参考(Cormen 等人 1990，第 19 章)。

校正算法，该算法基于最高选中次数选择最有可能的正确结果。在有多个索引和无效位向量的情况下，得到正确的词项选中数目不再是一个简单的查找过程。实际上，采用对数合并方法，信息检索系统的各个方面，包括索引维护、查询处理、分布等等，都要复杂得多。

由于动态索引构建的复杂性，一些大型搜索引擎采用从头开始重构索引而不是动态构建索引的方法。它们会周期性地构建一个全新的索引，然后将查询处理转到新索引上去，同时将旧索引删除。

? 习题 **4-4** 给定 $n = 2$ 及 $1 \leqslant T \leqslant 30$，对图 4-7 的算法进行逐步模拟。画出一个表格，给出在给定 $T=2*k$ 个词条已处理的时（$1 \leqslant k \leqslant 15$）所用到的 $I_0,\ldots,I_3$ 中的索引。该表格的前三行如下：

	I_3	I_2	I_1	I_0
2	0	0	0	0
4	0	0	0	1
6	0	0	1	0

4.6 其他索引类型

本章前面部分仅仅讨论了不包含位置信息的索引构建，而包含位置信息的索引结构与无位置信息的索引结构相比，除了需要考虑加入位置信息带来的更大数据开销之外，最主要的区别是将原来的（词项 ID，文档 ID）二元组变成了（词项 ID，文档 ID，（位置 1，位置 2，…））三元组，也就是每个文档 ID 后面附有词项 ID 在文档中的位置信息。这样一来，这里所讨论的所有算法都能应用到带位置信息的索引结构上。

在目前已讨论过的索引结构中，倒排记录表中的所有记录会按照文档 ID 排序。在第 5 章我们将会看到，这样做会对压缩提供方便，即可以不直接保存文档 ID 而是保存文档 ID 之间的间隔来进行压缩，这样就能够减少索引的存储空间。然而，这种结构对于排序式检索（ranked retrieval）系统（非布尔检索系统）而言（参见第 6 章和第 7 章）却不是最优的。在排序式检索系统中，倒排记录往往基于权重或者影响度排序，其中权重最大的倒排记录排在首位。因此，在这种结构下进行查询时，并不一定需要对所有的倒排记录进行扫描，而是在遇到权重足够小的文档后即可停止，因为后面的文档和查询的相似度可能都较低（参见第 6 章）。在以文档 ID 排序的索引中，新文档的加入往往只需在倒排记录表最后增加信息。而在以文档影响度排序的索引中，新文档的加入会导致在任意可能的地方插入信息，因此会大大增加倒排表更新的复杂度。

安全性是很多企业检索系统所必须重点关注的问题之一。一个低级别的员工不能对公司所有人的工资进行搜索，而只有授权的员工才能这样做。用户的搜索结果中不能出现禁止公开的文档，文档的存在本身也可能就是敏感信息。

用户授权往往通过 ACL（access control list，访问控制表）来实现。对于信息检索系统来说，

ACL 可以通过将每篇文档表示成所有能访问该文档的用户集合来实现（参见图 4-8），然后再对结果用户–文档矩阵进行转置。转置后的 ACL 索引中有一个每个用户都可以访问的所有文档组成的“倒排记录表”，这个表就是用户的访问表。搜索结果会和这个表进行求交集运算。然而，维护这样的索引是比较困难的，特别是在访问权限发生变化的时候。在 4.5 节介绍常规倒排索引的增量式构建问题时，我们就提到了上述困难。有些用户的可访问文档比较多，此时就要对长倒排记录表进行额外处理。在查询时，用户的资格往往通过直接从文件系统返回的访问信息来验证——即使这样会降低检索的速度。

另外，第 3 章我们还介绍了用于存储和检索词项（与存储和检索文档正好相反）的索引结构。

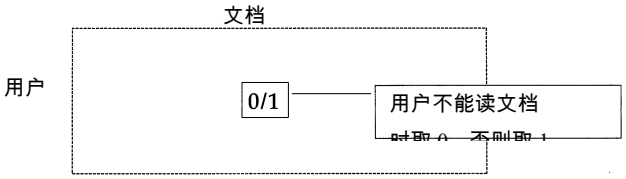


图 4-8　用户–文档的访问控制矩阵，如果用户 i 具有访问文档 j 的权限，那么矩阵元素（i, j）取 1，否则取 0。对查询进行处理时，用户的访问控制倒排记录表将和采用文本倒排索引得到的结果求交集

? 习题 **4-5**　拼写校正会不会危及到文档级的安全？考虑当拼写校正要基于用户无权访问的文档的这种情况。

习题 **4-6** 基于块的排序索引方法的总索引时间在表 4-3 中做了分解。假定系统采用表 4-1 中的参数，请在下表中补充对 Reuters- RCV1 处理的时间。

表4-3　采用基于块的排序索引方法对Reuters-RCV1进行索引构建时的5个步骤，这里的行号与图4-2对应

步　骤	时　间
1 文档集读取（对第4行）	
2 每10^7条记录进行10次初始排序（第5行）	
3 写入10个块（第6行）	
4 合并中的所有磁盘传输时间（第7行）	
5 实际合并的时间（第7行）	
总时间	

? 习题 **4-7**　对表 4-4 中的文档集重做习题 4-6。选择当前真实技术条件下的块大小（这里注意的是，块应该很容易放入主存），这样大小的块共有多少？

习题 **4-8**　假定我们有一个规模适度的文档集，它能使用图 1-4 所示的简单的内存式索引算法进行索引构建（参见第 1 章），计算该算法的内存、磁盘和时间开销并与基于块的排序索引算法相比较。

习题 **4-9**　假定 MapReduce 构架下的每台机器都有 100 GB 磁盘空间，而词项 the 的倒排记录表大小是 200GB。这种情况下，本章介绍的 MapReduce 方法就不能完成索引的构建，请问如何修改 MapReduce 以使它能解决这种情况？

习题 **4-10** 为了对负载均衡进行优化，MapReduce 中的倒排器必须获得大小相近的分区倒排记录文件，但是对于一个新文档集，事先并不知道键–值对的分布，这种情况下应该如何解决这个问题？

习题 **4-11** 将 MapReduce 用于计算每个词项在某个文件集中的出现次数，详细说明该任务中的 Map 和 Reduce 操作。按照图 4-6 的方式给出一个例子。

习题 **4-12** 我们认为辅助索引会削弱文档统计信息的质量。一个例子就是词项权重计算因子 idf，它定义成 $\log(N/\mathrm{df}_i)$，其中 N 是文档集中的文档总数，df_i 是第 i 个词项出现的文档数目。请说明如果仅利用主索引进行 idf 计算的话，那么即使很小的辅助索引也会导致显著的计算错误。请考虑一个罕见的词项突然在某些文档频繁出现的情况（如 Flossie 在 Tropical Storm Flossie 中的出现情况）。

4.7 参考文献及补充读物

Witten 等人（1999, 第 5 章）全面介绍了索引构建这个话题，并给出了能在内存开销、磁盘开销和时间开销之间采用不同折衷策略的索引算法。通常而言，基于块的排序索引算法在三者折衷方面做得很好。然而，如果将节约内存或者磁盘空间作为主要的考虑问题，那么其他算法可能会是更好的选择。在 Witten 等人（1999）的表 5-4 及表 5-5 中，我们会发现 BSBI 和“基于排序的多路归并”方法极其类似，但是这两个算法在词典结构及压缩的使用上都有所区别。

Moffat 和 Bell（1995）给出了一个所谓“原地”（in situ）构建索引的方法，即构建一个与磁盘空间大小相近的索引结构，并使得中间使用的额外临时文件的开销最小（也请参见 Harman 和 Candela（1990））。他们也把首先在索引构建中引入排序这一荣誉归于 Lesk（1988）及 Somogyi（1990）。

4.3 节讨论的 SPIMI 方法来自 Heinz 和 Zobel（2003）。在介绍时，我们在不少方面做了简化，比如：压缩技术的使用、词项的数据结构中应该也包含除了倒排记录表之外的其他信息，如文档频率、管理信息等。索引构建方面的最新、最详细的介绍材料推荐 Heinz 和 Zobel（2003）及 Zobel 和 Moffat（2006）。其他的对于词汇表大小来说具有良好扩展性的算法往往需要多遍数据扫描过程，如 FAST-INV 算法（参见 Fox 和 Lee，1991；Harman 等人，1992）。

MapReduce 架构在 Dean 和 Ghemawat（2004）中介绍。一个开源的 MapReduce 实现的地址是 http://lucene.apache.org/hadoop/。Ribeiro-Neto 等人（1999）和 Melnik 等人（2001）介绍了一些其他的分布式索引方法。有关分布式 IR 的介绍可以参见（Baeza-Yates 和 Ribeiro-Neto 1999, 第 9 章）、（Grossman 和 Frieder 2004, 第 8 章）及 Callan（2000）。

Lester 等人（2005）及 Büttcher 和 Clarke（2005a）分析了对数合并算法并与其他算法进行了比较。该方法的一个最早应用是 Lucene（http://lucene.apache.org）。其他的索引动态更新算法参见 Büttcher 等人（2006）和 Lester 等人（2006）。该文章还讨论了将旧索引替换成从头构建的新索引的策略。

Heinz 等人（2002）比较了内存中词汇表不断累加情况下的数据结构。Büttcher 和 Clarke（2005b）给出了普通倒排索引在面对多用户情况下的安全模型。对于 Reuters-RCV1 语料的一个详细介绍请参考 Lewis 等人（2004），而 NIST 授权分发该数据（参见 http://trec.nist.gov/data/reuters/reuters.html）。

Garcia-Molina 等人（1999, 第 2 章）深入讨论了系统设计相关的计算机硬件问题。

一个好的面向企业搜索的索引必须能够同企业内部各种不同的包含文本数据的应用进行数据交换，这些应用包括 Microsoft 的 Outlook、IBM 的 Lotus、诸如 Oracle 及 MySQL 的数据库软件、类似 Open Text 的内容管理系统及类似 SAP 的企业资源规划软件等等。

第5章

索引压缩

第 1 章介绍了信息检索系统中的两个主要数据结构：词典及倒排索引。本章将介绍对这两个数据结构的各种压缩技术，这些技术对于构建高效的 IR 系统非常关键。

进行压缩的一个优点显而易见：它能够节省磁盘空间。在本章我们即将看到，要达到 1∶4 的压缩比是非常容易的，也就是说可以降低 75%的索引存储开销。

索引压缩还有两个隐含的优点。第一是能增加高速缓存（caching）技术的利用率。在搜索系统中，词典中某些条目及其索引往往比其他条目及其索引的使用更频繁。例如，如果将一个频繁使用的查询词项 t 的倒排记录表放到高速缓存中，那么对仅由 t 构成的查询进行应答所需要的计算完全可以在内存中完成。如果采用压缩技术，那么高速缓存中就可以放更多的信息。当查询词项 t 的信息放在高速缓存时，处理查询 t 便不再需要进行磁盘操作，而只需将其倒排记录表在内存中解压缩即可。在后面我们将会看到，有很多简单高效的解压缩方法使得解压缩的代价非常低。因此，通过这种方式，我们能充分减少 IR 系统的应答时间。由于内存比磁盘更贵，所以，相对于磁盘空间的减少，采用高速缓存技术带来的速度提升是采用压缩技术的更主要的原因。

第二个隐含的优点是，压缩能够加快数据从磁盘到内存的传输速度。高效的解压缩算法在现代硬件上运行相当快，这样将压缩的数据块传输到内存并解压所需要的总时间往往会比将未压缩的数据块传输到内存要快。举例来说，即使会增加在内存进行解压缩的开销，我们也可以通过加载一个小很多的压缩倒排记录表来减少 I/O 时间。因此，在大部分情况下，使用压缩倒排记录表的检索系统会比没用压缩的系统的运行速度要快。

如果压缩的主要目的是为了节省磁盘空间，那么压缩算法的速度就不用特别考虑。但是，如果要提高高速缓存利用率和磁盘到内存的传输率，则解压缩的速度必须要快。本章介绍的压缩算法都非常高效，都可以达到上面提到的索引压缩的全部 3 个目标。

本章当中，倒排记录表中的每条记录定义成文档 ID。例如，一个倒排记录表为（6; 20, 45, 100），其中 6 是词项 ID，它包含 3 条记录，分别代表不同的文档 ID。正如 2.4.2 节所提到的那样，很多搜索系统中还包括词频和位置信息，但是这里我们只考虑简单的文档 ID 信息。如果对压缩词频和位置信息感兴趣，可以参看 5.4 节提到的相关参考文献。

本章一开始在 5.1 节中给出了在大规模文档集中待压缩对象（包括词项和倒排记录）的统计特性。然后在 5.2 节给出了采用“将词典视为长串”及按块存储的词典压缩技术。最后在 5.3 节介绍了两种倒排记录表压缩的方法：变长字节编码和 γ 编码。

5.1　信息检索中词项的统计特性

同上一章一样，这里仍然使用 Reuters-RCV1（参见表 4-2）语料作为我们的样本文档集。有关词项和倒排记录的统计信息列在表 5-1 中。“△%”表示和前一行相比数目的减少比率，“T%”表示以未过滤情况为基准的词项数目的累积减少比率。

表**5-1**　对**Reuters-RCV1**进行预处理前后词项、无位置信息倒排记录及词条的数目

	不同词项			无位置信息倒排记录			词条[①]		
	数目	△%	T%	数目	△%	T%	数目	△%	T%
未过滤	484 494			109 971 179			197 879 290		
无数字	473 723	–2	–2	100 680 242	–8	–8	179 158 204	–9	–9
大小写转换	391 523	–17	–19	96 969 056	–3	–12	179 158 204	–0	–9
30个停用词	391 493	–0	–19	83 390 443	–14	–24	121 857 825	–31	–38
150个停用词	391 373	–0	–19	67 001 847	–30	–39	94 516 599	–47	–52
词干还原	322 383	–17	–33	63 812 300	–4	–42	94 516 599	–0	–52

注：“△%”表示和前一行相比数目的减少比率，而“30 个停用词”和“150 个停用词”这两行均使用“大小写转换”那行作为对比基准。“T%”表示从未做任何过滤开始的数目累积减少比率。处理过程中采用 Porter 工具进行了词干还原（参考第 2 章）。

表 5-1 中的第 2 列给出了经过不同程度预处理后的词项数目，很显然，该数目是决定词汇表大小的主要因素。第 3 列给出了无位置信息倒排记录的数目，它反映了文档集无位置信息索引的预期大小，而包含位置信息的索引大小则与第 4 列所给出的位置信息的数目有关。

总的来说，表 5-1 中的统计数字表明，预处理对词典大小和无位置信息倒排记录的数目影响很大。词干还原和大小写转换分别会将不同词项的数目降低 17%，而将无位置信息倒排记录的个数分别降低 4%和 3%。高频词的处理也非常重要。三十定律（rule of 30）是指，出现频率最高的 30%个词在书面文本占了 30%的出现比例（表中的精确数字是 31%）。如果剔除出现频率最高的 150 个词（将它们当成停用词，参考 2.2.2 节），则无位置信息的倒排记录数目会减少 25%～30%。然而，尽管通过去除频率最高的 150 个停用词能够将倒排记录的个数减少 1/4 或更多，但是索引的压缩比例并不能达到这种减少量。在本章的后面部分我们将会看到，压缩之后高频词的每条倒排记录只占几个比特的空间[②]。

表格中的△值都在大规模文档集的典型的数值范围之内。然而，需要注意的是，代表减少率的 T 值对于不同的文本文档集来说差异可能很大。举例来说，由于法语相对于英语而言形态上更加丰富，所以采用词形归并工具对包含大量法语文本的网页集合进行处理，会比采用词干

① 词条的数目实际上等于倒排记录表中的位置信息个数。

② 也就是说，由于高频词的文档 ID 间隔小，压缩之后高频词所占用的空间少。因此，剔除高频词只能减少一小部分压缩空间。——译者注

还原工具 Porter 对仅仅包含英语的文档集进行处理所引起的词汇量的减少幅度要大得多。

本章介绍的压缩技术均属于无损压缩（lossless compression）技术，也就是说，压缩之后所有的原始信息都被保留。而与之相对的有损压缩（lossy compression）技术则会丢掉一些信息，因此可能会取得更高的压缩率。大小写转换、词干还原和停用词剔除均可看成是有损压缩技术。类似地，第 6 章介绍的向量空间模型和诸如 LSA（latent semantic analysis, 隐性语义分析，参见第 18 章）的降维技术能够建立更紧凑的文本表示，而基于这些表示方法不可能完全恢复到原始的文档集。当损失的信息不会被检索系统所用时，有损压缩是很有意义的。例如，对于 Web 搜索来说，其主要特点是文档数目巨大、查询短、用户只关注头几页结果等等，因此，可以将那些排名不可能靠前的文档的倒排记录给去掉。所以，在某些检索场景下可以使用有损压缩技术，而不会对检索效果造成任何损失。

在介绍词典的压缩技术之前，我们先估计一下文档集中不同词项的数目M，有时候我们也说成是估计语言的词汇量大小。牛津英语词典（*Oxford English Dictionary*）第 2 版定义的单词数目超过了 600 000，而对于大部分大规模的文档集来说，其词汇量会远远大于这个数目。牛津英语词典中并没有包括大部分人名、地名、产品名或包括基因名在内的大部分科学实体名称，而这些名称却都必须要包含在倒排索引中，这样用户才能对它们进行搜索[①]。

5.1.1 Heaps 定律：词项数目的估计

一个对词项数目 M 进行更好估计的方法是采用 Heaps 定律（Heaps' law），它将词项的数目估计成文档集大小的一个函数：

$$M = kT^b \qquad (5\text{-}1)$$

其中，T 是文档集合中的词条个数。参数 k 和 b 的典型取值为：$30 \leq k \leq 100$，$b \approx 0.5$。Heaps 定律认为，文档集大小和词汇量之间可能存在的最简单的关系是它们在对数空间（log-log space）中存在线性关系，并且这种线性关系往往和实际情况相吻合。图 5-1 给出了以 Reuters-RCV1 文档集为例的文档集大小和词汇量之间的关系示意图。从该图可以看出，当 $T > 10^5 = 100\,000$ 时，曲线和真实情况非常吻合，此时的参数值为 $b = 0.49$，$k = 44$。因此，对于前 1 000 020 个词条，Heaps 定律会估计得到大约 38 323 个词项，即

$$44 \times 1\,000\,020^{0.49} \approx 38\,323。$$

而实际的数目是 38 365，和估计值非常接近。

① 作者说这段话的目的是为了说明索引时词项的数量很大，当然实际当中，也可以不对这些名称直接索引，比如按 k-gram 或者汉语字索引等来搜索出这些名称。——译者注

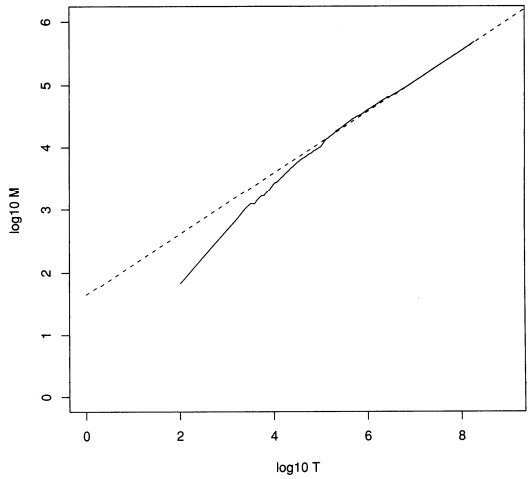


图 5-1　Heaps 定律，其中，词汇量 M 是文档集大小 T（以词条为单位）的函数。对于图中数据，虚线 $\log_{10}M = 0.49 \times \log_{10}T + 1.64$ 是其基于最小二乘法的拟合结果。因此，$k = 10^{1.64} \approx 44$，$b = 0.49$

不同的文档集下参数 k 的取值差异会比较大，这是因为词汇量的大小依赖于文档集本身以及对它进行处理的方法。大小写转换和词干还原会降低词汇量的增长率，而允许加入数字和容忍拼写错误则会增加该增长率。不论在某个特定的文档集中的参数取值如何，Heaps 定律提出了如下两点假设：

(i) 随着文档数目的增加，词汇量会持续增长而不会稳定到一个最大值；

(ii) 大规模文档集的词汇量也会非常大。

上述两个假设在经验上被认为是正确的，并且已经在很多大规模文档集上都得到了验证（参见 5.4 节）。因此，在大规模文档集情况下，词典压缩对于建立一个有效的信息检索系统来说是非常重要的。

5.1.2　Zipf 定律：对词项的分布建模

除了对词汇量进行估计之外，我们还想了解词项在文档中的分布情况。这可以为 5.3 节所介绍的倒排记录表的压缩算法提供支持。

一个常用的估计词项在文档中分布的模型是 Zipf 定律（Zipf's law），它的主要内容如下：如果 t_1 是文档集中的出现最多的词项，t_2 是文档集中的出现第二多的词项，依此类推，那么，排名第 i 多的词项的文档集频率 cf_i 与 $1/i$ 成正比，即

$$cf_i \propto \frac{1}{i}。 \qquad (5\text{-}2)$$

因此，如果出现最多的词项的出现次数是 cf_1 的话，出现第二多的词项的出现次数就是 cf_1 的一半，出现第三多的词项出现次数会是 cf_1 的 1/3，其余均可依此类推。直觉告诉我们，随着词项出现次数排名的下降，其出现频率也迅速下降。公式（5-2）是一个刻画这两种下降关系的最好方法之一，并且在实际中得到了广泛验证。

对公式(5-2)进行等价变换，可以将 Zipf 公式写成 $cf_i = ci^k$ 或者 $\log cf_i = \log c + k \log i$，其中 $k = -1$，c 是一个常数，它的定义将会在 5.3.2 节介绍。因此，*Zipf* 定律实际上是一个 $k = -1$ 情况下的幂定律（power law）。对于其他情况下的幂定律，可以参看第 19 章有关网页链接分布特性的内容。

图 5-2 给出了 Reuters-RCV1 文档集在对数空间下词项的文档频率和其出现次数排名之间的关系，图中同时也给出了一条 Zipf 定律所对应的直线 $\log cf_i = \log c - \log i$。从图中可以发现，实际数据和直线并不十分吻合，但是这对于 5.3 节的词项分布建模来说已经足够了。

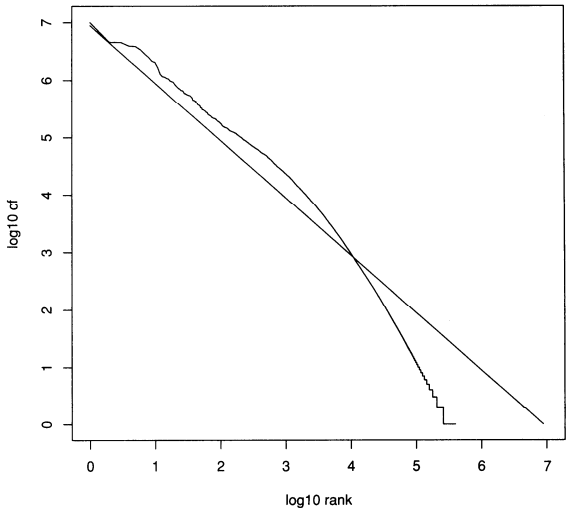


图 5-2 Reuters-RCV1 文档集上的 Zipf 定律，即词项的出现频率是其排名的函数，其中直线代表的是 Zipf 定律的预测分布结果（采用加权的最小二乘法拟合，截距是 6.95）

? 习题 **5-1** [*] 假定每个倒排记录占一个机器字长，那么基于表 5-1 的不同词条化方法得到的未压缩（无位置信息）索引的大小是多少？将这些数字和表 5-6 中的数字进行比较。

5.2 词典压缩

本节将给出词典的一系列数据结构，它们能够达到越来越高的压缩比。和倒排记录表的大小相比，词典只占非常小的空间（参见表 5-1）。那么，为什么还要对词典进行压缩呢？

这是因为决定信息检索系统查询响应时间的一个重要因素是磁盘的访问次数，而如果有部分词典存在于磁盘上，那么在处理查询时就需要更多的磁盘访问次数。因此，词典压缩的主要目的是将词典放入内存，或者说至少要把大部分词典放入内存，这样才能获得很高的查询吞吐率。尽管即使是非常大规模的文档集的词典也往往能够放入一台标准台式计算机的内存，但是在很多其他场景下情况并非如此。例如，大公司的一台企业搜索服务器也许要索引数太字节的文档，由于文档中可能包含多种不同语言，所以最后的词汇量可能会很大。另外，有些搜索系

统的设计要考虑到硬件条件的限制，比如手机或者车载计算机上的搜索系统等。压缩词典节省内存的其他原因还包括快速启动以及与其他应用程序共享资源等。运行搜索系统的 PC 上可能同时也在运行作为内存占用“ 大户” 的字处理软件，它们之间也要“ 和平共处”。

5.2.1　将词典看成单一字符串的压缩方法

最简单的词典的数据结构是 ,整个词典采用定长数组来存储且所有词项按照词典序排序(如图 5-3 所示)。假定对每个词项采用 20B 的固定长度 (因为英文中很少有长度大于 20B 的词) 存储，文档频率采用 4B 存储，词项到倒排记录表的地址指针也采用 4B 存储。这里的 4B 的指针能够访问 4GB 的地址空间。当然 , 对于像 Web 一样的超大文档集来说 , 则需要更大的字节数来存储指针。对于图 5-3 中的所示的数组，显然可以采用二分法来查找词典中的词项。在上述压缩机制下，Reuters-RCV1 文档集的词典存储空间总共为 $M\times$（20 + 4 + 4）= 400 000 × 28 = 11.2 MB。

	词项	文档频率	指向倒排记录表的指针
	a	656 265	→
	aachen	65	→
	…	…	…
	zulu	221	→
所需空间：	20B	4B	4B

图 5-3　采用定长数组存储的词典示意图

很显然，采用定长方法来存储词项存在着明显的空间浪费。英语中词项的平均长度大概是 8 字符（参见表 4-2），因此在上述定长存储机制下，每个词项平均会有 12 个字符的空间浪费。另外，上述定长机制也无法存储长度超过 20 字符的词项（如 hydrochlorofluorocarbons 和 supercalifragili–sticexpialidocious）。一种解决上述缺陷的方法是，将所有的词项存成一个长字符串（如图 5-4 所示）并给每个词项增加一个定位指针，它在指向下一词项的指针同时也标识着当前词项的结束。同以往一样，仍然可以通过二分查找法定位所需的词项，但是现在的表更小。由于每 20B 能够节省下 12B，所以相对于前面的定长机制而言，这种机制能够在词项存储上节省大约 60%的空间。当然，以上计算没有包括指向每个词项的指针所消耗的空间。所有这些指针寻址的空间大小为 400 000 × 8 = 3.2 × 10^6，因此一个指针可以用 $\log_2 3.2 \times 10^6 \approx 22$b 或者 3B 来表示。

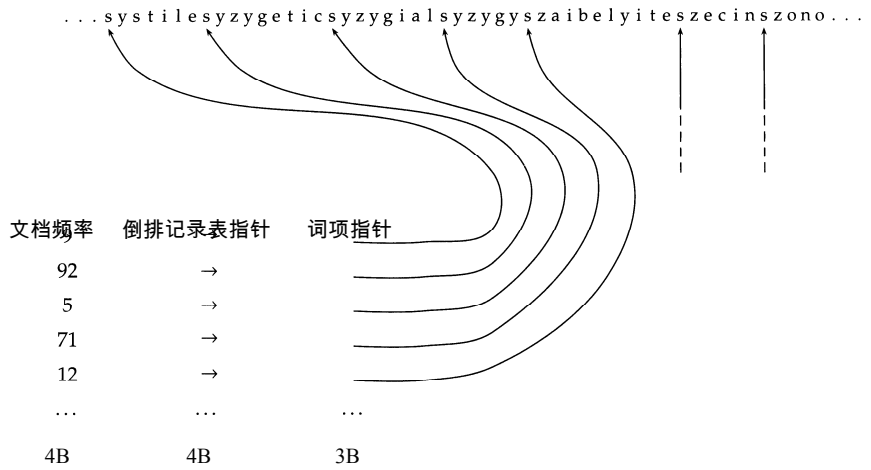


图 5-4 将整个词典看成一个长字符串的词典存储方式，其中指向下一词项的指针同时也标识着当前词项的结束。例子中的前 3 个词项是 systile、syzygetic 及 syzygial

在这种新机制下，对于 Reuters-RCV1 文档集需要 400 000 × （4 + 4 + 3 + 8）= 7.6 MB 的词典存储空间，其中词项文档频率、倒排记录表指针、词项指针分别用 4B、4B 和 3B 来表示，同时每个词项的平均长度是 8B。相对于前面所介绍的定长存储方法，词典存储空间从 11.2MB 压缩到了 7.6MB，压缩了大概 1/3。

5.2.2 按块存储

我们可以对 5.2.1 节的词典进行进一步的压缩：将长字符串中的词项进行分组变成大小为 k 的块（即 k 个词项一组），然后对每个块只保留第一个词项的指针（参见图 5-5）。同时，我们

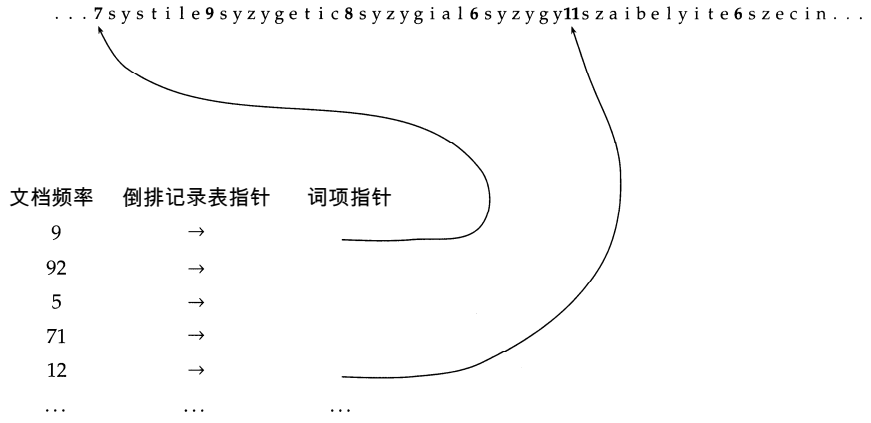


图 5-5 将每 4 个词项组成一个块的按块存储方式。第一个块包含 systile、syzygetic、syzygial 及 syzygy，它们的长度分别是 7B、9B、8B 和 6B。每个词项的前面都有一个字节来存储该词项的长度，根据这个长度可以跳到下一个词项

用一个额外字节将每个词项的长度存储在每个词项的首部。因此，对每个块而言，可以减少

k−1 个词项指针，但是需要额外的 kB 来保存 k 个词项的长度。对于 k=4，词项指针的存储将会减少（k−1）× 3 = 9B，但是同时需要增加 k=4B 来存储词项的长度。因此，在采用按块存储（blocked storage）的方式下，每 k=4 个词项就会节省出 5B，所以对于 Reuters-RCV1 来说，总共节省的空间为：400 000 × 1/4 × 5 = 0.5 MB。这样，原来的 7.6MB 存储空间可以进一步降低到 7.1MB。

显然，k 越大，压缩率也越高。但是，在压缩和词项查找速度之间必须要保持某种平衡。对于图 5-6 所示的 8 个词项组成的词典，采用二分法搜索的步骤在图中以双线表示，而采用表搜索（list search）的步骤则以单线表示。

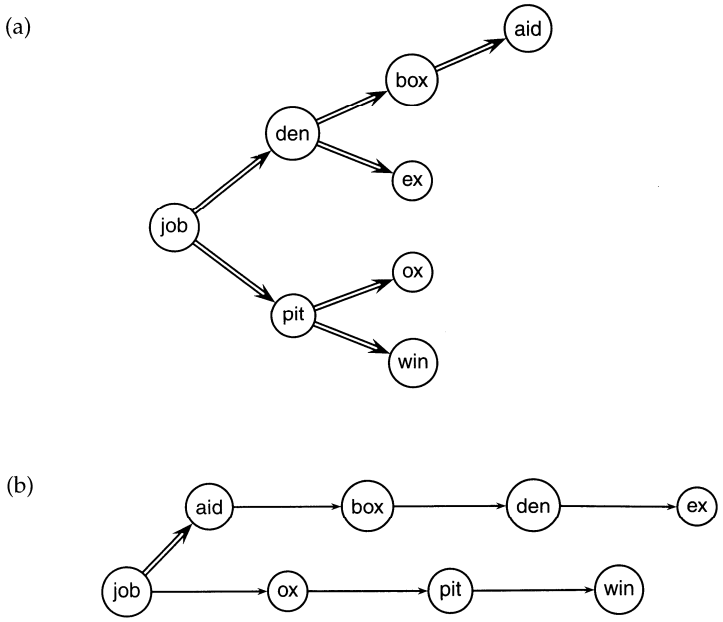


图 5-6　词典搜索示意图

（a）未压缩词典的搜索（b）采用按块存储方式下词典的搜索，其中 k = 4

当词典未压缩时，可以采用二分查找方法搜索词项（参考图 5-6 a）。在压缩后的词典中，首先通过二分查找得到块的入口位置，然后在块内进行线性查找得到最后的词项位置（参考图 5-6 b）。假定图 5-6（a）中每个后续词项的出现概率都相等的话，那么在未压缩的词典中查找的平均时间为(0 + 1 + 2 + 3 + 2 + 1 + 2 + 2) / 8≈ 1.6 步。比如，查找 aid 及 box 两个词项，就分别需要 3 步和 2 步。如果块的大小为 4，则在图 5-6（b）中所示的结构中的平均查找时间为(0 + 1 + 2 + 3 + 4 + 1 + 2 + 3) / 8 = 2 步，和前面相比，多花费了近 25%的时间。比如，寻找 den 需要 1 步二分搜索加上 2 步块内搜索。通过增加 k，可以把词典压缩到无限趋近于最小值 400 000 ×（4 + 4 + 1 + 8）= 6.8 MB，但此时词典的查找会因为 k 很大而慢得不可忍受。

迄今为止，词项之间的冗余性信息还没有利用，实际上，按照词典顺序排序的连续词项之

间往往具有公共前缀。因此，可以采用一种称为前端编码（front coding）的技术（参见图 5-7）。公共前缀被识别出来之后，后续的词项中便可以使用一个特殊的字符来表示这段前缀。我们在实验中发现，对于 Reuters-RCV1 文档集，采用前端编码又可以节省 1.2MB 的存储空间。

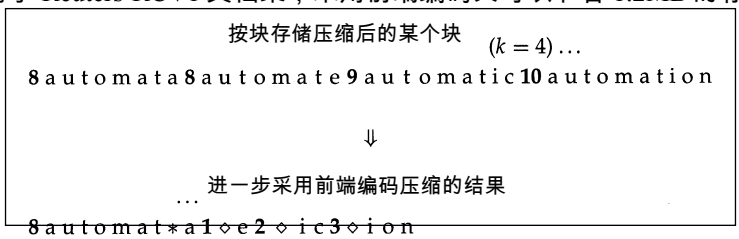


图 5-7 前端编码示意图。图中多个连续词项具有公共前缀 automat，那么在前缀的末尾用“*”号标识，在后续的词项中用“◇”表示该前缀。和前面一样，每个词项的前面第一个字节存储了该词项的长度

一些其他的具有更高压缩率的方法依赖于最小完美哈希（minimal perfect hashing）方法的使用，该哈希函数将 M 个词项映射到[1,…,M]上，并且不会发生任何冲突。但是，当插入新词项时，显然会发生冲突，此时不能对原有的完美哈希结果进行增量式修改而只能重新构造新的完美哈希函数。因此，完美哈希的方法无法在动态环境下使用。

即使采用最好的压缩方法，在很多情况下（比如文档集规模很大而内存很小时）要将所有词典存入内存也是不可能的。如果不得不把词典划分成不同页存储在磁盘上，则可以采用 B-树对每页的第一个词项进行索引。在处理大多数查询时，搜索系统必须要到磁盘获取倒排记录表，而在分页方式下还需要首先从磁盘获取词项所在的词典页，这会显著增加查询处理的时间，不过这种增加是可以忍受的。

表 5-2 总结了采用上面 4 种不同词典数据结构对 Reuters-RCV1 文档集的压缩结果。

表5-2 Reuters-RCV1文档集在采用不同压缩方法下的词典压缩结果

数据结构	压缩后的空间大小（单位：MB）
词典，定长数组	11.2
词典，长字符串+词项指针	7.6
词典，按块存储，k=4	7.1
词典，按块存储+前端编码	5.9

? 习题 **5-2** 估计 Reuters-RCV1 文档集词典在两种不同按块存储压缩方法下的空间大小。其中，第一种方法中 $k=8$，第二种方法中 $k=16$。

习题 **5-3** 将 Reuters-RCV1 文档集按块压缩存储，分别估计当块大小 k=4（即图 5-6 b）、k=8 及 k=16 情况下的词项查询时间，并与 k=1 情况下（图 5-6 a）的时间进行比较。

5.3　倒排记录表的压缩

回顾一下表 4-2，Reuters-RCV1 文档集有 800 000 篇文档，每篇文档有 200 个词条，每个词条有 6 个字符，倒排记录数目为 100 000 000。在本章中，每个倒排记录仅仅用文档 ID 来定义，也就是说，这里我们暂不考虑词项在文档内的频率和位置信息。上述这些数字对应表 5-1 的第 3 行（即“大小写转换”）。文档标识符（文档 ID）的长度为 $\log_2 800\,000 \approx 20$ 比特。因此，整个文档集大概有 800 000× 200 × 6B=960MB，而未压缩的倒排记录表的大小大概是 100 000 000 × 20/8 = 250 MB。

为设计出一个更高效的倒排文件表示方式，可以考虑每篇文档采用少于 20 比特的表示方式，观察中发现，高频词出现的文档 ID 序列值之间相差不大。为理解这一点，想象一下在文档集中遍历文档来寻找某个高频词项（如 computer）的过程：我们会找到一篇包含 computer 的文档，然后可能会跳过几篇不包含它的文档，之后又会找到另一篇包含 computer 的文档。这个过程可以不断循环下去（参见表 5-3）。这里面最关键的思路就是（一些词项对应的）倒排记录表中文档 ID 之间的间距（gap）不大，因此可以考虑用比 20 比特短很多的位数来表示它。实际上，对于一些高频词（如 the 和 and）来说，绝大部分间距都是 1。当然，对于只在文档集中出现一两次的罕见词（如表 5-3 中的 arachnocentric），其间距的数量级和文档 ID 的数目是一样的，因此仍然需要 20 比特。为了对这种间距分布的情况进行空间压缩，需要使用一种变长编码方法，它可以对短间距采用更短的位数来表示。

表5-3　对文档ID的间距而不是文档ID进行编码

	编码对象	倒排记录表					
the	文档ID	…	283042	283043	283044	283045	…
	文档ID间距			1	1	2	…
computer	文档ID	…	283047	283154	283159	283202	…
	文档ID间距			107	5	43	…
arachnocentric	文档ID	252000	500100				
	文档ID间距	252000	248100				

注：比如，对于 computer，存储间距序列 107, 5, 43, . . .,而不是文档 ID 序列 283154, 283159, 283202,…。当然，第一个文档 ID 仍然被保留（表中仅显示了 arachnocentric 的第一个文档 ID）。

为了对小数字采用比大数字更短的编码方式，本章主要考察了两类方法：按字节压缩及按位压缩。正如它们的名字所体现的那样，它们试图对间距分别采用最短的字节方式或位方式进行编码。

5.3.1　可变字节码

VB（*Variable byte*, 可变字节）编码利用整数个字节来对间距编码。字节的后 7 位是间距的

有效编码区，而第 1 位是延续位（continuation bit）。如果该位为 1，则表明本字节是某个间距编码的最后一个字节，否则不是。要对一个可变字节编码进行解码，可以读入一段字节序列，其中前面的字节的延续位都为 0，而最后一个字节的延续位为 1。根据上述标识可以把每个字节的 7 位部分抽取出来并连接在一起形成编码。图 5-8 给出了VB编码和解码的伪代码，表 5-4 给出了一个采用VB编码的例子 ①。

表5-4 VB编码。间距采用整数字节进行编码。每个字节中第一位为延续位，标识本次编码的结束(1)与否(0)

文档ID	824	829	215406
间距		5	214577
VB编码	00000110 10111000	10000101	00001101 00001100 10110001

```
VBEncodeNumber(n)
1  bytes ← ⟨⟩
2  while true
3  do Prepend(bytes, n mod 128)
4     if n < 128
5        then Break
6     n ← n div 128
7  bytes[Length(bytes)] += 128
8  return bytes

VBEncode(numbers)
1  bytestream ← ⟨⟩
2  for each n ∈ numbers
3  do bytes ← VBEncodeNumber(n)
4     bytestream ← Extend(bytestream, bytes)
5  return bytestream

VBDecode(bytestream)
1  numbers ← ⟨⟩
2  n ← 0
3  for i ← 1 to Length(bytestream)
4  do if bytestream[i] < 128
5        then n ← 128 × n + bytestream[i]
6        else n ← 128 × n + (bytestream[i] − 128)
7             Append(numbers, n)
8             n ← 0
9  return numbers
```

图 5-8 采用 VB 的编码和解码过程。div 和 mod 函数分别计算整数相除和相除后的余数。P_{REPEND} 在表头部增加一个元素。比如 $P_{REPREND}((1,2),3) = (3,1,2)$。$E_{XTEND}$ 表示将表合并，比如，$E_{XTEND}((1,2),(3,4)) = (1,2,3,4)$

采用 VB 编码压缩，我们在实验中发现，Reuters-RCV1 语料库的索引可以压缩到 116MB，相对于未压缩的索引，压缩率超过了 50%（参见表 5-6）。

① 请注意本表中以 0 为起始值开始计数。因为不需要考虑间距等于 0 的情况，实际上间距的起始值为 1。因此，编码 10000000 代表 1，10000101 实际上代表的间距是 6（而不是表中显示的 5），其他依此类推。

VB 编码的思想也可以应用于比字节更大或更小的单位上，比如 32 比特位字、16 比特位字和 4 比特位字（nibble，也称半字节）等等。编码单位越长，那么所需的位操作次数也越少，但是同时压缩率会降低甚至没有压缩。更短的编码单位会得到更高的压缩率，但同时位操作的次数也越多。总而言之，以字节为单位在压缩率和解压缩的速度之间提供了一个很好的平衡点。

对大多数信息检索系统来说，可变字节码压缩方法能够在时间和空间之间达到一个非常好的平衡点，在实现时也非常简单，而 5.4 节提到的很多其他方法都非常复杂。当然，如果磁盘空间稀缺的话，我们可以采用基于位的编码以得到更高的压缩率，特别是两个关系非常密切的编码方式：下一节将要介绍的 γ 编码及习题 5-9 提到的 δ 编码。

5.3.2　γ 编码

VB 编码能够根据间距的大小采用合适的字节数来编码。而基于位的编码能够在更细的位粒度上进行编码长度的自适应调整。最简单的位编码是一元编码（unary code）。数 n 的一元编码为 n 个 1 后面加个 0 组成的字符串（参考表 5-5 的前两列）。很显然，这种编码的效率不高，但是它会在后面用到。

表5-5　一些一元编码和γ编码的例子

数字	一元编码	长度	偏移	γ编码
0	0			
1	10	0		0
2	110	10	0	10,0
3	1110	10	1	10,1
4	11110	110	00	110,00
9	1111111110	1110	001	1110,001
13		1110	101	1110,101
24		11110	1000	11110,1000
511		111111110	11111111	111111110,11111111
1025		11111111110	0000000001	11111111110 ,0000000001

注：表中只给出了小数字的一元编码结果，γ 编码结果中的逗号只是为了方便阅读，并不是编码的内容。

一个明显的问题就是，某种编码在理论上能够达到怎样的性能？假定有 2^n 个间距，每个间距 G 满足 $1 \leqslant G \leqslant 2^n$，且 G 取其中每个值的可能性相等，那么对每个 G 的最优编码长度是 n 位来表示。因此，有些间距（这里是 $G = 2^n$）的编码不可能少于 $\log_2 G$ 位。我们的压缩目标就是尽可能地接近这个下界[①]。

一个和最优编码长度差距在常数倍之内的方法是 γ 编码。γ 编码将间距 G 表示成长度（length）和偏移（offset）两个部分进行变长编码。G 的偏移实际上是 G 的二进制编码，但是前端的 1 被

① 这里的意思似乎是为了说明 G 的表示不能少于 $\log_2 G$ 位。——译者注

去掉[①]。比如，对 13（二进制为 1101）进行编码，其偏移为 101。G的长度指的是偏移的长度，并采用一元编码。对于刚才的例子，偏移的长度是 3 位，因此其长度部分的编码是 1110。因此，13 的整个γ 编码是 1110101，即长度部分 1110 和偏移部分 101 的连接。表 5-5 中的右列中给出了一些其他数的γ 编码的例子。

对 γ 编码解码时，首先读入一元编码直至遇到 0 结束，比如在对 1110101 解码时，会一开始读入前 4 位 1110。然后便知道后面的偏移部分的长度是 3，因此，再正确读入后续的 3 位编码 101，补上原来去掉的前端的 1，最后可以得到 101→1101 = 13。

很显然，偏移部分的编码长度是$\lfloor \log_2 G \rfloor$位，而长度部分的编码长度为$\lfloor \log_2 G \rfloor$+1，因此，全部编码的长度为 $2\times\lfloor \log_2 G \rfloor$ + 1 位。γ 编码的长度永远都是奇数位，而且它与我们前面提到的最优编码长度 $\log_2 G$ 只相差一个因子 2。然而，我们是在 1 到 2^n 之间的所有 2^n 个间距是均匀分布的假设条件下得到上述最优结果的。实际中的情况往往并非如此。一般而言，我们事先并不知道间距的先验分布。

对于离散的概率分布 [②]P，它的熵（entropy）$H(P)$决定其编码性质（包括某个编码是否最优），其定义如下：

$$H(P) = -\sum_{x\in X} P(x)\log_2 P(x)$$

其中，X是所有需要编码的数字集合（因此，$\sum_{x\in X} P(x) = 1.0$）。熵是不确定性的一种度量方式。对于图 5-9 中的例子，分布 P 只得到两种结果，也就是说 $X=\{x_1, x_2\}$。当 $P(x_1) = P(x_2) = 0.5$ 时，熵取得最大值 $H(P) = 1$，此时对于后一个即将出现 x_1 还是 x_2 的不确定性最大。当 $P(x_1) = 1, P(x_2) = 0$ 或 $P(x_1) = 0, P(x_2) = 1$ 时，熵取得最小值 $H(P) = 0$。此时没有不确定性。

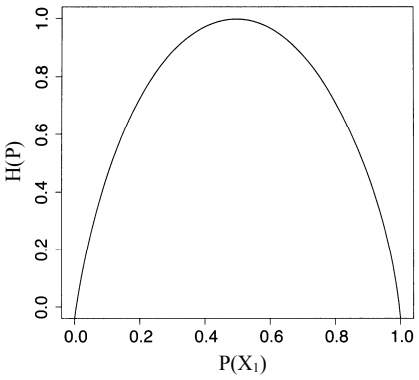


图 5-9 当随机变量只能取 x_1 及 x_2 两种结果时，熵 H（P）作为 P（x_1）的函数示意图

可以证明，在某些条件成立时，编码长度 L 的期望 $E(L)$ 的下界是 $H(P)$（参见参考文献）。

① 我们假设 G 前端不会以 0 开始。如果有 0 的话，那么要在删除前端的第一个 1 之前先删除前端所有的 0。

② 对概率论的基本概念感兴趣的读者可以参考 Rice(2006)或 Ross(2006)。需要指出的是，尽管这里仅仅对整数的概率分布感兴趣（间距、频率等），但是一个概率分布的编码性质与随机变量取值结果是否为整数无关。

可以进一步证明，对于 $1 < H(P) < \infty$, γ 编码方法的长度在最优编码长度的 3 倍之内，如果 $H(P)$ 较大，那么这个值接近 2 倍（给校对者，一校中下公式有误）：

$$\frac{E(L_\gamma)}{H(P)} \leq 2 + \frac{1}{H(P)} \leq 3 \text{。}$$

上述结果的一个不凡之处在于它对任何概率分布 P 都成立。因此，即使对间距的分布事先一无所知，也可以确信利用 γ 编码方式能够达到最优编码的 2 倍左右（最优编码对应的分布的熵较大）。对于任意分布，像 γ 编码这种编码长度在最优编码长度的某个倍数之内的编码方式，被称为通用性编码（universal code）。

除了通用性之外，γ 编码还具有两种适合于索引压缩的性质。第一，γ 编码方法是前缀无关码（prefix-free code，也称 prefix code），即一个 γ 编码不会是另一个 γ 编码的前缀。这也意味着对于一个 γ 编码序列来说，只可能有唯一的解码结果，不需要对编码进行切分，而如果切分则会降低解码的效率。γ 编码方法的第二个性质是参数无关（parameter free）性。对于很多其他高效的编码方式，需要对模型（比如二项式分布模型）的参数进行拟合使之适应于索引中间距的分布情况，而这样做会加大压缩和解压缩的实现复杂性，比如，必须对这些参数进行存储和检索。另外，在动态索引环境下，间距的分布会变化，因此原有的参数可能不再合适。而对于参数无关编码方法来说，上述问题就不存在。

那么，采用 γ 编码对倒排索引进行压缩能够得到多高的压缩率？为回答这个问题，我们使用了 5.1.2 节中介绍的有关词项分布的 Zipf 定律。按照该定律，文档集频率 cf_i 正比于其序值的倒数，也就是说，存在常数 c' 使得下式成立：

$$cf_i = \frac{c'}{i} \text{。} \tag{5-3}$$

可以选择另一个合适的常数 c，使得所有的 c/i 成为相对频率，总和为 1（也就是说 $c/i=cf_i/T$）：

$$1 = \sum_{i=1}^{M} \frac{c}{i} = c\sum_{i=1}^{M} \frac{1}{i} = cH_M \text{。} \tag{5-4}$$

$$c = \frac{1}{H_M} \tag{5-5}$$

其中，M 是不同词项的个数，H_M 是第 M 个调和数（harmonic number）[①]。对于 Reuters-RCV1 文档集来说，M=400 000，由于 $H_M \approx \ln M$，所以我们有

$$c = \frac{1}{H_M} \approx \frac{1}{\ln M} = \frac{1}{\ln 400\ 000} \approx \frac{1}{13} \text{。}$$

因此，频率排名第 i 位的词项的相对频率大约是 $1/(13i)$，其在长度为 L 的文档中出现的期望次数为：

① 对于正整数 M，第 M 个调和数 H_M=1/1+1/2+…+1/M。需要注意的是，在这里熵和调和数都用了大写字母 H 来表示，其具体含义要根据上下文来确定。

$$L\frac{c}{i}\approx\frac{200\times\frac{1}{13}}{i}\approx\frac{15}{i}。$$

上述变换中，词项的相对频率看成其出现的概率，而 200 是 Reuters-RCV1 文档集每篇文档的平均词条数目（参见表 4-2）。

迄今为止，我们已经推导出了文档集中的词项分布情况，并且进一步推出了倒排记录表的间距分布情况。通过这些统计数字可以计算出采用γ编码进行倒排索引压缩的空间需求。首先，将所有词汇表划分成段，每段有L_c=15 个词汇。从上面的推导有，第i个词项在每个文档中出现的平均次数是 15/i。因此，对于词汇表的第 1 段词汇（即前 15 个词）来说，其在每篇文档中出现的平均次数$\bar{f}$满足 1≤ $\bar{f}$，也就是说对于这些词汇来说，产生的间距个数为N。同样，可以依此类推，对于词汇表中第 2 段词汇来说，其在每篇文档中出现的平均次数$\bar{f}$满足 1/2≤ $\bar{f}$ ≤ 1，也就是说对于这些词汇来说，产生的间距个数为N/2。对于词汇表中第 3 段词汇来说，其在每篇文档中出现的平均次数$\bar{f}$满足 1/3≤ $\bar{f}$ ≤ 1/2，也就是说对于这些词汇来说，其产生的间距个数为N/3，依次类推。需要注意的是，为了后面计算的简便，上述推导中我们都只采用了下界值。当然，后面我们就会看到，即使在这样的假设条件下，最终的估计也过于保守[①]。我们给出的另外一个并不符合实际情况的假设就是，假设对于给定的词项，所有的间距都具有相同的值（如图 5-10 所示）。在这种均匀分布的假设下，可以认为第 1 段词汇的间距值为 1，第 2 段为 2，依次类推。

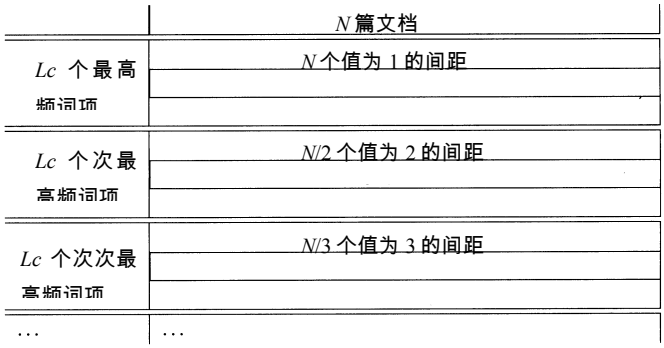


图 5-10 基于词项分段对倒排索引进行 γ 编码压缩的大小估计

对于间距值为 j 的 N/j 个间距采用 γ 编码方法，对于第 j 段的每个词项的倒排记录表进行压缩所需要的空间开销为：

$$bits\text{-}per\text{-}row=\frac{N}{j}\times(2\times\lfloor\log_2 j\rfloor+1)$$
$$\approx\frac{2N\log_2 j}{j}。$$

① 即估计出来的压缩大小仍然高于实际压缩值。——译者注

对每段进行编码需要 $Lc\times(2N\log_2 j)/j$ 位，总共有 $M/(Lc)$个段，因此，所有的倒排记录表所占的空间为：

$$\sum_{j=1}^{\frac{M}{Lc}}\frac{2NLc\log_2 j}{j}。\qquad(5\text{-}6)$$

对于 Reuters-RCV1 文档集，$\frac{M}{Lc}\approx\frac{400\ 000}{15}\approx 27\ 000$，因此

$$\sum_{j=1}^{27,000}\frac{2\times10^6\times15\log_2 j}{j}\approx 224MB。\qquad(5\text{-}7)$$

因此，960 MB 原始文档集的倒排索引能够压缩到大约 224 MB，差不多是原始文档集大小的 1/4。

实际中采用 γ 编码对 Reuters-RCV1 文档集进行索引压缩时得到的索引更小，只有大约 101 MB，差不多是原始文档集大小的 1/10。造成这种预期大小和实际大小不一致的主要原因包括：

(i) Zipf 定律对 Reuters-RCV1 文档集的词项频率实际分布的估计并不是非常准确；

(ii) 间距并不满足均匀分布。

对于表 4-2 中未经舍入处理的原始数字，采用 Zipf 定律进行预测，得到的索引大小为 251 MB。如果采用 Zipf 定律来产生词项的频率信息，并且对这些人造的词项的索引进行压缩，那么压缩后的索引大小是 254 MB。所以从某种程度上来说，如果关于词项分布的假设是精确的话，模型的预测结果也会准确。

表 5-6 概括了本章中所有的压缩技术。对于 Reuters-RCV1 文档集来说，词项关联矩阵（参见图 1-1）所占据的空间有 $400\ 000\times 800\ 000 = 40\times 8\times 10^9$bit，即 40 GB。

表5-6　Reuters-RCV1中的索引及词典压缩

数据结构	压缩后的空间大小（单位：MB）
词典，定长数组	11.2
词典，长字符串+词项指针	7.6
词典，按块存储，k=4	7.1
词典，按块存储+前端编码	5.9
文档集（文本、XML标签等）	3 600.0
文档集（文本）	960.0
词项关联矩阵	40 000.0
倒排记录表，未压缩（32bit位字）	400.0
倒排记录表，未压缩（20bit位）	250.0
倒排记录表，可变字节码	116.0
倒排记录表，γ 编码	101.0

注：压缩率取决于文档集中真实文本的比例。对于 Reuters-RCV1 文档集而言，它包含了大量 XML 标记。倒排记录和词典分别采用最好的两种压缩编码——γ 编码及按块存储前端编码对 Reuters-RCV1 文档集进行编码，可以获得非常高的压缩率（压缩后索引大小与原始文档集大小的比值，也常常称为压缩比）：（101+59）/3 600≈0.03。

对于 Reuters-RCV1 文档集，γ 编码技术能够取得比可变字节码更高的压缩率，提高的比率

大概是 15%。但是，解压的消耗会更高，主要原因是因为采用位编码时，两个编码之间的分界点可能在某个机器字当中，因此在对 γ 编码序列解码时需要很多诸如移位或者位掩码之类的位操作。随之而来的结果就是，在查询处理时，采用 γ 编码的消耗也比采用可变长字节编码要大。实际中到底采用哪种编码方式取决于应用的特点，比如，在具体应用时，节省磁盘空间与提高查询响应时间这两者在应用中到底孰轻孰重。

在表 5-6 中，未压缩的原始倒排索引的每条倒排记录都采用 32 位字存储，因此，整个索引的大小是 400 MB。而采用γ编码压缩后的索引大小为 101 MB，采用可变长字节码压缩后的索引大小为 116 MB，因此两种方式下的索引压缩率都大约为 25%。这也表明上述两种编码方式都能达到本章一开始所提到的压缩率为 1/4 的目标。通过减少索引的磁盘存储空间、增加高速缓存中的信息存放量和加快从磁盘到内存的数据传输速度，索引压缩能够大大提高索引的时空效率。

习题 **5-4 [*]** 写出表 5-3 及表 5-5 中的可变字节编码。

习题 **5-5 [*]** 写出倒排记录表（777，17743，294068，31251336）的可变字节编码及 γ 编码。在可能的情况下对间距而不是文档 ID 编码。写出 8 位块的二元码。

习题 **5-6** 考虑倒排记录表（4，10，11，12，15，62，63，265，268，270，400）及其对应的间距表（4，6，1，1，3，47，1，202,3，2，130）。假定倒排记录表的长度和倒排记录表分开独立存储，这样系统能够知道倒排记录表什么时候结束。采用可变字节码：

(i) 能够使用 1 字节来编码的最大间距是多少？

(ii) 能够使用 2 字节来编码的最大间距是多少？

(iii) 采用可变字节编码时，上述倒排记录表总共需要多少空间（只计算对这些数字序列进行编码的空间消耗）？

习题 **5-7** 由于间距的长度不可能是 0，所以经过移位以后得到的待编码值也不可能是 0，基于上述观察，可以采用一些小技巧对原有可变字节码进行修改：

(i) 对可变字节码进行修改，使之在同样的空间消耗下能够对更大的间距进行编码；

(ii) 能够使用 1 字节来编码的最大间距是多少？

(iii) 能够使用 2 字节来编码的最大间距是多少？

(iv) 采用可变字节编码时，上述倒排记录表总共需要多少空间（只计算对这些数字序列进行编码的空间消耗）？

习题 **5-8 [*]** 对于下列采用 γ 编码的间距编码结果，请还原原始的间距序列及倒排记录表。

1110001110101011111101101111011

习题 **5-9** 对于大数字（比如，表 5-5 中的 1025）来说，γ 编码的效率相对较低，这是因为在对偏移部分长度进行编码的时候采用了效率并不高的一元编码，在这点上，δ 编码和 γ 编码不一样，即偏移长度部分进行编码并不采用一元编码，而是采用 γ 编码。比如，7 的 δ 编码结果是 10,0,11（为了阅读方便，这里加了逗号）。其中 10,0 是数字 2 的 γ 编码结果，而偏移部分的 11 并没有发生变化。

(i) 计算表 5-5 中数字的 δ 编码，并回答在什么数字范围内，δ 编码的长度会小于 γ 编码。

(ii) 在表 5-6 中，γ 编码与可变字节码相比表现出优势，这是因为索引中包含很多停用词，它们对应的倒排记录表的间距都较小。

(iii) 在大间距值占主导地位的情况下，对 δ 编码和可变字节码进行比较。

习题 **5-10 [*]**　本章中将一元编码定义为“ 10”模式，即多个 1 后面跟一个表示结束的 0。如果将 0 和 1 的角色进行置换将会得到所谓“ 01”模式的一元编码。在使用这种编码的情况下，构建 G 的 γ 编码的需要两步：(1) 将 G 写成二进制形成，这需要 $b=\lfloor \log_2 G \rfloor+1$ 位；(2) 在(1)的结果前面追加$(b-1)$个 0。

(i) 使用上述 γ 编码对表 5-5 中的数字进行表示。

(ii) 证明上述编码和原来的 γ 编码的长度一样，解码也具有唯一性。

习题 **5-11 [***]**　在上述定义所表示的意义下，一元编码不是一种通用性编码。然而，存在某种间距分布，使得一元编码能够达到最优编码的效果，给出这种分布。

习题 **5-12**　请给出一些词项的例子，它们不满足文中所提到的“ 全部间距都具有相同值”的假设（该假设当时用于 γ 编码方式下的索引空间计算）。这些词项的一般特点是什么？

习题 **5-13**　考虑某个词项，它的倒排记录表的长度是 n，假定 $n = 10\,000$，试比较当间距符合均匀分布（即所有间距值相同）和不符合均匀分布这两种情况下，采用 γ 编码对倒排记录表进行压缩后得到的结果大小。哪种情况下得到的压缩倒排记录表更小？

习题 **5-14**　对表 4-2 的数据采用公式 (5-7) 求和，其结果应该约等于 251 MB。利用表 4-2 的数据时，不对 Lc、c 和词汇表的分块数目进行舍入处理。

习题 **5-15**　详细检查习题 5-14 中索引大小的计算过程，请给出公式（5-6）所用的全部近似值。

习题 **5-16**　选择一个文档集，确定文档的数目、词项的数目及每篇文档的平均长度，并回答如下问题：

(i) 按照公式（5-6）计算，压缩后的倒排索引的预期大小是多少？

(ii) 实现一个采用 γ 编码方式进行索引压缩的索引器，在该索引器下得到的实际索引大小是多少？

(iii)实现一个采用可变字节编码方式进行索引压缩的索引器，在该索引器下得到的实际索引大小是多少？

习题 **5-17** 为了能在主存储器中容纳更多的倒排记录，对在索引构建过程中产生的中间文件进行压缩是一个很好的思路。

(i) 采用上述做法会增加基于排序的块索引中合并过程的复杂度。请写出表 5-7 中 γ 编码表示的间距序列的合并过程。

(ii) 当采用压缩技术时，索引构建过程在空间角度上说是非常高效的，那么能否期望这个过程更快？

表5-7　基于块的排序索引中待合并的两个间距序列

运行1的采用γ编码的间距序列	1110110111111001011111111110100011111001
运行2的采用γ编码的间距序列	1111101000011111100010001111111001000001111101010 1

? 习题 **5-18**

(i) 请证明如果按照 Zipf 定律，词汇表的大小是有限的。而按照 Heaps 定律，词汇表的大小却是无限的。

(ii) 能否由从 Zipf 定律推导出 Heaps 定律？

5.4 参考文献及补充读物

Heaps 定律出自 Heaps（1978），关于它可以同时参考 Baeza-Yates 和 Ribeiro-Neto（1999）。在大文档集情况下词汇表增长的详细研究参见 Williams 和 Zobel（2005）。Zipf 定律归功于 Zipf（1949）。Witten 和 Bell（1990）考察了该定律的拟合效果。其他一些有关词项分布的模型，包括 K 混合模型及双泊松模型等，在 Manning 和 Schütze（1999, 第 15 章）有讨论。Carmel 等人（2001）、Büttcher 和 Clarke（2006）、Blanco 和 Barreiro（2007）及 Ntoulas 和 Cho（2007）都表明有损压缩能够达到非常好的压缩效果且不会降低或显著降低检索的效果。

有关词典压缩的详细介绍参见 Witten 等人（1999, 第 4 章），我们也推荐该读物作为本章的重要补充。

5.3.1 节的内容主要基于 Scholer 等人（2002）。论文作者发现，采用可变字节编码来处理查询比采用位压缩编码或不压缩快 2 倍，这时相对于最好的位编码来说，压缩率上会差 30%。他们同时也发现，相对于采用未压缩索引，采用压缩索引不仅能够节省磁盘空间，还能加快查询处理的速度。Anh 和 Moffat（2005）中的实验表明，相对于可变字节编码，可变半字节编码能够在压缩率上提高 5%～10%，同时最多会造成检索效果下降 1/3。Trotman（2003）也建议在磁盘空间并不紧张的情况下采用 VB 编码。近年来，Anh 和 Moffat（2005, 2006a）和 Zukowski 等人（2006）都建立了位对齐的二元编码，它们在解压速度上都比 VB 编码要快，效率上至少和 VB 一样好。Zhang 等人（2007）考察了在现代硬件条件下，采用一系列压缩技术对倒排记录表进行压缩时通过高速缓冲技术所带来的效果提高情况。

δ 编码（习题 5-9）和 γ 编码都源自 Elias（1975），文中同时证明了两种编码都具有通用性。需要补充说明的是，在 $H(P)\rightarrow\infty$ 时，δ 编码是渐进最优的。在大数字（大于 15）占主要地位时，δ 编码的性能优于 γ 编码。信息论（包括熵的概念）的有关介绍参见 Cover 和 Thomas（1991）。对任意分布 P，Elias 系列编码只是渐进最优，而可以构建任意接近最优值 $H(P)$ 的算术编码（参见 Witten 等人，1999，2.4 节）。

其他一些索引压缩技术可以参见 Witten 等人（1999，3.3 节、3.4 节及第 5 章），他们建议对索引压缩采用参数编码方式，此时可以明确地对每个词项间距的分布建模。例如，他们证明，在大规模文档集上，采用 Golomb 编码能够比采用 γ 编码获得更高的压缩率。Moffat 和 Zobel（1992）比较了几种参数编码方法，这其中包括 LLRUN（Fraenkel 和 Klein，1985）。

倒排记录表的间距分布取决于文档 ID 的分配。一系列的研究都关注如何分配文档 ID 来提高间距序列的压缩性能，这些研究包括（Moffat 和 Stuiver，1996；Blandford 和 Blelloch，2002；

Silvestri 等人，2004；Blanco 和 Barreiro，2006；Silvestri，2007）。这些技术将一个小范围的文档 ID 分配给某个文档簇中的所有文档，文档簇可以由给定时间内、特定网站内或共享其他一些属性的所有文档组成。因此，当簇内的文档在同一倒排记录表中出现的话，间距值会较小从而可以提高压缩效果。

另外，有一些压缩方法主要针对文档频率和词的位置信息，而不是文档 ID 信息，这方面的内容参见 Scholer 等人（2002）及 Zobel 和 Moffat（2006）。其中后者可以是目前最新的有关倒排索引深入介绍的读物，这其中的内容当然也包括索引压缩。

本章主要考虑了布尔检索条件下的索引压缩。对排序式检索（参见第 6 章）而言，按照词项频率对倒排记录表排序比按文档 ID 排序要好。在查询处理中，对倒排记录表的扫描可以提前停止，这是因为排在后面的倒排记录的权重较小，不足以改变现有的前 k 个文档的排序。将权重信息（与频率信息相反，它们大都是浮点数）预先计算好并存到索引中并不是一个好的做法，这是因为它们不能像整数一样被压缩得那么好（参见 7.15 节）。

在高效的信息检索系统中，文档的压缩也非常重要。De Moura 等人（2000）和 Brisaboa 等人（2007）给出了可以在压缩文本中直接进行词项和短语搜索的压缩机制，这些机制在一些常规的文本压缩工具（如 gzip 和 compress）中是无法做到的。

第 6 章

文档评分、词项权重计算及向量空间模型

迄今为止，我们介绍了支持布尔查询的索引处理办法，给定一个布尔查询，一篇文档要么满足查询的要求要么不满足。在文档集规模很大的情况下，满足布尔查询的结果文档数量可能非常多，往往会大大超过用户能够浏览的文档的数目。因此，对搜索引擎来说，对文档进行评分和排序非常重要。为此，对于给定的查询，搜索引擎会计算每个匹配文档的得分。本章将开始考察给每个（查询，文档）对进行评分的办法，主要内容包括以下 3 个部分。

1.6.1 节引入了参数化索引及域索引的概念，主要为达到两个目的：第一，采用上述索引方法可以通过元数据（比如文档的语言类型）对文档进行索引和检索；第二，上述索引能够提供一个简单的文档评分方法。

2.6.2 节引入了词项在文档中的权重的概念，并通过其出现的统计信息进行权重计算。

3.6.3 节中，每篇文档被表示为上述权重计算结果的向量，通过它可以计算查询和每篇文档的相似度。这就是为众所周知的向量空间方法。

6.4 节给出了向量空间模型中的权重计算方法的各种变形。第 7 章将主要介绍与向量空间模型实现时的计算开销及相关的话题。

在本章的不同部分当中，查询的概念有一些细微的差别。比如，6.1 节的查询的词项要求出现在文档的某个字段或域中。而从 6.2 节开始，这种要求被放松，我们主要讨论自由文本查询，在这类查询中，词项之间的相对次序、重要性以及其在文档中出现的位置并不作特别指定。本章当中，我们主要考察后一种查询的概念，在这种概念下，查询实际上就是多个词项构成的集合。

6.1 参数化索引及域索引

迄今为止，我们都将文档看成一系列词项的序列。实际上，大多数文档都具有额外的结构信息。数字文档通常会把与之相关的元数据（metadata）以机读的方式一起编码。所谓元数据，指的是和文档有关的一些特定形式的数据，比如文档的作者、标题以及出版日期等等。这些元数据通常会包括字段（field）信息，如文档的创建日期、文档格式、作者信息等等，文档的标题有时可以看成字段信息。字段的取值通常是有限的，比如，创作的时间集合显然是有限的。

考虑查询“ 寻找由 William Shakespeare 于 1601 年撰写、其中包含短语 alas poor Yorick 的文档”。和通常一样，查询的处理过程需要进行倒排记录表的合并操作，但是不同的是，这里在处理上述查询时还会涉及到参数化索引（parametric index）上的合并操作。对每个字段（比如文档

创建时间）都存在一个与之对应的参数化索引，通过它我们只会选择在时间字段上满足查询需求的文档。图 6-1 给出了这样一个参数化搜索的人机交互界面。有些字段为有序值，比如日期。上面给出的查询中，年份 1601 就是这样一个字段的取值。搜索引擎可以支持对一些有序字段在某个取值范围内搜索。为达到这个目的，可以对该字段的词典采用类似 B-树的数据结构进行组织。

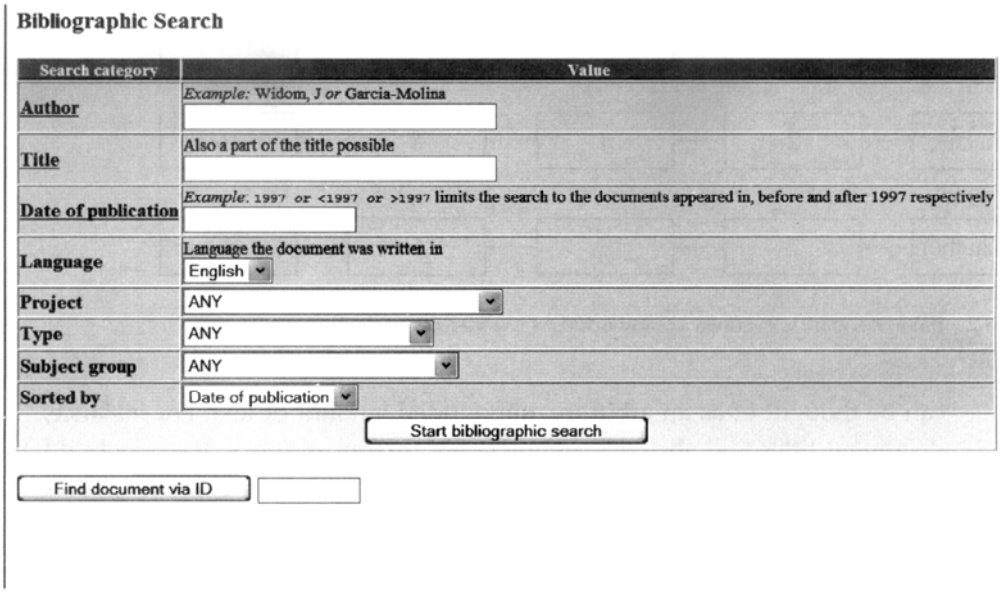


图 6-1 参数化搜索的一个例子。本例当中，允许按照作者或者语言种类对文档集进行搜索

域（zone）和字段很相似，只是它的内容可以是任意的自由文本。字段通常的取值可能性相对较小，而域可以由任意的、数目无限制的文本构成。比如，通常可以把文档的标题和摘要看作域。我们可以对文档的不同域构建独立的倒排索引，如果要支持类似“寻找标题中出现 merchant、作者中存在 william 且正文中存在短语 gentl rain 的文档”的查询，那么就需要建立图 6-2 这样的索引结构。参数化索引中，词典常常来自固定的词汇表（比如语言种类的集合、日期的集合等），而在域索引中，词典中应该收集来自域中自由文本的所有词汇。

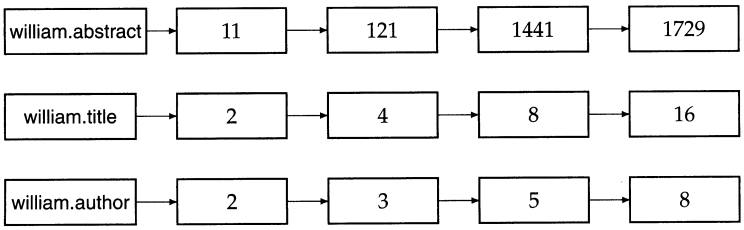


图 6-2 基本的域索引示意图，每个域采用词典项的某种扩展表示方法

实际上，可以通过对域进行编码来减少上述索引中词典的规模。图 6-3 给出了一个例子，可以将 william 在不同文档的 title 和 author 域中的出现情况进行统一编码。当词典大小为主要关

注目标时，这种编码方式就非常有用（比如我们要将词典放入内存）。采用这种编码的另外一个重要的原因是它能支持域加权评分（weighted zone scoring）技术的使用。

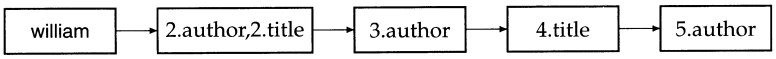


图 6-3　一种域索引的实现方法，其中域的信息存放在倒排记录表而不是词典中

6.1.1　域加权评分

前面我们主要关注基于字段或域的布尔查询的检索过程。接下来我们讨论域及字段的另外一种应用。

给定一个布尔查询 q 和一篇文档 d，域加权评分方法给每个（q,d）对计算出一个[0,1]之间的得分，该得分由每个域上的得分线性组合而成，而每个域上的得分取布尔值：要么是 0，要么是 1。更具体地，给定一系列文档，假定每篇文档有 l 个域，其对应的权重分别是 $g_1, \ldots, g_l \in [0, 1]$，它们满足 $\sum_{i=1}^{l} g_i = 1$。令 s_i 为查询和文档的第 i 个域的匹配得分（1 和 0 分别表示匹配上和没匹配上）。例如，在 AND 查询下，如果所有查询词项都出现在某个域中，则这个域的对应得分为 1；否则为 0。实际上，域中查询词项到{0,1}集合的出现关系映射可以是其他任何布尔函数而不只是 AND 函数。于是，域加权评分方法可以定义为：

$$\sum_{i=1}^{l} g_i s_i \text{。} \qquad (6\text{-}1)$$

该方法有时也称为排序式布尔检索（ranked Boolean retrieval）。

例 6-1　考虑一个文档集，其中每篇文档都有 3 个域——author、title 和 body，考虑查询 shakespeare。对于每个域，如果出现 shakespeare 时得分为 1，否则为 0。对于该文档集，有 3 个权重系数 g_1、g_2 和 g_3，它们分别对应 author、title 和 body 域。假定 g_1=0.2，g_2=0.3，g_3=0.5，这表示在匹配过程中 author 域的重要性最小，title 域相对大点，而 body 域重要性最大。

因此，如果 shakespeare 出现在某文档的 title 和 body 域，那么该文档最后的总得分是 0.8。

在实际中如何实现域加权评分方法？一个简单的方法就是依次扫描每篇文档并对其评分，评分时将不同域的得分进行累加汇总。这种方法简单但是不够直接，下面我们将介绍一种通过倒排索引直接计算域加权评分的方法。图 6-4 给出了这种思路的一个实现方法。本算法主要考虑两个词项 q_1 和 q_2，它们之间存在 AND 关系：如果两个词项都出现在文档同一域中，那么文档的得分为 1；否则为 0。在介绍这个算法之后，我们将介绍面向更复杂的查询和布尔函数的扩展方法。

```
ZoneScore(q1, q2)
 1  float scores[N] = [0]
 2  constant g[ℓ]
 3  p1 ← postings(q1)
 4  p2 ← postings(q2)
 5  // scores[] is an array with a score entry for each document, initialized to zero.
 6  //p1 and p2 are initialized to point to the beginning of their respective postings.
 7  //Assume g[] is initialized to the respective zone weights.
 8  while p1 ≠ NIL and p2 ≠ NIL
 9  do if docID(p1) = docID(p2)
10        then scores[docID(p1)] ← WeightedZone(p1, p2, g)
11             p1 ← next(p1)
12             p2 ← next(p2)
13        else if docID(p1) < docID(p2)
14                then p1 ← next(p1)
15                else p2 ← next(p2)
16  return scores
```

图 6-4 对两个倒排记录表计算域加权评分的算法，其中函数 WeightedZone 按照公式(6-1)计算

读者可能会注意到，图 6-4 中的算法和图 1-6 中的算法极其类似。实际上，两个算法中的倒排记录表的遍历过程是一样的，但是与图 1-6 所示算法不同的是，这里不仅仅是将满足 AND 关系的文档挑出来加入到结果集合中，而且还要计算该文档的得分。有些文献将图 6-4 中的得分数组 scores[]称为累加器（accumulator）集合，到我们后面谈到比 AND 更复杂的布尔函数时就会明白上述称呼的原因。这样，即使一篇文档并不包括所有词项，它也可以被赋予一个非零的得分。

6.1.2 权重学习

上面的计算中用到了不同域的权重，那么如何确定这些权重的值呢？这些权重可以由专家来设定，当然，理论上也可以由一般用户来指定。但是，人们越来越倾向于从人工标注好的训练数据中学习到这些权重。这种方法属于信息检索中一类被称为机器学习相关性（machine-learned relevance）评分及排序方法的范畴。由于域加权评分方法已经足以引出对机器学习相关性的介绍，在这里我们先简述这个主题的基本情况，要详细了解机器学习相关的内容请参见第 15 章。

1. 给定一批训练样本（training example），每个样本可以表示成一个三元组<查询 q, 文档 d，q 和 d 的相关性判断>。最简单的情况下，相关性判断的结果要么是相关（relevant） 要么是不相关（nonrelevant）。更细致的方法在实现时会利用更多级别的相关性判断结果。

2.利用上述训练样本集合学习到权重 g_i，使得利用这些权重在训练集中计算到的每篇文档的得分尽量接近事先给出的相关性判断结果。

对于域加权评分来说，上述过程实际上是在学习一个线性函数，它能够组合不同域的布尔得分结果。这种学习方法中的较高代价主要来自人工进行的相关性判断，这些判断需要消耗大量人力，尤其在文档集（如 Web）频繁变化的情况下这种代价更大。下面将基于一个简单的例子，来详细介绍如何把上述权重学习过程转变为一个简单的优化问题来进行求解。

考虑一个简单的域加权评分的例子，其中每篇文档只包含 title 和 body 两个域。给定查询 q

和文档 d，根据 title 及 body 域是否和 q 匹配，利用布尔匹配函数分别计算出布尔变量 $s_T(d,q)$ 和 $s_B(d,q)$。布尔匹配函数的例子可以参见图 6-4 中的算法，它使用了词项的 AND 关系作为布尔匹配函数。最后，对于查询 q 和文档 d，可以使用一个常数 $g\in[0,1]$将上面的 $s_T(d,q)$ 和 $s_B(d,q)$ 组合成一个最终得分：

$$socre(d,q) = g \cdot s_T(d,q) + (1\text{-}g)\, s_B(d,q)。 \qquad (6.2)$$

下面介绍如何根据训练样本来确定常数 g，其中每个训练样本可以表示成三元组 $\Phi_j=(d_j,q_j,r(d_j,q_j))$。对于给定的 d_j 和 q_j，$r(d_j,q_j)$是人工判断的相关性结果，它是个二值变量：相关或不相关。图 6-5 给出了一个包含 7 个训练样本的例子。

样本	文档ID	查询	s_T	s_B	相关性判断
Φ_1	37	linux	1	1	相关
Φ_2	37	penguin	0	1	不相关
Φ_3	238	system	0	1	相关
Φ_4	238	penguin	0	0	不相关
Φ_5	1741	kernel	1	1	相关
Φ_6	2094	driver	0	1	相关
Φ_7	3191	driver	1	0	不相关

图 6-5　一个训练样本集的例子

对于每个训练样本 Φ_j，我们会得到两个布尔值 $s_T(d_j,q_j)$和 $s_B(d_j,q_j)$，利用公式(6-2)进行如下计算：

$$score\,(d_j, q_j) = g \cdot s_T\,(d_j, q_j) + (1 - g)\, s_B(d_j, q_j)。 \qquad (6\text{-}3)$$

对于上述计算结果，可以将它与人工判断的结果 $r(d_j,q_j)$进行比较，后者取值要么为 0（代表不相关），要么为 1（代表相关）。定义公式(6-3)评分计算结果的误差函数为

$$\varepsilon(g, \Phi_j) = (r(d_j,q_j) - score\,(d_j, q_j))^2，$$

因此，所有训练样本的误差函数为

$$\sum_j \varepsilon(g,\Phi_j)。 \qquad (6\text{-}4)$$

至此，从给定训练集中学习常数 g，就变成了一个选择 g 使得公式(6-4)所表示的误差函数取最小值的过程。

对公式(6-4)选择最佳 g 值实际上可以看成是一个 $g\in[0,\ 1]$的二次规划问题的求解过程，这将在 6.1.3 节做详细介绍。

6.1.3　最优权重 g 的计算

可以注意到，对于某个样本 Φ_j，如果 $s_T(d_j,q_j)$ =0 且 $s_B(d_j,q_j)$ =1，那么利用公式(6-2)进行计算得到的分数是 1-g。对于其他 3 种可能的情况，利用公式(6-2)得到的结果和刚才所讲情况的结果可以一起概括到下图中。

s_T	s_B	Score 得分
0	0	0
0	1	$1-g$
1	0	g
1	1	1

图 6-6 s_T和 s_B的 4 种可能的取值组合

令 n_{01r}表示当 $s_T(d_j,q_j)$=0 及 $s_B(d_j,q_j)$=1 且人工判断为相关的样本个数，而 n_{01n}表示此时人工判断为不相关的样本个数。因此，公式(6-4)中由 $s_T(d_j,q_j)$ =0 及 $s_B(d_j,q_j)$ =1 的训练样本带来的误差为：

$$[1-(1-g)]^2 n_{01r} + [0-(1-g)]^2 n_{01n}。$$

同样可以写出 s_T和 s_B的其他 3 种取值组合的情况所带来的误差，因此，总的误差为：

$$(n_{01r} + n_{10n})\, g^2 + (n_{10r} + n_{01n})\,(1-g)^2 + n_{00r} + n_{11n}。 \quad (6\text{-}5)$$

对 g 求导数，并令其为 0，则可以解出最优的 g 值如下：

$$\frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}。 \quad (6\text{-}6)$$

习题 **6-1** 当使用域加权评分时，是不是必须对每个域使用同样的布尔匹配函数？

习题 **6-2** 上面的例 6-1 中，如果 $g_1 = 0.2$, $g_2 = 0.31$ 及 $g_3 = 0.49$，那么对于一个文档来说所有可能的不同得分有多少？

习题 **6-3** 重写图 6-4 中的算法使之能够处理 2 个以上的词项构成的查询。

习题 **6-4** 写出图 6-4 中两个倒排记录表情况下的 WEIGHTEDZONE 函数的伪代码。

习题 **6-5** 将公式(6-6)应用到图 6-5 的样本集上，估计该样本集下的最优 g 值。

习题 **6-6** 对于习题 6-5 中估计到的 g，对每个二元组 <查询，文档>计算域加权评分值，并将之与图 6-5 中人工判断结果（0 或 1）相比较，确定它们之间的关系。

习题 **6-7** 为什么公式(6-6)中不包含 $s_T(d_t, q_t)$和 $s_B(d_t, q_t)$ 在取值相等情况下的训练样本个数？

6.2 词项频率及权重计算

到目前为止，我们只考虑了词项在文档域中出现与否这两种情况。本节将在此基础上往前更进一步，如果文档或者域中词项出现的频率越高，那么该文档或者域的得分也越高。为了说明这一点，我们先回忆一下 1.4 节提到的自由文本查询，即在搜索界面上自由输入的一个或者多个没有通过任何搜索连接符（如布尔操作符）连接的词项。这种类型的查询在 Web 上非常流行，它将查询简单地看成是多个词组成的集合。那么，一个看上去很合理的方式是先基于每个查询词项与文档的匹配情况对文档打分，然后对所有查询词项上的得分求和。

这样，我们对文档中的每个词项都赋予了一个权重，它取决于该词项在文档中出现的次数。

首先，我们对于词项 t，根据其在文档 d 中的权重来计算它的得分。最简单的方式是将权重设置为 t 在文档中的出现次数。这种权重计算的结果称为词项频率（term frequencey），记为 $\mathrm{tf}_{t,d}$，其中的两个下标分别对应词项和文档。

对于文档 d，利用上述 tf 权重计算方式（或者任意一个将 tf 映射成实数的权重计算函数）得到的权重集合可以看成是文档的一个经过量化以后得到的浓缩版本。在这种通常称为词袋模型（bag of words model）的文档视图的情况下，词项在文档中的出现次序被忽略，但是出现的次数非常重要，这和布尔检索形成了鲜明对比。需要指出的是，这里我们只保留了词项在文档中出现的次数。因此，在这种情况下，文档 Mary is quicker than John 和 John is quicker than Mary 的文档视图完全等价。尽管如此，直觉上仍认为在词袋模型下表示相近的文档的内容具有相似性。我们在 6.3 节将会进一步讨论这个问题。

在这之前，我们首先考察如下这个问题：文档中所有词项的重要性是一样的吗？答案显然是否定的。在 2.2.2 节中，我们曾讨论过停用词，不需要对它们建立索引，因为它们一般不会对检索和评分造成大的影响。

6.2.1　逆文档频率

原始的词项频率会面临这样一个严重问题，即在和查询进行相关度计算时，所有的词项都被认为是同等重要的[①]。实际上，某些词项对于相关度计算来说几乎没有或很少有区分能力。例如，在一个有关汽车工业的文档集中，几乎所有的文档都会包含 auto，此时，auto就没有区分能力。为此，下面我们提出一种机制来降低这些出现次数过多的词项在相关性计算中的重要性。一个很直接的想法就是给文档集频率（collection frequency）较高的词项赋予较低的权重，其中文档集频率指的是词项在文档集中出现的次数。这样，便可以降低具有较高文档集频率的词项的权重。

实际中，一个更常用到的因子是文档频率（document frequency）df_t，它表示的是出现 t 的所有文档的数目。这是因为文档评分的目的是区分文档，所以最好采用基于文档粒度的统计量（比如出现 t 的所有文档的数目），而不是采用基于整个文档集的统计量来计算。图 6-7 也通过一个例子来说明了为什么采用 df 而不是 cf 的原因，该例子表明 df 和 cf 可能会相差很大。具体来说，try 和 insurance 的 cf 值基本相当，但是它们的 df 值却相差很大。直观上说，对于查询 insurance，由于包含 insurance 的文档只有少数，所以我们希望这些文档的得分能够得到提升；而对于查询 try，由于包含 try 的文档数目较大，因此它们的得分提升幅度相对不大。

词	cf	df
try	**10422**	**8760**
insurance	**10440**	**3997**

① 这里的意思是如果仅仅采用 tf 表示的话，相当于默认所有词项在全局中的重要性是一样的。当然，不同词项在同一篇文档的 tf 不一样，其贡献也不一样。——译者注

图 6-7 Reuters RCV1 文档集中文档集频率（cf）及文档频率（df）表现迥异的例子

由于 df 本身往往较大，所以通常需要将它映射到一个较小的取值范围中去。为此，假定所有文档的数目为 N，词项 t 的 idf（inverse document frequency, 逆文档频率）的定义如下：

$$idf_t = \log \frac{N}{df_t}。 \quad (6\text{-}7)$$

因此，一个罕见词的 idf 往往很高，而高频词的 idf 就可能较低。图 6-8 给出了包含 806 791 篇文档的 Reuters-RCV1 文档集上的 idf 的例子。其中该例中的对数以 10 为底。实际上，我们将在习题 6-12 中看到，对数的底并不会对文档的相对排序有实际影响。在 11.3.3 节中，我们将会给出公式（6-7）中 idf 的具体计算方法的一个合理性分析。

词项	dft	idft
car	**18 165**	**1.65**
auto	**6723**	**2.08**
insurance	**19 241**	**1.62**
best	**25 235**	**1.5**

图 6-8 idf 值的例子，本图中给出了 Reuters-RCV1 文档集中不同频率的词项的 idf 值

6.2.2 tf-idf 权重计算

对于每篇文档中的每个词项，可以将其 tf 和 idf 组合在一起形成最终的权重。tf-idf 权重机制对文档 d 中的词项 t 赋予的权重如下：

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t。 \quad (6\text{-}8)$$

换句话说，$\text{tf-idf}_{t,d}$ 按照如下的方式对文档 d 中的词项 t 赋予权重：

(1) 当 t 只在少数几篇文档中多次出现时，权重取值最大（此时能够对这些文档提供最强的区分能力）；

(2) 当 t 在一篇文档中出现次数很少，或者在很多文档中出现，权重取值次之（此时对最后的相关度计算作用不大）；

(3) 如果 t 在所有文档中都出现，那么权重取值最小。

这样，就可以把文档看成是一个向量（vector），其中的每个分量都对应词典中的一个词项，分量值为采用公式(6-8)计算出的权重值。当某词项在文档中没有出现时，其对应的分量值为 0。这种向量形式对于评分和排序十分重要，关于这一点我们会在 6.3 节中做进一步介绍。首先介绍重合度评分指标（overlap score measure）[①]：文档d的得分是所有查询词项在文档中的出现次数tf之和。当然，我们可以对这种方法进行修正，即不采用tf而采用tf-idf权重求和：

$$Score(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d}。 \quad (6\text{-}9)$$

① 根据双方的重合程度来计算相似度的一种指标。——译者注

6.3 节将给出公式（6-9）的一个更严密的形式。

习题 **6-8**　为什么 idf 的大小总是有限的？

习题 **6-9**　出现在所有文档中的词项的 idf 值是多少？将计算结果和使用停用词的方式进行比较。

习题 **6-10**　考虑图 6-9 中的 3 篇文档 Doc1、Doc2、Doc3 中几个词项的 tf 情况，采用图 6-8 中的 idf 值来计算所有词项 car、auto、insurance 及 best 的 tf-idf 值。

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

图 6-9　习题 6-10 中所使用的 tf 值

习题 **6-11**　词项的 tf-idf 权重能否超过 1？

习题 **6-12**　公式（6-7）中对数的底对公式（6-9）会有什么影响？对于给定查询来说，对数的底是否会对文档的排序造成影响？

习题 **6-13**　假定公式（6-7）中的对数以 2 为底，给出 idf 的一个简单近似值。

6.3　向量空间模型

6.2 节提出了文档向量的概念，其中每个分量代表词项在文档中的相对重要性。一系列文档在同一向量空间中的表示被称为向量空间模型（vector space model，简称 VSM），它是信息检索领域一系列相关处理的基础，比如文档的评分、文档的分类及聚类等。接下来我们首先给出向量空间模型评分方法的基本思路，其中最关键的步骤是将查询也看成同一空间下的向量（参见 6.3.2 节）。

6.3.1　内积

假设文档 d 对应的向量用 $\vec{V}(d)$ 表示，其中每个分量对应一个词项。如果不特别说明的话，那么假定以下的向量分量均采用 tf-idf 权重计算方式。当然，具体的权重计算方式对下面的讨论本身并没有实质性的影响。至此，一组文档的集合可以看成向量空间中的多个向量，每个词项对应一个坐标轴。这种表示忽略了词项在文档中的相对次序，在 6.2 节也曾提到过一个例子，即在这种词袋模型表示下，文档 Mary is quicker than John 和 John is quicker than Mary 是等价的。

在向量空间下，如何对两篇文档的相似度进行计算？我们可以首先考虑采用两个文档向量差向量的大小进行计算。但是这种计算方法有一个缺点：两篇内容相似的文档向量的差向量可能很

大，这是因为一篇文档可能比另一篇文档要长得多。因此，尽管两个词项在每篇文档中的相对分布完全一样，但是其中一个词项的绝对词频有可能远远大于另一个，于是计算出的差向量也就很大。

为了弥补文档长度给上述相似度计算所带来的负面效果，计算两篇文档 d_1 和 d_2 相似度的常规方法是求向量 $\vec{V}(d_1)$ 和 $\vec{V}(d_2)$ 的余弦相似度（cosine similarity）：

$$sim(d_1,d_2)=\frac{\vec{V}(d_1)\cdot\vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}\ , \qquad (6\text{-}10)$$

其中，分子是向量 $\vec{V}(d_1)$ 和 $\vec{V}(d_2)$ 的内积（inner product）或称点积（dot product），分母是两个向量的欧几里得长度（Euclidean length，简称欧氏长度）的乘积。两个向量的内积 $\vec{x}\cdot\vec{y}$ 定义为 $\sum_{i=1}^{M}x_iy_i$，文档 d 对应的向量表示为 $\vec{V}(d)$，它是一个 M 维的向量 $\vec{V}_1(d)\ldots\vec{V}_M(d)$，$d$ 的欧几里得长度定义为 $\sqrt{\sum_{i=1}^{M}V_i^2(d)}$。

公式(6-10)中除以分母的效果实际上相当于将向量 $\vec{V}(d_1)$ 和 $\vec{V}(d_2)$ 进行长度归一化（称为欧氏归一化），得到单位向量：$\vec{v}(d_1)=\vec{V}(d_1)/|\vec{V}(d_1)|$，$\vec{v}(d_2)=\vec{V}(d_2)/|\vec{V}(d_2)|$。因此，公式(6-10)可以重写成：

$$sim(d_1,d_2)=\vec{v}(d_1)\cdot\vec{v}(d_2)\text{。} \qquad (6\text{-}11)$$

例 6-2 考虑图 6-9 中的文档，采用欧氏归一化方式对每篇文档中的 tf 值进行归一化，得到的 3 个文档 Doc1、Doc2 和 Doc3 的 $\sqrt{\sum_{i=1}^{M}V_i^2(d)}$ 值分别为 30.56、46.84 和 41.30。最后的 tf 归一化结果如图 6-10 所示。

	Doc1	Doc2	Doc3
car	0.88	0.09	0.58
auto	0.10	0.71	0
insurance	0	0.71	0.70
best	0.46	0	0.41

图 6-10 图 6-9 中 tf 值的欧氏归一化结果

因此，公式(6-11)可以看成是两个归一化以后的文档向量的内积，也就是计算两个向量的夹角余弦（如图 6-11 所示）。既然检索过程是计算查询和文档的相似度的，那么计算出两篇文档的相似度 sim(d_1,d_2)的作用何在？我们考虑在给定文档 d（也许是文档集中的某篇文档 d_i）的前提下在文档集中搜索与之相近的文档的过程。如果在某个系统中，用户先确定一篇文档然后查

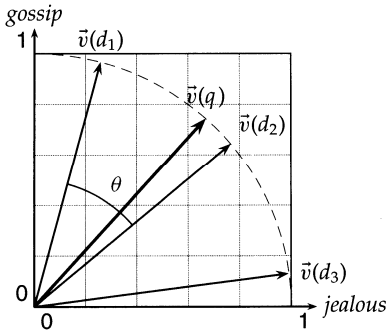


图 6-11　两个向量的夹角余弦相似度计算：$\text{sim}(d_1, d_2) = \cos\theta$

找与之相似的文档，那么上面给出的文档相似度计算就非常有用。在一些搜索引擎的返回结果当中有时就能看到这种搜索方式，比如一些搜索引擎的 more like this（“类似网页”）按钮所对应的搜索功能。于是，查找与 d 最相似的文档这个问题可以归结成寻找和 d 有最大内积结果 $\vec{v}(d)\cdot\vec{v}(d_i)$ 的文档过程。因此，需要首先计算 $\vec{v}(d)$ 与每篇文档 $\vec{v}(d_1)$,…, $\vec{v}(d_N)$ 的内积，然后选择具有最大值的结果。

例 6-3　图 6-12 中给出了 3 个词项 affection、jealous 及 gossip 在如下 3 篇小说中的出现次数：Jane Austen 的 *Sense and Sensibility*（SAS）和 *Pride and Prejudice*（PAP）及 Emily Brontë 的 *Wuthering Heights*（WH）。当然，这些小说中还存在其他词项。本例当中，这些小说都以一个 3 维单位向量来表示，每一维对应一个词项，这里我们只使用的原始的 tf 值，没有考虑乘上 idf 因子。最终的权重值如图 6-13 所示。

词项	SAS	PAP	WH
affection	**115**	**58**	**20**
jealous	**10**	**7**	**11**
gossip	**2**	**0**	**6**

图 6-12　3 部小说中的词项频率，这 3 部小说分别是：Jane Austen 的 *Sense and Sensibility*（中文名为《理智与情感》，简记为 SAS）和 *Pride and Prejudice*（中文名为《傲慢与偏见》，简记为 PAP）及 Emily Brontë 的 *Wuthering Heights*（中文名为《呼啸山庄》，简记为 WH）

词项	SAS	PAP	WH
affection	0.996	0.993	0.847
jealous	0.087	0.120	0.466
gossip	0.017	0	0.254

图 6-13　图 6-12 中 3 部小说对应的词项向量。这些向量基于原始的 tf 值进行归一化计算，并假定文档集中没有出现其他的词项。由于 affection 和 jealous 在所有 3 篇文档中都出现了，所以如果采用大多数 tf-idf 权重计算方法时，其权重值都为 0

现在考虑上面 3 个 3 维向量两两之间的余弦相似度。简单的计算得出，$sim(\vec{v}(\text{SAS}),\vec{v}(\text{PAP}))=0.999$，$sim(\vec{v}(\text{SAS}),\vec{v}(\text{WH}))=0.888$。可见，同一作者 Austen 所写的两本书 SAS 和 PAP 之间相对比较接近，而 Bronë 所写的书 WH 则与它们的距离较远。实际上，当仅仅考上述 3 个词项时，SAS 和 PAP 的相似度几乎接近最大的相似度取值 1。需要指出的是，这里我们仅仅考虑了 tf 权重，也可以使用其他词项权重计算方法。

很自然地，将一个包含 N 篇文档的文档集看成向量的集合相当于将整个文档集看成一个 $M\times N$ 的词项–文档矩阵（term-document matrix），其中矩阵的每行代表一个词项，每列代表一篇文档。通常，词项可以在索引前进行词干还原处理，因此，jealous 和 jealousy 在词干还原之后可能会被合并成一个词项，从而构成向量空间的一维。我们将会在第 18 章看到，采用上述矩阵来表示文档集的方法非常有用。

6.3.2 查询向量

将文档表示成向量的一个令人信服的理由是也可以将查询表示成向量。考虑查询 q=jealous gossip，那么可以将查询转化为单位向量 $\vec{v}(q)=(0,0.707,0.707)$，其中每一维所对应的词项与图 6-12 及图 6-13 保持一致。现在，最关键的思路是按照和 q 的内积计算结果对每篇文档 d 进行评分：

$$\vec{v}(q)\cdot\vec{v}(d)\text{。}$$

在图 6-13 所示的例子中，对上述查询而言，文档 *Wuthering Heights* 会得到最高分 0.509；而文档 *Pride and Prejudice* 则排名第二，得分是 0.085；文档 *Sense and Sensibility* 以 0.074 分排名垫底。上述查询的例子可能在某种程度上会给人造成误解，因为实际当中向量的维数远远超过 3，它应该等于词汇表的大小 M。

概括来说，只要将查询看成词袋，那么就能将它当成一篇极短的文档来处理。因此，可以通过计算给定的查询向量和每个文档向量的相似度来对所有文档进行排名，最终的结果可以用于选择排名靠前的一些文档。于是，我们有

$$score(q,d)=\frac{\vec{V}(q)\cdot\vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}\text{。}\qquad(6\text{-}12)$$

即使不包含所有的查询词项，一篇文档也可能会获得较高的余弦得分[①]。需要提醒的是，刚才的讨论中并没有涉及到查询向量[②]中词项的具体权重计算方式，实际上我们可以想象成这其中采用了tf或tf-idf权重。当然，对于查询向量来说，它和文档向量一样也可以有很多种权重计算方式（参见例 6-4），关于这一点我们将在 6.4 节进一步讨论。

这样的话，整个检索过程就是：计算查询向量和文档集中每个文档向量的余弦相似度，结果按照得分排序，并选择得分最高的 K 篇文档。这个过程的过程代价很大，这是因为每次相似

① 比如文档虽然只含有部分词项，但是这些词项的权重较大。——译者注

② 此处应为查询向量，原文误为文档向量。——译者注

度计算都是数万维向量之间的内积计算，这需要数万次的算术操作。7.1 节将讨论如何利用倒排索引完成上述检索过程，并讨论一系列提高检索性能的启发式策略。

例 6-4　考虑一个假想的文档集，其中 N=1 000 000，词项 auto、best、car 及 insurance 的文档频率分别是 5 000、50 000、10 000 及 1 000。考虑查询 best car insurance。

词项	查询				文档			内积
	tf	**df**	**idf**	**$w_{t,q}$**	**tf**	**wf**	**$w_{t,q}$**	
auto	**0**	**5000**	**2.3**	**0**	**1**	**1**	**0.41**	**0**
best	**1**	**50000**	**1.3**	**1.3**	**0**	**0**	**0**	**0**
car	**1**	**10000**	**2.0**	**2.0**	**1**	**1**	**0.41**	**0.82**
insurance	**1**	**1000**	**3.0**	**3.0**	**2**	**2**	**0.82**	**2.46**

上例中，查询中词项的权重采用 idf 表示，如果词项在查询中不存在（如 auto），则其权重置为 0。$w_{t,q}$ 对应的那一列给出了查询中词项的权重。而对于文档，则采用 tf 权重计算方式（不含 idf）并做欧氏归一化处理。归一化前、归一化后的权重分别列在 wf 及 $w_{t,d}$ 所在的列中。基于上述结果，可以采用公式（6-12）对查询和文档的相似度进行计算，最后得到的结果为 0 + 0 + 0.82 + 2.46 = 3.28。

6.3.3　向量相似度计算

通常情况下，一个典型检索系统的配置包括：一批文档组成的文档集，其中每篇文档表示成一个向量；一个自由文本查询，也表示成一个向量；正整数 K。检索系统的目标是，给定查询，从文档集合中返回得分最高的 K 篇文档。下面将讨论，在给定查询的情况下如何根据向量的相似度来选择得分最高的 K 篇文档。通常系统会按照降序给出得分最高的 K 篇文档，比如在很多搜索引擎中 K=10，即返回第一页的 10 篇得分最高的文档。这里先给出一个基本算法，第 7 章将就如何提高该算法的效率进行进一步的讨论。

图 6-14 给出了计算向量相似度的一个基本算法，每一行代表算法的一步。Length 数组中存放的是每个文档向量的长度（即归一化因子），而 Scores 数组放的是每篇文档的得分。在第 9 步计算出所有文档的得分以后，第 10 步则从中选出排名最高的 K 篇文档。

```
CosineScore(q)
1  float Scores[N] = 0
2  Initialize Length[N]
3  for each query term t
4  do calculate w_{t,q} and fetch postings list for t
5     for each pair(d, tf_{t,d}) in postings list
6     do Scores[d] += wf_{t,d} × w_{t,q}
7  Read the array Length[d]
8  for each d
9  do Scores[d] = Scores[d]/Length[d]
10 return Top K components of Scores[]
```

图 6-14 向量相似度的基本算法

第 3 步是外循环，即对于每个查询词项t，依次反复更新Scores数组。第 4 步，计算t在查询向量中的权重，第 5～6 步，根据t的贡献更新每篇文档的得分。这种根据每个t的贡献对文档得分进行累加的方法有时也称为以词项为单位(term-at-a-time)的评分或累加方法，因此数组Scores的N个元素也称为累加器（accumulator）。为了达到这个目的，看上去似乎很有必要存储每个倒排记录中的权重值，即t在d中的权重$w_{t,d}$（到现在为止，我们讨论的权重计算方法是tf或者tf-idf，其他权重计算方法留到 6.4 节再讨论）。实际上，因为存储权重可能需要浮点数，这会造成空间的浪费。有两种方法可以用来缓解这种空间上可能带来的问题。第一，如果使用逆倒排文档频率的话，则不必事先计算出idf_t的值，而只需要将N/df_t存储在t对应的倒排记录表的头部就已经足够；第二，在每个倒排记录中存储词项频率$tf_{t,d}$[①]。最后，第 10 步是抽取最高的K个得分，这需要一个优先队列数据结构，往往通过堆来实现。这样一个堆结构，建立时不会超过 $2N$次比较，每个排名在前K个之内的分数能通过$O(\log_2 N)$次比较得到。

需要指出的是，图 6-14 只给出了一个通用的算法，并没有给出对于不同查询词项的倒排记录表进行遍历的具体实现。我们可以像第 3 步开始的循环那样一次访问一个查询词项，也可以像图 1-6 那样实现并发访问。在并发访问的情况下，每次计算一篇文档的得分，因此这种方式有时也被称为以文档为单位（document-at-a-time）的评分方法。7.1.5 节中将对该方法做进一步介绍。

习题 **6-14** 如果在建立向量空间之前，将 jealous 和 jealousy 还原为同一词干，请叙述如何对 tf 及 idf 值进行修改。

习题 **6-15** 回到习题 6-10 中的 tf-idf 权重计算，试计算采用欧氏归一化方式处理后的文档向量，其中每个向量有 4 维，每维对应一个词项。

习题 **6-16** 请验证习题 6-15 中的向量的分量平方和等于 1（在舍入误差允许的范围内）。为什么？

习题 **6-17** 基于习题 6-15 的词项权重计算结果，对于查询 car insurance 计算 3 篇文档的得分并进行排序。计算时，查询词项的权重计算分别采用如下方法：

1. 查询中出现的词项权重为 1，否则为 0；

① 这样每个 $tf_{t,d}$ 采用整数存储，而倒排记录表的头部只需保存一个浮点数。因此，可以节省空间。——译者注

2. 采用欧氏方式对 idf 进行归一化。

6.4　其他 tf-idf 权重计算方法

对每篇文档的每个词项赋予一个权重，除原始的 tf 及 tf-idf 之外还有很多其他方法。本节只讨论一些主要的其他方法，更全面的方法介绍参见第 11 章。6.4.3 节对本节讨论的方法进行了小结。

6.4.1　tf 的亚线性尺度变换方法

显而易见，即使一个词项在文档中出现了 20 次，它所携带信息的重要性也不可能是只出现 1 次的词项的 20 倍。因此，有很多研究对原始的词项频率进行修改，而不仅仅是计算词项出现的次数。一个常用的修改方法是采用原始词项频率的对数函数，此时的权重计算方法如下：

$$wf_{t,d}=\begin{cases}1+\log tf_{t,d}, & tf_{t,d}>0,\\ 0, & \text{其他。}\end{cases} \tag{6-13}$$

上述方式下，tf 就可以用类似公式(6-13)中的 wf 来替代，于是可以得到

$$\text{wf-idf}_{t,d}=\text{wf}_{t,d}\times\text{idf}_t。 \tag{6-14}$$

公式(6-9)中的 tf-idf 可以用公式(6-14)中定义的 wf-idf 来代替。

6.4.2　基于最大值的 tf 归一化

另一种被充分研究的 tf 权重归一化方法是，采用文档中最大的词项频率对所有词项的频率进行归一化。对每篇文档 d，假定 $\text{tf}_{\max}(d)=\max_{\tau\in d}\text{tf}_{\tau,d}$，其中 τ 可以是 d 中的任一词项。于是，可以对文档中的每个词项 t 计算归一化后的 tf，计算公式如下：

$$\text{ntf}_{t,d}=a+(1-a)\frac{\text{tf}_{t,d}}{\text{tf}_{\max}(d)}\ , \tag{6-15}$$

其中，阻尼系数 a 是一个 0 到 1 之间的数，通常取 0.4，而在一些早期的工作中使用的是 0.5。a 主要起平滑（smoothing）作用，它在公式（6-15）中主要是抑制后一部分的贡献，而后一部分可以看成是通过 d 中最大的 tf 值来对词项的 tf 进行缩减的结果。在第 13 章介绍分类时，我们还会遇到平滑问题。这里进行平滑的基本思路是当 $\text{tf}_{t,d}$ 适度变化时（如从 1 变到 2），使用平滑技术来保证因其引起的 $\text{ntf}_{t,d}$ 的波动不会非常剧烈。我们发现，由于长文档中词项反复出现的可能性大，所以长文档中的词项频率倾向于取更大的值。这显然是不公平的，而最大 tf 归一化方法的主要思路就是减轻这种不公平所带来的影响。为了说明这一点，我们看一个例子。假定将文档 d 复制一份并和 d 合成文档 d'，那么很显然，尽管对任何查询而言，文档 d' 都不比文档 d 的相似度要大，但是，如果使用公式(6-9)来计算，那么 d' 的得分将会是 d 的得分的 2 倍。而此时如果将公式(6-9)中的 $\text{tf}_{t,d}$ 替换成公式(6-15)的中 $\text{ntf}_{t,d}$，则不会出现这样异常的结果。基于最大值

的 tf 归一化方法使用时会受以下情况的影响。

1. 停用词表的变化将引起词项权重的显著变化，从而也造成文档排序的显著变化。从这个意义上说，该方法不稳定。因此，这种方法很难调节。

2. 某篇文档可能包含一个异常词项，它的出现次数非常多，但是它并不代表文档的内容。

3. 更一般的情况是，如果一篇文档包含的最高频词项和其他词项出现次数相差不大，即词项的分布比较均衡，那么该文档应该和那些词项分布不均衡的文档区别对待。

6.4.3 文档权重和查询权重机制

公式(6-12)是基于任何向量空间评分方法的 IR 系统的基本计算公式。向量空间中各种相似度评分方法的不同，主要在于向量 $\vec{V}(d)$ 和 $\vec{V}(q)$ 中的权重计算机制的不同。图 6-15 中列出一些目前使用的有关 $\vec{V}(d)$ 和 $\vec{V}(q)$ 的主要权重计算方法，其中对于每种不同的权重计算方法采用了不同标记，这种标记系统有时也称为 SMART 记号，来源于一个早期的信息检索系统 SMART。这种标记中，文档向量和查询向量权重计算方法的组合字母表示为 *ddd.qqq*，前 3 位字母代表文档向量的权重计算方法，而后 3 位字母代表查询向量的权重计算方法。每个 3 位字母组合中的第 1 位字母表示权重计算中的 tf 因子，第 2 位表示 df 因子，第 3 位表示归一化形式。在实际中，对查询向量和文档向量分别采用不同的归一化函数是很常见的事情。比如：一个普遍使用的权重计算机制是 lnc.ltc，这表明文档向量采用了对数 tf 计算方法、没有采用 idf 因子（同时基于效率和效果的考虑）及余弦归一化方法，而此时查询向量则采用了对数 tf 计算方法、idf 权重因子及余弦归一化方法。

词项频率tf	文档频率df	归一化方法
n(natural) $\mathrm{tf}_{t,d}$	n(no) 1	n(none) 1
l(logarithm) $1+\log(\mathrm{tf}_{t,d})$	t(idf) $\log\frac{N}{\mathrm{df}_t}$	c(cosine) $\frac{1}{\sqrt{w_1^2+w_2^2+\cdots+w_M^2}}$
a(auamented) $0.5+\frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p(prob idf) $\max\left\{0,\log\frac{N-\mathrm{df}_t}{\mathrm{df}_t}\right\}$	u(pivoted unique) $1/u$(Section 17.4.4)
b(boolean) $\begin{cases}1 \text{ if } \mathrm{tf}_{t,d}>0\\0 \text{ otherwise}\end{cases}$		b(byte size) $1/CharLength^a, a<1$
L(log ave) $\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$		

图 6-15 不同 tf-idf 方法的 SMART 系统记号，其中 CharLength 指的是以字节为单位的文档长度

6.4.4 文档长度的回转归一化

在 6.3.1 节中，我们提到可以基于欧氏长度将每个文档向量归一化成单位向量，这样做会丢失原始的文档长度信息，也可能会隐藏长文档的一些细微性质：第一，由于长文档包含更多的词项数目，因此长文档中词项的频率 tf 可能更高；第二，长文档可能包含更多的不同词项，即

词汇量可能更大。这些因素会提高长文档的评分结果，这至少对某些信息需求来说是很不正常的。长文档可以大致归成两类：第一类是那些同一内容反复出现的冗余性文档，文档的长度不会改变词项的相对权重；第二种是那些包含多个不同主题的文档，查询词项可能只能和文档的部分内容相匹配。第二种文档中词项的相对权重，就会和一篇与查询相匹配的短文档迥然不同。可以采用一种与查询频率及文档频率都无关的文档长度归一化方法对上述现象进行修正。为此，下面我们引入这种对文档集中的文档向量进行长度归一化的方法，这种方法并不要求归一化后的结果文档都必须是单位长度。然后，当计算查询单位向量和上述归一化后的文档向量的内积时，评分方法能够合理体现文档长度对相关性的影响。这种对文档长度的修正方法称为回转文档长度归一化（pivoted document length normalization）。

考虑一个文档集以及一系列查询构成的查询集。假定对于每个查询 q 和每篇文档 d，都事先给定了 q 和 d 的相关性判断结果，这个结果是个布尔量，分别表示相关或不相关。第 8 章会详细介绍如何构造这个相关性判断集合，这里就先不赘述。给定相关性判断之后，可以将相关概率（probability of relevance）看成文档长度的函数，并且在所有查询上进行平均求得结果。然后将结果描成一条曲线（该曲线看上去像图 6-16 中那条较粗的曲线）。具体地，为了计算出这条曲线，我们将文档按照长度分组，然后计算每个组相关文档的比例，最后将该组文档长度的中值作为横坐标，刚才得到的比例值作为纵坐标在平面上描点。因此，尽管图 6-16 中的曲线看上去是连续的，但实际上是将文档按照长度进行分组得到的离散分组的直方图。

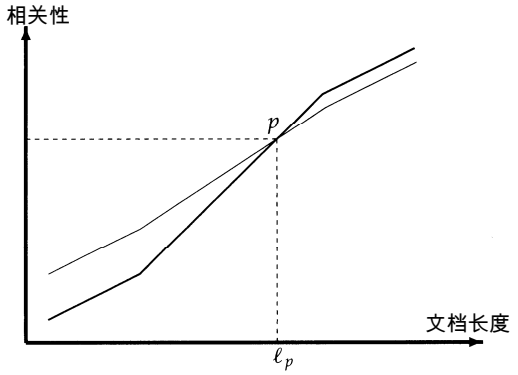


图 6-16　回转文档长度归一化

另一方面，在同样的文档集和查询集上，我们采用公式（6-12）中余弦归一化公式计算得到相关度，将该结果描出来之后便得到图 6-16 中细线对应的曲线。因此，余弦归一化得到的计算结果的相关度和真实的相关度之间存在着差异。两条曲线相交于点 p，该点对应的文档长度记为 l_p，称为回转长度（pivot length）。图中虚线标出了该点对应的横坐标和纵坐标。回转文档长度归一化的思路是，将余弦归一化结果曲线以 p 点为轴逆时针旋转，使之能够和真实的基于文档长度的相关度曲线高度吻合。正如本节一开始所提到的那样，我们可以通过对公式（6-12）中的每篇文档的向量 $\vec{V}(d)$ 引入一个归一化因子来达到我们的目标，这个归一化因子不是向量的

欧氏长度，而是另外一种长度计算方法，它在文档长度小于 l_p 的时候取值大于欧氏长度，而在文档长度大于 l_p 的时候取值却小于欧氏长度。

为了达到上述目的，首先我们注意到公式(6-12)的分母中 $\vec{V}(d)$ 的归一化因子是欧氏长度，记为 $|\vec{V}(d)|$。最简单的回转长度归一化方法是在分母中使用 $|\vec{V}(d)|$ 的斜率小于 1 的线性函数(如图 6-17 所示)。图 6-17 中，横轴表示 $|\vec{V}(d)|$，纵轴表示可能的归一化因子。曲线 $y=x$ （图中细线）表示使用的是余弦归一化，而粗线表示的是回转文档长度归一化。这里需要注意以下几个方面。

(1) 它是文档长度的线性函数，形式为：

$$a|\vec{V}(d)|+(1-a)\text{piv} \quad , \tag{6-16}$$

其中，piv 是两条曲线交点的余弦归一化因子值。

(2) 该直线斜率 $a<1$。

(3) 它和直线 $y=x$ 在 piv 处相交。

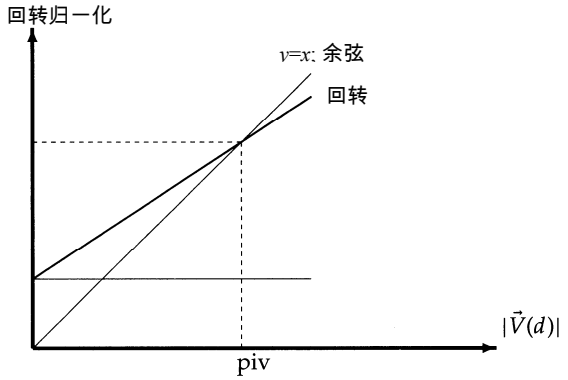


图 6-17 线性回转长度归一化方法的实现

有人认为，在实际当中公式(6-16)可以通过下式来很好地近似：

$$au_d+(1-a)\text{piv} \quad ,$$

其中，u_d 是文档 d 中不同的词项的数目（即词汇量）。

当然，回转长度归一化方法并不适合于任何应用。比如，一个由很多 FAQ 的答案组成的文档集（如客户服务网站）中，相关性可能基本上和文档长度无关。而其他情况下，这种依赖关系可能又过于复杂，不可能通过简单的线性回转归一化方式来实现。此时，文档长度可以作为一个特征在基于机器学习的评分方法中使用（参见 6.1.2 节）。

? 习题 **6-18** 另一种计算向量相似度的方法称为欧几里得距离（Euclidean distance，或称 L_2 距离或欧氏距离），计算公式如下：

$$|\vec{x}-\vec{y}|=\sqrt{\sum_{i=1}^{M}(x_i-y_i)^2} \quad 。$$

给定查询 q 和文档集 $d_1,d_2,\ldots$，可以按照它们和查询之间的欧氏距离输出有序的结果。假定 q 和 d_i 都归一化成单位向量，请证明通过欧氏距离计算出的文档排名和使用余弦相似度所得的结果完全一致。

习题 **6-19**　计算查询 digital cameras 及文档 digital cameras and video cameras 的向量空间相似度并将结果填入表 6-1 的空列中。假定 N=10 000 000，对查询及文档中的词项权重（wf 对应的列）采用对数方法计算，查询的权重计算采用 idf，而文档归一化采用余弦相似度计算。将 and 看成是停用词。请在 tf 列中给出词项的出现频率，并计算出最后的相似度结果。

表6-1　习题6-19中的余弦相似度计算

词	查　询					文　档				$q_i \cdot d_i$
	tf	wf	df	idf	q_i=wf-idf	tf	wf	d_i=归一化的wf	wf	
digital			10 000							
video			100 000							
camera			50 000							

习题 **6-20**　请说明对于查询 affection，图 6-13 中的 3 篇文档的得分顺序正好和查询 jealous gossip 的结果的次序相反。

习题 **6-21**　在将图 6-13 的查询转变成单位向量时，我们给查询中的每个查询词项赋予了相等的权重，有没有其他理论上可行的方法？

习题 **6-22**　考虑有一个查询词项不包含在 M 个索引词项中的情况，因此，一般情况下，构建出的查询向量 $\vec{V}(q)$ 和文档集并不在同一空间。如何调整向量空间表示来对该情况进行处理？

习题 **6-23**　考虑习题 6-10 中 4 个词项和 3 篇文档中的 tf 和 idf 值，采用如下权重计算机制来计算获得得分最高的两篇文档：(i) nnn.atc ；(ii) ntc.atc。

习题 **6-24**　假定词 coyote 在习题 6-10 及习题 6-23 中的文档集中没有出现，那么如何采用 ntc.atc 权重计算方法来计算文档和查询 coyote insurance 的相似度？

6.5　参考文献及补充读物

第 7 章将介绍向量空间模型实现时的计算开销问题。Luhn（1957, 1958）介绍了有关词项权重应用的一些最早报道。他的论文详细讨论了中频词项（即出现既不特别频繁也不特别稀少的词项）的重要性，可以认为这项工作为后来的 tf-idf 及相关权重计算方法奠定了基础。Spärck Jones（1972）通过细致的实验给出了 idf 的使用方式，也建立了上述直观的结果。对 idf 的一系列扩展和理论分析参见 Salton 和 Buckley（1987）、Robertson 和 Jones（1976）、Croft 和 Harper（1979）及 Papineni（2001）。Robertson 维护着一个包含 idf 发展史的网页（www.soi.city.ac.uk/~ser/idf.html），其中包含了一些电子版本出现之前期刊文章的软拷贝。Singhal 等人（1996a）提出了回转长度归一化方法。第 11 章的概率语言模型中发展的权重计算方法与 tf-idf 相比有更细微的差别，读者将会在 11.4.3 节找到相关细节。

我们观察到，对一篇文档的每个词项赋予一个权重后，文档就可以看成词项权重的向量。第一次将文档看成权重向量或许始于 Salton 及其同事在康奈尔大学开发的 SMART 系统（ Salton，1971b ）中。6.3.3 节介绍的基本余弦相似度计算方法归功于 Zobel 和 Moffat （ 2006 ）。有关以词项为单位和以文档为单位的两类查询处理策略的讨论参见 Turtle 和 Flood （ 1995 ）。

图 6-15 中 SMART 系统的 tf-idf 权重计算机制的标记方法出现在（ Salton 和 Buckley，1988；Singhal 等人，1995, 1996b ）中。并非所有的标记的版本都一致，本书中主要参考的是（ Singhal 等人 1996b ）中的标记体系。Moffat 和 Zobel（ 1998 ）提出了一个更详尽的标记体系，其中考虑各种词项频率和文档频率权重的计算方法。除了标记体系外，Moffat 和 Zobel（ 1998 ）还试图建立一个可行的权重函数空间，一开始给出一个较好的权重计算函数，然后通过爬山法（ hill-climbing method ）来求解，最后通过局部点上的提高来获得最佳的组合方式。然而，他们报告说，这些爬山方法并不能得到任何有关最佳权重机制的结论。

第 7 章

一个完整搜索系统中的评分计算

第 6 章给出了文档评分中词项权重计算的理论，并由此导出了向量空间模型和基本的余弦相似度评分算法（参见 6.3.3 节）。本章一开始将在 7.1 节中介绍一些启发式策略，这些策略能够加快评分算法的实现速度，当然其中的不少策略有可能不会精确返回与查询相匹配的前 K 篇文档，一些策略也可以推广到余弦相似度计算之外的其他场合中去。在 7.1 节中，我们实际上已经拥有了一个搜索引擎所需要的全部部件。因此我们回过头来考虑余弦相似度评分计算问题，并将之过渡到搜索引擎中文档评分这个更具一般性的问题。在 7.2 节中，我们将给出一个搜索引擎的完整描述，包括既支持余弦相似度计算也支持其他一般评分因子（如查询词项之间的邻近度）的索引和数据结构。7.2.4 节介绍如何将不同部件组成一个完整的搜索系统。7.3 节中给出本章的结论，并讨论能否让支持自由文本查询的向量模型来支持常用的查询操作符。

7.1 快速评分及排序

首先我们回顾图 6-4 所给出的算法。对于查询 q = jealous gossip，能够直接观察到如下两种现象：

1. 单位向量 $\vec{v}(q)$ 只有 2 个非零分量；

2. 不考虑查询词项的任何权重机制时，$\vec{v}(q)$ 中的 2 个非零分量相等，在这里都等于 0.707。

给定查询时要实现文档排序的目标，实际上只需要得到文档之间的相对而不是绝对得分。因此，只需要计算 $\vec{v}(d)$ 和 $\vec{V}(q)$（和单位向量 $\vec{v}(q)$ 不同，$\vec{V}(q)$ 是一个布尔向量，其分量值非 0 则 1）的余弦相似度即可。对于任意两篇文档 d_1、d_2，显然有

$$\vec{V}(q)\cdot\vec{v}(d_1)>\vec{V}(q)\cdot\vec{v}(d_2)\Leftrightarrow\vec{v}(q)\cdot\vec{v}(d_1)>\vec{v}(q)\cdot\vec{v}(d_2)。\qquad(7\text{-}1)$$

这样，对于任一文档 d，计算余弦相似度 $\vec{V}(q)\cdot\vec{v}(d)$ 都相当于对 q 中所有的查询词项，计算它们在文档 d 的权重之和。该过程实际上可以通过图 6-14 中算法中的倒排记录表合并操作来完成，不同的是将原算法第 7 行中的 $w_{t,q}$ 设为 1。因此，原算法中的先乘后加操作在这里仅需要加法操作。图 7-1 给出了新算法的实现过程。在倒排索引中，遍历 q 中所有查询词项对应的倒排记录表，并将每篇文档上的得分进行累加。这和布尔查询的处理方法基本一致，所不同的是，这里对每条倒排记录对应的文档都会赋予一个正分。正如 6.3.3 节所提到的那样，对于词典中的每个词，我们都保存其 idf 值，而每条倒排记录中都会保存词项在当前文档中的 tf 值。这种机制会对任一查询词项对应的倒排记录表中的每篇文档进行评分，而这类文档的总数可能会远远小于 N。

计算出满足条件文档的得分以后，最后一步就是选出得分最高的 K 篇文档呈现给用户。尽管可以先对上述所有得分进行排序然后再挑选出前 K 个结果，但是一个更好的方法是通过某种堆结构只返回头 K 篇文档。假定余弦相似度得分非零的文档数目是 J，那么建立这样的堆结构需要 $2J$ 次比较，对于排名前 K 的每篇文档从上述堆结构中返回时都需要进行 $\log J$ 次比较。

```
FastCosineScore(q)
 1  float Scores[N] = 0
 2  for each d
 3  do Initialize Length[d] to the length of doc d
 4  for each query term t
 5  do calculate w_{t,q} and fetch postings list for t
 6     for each pair(d, tf_{t,d}) in postings list
 7     do add wf_{t,d} to Scores[d]
 8  Read the array Length[d]
 9  for each d
10  do Divide Scores[d] by Length[d]
11  return Top K components of Scores[]
```

图 7-1　一个向量空间评分的快速实现算法

7.1.1　非精确返回前 K 篇文档的方法

迄今为止，我们主要关注给定查询后精确返回前 K 篇得分最高的文档的方法。本节主要考察产生“ 可能” 排名最高的 K 篇文档的方法。这样做的目的在于，显著降低输出前 K 篇文档所需要的计算复杂度，同时并不让用户感觉到前 K 个结果的相关度有所降低。由于采用这种机制能够返回和真实的前 K 篇文档得分非常接近的 K 篇文档，因此在大多数应用中，这样做能够充分满足应用的需要。接下来，我们将详细介绍非精确返回前 K 篇文档的方法，我们会尽量避免对文档集中的大多数文档进行评分计算。

另一方面，从用户角度看，这种非精确的前 K 篇文档检索并不一定是坏事。给定查询情况下，余弦相似度计算得分最高的 K 篇文档在很多情况下不一定就是最好的 K 篇文档，余弦相似度只不过是用户所感觉到的相似度的一个替代品。从 7.1.2 节一直到 7.1.6 节，我们将介绍可能返回与前 K 篇文档得分接近的 K 篇文档的一系列启发式策略。计算前 K 篇得分最高文档的主要计算开销源于大量文档都参与的余弦相似度计算，这同样也会增加从堆中选择最后 K 篇文档所需要的计算开销。下面介绍减少参与计算的文档数目的一些方法，它们主要包括如下两个步骤。

(1) 找到一个文档集合 A，它包含了参与最后竞争的候选文档，其中 $K < |A| << N$。A 不必包含前 K 篇得分最高的文档，但是它应该包含很多和前 K 篇文档得分相近的文档。

(2) 返回 A 中得分最高的 K 篇文档。

从上述两步的描述中不难理解，很显然上述方法需要针对当前文档集和应用来调节相关参数。至于如何设定这些经验性参数，则可以参考本章的最后部分。需要指出的是，这些启发式方法中的大部分都非常适合于自由文本查询，但是并不适合于布尔查询或短语查询。

7.1.2 索引去除技术

对于一个包含多个词项的查询来说，很显然我们可以仅仅考虑那些至少包含一个查询词项的文档，于是可以进一步考虑使用如下的启发式方法。

(1) 只考虑那些词项的 idf 值超过一定阈值的文档。因此，在进行倒排索引遍历时，我们仅考虑那些 idf 值较高的词项所对应的倒排记录表。这样做具有非常显著的好处：那些低 idf 值词项的倒排记录表往往比较长，如果将它们剔除，那么需要计算余弦相似度的文档数目将大大减少。从另一个角度看，idf 值低的词项也可以看成停用词，它们对评分结果没有什么贡献。例如，查询 catcher in the rye 中，只有 catcher 和 rye 的倒排记录表才会被遍历。显然，阈值可以根据不同的查询来调节取值。

(2) 只考虑包含多个查询词项（一个特例是包含全部查询词项）的文档。这可以在倒排记录表遍历过程中实现，我们仅考虑对那些包含较多（或全部）查询词项的文档进行计算。这种方法存在的一个风险是，如果仅仅对这些文档进行相似度计算，那么很有可能最后的候选结果文档数目少于 K 个。关于这个问题的进一步讨论参见 7.2.1 节。

7.1.3 胜者表

胜者表（champion list），有时也称为优胜表（fancy list）或高分文档（top doc），它的基本思路是，对于词典中的每个词项 t，预先计算出 r 个最高权重的文档，其中 r 的值需要事先给定。对于 tf-idf 权重计算机制而言，词项 t 所对应的 tf 值最高的 r 篇文档构成 t 的胜者表。

给定查询 q，对查询 q 中所有词项的胜者表求并集，并可以生成集合 A。只有 A 中的文档才可以参加最后的余弦相似度运算。这种机制中的一个关键参数是 r，它与应用高度相关。直观上说，r 应该比 K 大，尤其是在使用 7.1.2 中提到的任一索引去除技术时更要如此。一个可能的问题是，r 的值往往在索引构建时就已经设定，而取决于应用本身的 K 值往往要到接收到查询后才确定。因此，这种情况下（在索引去除的时候也可能会遇上这种情况）得到的集合 A 中的元素个数可能会少于 K。另外，没有必要将所有词项的 r 设成一样的值，比如对于罕见词项，r 值可以适当设大一些。

7.1.4 静态得分和排序

这节中我们将提出一个在某种程度上说更一般的概念：静态质量得分（static quality score，简称静态得分），通过它可以进一步扩展胜者表的思路。很多搜索引擎中，每篇文档 d 往往都有一个与查询无关的静态得分 $g(d)$。该得分函数的取值往往在 0 到 1 之间。比如，对于 Web 上的新闻报道，$g(d)$ 可以基于用户的正面评价次数来定义。4.6 节已经对该话题进行了深入的讨论，而第 21 章会在 Web 搜索这个大背景下再次讨论这个话题。

这样，一篇文档 d 的最后得分就可以定义为 $g(d)$ 和某个与查询相关的得分（可以参考公

式 6-12) 的某种组合。精确的组合方式可能需要机器学习 (参见 6.1.2 节) 方法，进一步的讨论参见 15.4.1 节。这里为了说明的需要，仅仅考虑简单的求和式组合方法。

$$\text{net-score}(q,d)=g(d)+\frac{\vec{V}(q)\cdot\vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}。\qquad (7\text{-}2)$$

在这种简单形式中 ,静态得分 $g(d)$和利用公式(6-10)计算出的与查询相关的得分的取值都在 0 到 1 之间，并且它们具有同等的作用。当然，也可以有其他的权重因子，上述启发式方法的有效性依赖于具体的相对权重因子。

首先，将所有文档按照其静态得分 $g(d)$在倒排记录表中降序排列，然后利用图 1-6 所示的算法进行倒排记录表的合并操作。为了只对每个查询词项的倒排记录表进行一次扫描，图 1-6 中的算法要求倒排记录表已经按照文档 ID 排序。然而 ,实际上我们只要求所有的倒排记录表按照一种相同的方式排序即可。这里 , 我们按照 $g(d)$的大小进行统一排序。图 7-2 给出了按照 $g(d)$将文档进行降序排列的一段倒排记录表的例子。

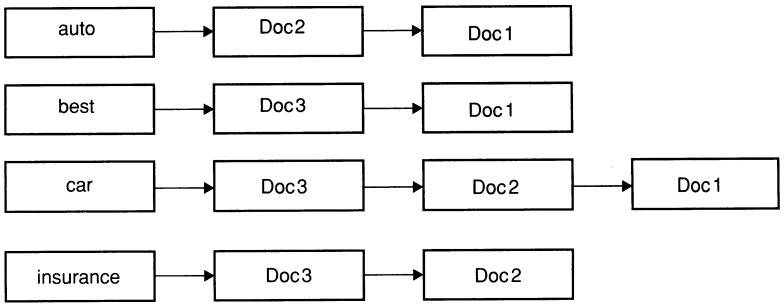


图 7-2　按照静态得分排序的索引结构，其中假定 Doc1、Doc2、Doc3 的静态得分分别是 $g(1) = 0.25$、 $g(2) = 0.5$ 及 $g(3) = 1$

下面，我们将对胜者表进行进一步扩展：对于精心选择好的 r 值，对每个词项 t 构建一个全局胜者表 (global champion list) , 其中包含了 $g(d)$+tf-idf$_{t,d}$ 得分最高的 r 篇文档。胜者表本身则像所有的倒排记录表一样，都采用统一的排序方式 (使用文档 ID 或者是静态得分值)。于是，当查询提交以后，只需要利用公式(7-2)对所有全局胜者表的并集中的文档计算其最后得分。直观上说，上述方法实际上只集中关注那些可能具有很高最终得分的文档。

接下来我们来介绍另外一种思路，并以此来结束有关全局胜者表的讨论。对每个词项 t，我们维持两个无交集的倒排文件表，每个表均以 $g(d)$值来排序。第一张表，我们称为高端表，由 m 篇具有最高 tf 值的文档组成；而第二张表，我们称为低端表，则由剩下的包含 t 的文档组成。在对查询进行处理时，首先对词项的高端表进行扫描，计算在所有或超过给定数目的查询词项的高端表中的每篇文档的最后得分值。如果该过程能够产生前 K 篇得分最高的文档，则处理结束。否则 ,需要继续扫描低端表并计算其中文档的得分。该方法将会在 7.2.1 节进一步讨论。

7.1.5　影响度排序

在上面介绍的所有倒排记录表中，文档通常采用同一种排序方式排序，典型的方法包括按文档 ID 或 7.1.4 节介绍的静态得分来排序。如同在 6.3.3 节结尾所提到的那样，这种统一排序方法能够支持多个查询词项倒排记录表的并发扫描，在遇到每一篇文档时能够计算它的得分。这种计算文档得分的方式有时也称为以文档为单位（document-at-a-time）的评分方法。现在我们要介绍的是，当所有倒排记录表并不完全使用统一排序方式时，如何非精确地计算出前 K 篇文档。这时我们就不能对多个倒排记录表进行并发扫描。此时，我们需要在遇到每个词项时得分能够逐渐累加（参见图 6-14），因此，我们得到以词项为单位（term-at-a-time）的得分计算方法。

具体的思路是，将词项 t 对应的所有文档 d 按照 $\mathrm{tf}_{t,d}$ 值降序排列。这样的话，不同词项倒排记录表中文档所采用的排序方式就不是统一的，于是就不能通过并发扫描多个倒排记录表的方式来计算文档的得分。给定按照 $\mathrm{tf}_{t,d}$ 值降序的倒排记录表，已经发现有两种思路可以显著降低用于累加得分的文档数目：(1) 对某个查询词项 t 对应的倒排记录表进行从前往后扫描时，可以在某个阶段停止，此时可能是扫描了 r 篇固定数目的文档，也有可能是当前记录的 $\mathrm{tf}_{t,d}$ 已经低于某个阈值；(2) 当在图 6-14 中所示的外循环累加文档得分时，词项按照 idf 降序排列，因此，对最终得分贡献最大的查询词项首先被考虑。后一种方法可以在查询处理过程中进行自适应处理，当遇到具有较低 idf 值的查询词项时，可以根据和前一个查询词项的文档得分的改变值来决定是否需要处理。如果改变值达到最小限度时，就可以忽略剩下的词项，或者只对它们的倒排记录表的更前面的记录进行处理。

上述思路给出 7.1.2 节到 7.1.4 节所有方法所共有的一个一般形式。我们也可以实现另外的静态得分情况下的倒排索引，其中每个倒排记录表按照静态得分及查询相关得分的相加值进行排序。这种情况下，显然不同倒排记录表中的文档排序方式也不尽一致。此时，就必须以查询词项为单位进行处理，然后对所有的文档计算累加得分。基于具体的评分计算方法，倒排记录表可以采用不同于 tf 的排序方式，这种更一般的设置情况下的排序方法被称为影响度排序（impact ordering）。

7.1.6 簇剪枝方法

在簇剪枝方法（cluster pruning）中，我们先对文档向量聚类来进行预处理操作，然后，在查询处理时，我们只考虑利用少数几个簇中的文档进行余弦相似度计算。具体的预处理步骤如下：

(1) 从 N 篇文档组成的文档集中随机选出 $\sqrt{N}$ 篇文档，它们称为先导者（leader）集合；

(2) 对于每篇不属于先导者集合的文档，计算离之最近的先导者。

不属于先导者集合的文档称为追随者（follower）。直观地看，对于 $\sqrt{N}$ 篇随机选出的先导者文档，其期望分配到的追随者个数大约为 $N/\sqrt{N}=\sqrt{N}$。于是，查询处理过程如下：

(1) 给定查询 q，通过与 $\sqrt{N}$ 个先导者计算余弦相似度，找出和它最近的先导者 L；

(2) 候选集合 A 包括 L 及其追随者，然后对 A 中的所有的文档计算余弦相似度。

聚类过程中随机选择先导者的方法速度要很快，并且要尽可能反映文档向量的分布：文档

向量密集分布的区域很可能会产生多个先导者，因此需要细分成多个子区域。该过程参见图 7-3。

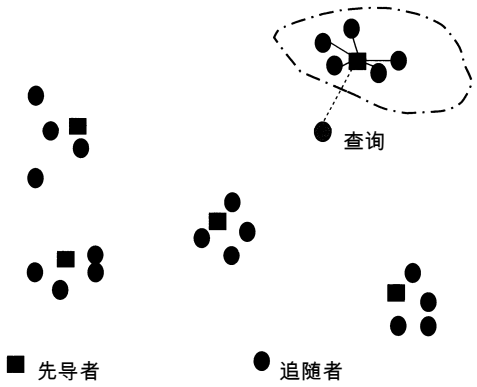


图 7-3　簇剪枝方法

在上述簇剪枝方法的一些变形当中，会另外引入两个参数 b_1 和 b_2，它们都是正整数。预处理时，我们将每个追随者分配给离它最近的 b_1 个先导者，而不是像上面一样只分配给一个最近的先导者。查询处理时，我们将考虑和查询 q 最近的 b_2 个先导者。很显然，前面讲到的方法只是该方法在 $b_1 = b_2 = 1$ 情况下的一个特例。进一步说，增加 b_1 和 b_2 会增加找到真实的前 K 篇文档的可能性，当然此时也会消耗更大的代价。在第 16 章讨论聚类算法时还会提到这种聚类方法。

习题 **7-1**　图 7-2 中倒排记录表均按照静态得分 $g(d)$的降序排列，为什么不采用升序排列？

习题 **7-2**　讨论胜者表时，我们只是简单地选择 r 篇 tf 值最高的文档来生成胜者表，但是，在考虑全局胜者表时，我们也使用了 idf，具体地是采用 $g(d)$+tf-idf$_{t,d}$ 进行排序，那么这两种情况有什么区别？

习题 **7-3**　给定单个词项组成的查询，请解释为什么采用全局胜者表（r=K）已经能够充分保证找到前 K 篇文档。如果只有 s 个词项组成的查询（s>1），如何对上述思路进行修正？

习题 **7-4**　请解释为什么高端表和低端表中均采用统一的 $g(d)$函数进行排序能够提高评分计算的效率。

习题 **7-5**　重新考察习题 6-23 中基于 nnn.atc 权重计算的数据，假定 Doc1 和 Doc2 的静态得分分别是 1 和 2。请确定在公式(7-2)下，如何对 Doc3 的静态得分进行取值，才能分别保证它能够成为查询 best car insurance 的排名第一、第二或第三的结果。

习题 **7-6**　画出图 6-9 中数据按照词项频率排序的倒排记录表。

习题 **7-7**　设定图 6-10 中 Doc1、Doc2 和 Doc3 的静态得分分别是 0.25、0.5 和 1，画出当使用静态得分与欧几里得归一化 tf 值求和结果进行排序的倒排记录表。

习题 **7-8**　平面上的最近邻问题如下：在平面上给出 N 个数据点并将它们预处理成某种数据结构，给定查询点 Q，在 N 个点中寻找与 Q 具有最短欧氏距离的点。很显然，如果我们希望能够避免计算 Q 和所有平面上的点的距离时，簇剪枝就能够作为最近邻问题的一种处理方法。请给出一个简单的例子来说明：如果只选择最近的两个先导者，那么簇剪枝方法可能会返回错误的结果（也就是说返回的不是离 Q 最近的数据点）。

7.2 信息检索系统的组成

本节将前面介绍的各种思路组合到一个基本的信息搜索系统中，该系统能够检索文档并对文档进行评分。首先，我们将进一步讨论除向量空间之外的评分方法；接着，我们将各种元素组合到一个完整的系统中。由于考虑的是一个完整的检索系统，我们并不局限于基于向量空间模型的系统。实际上，这个完整的检索系统不仅支持向量空间模型，也支持其他的查询操作符和检索形式。7.3 节中，我们回到向量空间模型下的查询，并讨论它们和其他查询操作符的结合方法。

7.2.1 层次型索引

7.1.2 节曾经提到，当利用诸如索引去除的启发式技术来进行非精确的前 K 篇文档检索时，可能会出现通过这些启发式方法得到的候选集合 A 中元素的个数比 K 小的情况。通常的解决方法是使用层次型索引（tiered index），这种结构可以看成是优胜表的一般化形式。图 7-4 给出了对应图 6-9 中文档和词项表示的层次型索引的一个例子。该例中，第 1 层索引中的 tf 阈值是 20，第 2 层阈值是 10。这意味着第 1 层索引只保留 tf 值超过 20 的倒排记录，而第 2 层的记录只保留 tf 值超过 10 的倒排记录。本例中每层的倒排记录表均按照文档 ID 排序。

7.2.2 查询词项的邻近性

对于检索中的查询，特别是Web上的自由文本查询来说（参见第 19 章），用户往往希望返回的文档中大部分或者全部查询词项之间的距离比较近，因为这表明返回文档中具有聚焦用户查询意图的文本。考虑一个由两个或者多个查询词项构成的查询$t_1, t_2, \ldots, t_k$。令文档d中包含所有查询词项的最小窗口大小为ω，其取值为窗口内词的个数。例如，假设某篇文档仅仅包含一个句子The quality of mercy is not strained，那么查询strained mercy 在此文档中的最小窗口大小是 4。直观上讲，ω的值越小，文档d和查询匹配程度更高。如果文档中不包含所有的查询词项，那么此时可以将ω设成一个非常大的数字。在计算时，还可以考虑各种可能的策略变化，比如在以单词个数来计算窗口宽度ω时，可以不考虑停用词的数目。这种基于邻近性（*proximity*）加权的评分函数已经和纯余弦相似度计算方法有所不同，而更接近于目前包括Google在内的很多搜索引擎所提供并明显在使用的“软合取”（soft conjunctive）[①]语义。

① 所谓“软合取”指的是在对一个包含多个词项的查询进行检索时，检索中的文档中只要出现大部分查询词项即可，并不要求出现全部查询词项。

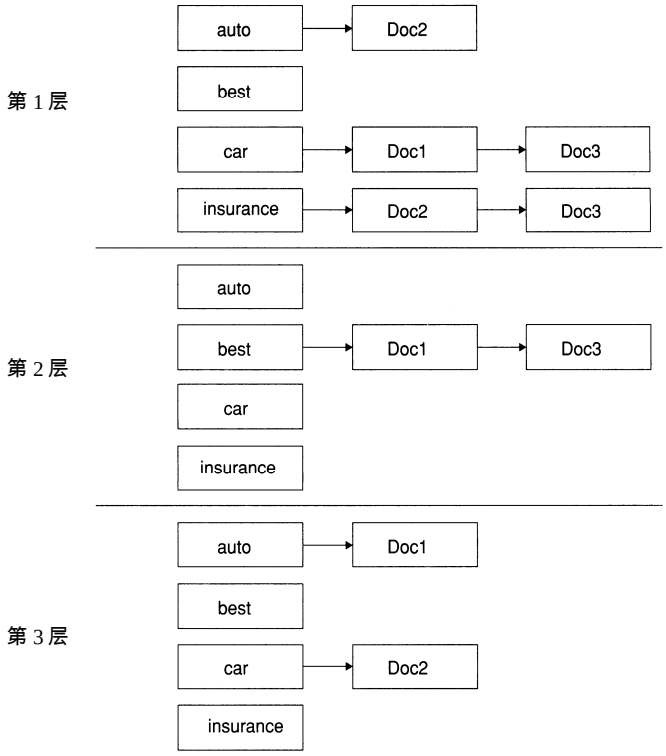


图 7-4　层次型索引结构。如果基于第 1 层索引返回的结果数小于 K，那么查询处理就会回退到第 2 层进行处理，如此循环下去。每层中，倒排记录表均按照文档 ID 排序

如何设计一个基于邻近关系程度 ω 的加权评分函数？最简单的方法依赖于 7.2.3 节将要介绍的一种称为手工编码（handing coding）的技术。一个更有扩展性的方法可以回到 6.1.2 节，在那里，整数 ω 可以看成评分函数的另一个特征并通过机器学习的方法得到其权重，15.4.1 节中将会进一步讨论这个问题。

7.2.3　查询分析及文档评分函数的设计

一般的搜索界面，特别是 Web 上面向消费者的搜索应用界面，都倾向于对最终用户屏蔽查询操作符。这样做的目的是在面向大量的非专业用户时能够隐藏查询操作符的复杂性，从而导致自由文本查询（free text query）的普遍应用。给定此类搜索界面的情况下，一个搜索系统拥有支持各种查询操作符处理的索引结构，此时它应该如何处理 rising interest rates 之类的查询？更一般地，给定我们前面介绍过的各种能够影响文档得分的因素，如何对这些因素进行组合？

上述问题的答案当然依赖于用户数量、查询分布及文档集本身。通常情况下，会有一个查询分析器（query parser）将用户输入的关键词转换成带操作符的查询，该查询能够基于底层的索引结构进行处理。有时，这种处理过程可能需要基于底层索引结果对多个查询进行处理，比如，查询分析器可能会产生如下的一系列查询。

1. 将用户输入的查询字符串看成一个短语查询。利用向量空间模型求解，此时输入查询向量是以 rising interest rates 为基的 1 维向量。

2. 如果包含短语 rising interest rates 的文档数目少于 10 篇，那么会将原始查询看成 rising interest 和 interest rates 两个查询短语，同样通过向量空间方法来计算。

3. 如果结果仍然少于 10 个，那么重新利用向量空间模型求解，这时候认为 3 个查询词项之间是互相独立的。

上面的每一步在调用的情况下都会产生一系列带得分的文档列表结果，而每个得分必须融合向量空间计算、静态得分、邻近度加权或其他可能的因素，特别地，一篇文档可能在上述的多个步骤结果列表中同时出现。这时就要求有一个综合得分函数能够融合不同来源的得分。那么应该如何来设计查询分析器？又如何来设计综合评分函数呢？

答案取决于具体的应用要求。在企业级应用背景下，开发者能够利用评分工具集及查询分析层来手工配置查询分析和评分函数。这些应用的开发者能够利用域、元数据、代表文档及代表查询的知识来对查询分析和评分进行调节。在文档集特征变化不频繁的情况下（在企业级应用背景下，文档集和查询通常只在新文档格式或新文档管理系统的引入、企业兼并时才会发生显著变化，显然这种变化是不频繁的）可以采用上述做法。而与之相反的是 Web 搜索，其文档集会不断变化，而新特征一直也都在不断产生。这种情况下，评分因子可能会有数百个，这大大增加了手工调节打分的难度。为了解决这个问题，目前普遍的做法是通过机器学习方法来评分，这是对 6.1.2 节介绍的思路的扩展，关于这一点我们还要在 15.4.1 节中再次讨论。

7.2.4 搜索系统的组成

到目前为止，我们考察了一个能够支持自由文本查询、布尔查询、域查询及字段查询的基本搜索系统所需要的各个部件。下面我们将简单介绍如何将这些部件组装成一个整体的系统，整个过程参见图 7-5。

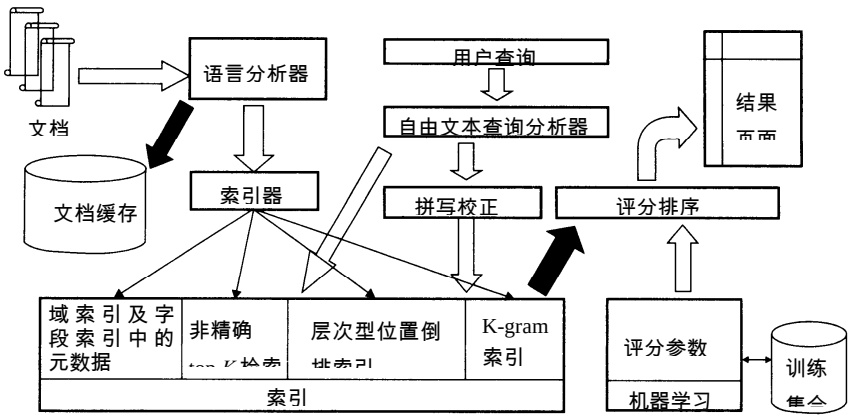


图 7-5 一个完整的搜索系统，其中主要给出了针对自由文本查询的数据传递路径

在图 7-5 中，文档不断从左边流入到分析和语言学处理模块（语种识别、格式识别、词条化及词干还原等等），处理所输出的结果词条序列又会输送给两个后续模块：首先，文档缓存中保留了每个分析过的文档，用于产生结果摘要片段（result snippet），即伴随每篇文档一起出现的文档摘要，它通过一段简洁的文本来说明文档和查询相关的原因。有关文档摘要的自动生成方法请参见 8.7 节的内容，这里不再介绍。所有词条的另一个副本输入则到一系列索引器中，从而产生一系列索引结构，包括能够存储文档元数据的域索引及字段索引、（层次型的）位置信息倒排索引、用于支持拼写校正与其它容错式检索的索引，以及产生非精确前 K 篇文档检索的索引结构等等。自由文本查询（图上部中间）输送给索引器，并通过一个模块产生拼写校正后的候选查询。正如第 3 章提到的那样，候选查询仅仅在原始查询返回的结果数目不足时才启用。返回的文档输送给评分模块（黑箭头），该模块通过机器学习的方法进行计算，该计算方法建立于 6.1.2 节并将在 15.4.1 节进一步讨论。最后，排序文档结果会通过结果页面呈现给用户。

? 习题 **7-9**　修改 1.3 节的倒排记录表合并算法，使之能够寻找包含所有查询词项的最小窗口值 ω。

习题 **7-10**　当所有的查询词项并不同时出现在同一文档中时，试对上题中的算法进行修改。

7.3　向量空间模型对各种查询操作的支持

前面我们介绍了支持自由文本查询的向量空间模型。本章的最后，我们将介绍如何将向量空间模型同前面提到的的查询操作符关联起来。向量空间模型一般只支持自由文本查询，而前面介绍的几种查询操作，则实际上增强了查询的表达能力。因此，下面主要两个层面上来考察：第一，能否在在用向量空间模型的同时，支持这些查询操作，从而增强查询的表达能力？第二，如果可以的话，如何对现有的索引结构进行修改？在建立一个搜索引擎时，我们往往希望能够支持用户的多种查询操作符。为此，必须能够理解哪些索引部件能够在执行不同查询操作符时进行共享、如何处理混合了各种查询操作符的用户查询。

向量空间模型能够支持一种称为自由文本的检索，其中查询表示为一系列词的集合，词之间没有任何操作符进行连接。在这种模型下，就能够对文档进行评分和排序，这和以前讨论过的布尔查询、通配符查询及短语查询的处理有所不同。在传统意义上说，输入一个自由文本查询意味着每篇返回文档中至少要包含一个查询词项。但是，近年来，包括Google在内的搜索引擎已经让如下的概念深入人心：输入框中输入的一系列关键词查询（即自由文本查询）携带了类似合取操作的语义，因此，系统只返回包含全部或绝大部分查询词项的文档[①]。

7.3.1　布尔查询

① 这是因为 Web 上满足查询的文档一般都过多，这样的话包含所有或大部分查询词项的文档优先被考虑。这也是前面 Top K 检索的一般思路。——译者注

很显然，当词项t出现在文档d中时，t在d中的向量空间权重便不为 0，因此，向量空间模型中的索引结构显然能够用于处理布尔查询，但反之并不成立。布尔模型的索引在默认情况下并不包含词项的权重信息。站在用户角度来看，将向量空间模型和布尔查询融合并非易事：一方面，向量空间查询的处理基本上是证据累加（evidence accumulation）的方式，即多个查询词项的出现会增加文档的得分；另一方面，布尔检索需要用户指定一个表达式，通过词项的出现与不出现的组合方式来选择最终的文档，文档之间并无次序可言。数学上实际上存在一种称为p范式（p-norm）[①]的方法来融合布尔和向量空间查询，但是据我们所知，目前还没有实际系统这样做。

7.3.2 通配符查询

通配符查询和向量空间查询需要不同的索引结构来完成，当然，从基本层面上说，它们的索引结构都包括一部词典和一系列倒排记录表（比如，对于通配符查询而言，索引结构可能包含由 3-gram 组成的词典）。如果搜索引擎允许用户在自由文本查询中同时给定通配符（比如，输入查询 rom* restaurant），那么就可以把查询中的通配符解释成向量空间模型中的一系列查询词项（上例中，rome 和 roman 就是两个可能的查询词项），然后将所有查询词项加入到查询向量中去。最后，上述向量空间查询按照通常的方式进行处理，结果文档评分并排序输出。显然，一篇同时包含 rome 和 roma 的文档要比只包含其中一个词的文档的得分要高。当然，最终文档的精确排序主要取决于不同词项在文档中的相对权重。

7.3.3 短语查询

由于词项的次序信息在文档转换成向量的过程中被丢失，所以将文档表示成向量在理论上存在着明显的损失。即使我们在某种程度上将二元词（即两个连续出现的词组成的词对）也看成词项（即向量空间中的坐标轴），不同轴的权重也不是互相独立的。比如，短语 German shepherd 可以在 german shepherd 所对应的轴上有值，但同时在 german 和 shepherd 对应的轴上的权重值都非零。另外，诸如 idf 的概念也可以扩展到这种二元词上。因此，用于向量空间方法的索引通常并不能用于短语查询的处理。此外，因为我们仅仅知道每个词项在文档中的相对权重，所以无法得到短语查询的向量空间评分结果。

对于查询 german shepherd，可以通过向量空间检索来找到这两个词项权重较高的文档，但是无法要求这两个词项连续出现。另一方面，短语检索告诉我们短语 german shepherd 在文档中的出现，但是并没有任何有关短语相对频率及权重的提示信息。尽管短语检索和向量空间检索从索引结构和检索算法上来说存在较大差异，但是有时可以像 7.2.3 节给出的三步操作的例子那

① 这种方法是扩展的布尔模型(Extended Boolean Model)的一种，其基本思路是对布尔检索的结果进行排序。关于 p-norm 模型的详细介绍请参见 Baeza-Yates, Ricardo and Berthier Ribeiro-Neto. 1999. Modern Information Retrieval. Addison-Wesley, 2.6.2 节(P38)。——译者注

样进行有效的组合。

7.4　参考文献及补充读物

通过提前终止计算来进行快速查询处理的启发式方法的介绍参见 Anh 等人(2001)、Garcia 等人(2004)、Anh 和 Moffat(2006b)及 Persin 等人(1996)。有关簇剪枝方法的研究请参考 Singitham 等人（2004）及 Chierichetti 等人（2007）, 相关内容将会在 16.6 节中进一步讨论。胜者表的介绍参见 Persin（1994） 及 Brown（1995）, 在 Brin 和 Page（1998）及 Long 和 Suel（2003）中得到进一步发展。尽管这些启发式策略适合于可以看成向量的自由本文查询 , 它们同时也加大了短语查询的复杂性。Anh 和 Moffat（2006c）给出了一个能同时支持加权及布尔/短语查询搜索的索引结构。Carmel 等人（2001）、Clarke 等人（2000）及 Song 等人（2005）将查询词项的邻近关系用于相关性评估上。通过机器学习方法获得排序函数的开创性工作参见 Fuhr(1989)、Fuhr 和 Pfeifer（1994）、Cooper 等人（1994）、Bartell（1994）、Bartell 等人（1998）及 Cohen 等人（1998）。

第 8 章

信息检索的评价

前面各章介绍了信息检索系统设计中的各种方法。怎样才能知道其中哪些技术在哪些应用中有效？检索中是否应该使用停用词表？是否需要词干还原？在权重计算时是否应该使用逆文档频率？信息检索已经发展成为一门高度经验性的学科，需要在具有代表性的文档集上进行全面细致的评价，从而论证新技术的应用所带来的性能提升。

本章首先讨论信息检索系统效果的评价指标（见 8.1 节）以及用于评价的常用的标准测试集（见 8.2 节）。接着介绍相关文档和不相关文档的概念以及目前用于无序检索结果的形式化评价方法（见 8.3 节）。这其中也介绍了用于文档检索以及相关任务（如文本分类）的各种评价指标，并且解释了这些指标适用于评价这些任务的原因。接下来的 8.4 节对前面的概念进行了拓展，并介绍了适用于有序检索结果的评价指标。8.5 节讨论了如何开发可靠的、内容充实的测试集。

随后，在 8.6 节我们回过头来介绍用户效用（User utility）的概念，并介绍如何通过文档的相关性来对它进行近似估计。最关键的效用指标是用户满意度。响应速度和索引大小是用户满意度的影响因子。“结果的相关性是最重要的因子”这个假设看上去很合理，因为响应速度很快但不相关的回答结果并不会令用户满意。然而，用户的满意度和系统设计者所理解的结果质量概念之间并不完全一致。比如，用户的满意度通常都很大程度上依赖于用户界面的设计，包括布局、架构清晰度及用户界面的响应等等因素，而这些因素与系统返回结果的质量无关。最后我们会谈及其他质量检索结果度量的技术（见 8.7 节），特别是高质量的文档摘要片段的生成方法，它对用户效用的影响很大，但是它并没有包括在基本的相关度排序模式中。

8.1 信息检索系统的评价

采用常规的方式来度量 ad hoc IR 系统的效果，需要一个测试集（test collection），它由以下 3 个部分构成：

(1) 一个文档集；

(2) 一组用于测试的信息需求集合，信息需求可以表示成查询；

(3) 一组相关性判定结果，对每个查询–文档对而言，通常会赋予一个二值判断结果——要么相关（relevant），要么不相关（nonrelevant）。

常规的IR系统评价方法主要是围绕相关和不相关文档的概念来展开。对于每个用户信息需求，将测试集中的每篇文档的相关性判定看成一个二类分类问题进行处理，并给出判定结果：

相关或不相关。这些判定结果称为相关性判定的黄金标准（gold standard）或绝对真理（ground truth）[①]。测试集中的文档及信息需求的数目必须要合理：由于在不同的文档集和信息需求上的结果差异较大，所以需要在相对较大的测试集合上对不同信息需求的结果求平均。经验上发现 50 条信息需求基本足够（同时 50 也是满足需要的最小值）。

需要指出的是，相关性判定是基于信息需求而不是基于查询来进行的[②]，比如，可能有这样一个信息需求：在降低心脏病发作的风险方面，饮用红葡萄酒是否比饮用白葡萄酒更有效（原文是 whether drinking red wine is more effective at reducing your risk of heart attacks than drinking white wine）。该需求可能会表达成查询 wine AND red AND white AND heart AND attack AND effective。

一篇满足信息需求的文档是相关的，但这并不是因为它碰巧都包含查询中的这些词。由于信息需求往往并不显式表达，上述区别在实际上常常被误解。尽管如此，信息需求却始终存在。如果用户向 Web 搜索引擎输入 python，那么他们可能想知道可以买宠物蛇的地方，或者想查找与编程语言 Python 相关的信息。对于单个词构成的查询，系统很难知道其背后的真实需求。当然，对于用户而言，他肯定有自己的信息需求，并且能够基于该需求判断返回结果的相关性。要评价一个系统，需要对信息需求进行显式的表达，以便利用它对返回文档进行相关性判定。迄今为止，我们对相关性都进行了简化处理，把相关性考虑为一个只具有如下尺度的概念：一些文档高度相关而其他却不太相关。也就是说，到现在为止，我们仅对相关性给出一个二值判定结果。应该说，这种简化具有一定的合理性。我们将会在后面的 8.5.1 节讨论这样简化的原因以及其他尺度的相关度判定结果。

许多系统都包含多个权重参数，改变这些参数能够调优系统的性能。通过调优参数而在测试集上获得最佳性能并报告该结果是不可取的。这是因为这种调节能在特定的查询集上获得最佳参数，而这些参数在随机给定的查询集上并不一定能够取得最佳性能，因此，上述做法实际上夸大了系统的期望性能。正确的做法是，给定一个或者多个开发测试集（development test collection），在这个开发测试集上调节参数直至最佳性能，然后测试者再将这些参数应用到最后的测试集上，最后在该测试集上得到的结果性能才是真实性能的无偏估计结果。

8.2　标准测试集

下面列出了一些常用的标准测试集及相关的评测会议，这里我们主要关注用于 ad hoc 信息检索系统评价的测试集，但是也提到了一些用于文本分类任务的测试集。

① 也就是说这些判定构成所谓的“标准答案”集合，对于每个系统的输出结果，可以利用这个集合进行评分。——译者注

② 也就说，一个结果的相关与否是针对信息需求而言的，而不管信息需求具体表达成何种查询形式。本节例子中，根据转换后的布尔查询返回的一些结果可以并不能满足信息需求。——译者注

CRANFIELD Cranfield 测试集，它是对信息检索系统的效果进行精确定量评价的首个测试集，但是对现在来说它的规模上已经非常小，只能用于最基本的试验性工作。该测试集于 20 世纪 50 年代末期在英国收集而得，总共包含 1 398 篇空气动力学期刊的文章摘要、225 个查询以及所有的（查询,文档）对的相关性判定结果。

TREC TREC（Text Retrieval Conference，文本检索会议），是从 1992 年开始由 NIST（National Institute of Standards and Technology，美国国家标准技术研究所）组织的大型 IR 系统的年度评测会议。这个框架下定义了很多任务，每个任务都有自己的测试集。但是，其中最著名的测试集还是用于 1992 到 1999 年间的最早 8 次 TREC Ad Hoc 任务的测试集。这些测试集总共由 6 张 CD 组成，包括 189 万篇文档（主要是新闻类文章，也包括其他类型的文章）、450 个信息需求及相关性判定，在 TREC 中每个信息需求也称为主题（topic），并用详细的文字进行描述。多个不同的测试集定义在上述数据集的子集上。早期的每届 TREC 都有 50 个信息需求，但是它们在不同的文档子集上进行测试，这些文档子集之间也有重合。从第 6 届 TREC 到第 8 届 TREC，总共提供了 150 个信息需求，文档集包含 528 000 篇新闻及美国中央情报局对外广播情报处（Foreign Broadcast Information Service）的文章。由于这个子测试集的规模最大，主题也比较一致，该子测试集可能是未来可用的最佳子集。由于 TREC 中的文档集都相当大，所以评测中并没有完整的相关性判定。实际上，给定信息需求，NIST 评估者仅仅针对 TREC 所提交的每个系统的前 k 篇返回文档做相关性判定。

GOV2 近年来，NIST 在较大规模的文档集上进行了评价测试，这其中包括 2 500 万网页组成的 GOV2 文档。从 TREC 一开始，NIST 就提供了比已有供研究者使用的数据大几个数量级的测试集，GOV2 是目前易获得的、用于研究目的的最大规模的 Web 文档集。然而，GOV2 在规模上仍然比现有的大型搜索引擎厂商索引的网页至少要小 2 个数量级。

NTCIR 日本国立情报研究所的信息检索测试集（NII Test Collections for IR Systems）的简称。NTCIR 项目构造了多个和 TREC 文档集规模相当的文档集，其中大部分文档都集中关注东亚语言和跨语言检索任务。这里提到的跨语言检索指的是这样的一种检索任务：查询是一种语言，而待检索文档由另外一种或者多种语言所构成。详细内容请参考 http://research.nii.ac.jp/ntcir/data/data-en.html。

CLEF 跨语言评价论坛（Cross Language Evaluation Forum）的简称。该评价会议系列主要关注欧洲语言及它们之间的跨语言检索任务。详情请参考 www.clef-campaign.org/。

Reuters 语料 包括 Reuters-21578 和 Reuters-RCV1 语料。对于文本分类任务，最常用的测试集一直是 Reuters-21578 语料，它由 21 578 篇新闻报道组成（参见第 13 章）。近年来，Reuters 发布了一个更大规模的语料库 RCV1（Reuters Corpus Volume1），它包括 806 791 篇文档（参见第 4 章）。RCV1 的规模和丰富的标注信息为未来的研究提供了更好的基础。

20 Newsgroups 这是另一个广泛使用的文本分类语料，来自 Ken Lang 所收集的新闻组语料。它从 20 个 Usenet 新闻组（新闻组的名字即类别的名称）的每个组中取 1 000 篇文章构成。

去掉重复文章以后，通常所用的语料中包含 18 941 篇文章。

8.3　无序检索结果集合的评价

在给定测试集的情况下，如何度量系统的效果？信息检索中最常用的两个基本指标是正确率和召回率。它们最早定义于一种非常简单的情况下：对于给定的查询，IR 系统返回一系列文档集合，其中的文档之间并不考虑先后顺序。后面将介绍如何将这些概念扩展到有序的检索中去。

正确率（Precision，简记为 P）是返回的结果中相关文档所占的比例，定义为：

$$\text{Precision} = \frac{\text{返回结果中相关文档的数目}}{\text{返回结果的数目}} = P(\text{relevant} \mid \text{retrieved})。 \quad (8\text{-}1)$$

而召回率（Recall，简记为 R）是返回的相关文档占所有相关文档的比例，定义为：

$$\text{Recall} = \frac{\text{返回结果中相关文档的数目}}{\text{所有相关文档的数目}} = P(\text{retrieved} \mid \text{relevant})。 \quad (8\text{-}2)$$

为了更好地理解上述概念[①]，可以对照如下列联表（contingency table）：

	相关(relevant)	不相关(nonrelevant)
返回(retrieved)	真正例(true positives, *tp*)	伪正例(false positives, *fp*)
未返回(not retrieved)	伪反例(false negatives, *fn*)	真反例(true negatives, *tn*)

(8-3)

于是有：

$$P = tp / (tp + fp)$$
$$R = tp / (tp + fn)。 \quad (8\text{-}4)$$

在这里，读者可能会想到的一个显然可以用于度量信息检索系统效果的指标是精确率（accuracy），也就是文档集中所有判断正确的文档所占的比例。如果采用上述表格中的符号，那么精确率的计算公式为：$(tp + tn)/(tp + fp + fn + tn)$。采用精确率来度量信息检索的效果看上去有一定的道理，这是因为上述检索系统中实际上存在着两个类别：相关类和不相关类，信息检索系统也因此可以看成一个针对二类问题的分类器，这个分类器实际上认为返回的文档是相关文档。精确率指标在很多机器学习问题中的使用非常普遍，是一个非常适合这类问题的效果度量指标。

然而，精确率对于信息检索来说并不是一个很好的度量指标。这一点也很容易解释：绝大多数情况下，信息检索中的数据存在着极度的不均衡性，比如通常情况下，超过 99.9%的文档都是不相关文档。这样的话，一个简单地将所有的文档都判成不相关文档的系统就会获得非常高的精确率值，从而使得该系统的效果看上去似乎很好。而即使系统实际上非常好，试图将某

① 公式(8-1)和公式(8-2)中都采用了条件概率的写法。另外一种常用的写法是 ,Precision=(retrieved relevant) /retrieved, Recall=(retrieved relevant)/relevant。这些写法的含义都是一致的。——译者注

些文档标注为相关往往也会导致很高的假阳率（rate of false positive）[①]。然而，将所有文档标注为不相关文档（即不返回任何文档）显然不能使检索系统的用户满意。用户往往想浏览某些文档，并且可以假设他们在浏览有用的信息之前能够在一定程度上容忍不相关文档的存在。正确率和召回率只对返回文档中的真正相关文档进行评估，它们分别度量的是返回文档中的相关文档比例和返回的相关文档占所有相关文档的比例。

同时采用正确率和召回率两个指标来度量效果有一个优点，即能够满足偏重其中一个指标的场景的需要。典型的 Web 检索用户希望第一页的所有的结果都是相关的，也就是说他们非常关注高正确率，而对是否返回所有的相关文档并没有太大的兴趣。相反地，一些专业的搜索人士（如律师助手、情报分析师等）却往往重视高召回率，有时甚至宁愿忍受极低的正确率也要获得高的召回率。对本机硬盘进行搜索的个人用户也常常关注召回率。然而，这两个指标之间显然存在着某种折衷。一个极端的情况是，对于某查询如果返回所有文档，显然此时的召回率为 1，但是此时的正确率往往却很低。一方面，相对于返回文档的数目而言，召回率是一个单调非减函数；另一方面，在一个好的系统中，正确率往往会随着返回文档数目的增加而降低。通常来说，我们希望在容许较小的错误率（此时会达到较高的正确率）的同时达到一定的召回率。

一个融合了正确率和召回率的指标是 F 值（F measure），它是正确率和召回率的调和平均值，定义如下：

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad , \tag{8-5}$$

其中，$\beta^2 = \frac{1-\alpha}{\alpha}$。

公式（8-5）中，$\alpha \in [0, 1]$，因此 $\beta^2 \in [0,\infty]$。默认情况下，平衡 F 值（balanced F measure）中正确率和召回率的权重相等，即 $\alpha = 1/2$ 或 $\beta = 1$。尽管是从 α 的角度来体现出 F 是正确率和召回率的调和平均值，但上述等权重情况下我们可以用 β 的取值来记 F 值，此时记为 F_1，它是 $F_{\beta=1}$ 的省略形式。当 $\beta = 1$ 时，F 的计算公式可以简化为：

$$F_{\beta=1} = \frac{2PR}{P+R}。 \tag{8-6}$$

然而，等权重取值并不是唯一选择。$\beta < 1$ 表示强调正确率，而 $\beta > 1$ 表示强调召回率。例如，如果强调召回率的话，β 可以取 3 或 5。召回率、正确率和 F 值的取值范围都是[0,1]，当然它们也常常可以写成百分数的形式。

为什么使用调和平均而不是其他简单的平均方法（如算术平均）来计算 F 值呢？前面我们提到过，我们可以通过返回所有文档来获得 100%的召回率，此时如果采用算术平均来计算 F 值，那么 F 值至少为 50%。这表明在这里使用算术平均显然是不合适的。而如果采用调和平均

① 假阳率也称为误判率，指的是所有不相关文档中错判为相关的文档（false positive，称伪正例或假正例）的比率。在医学领域这个指标也往往称为误诊率，即将实际无病判为有病的比率。计算公式为 *fp*/（*fp*+*tn*）。——译者注

来计算，假定 10 000 篇文档中只有 1 篇和查询相关，那么此时 F 值为 0.02%。调和平均值小于等于算术平均值和几何平均值。如果两个求平均的数之间相差较大，那么和算术平均值相比，调和平均值更接近其中的较小值（参见图 8-1）。

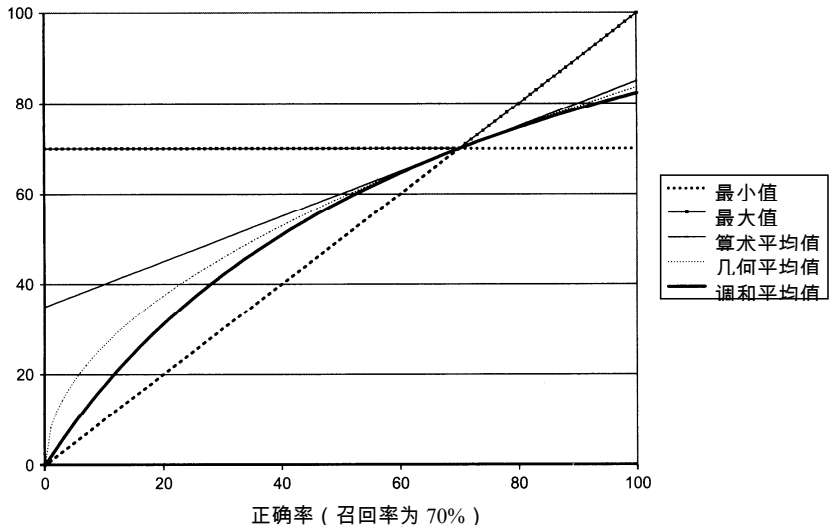


图 8-1　调和平均值和其他几种平均值的比较图。本图基于固定的召回率（70%），给出的是在正确率变化的情况下不同平均值的一段变化图。图中可以看到，调和平均值往往小于算术平均和几何平均值，并且常常与两个数的较小值更接近。图中也可以看出，当正确率也等于 70%时，各种度量值都相等

习题 **8-1** [*]　某个 IR 系统返回了 8 篇相关文档和 10 篇不相关文档。在文档集中总共有 20 篇相关文档。试计算本次搜索的正确率和召回率。

习题 **8-2** [*]　F_1 定义为正确率和召回率的调和平均值，采用调和平均而不是算术平均进行计算的好处是什么？

习题 **8-3** [**]　给定 $\alpha = 1/(\beta^2 + 1)$，试推导出公式(8-5)。

8.4　有序检索结果的评价方法

正确率、召回率和F值都是基于集合的评价方法，它们都利用无序的文档集合进行计算。当面对诸如搜索引擎等系统输出的有序检索结果时，有必要对上述方法进行扩展或者定义新的评价指标。在结果有序的情况下，通常很自然地会将前面k个（k=1,2,…）检索结果组成合适的返回文档子集。对每个这样的集合，都可以得到正确率和召回率，分别以它们作为纵坐标和横坐标在平面上描点并连接便可以得到如图 8-2 所示的正确率–召回率曲线（precision-recall curve）。正确率–召回率曲线往往会表现出明显的锯齿形状，这是因为如果返回的第(k+1)篇文档不相关，那么在(k+1)篇文档位置上的召回率和前k篇文档位置上的召回率一样，但是正确率

显然下降。反之，如果返回的第(k+1)篇文档相关，那么正确率和召回率都会增大，此时曲线又呈锯齿形上升。将这些细微的变化去掉往往非常有用，实现这一目的的常规方法是采用插值的正确率。在某个召回率水平r上的插值正确率（interpolated precision，记为p_{interp}）定义为对于任意不小于r的召回率水平r'所对应的最大正确率[①]，即

$$p_{\text{interp}}(r) = \max_{r' \geq r} p(r') \text{。} \qquad (8\text{-}7)$$

上述定义的合理性在于：如果多看一些文档会提高所看文档中相关文档的比例的话（也就是说在更大的集合中的正确率更高），那么大部分用户都会这样做。在图 8-2 中，插值正确率采用细线表示。依照上述定义，召回率为 0 所对应的正确率值也能得到很好的定义（参考习题 8-4）。

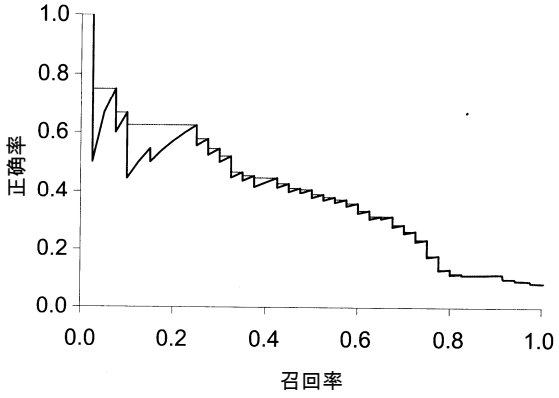


图 8-2 正确率–召回率曲线图

整个正确率–召回率曲线具有非常丰富的信息，但是人们往往期望将这些信息浓缩成几个甚至一个数字。传统的做法是（比如在前八次 TREC 的 Ad hoc 任务中）定义一个 11 点插值平均正确率（eleven-point interpolated average precision）。对每个信息需求，插值的正确率定义在 0.0, 0.1, 0.2, ..., 1.0 等 11 个召回率水平上。图 8-2 中的正确率–召回率曲线所对应的 11 点上的插值正确率在表 8-1 中列出。对于每个召回率水平，可以对测试集中每个信息需求在该点的插值正确率求算术平均。于是可以画出一条包含 11 个点的正确率–召回率合成曲线。图 8-3 就给出了 TREC8 中一个效果较好的系统的曲线的例子。

表8-1 11点插值平均正确率的计算。表中的值对应图8-2的正确率–召回率曲线

召回率	插值正确率	召回率	插值正确率
0.0	1.00	0.6	0.36
0.1	0.67	0.7	0.29
0.2	0.63	0.8	0.13
0.3	0.55	0.9	0.10
0.4	0.45	1.0	0.08
0.5	0.41		

① 需要指出的是，这里的插值正确率定义不同于 Baeza-Yates 在 *Modern Information Retrieval* 一书中的定义，那里定义的是上面一个小区间内最大的正确率，这里实际上算的是上面所有区间的最大正确率。——译者注

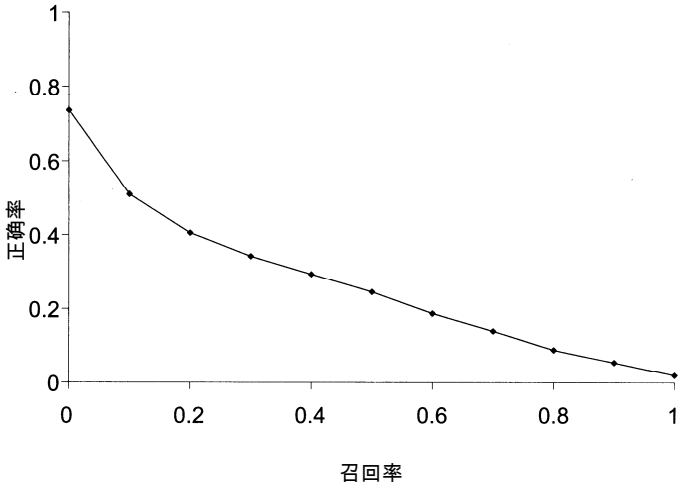


图 8-3　在 50 个查询上某 TREC 系统的 11 点插值正确率–召回率平均曲线，该系统的 MAP 值是 0.2553

近年来，一些其他的评价指标使用越来越普遍。TREC中最常规的指标是MAP（mean average precision，平均正确率均值），它可以在每个召回率水平上提供单指标结果 ①。在众多评价指标中，MAP被证明具有非常好的区别性（discrimination）和稳定性（stability）。对于单个信息需求，返回结果中在每篇相关文档位置 ②上的正确率的平均值称为平均正确率（average precision），然后对所有信息需求平均即可得到MAP。形式化地，假定信息需求$q_j \in Q$对应的所有相关文档集合为$\{d_1, \ldots d_{mj}\}$，R_{jk}是返回结果中直到遇见d_k后其所在位置前（含d_k）的所有文档集合，则有

$$\mathrm{MAP}(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j} Prcision(R_{jk}) \text{。} \qquad (8\text{-}8)$$

如果某篇相关文档未返回 ③，那么上式中其对应的正确率值为 0。对于单个信息需求来说，平均正确率是未插值的正确率–召回率曲线下面的面积的近似值，因此，MAP可以粗略地认为是某个查询集合对应的多条正确率–召回率曲线下面积的平均值。

使用 MAP，就不再需要选择固定的召回率水平，也不需要插值。某个测试集的 MAP 是所有单个信息需求上的平均正确率的均值。这导致的效果是，即使有些查询的相关文档数目较多而有些却很少，但是在最终的 MAP 指示报告中每个信息需求的作用却是相等的。单个系统在不同信息需求集上的 MAP 值往往相差较大（比如，在 0.1 到 0.7 之间变化）。实际上，与单个系统在不同信息需求上的 MAP 差异相比，不同系统在同一信息需求上的 MAP 差异反而相对要

① 有两种计算 MAP 的方法或者说有两种 MAP 的定义方法。每一种方法是在每篇相关文档所在位置上求正确率然后平均（公式 8-8)。另一种是在每个召回率水平上计算此时的插值正确率，然后求 11 点平均正确率，最后在不同查询之间计算平均。前者也称为非插值 MAP。一般提 MAP 都指前者。

② 如果相关文档没返回，则认为该相关文档在无穷远处，即此时该文档对应的正确率为 0。需要注意的是，该正确率也要用于最后的 MAP 计算。

③ 给定查询，一个系统往往不可能对文档集中所有文档进行排序，或不管怎样，评测可能只基于每个信息需求的前 k 个结果来进行。

小一些。这意味着用于测试的信息需求必须足够大、需求之间的差异也要足够大，这样的话系统在不同查询上体现出的效果才具有代表性。

上述指标实际是在所有的召回率水平上计算正确率。对于许多重要应用特别是 Web 搜索来说，该指标对于用户而言作用并不大，他们看重的是在第 1 页或前 3 页中有多少好结果。于是需要在固定的较少数目（如 10 或者 30 篇文档）的结果文档中计算正确率。该正确率称为前 k 个结果的正确率（precision at k，可简写成 P@k），比如 P@10。该指标的优点是不需要计算相关文档集合的数目，缺点就是它在通常所用的指标中是最不稳定的，这是因为相关文档的总数会对 P@k 有非常强的影响。

能够解决上述问题的一个指标是 R-precision。它需要事先知道相关的文档集 Rel，然后计算前|Rel|个结果集的正确率。其中 Rel 不一定是完整（complete）的相关文档集合，可以先将不同系统在一系列实验中返回的前 k 个结果组成缓冲池，然后基于缓冲池进行相关性判定从而得到相关文档的集合。R-Precision 能够适应不同的相关文档集大小的变化。一个完美系统的 R-Precision 值可以达到 1，而对于一个只包含 8 个相关文档的信息需求而言，最完美的系统的 P@20 值也只能达到 0.4。因此，对于 R-Precsion 指标来说，在不同查询上求平均才更有意义。另外，对用户来说，该指标比 P@k 难理解，但又比 MAP 好理解：对于某查询如果总共有|Rel|篇相关文档，而在前|Rel|个返回结果中有 r 篇相关文档，那么根据定义，不仅此时的正确率为 r/|Rel|（当然此时的 R-Precision 也是 r/|Rel|），而且召回率也等于这个值。因此，R-Precsion 和有时候用到的正确率召回率等值点（*break-even point*）的概念是一样的，后者指的是正确率和召回率相等的点。像 P@k 一样，R-Precision 描述的也是正确率–召回率曲线上的一个点，而不是对整条曲线求概括值。因此，在某种程度上很难说清楚为什么只对曲线上的等值点而不是最好的点（F 值最大的点）感兴趣，或是对某个特定应用的固定返回结果数目上的值感兴趣（如 P@k）。尽管如此，虽然 R-Precision 只度量了曲线上的一个点，但是在经验上却证实它和 MAP 高度相关。

另一个在评价时可能会用到的概念是 ROC（ROC 是 receiver operating characteristics 的缩写，但是知道这个对大多数人来说没什么用）曲线。ROC 曲线是基于假阳率（false positive rate）或 1-特异度（specificity）画出真阳率（true positive rate）或者敏感度（*sensitivity*）而得到的曲线。这里，敏感度只是召回率的另外一种叫法。假阳率的计算公式为 $fp/(fp+tn)$。图 8-4 给出了对应于图 8-2 正确率–召回率曲线的 ROC 曲线。ROC 曲线通常起于左下角而逐渐向右上角延伸。一个好的系统，曲线图的左部会比较陡峭。对于无序的检索来说，计算公式为 $tn/(fp+tn)$的特异度并不被看成一个很有用的概念。返回结果中真正不相关的文档集合通常很大，因此，对于所有的信息需求来说，特异度结果值接近 1，而假阳率接近于 0。所以，图 8-2 中有趣的部分是 0<召回率<0.4 这段，在图 8-4 中它被压缩到一个小角落。然而，如果把视角放宽到整个检索，那么 ROC 曲线是有意义的，它提供了对数据观察的另外一个角度。在很多领域，一个普遍使用的指标是计算 ROC 曲线下的面积，这可以看成是基于 ROC 的 MAP 版本。有时候，在非严格的

情况下，正确率–召回率曲线也被称为 ROC 曲线，在上下文环境下，这通常都能够被正确理解。

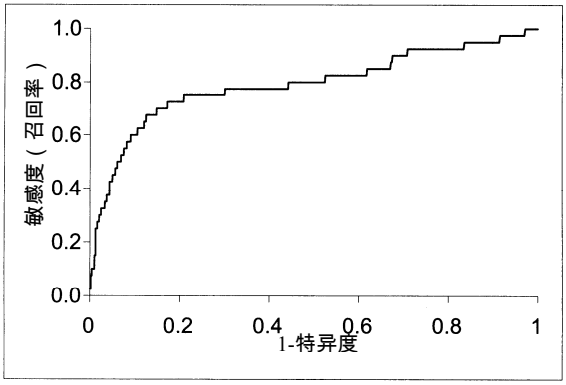


图 8-4　图 8-2 正确率–召回率曲线所对应的 ROC 曲线

最后介绍一种近年来逐渐被采用、往往应用在基于机器学习的排序方法中（参考 15.4 节）的指标——CG（cumulative gain，累积增益），一个具体的指标为NDCG（normalized discounted cumulative gain，归一化折损累积增益）。NDCG是针对非二值相关情况下的指标（参考 8.5.1 节）。同指标P@k一样，它基于前k个检索结果进行计算。设$R(j,d)$是评价人员给出的文档d对查询j的相关性得分，那么有[①]（给后来校对者，原书公式有误，Z少一个下标j）

$$\mathrm{NDCG}(Q,k)=\frac{1}{|Q|}\sum_{j=1}^{|Q|}Z_{j,k}\sum_{m=1}^{k}\frac{2^{R(j,m)}-1}{\log(1+m)}\ , \qquad (8\text{-}9)$$

其中，Z_k是归一化因子，用于保证对于查询j最完美系统的 NDCG at k 得分是 1，m 是返回文档的位置。如果某查询返回的文档数$k'<k$，那么上述公式中只需要计算到k'为止。

习题 **8-4** [*]　召回率水平为 0 所对应的插值正确率的可能取值为多少？

习题 **8-5** [**]　正确率和召回率之间是否一定存在等值点？说明为什么一定存在或给出反例。

习题 **8-6** [**]　正确率–召回率等值点与这点的 F_1 值之间是什么关系？

习题 **8-7** [**]　两个集合之间的 *Dice* 系数是度量其交集大小占两个集合大小之和的比率的一个指标，取值在 0 到 1 之间，其计算公式为

$$\mathrm{Dice}(X,Y)=\frac{2|X\cap Y|}{|X|+|Y|}。$$

请证明，召回率和正确率等权重情况下得到的 F_1 值等于检索结果文档集合和相关文档集合

① 本书给出的 NDCG 计算公式和一般文献(包括原始定义文献 Järvelin and Kekäläinen (2002))不同，简单地说，原始文献直接定义在相关度值上而不是以 2 为底的相关度幂值上。按照 W. B. Croft 等人 “Search Engines-Information Retrieval Practice”(第 320 页)给出的解释。本书的 NDCG 公式为很多搜索引擎公司所使用，也许会成为新的 NDCG 计算标准。Croft 指出，本书的公式和一般文献的公式在二值相关度情况下是等价的(经过译者推算，应该还有细微的区别，主要在分母 1+m 这块，原始文献应为 m)，但是在非二值情况下，本书公式对高相关度文档给予了更高权重。需要指出的是，这里的对数应该以 2 为底。——译者注

的 *Dice* 系数。

习题 **8-8** [*] 考虑一个有 4 篇相关文档的信息需求，考察两个系统的前 10 个检索结果（左边的结果排名靠前），相关性判定的情况如下所示：

系统 1 R N R N N N N N R R

系统 2 N R N N R R R N N N

a. 计算两个系统的 MAP 值并比较大小。

b. 上述结果直观上看有意义吗？能否从中得出启发如何才能获得高的 MAP 得分？

c. 计算两个系统的 R-precision 值，并与 a 中按照 MAP 进行排序的结果进行对比。

习题 **8-9** [**] 在 10 000 篇文档构成的文档集中，某个查询的相关文档总数为 8，下面给出了某系统针对该查询的前 20 个有序结果的相关（用 R 表示）和不相关（用 N 表示）情况，其中有 6 篇相关文档：

R R N N N N N N R N R N N N R N N N N R

a. 前 20 篇文档的正确率是多少？

b. 前 20 篇文档的 F_1 值是多少?

c. 在 25%召回率水平上的插值正确率是多少？

d. 在 33%召回率水平上的插值正确率是多少？

e. 假定该系统所有返回的结果数目就是 20，请计算其 MAP 值。

假定该系统返回了所有的 10,000 篇文档，上述 20 篇文档只是结果中最靠前的 20 篇文档，那么

f. 该系统可能的最大 MAP 是多少？

g. 该系统可能的最小 MAP 是多少？

h. 在一系列实验中，只有最靠前的 20 篇文档通过人工来判定，(e)的结果用于近似从(f)到(g)的 MAP 取值范围。对于上例来说，通过(e)而不是(f)和(g)来计算 MAP 所造成的误差有多大（采用绝对值来计算）？

8.5 相关性判定

为了对系统进行恰当地评价，所选取的测试信息需求一定要和测试文档集密切相关，并且能够对系统的使用情况进行预期。这些信息需求最好由领域专家来设计。通过随机组合查询词项的方式来生成信息需求并不是一个好的做法，这是因为这样随机组合生成的结果并不能代表信息需求的真实分布。

给定信息需求集及文档集，需要给出它们之间的相关性判定情况，这是一项需要人工参与的费时费力的工作。对于像 Cranfield 那样极小的文档集来说，可以人工对每对查询和文档进行相关性判定而得到结果。而对于当前的大规模文档集，通常的做法是只对一部分文档子集进行相关性判定。最常规的做法称为缓冲池法（pooling），即将一系列检索系统中每个系统所返回的前 k 篇文档合成一个文档子集，并对这个子集进行相关性判定。这些检索系统往往同时也是

需要评价和比较的对象。当然，有时候这个文档子集中也会加入通过布尔关键词查询得到的结果，或专家通过交互式检索方式得到的文档。

人不是机器，他们给出的文档和查询的相关性判定结果并不完全可靠。实际上，不论是人还是他们的相关性判定结果都极具主观性、差异很大。然而最终的分析表明，信息检索系统的成功与否取决于它能否满足每个人的特定的信息需求，因此上述的差异并不是个问题。

然而，一个有趣的话题是考虑不同人所做出的相关性判定之间的一致性。在社会科学中，一个用于度量这个一致性的常用指标是 *kappa* 统计量（kappa statistic）。它用于类别型判断结果（比如相关或不相关两类判断结果），是对随机一致性比率的一个简单校正。

$$kappa = \frac{P(A) - P(E)}{1 - P(E)}。 \qquad (8\text{-}10)$$

其中，$P(A)$是观察到的一致性判断比率，而$P(E)$是随机情况下所期望的一致性判断比率。对于后者的估计可以有很多选择。一种简单的情况是做二类判定，并不做任何其他假设，那么随机的一致性判断比率是 0.5。然而，通常情况下类别之间的分布是不均衡的，因此通常采用边缘统计量（marginal statistics）来计算随机一致性比率[①]。这里也有两种做法，一种做法是将每个评判人的边缘分布值进行迭加，另一种是单独计算每个评判人的边缘分布值。实际中两种方法都在使用，但这里我们采用了前者，这是因为不同评判人的边缘分布之间可能差别很大(即存在系统性差异)，而后者中单独计算的边缘分布可能更会受到这种差异的影响。*kappa*统计量的计算过程如表 8-2 所示。当两个判断之间一致时，*kappa*统计量取值为 1。如果判断之间的一致性和随机判断一样，则取值为 0。如果还不如随机判断，则取值为负值。如果有 2 个以上的判断，那么通常会计算两两之间*kappa*值的平均值。虽然具体的取值范围取决于数据使用的目的，但是经验上来说，一般如果*kappa*值大于 0.8，那么表示存在很好的一致性。若取值在 0.67 到 0.8 之间，表示存在较好的一致性。如果取值小于 0.67，那么结果的可靠性值得怀疑。

表8-2　*kappa*统计量的计算

		第2个人的相关性判定结果		
		yes	no	total
第1个人的相关性判定结果	yes	300	20	320
	no	10	70	80
	total	310	90	400

观察到的两个人的一致性判断比率

$P(A)=(300+70)/400=370/400=0.925$

边缘统计量

$P(nonrelevant)=(80+90)/(400+400)=170/800=0.2125$

$P(relevant)=(320+310)/(400+400)=630/800=0.7878$

两人的随机一致性比率

$P(E)=P(nonrelevant)^2+P(relevant)^2=0.2125^2+0.7878^2=0.665$

① 对于表 8-2 所示的列联表，边缘统计量可以对行或列求和而获得。比如，边缘 $a_{i,k} = \sum_j a_{ijk}$ 。

Kappa 统计量

$k=(P(A)-P(E))/(1-P(E))=(0.925-0.665)/(1-0.665)=0.776$

相关性判定的一致性度量在 TREC 评测和一些医学文档集上得到应用。按照上面提到的经验法则，这些场景下的一致性水平大都落在“较好”对应的那个区间（即介于 0.67～0.8 之间）上。在二值相关性判定中，人与人之间的一致性水平一般，这个事实也是在 IR 评估中不需要更细粒度的相关性定义的一个原因。那么，为什么在不同评估者之间判断存在差异的情况下信息检索的评价仍然有效呢？这是因为人们选择两个评价结果中的一个或另一个作为标准答案进行了评价，不同的选择方法会造成结果得分在绝对值上的不小差异，但是一般情况下却不会对两个系统或一个系统的不同变形情况下的相对排序产生影响。

8.5.1 相关性概念的正反辨析

基于相关和不相关文档进行系统评价的好处在于，可以在某个固定配置下使用不同的检索系统和参数来进行对比实验。相对基于检索结果的用户调查方法来说，这种做法花费更少，并且能够获得参数变化对系统影响的更清晰的分析和判断。实际上，一旦我们有一个形式化的指标并确信其能代表用户的满意度，那么就可以通过机器学习而不是手工调节参数的方法来对检索效果进行调优。当然，如果该形式化指标不能很好地反映用户所需，这样做并不能提高用户的满意度。我们的观点是，在实际中，标准的 IR 评价指标尽管简单，但是已经足够好，近年对这些指标进行优化的工作也取得了显著的成功。在形式化的评价中，也产生了大量用于提高检索效果的技术，比如在 TREC 中的文档长度归一化技术的应用（参见 6.4.4 节和 11.4.3 节）及在结果评分中用于权重参数调节的机器学习方法（参见 6.1.2 节）。

但是，这并不是说在上述形式化过程中没有任何问题。第一，上述过程中都假设文档集中文档和文档之间的相关性是相互独立的。实际上，该假设在很多检索系统中使用，其中文档往往是基于查询而不是其他文档来进行评分，在评价时也是如此。第二，上面的相关性判定结果都是二值的，即没有给出更多粒度的相关性。第三，上述文档和信息需求的相关性判定被看成是一个绝对的、客观的过程。但是，很显然相关性的判定是主观的，正如前面所讨论的一样，不同的人的判断不尽相同。实际中，人类也并非完美的测量仪器，易受理解能力和注意力方面的缺陷所影响。另外，前面我们也假设用户的信息需求在用户浏览检索结果过程中不会改变。基于一个文档集的任何结果都非常依赖于文档集、查询和相关性判定的选择，该结果不能从一个领域推广到另一个领域，也不能推广到另外一个用户群体。

当然，上面的有些问题是可以解决的。近年来，包括 INEX、TREC 及 NTCIR 在内的一系列评测引入了相关性的等级概念，它们把文档和查询的相关性分成 3 或 4 个等级，将轻度相关和高度相关的文档区别开来。这种思路在 INEX 评测中的详细使用方法请参看 10.4 节。

前面提到的相关性判定的方法存在一个明显的问题就是要区别相关性和边缘相关性（marginal relevance）。后者关注的是在看了一些其他文档之后文档是否仍然具有显著的价值（参

见 Carbonell 和 Goldstein，1998）。即使一篇文档高度相关，其信息也可能是冗余的，因为已存在于一篇已经看过的文档当中。最极端的例子是，两篇文档完全相同，这种现象在互联网上非常普遍。另外，当多篇文档对同一事件提供近似的介绍时也很容易发生这种现象。这种情况下，很显然边缘相关性对于用户而言是一个更好的指标。要使边缘相关性最大化，则要求返回的文档具有多样性（diversity）和新颖性（novelty）。一种做法是利用事实或实体作为评价的单位。这样做能更直接地度量了用户的真实效用，但是需要建立测试集，难度还是相当大的。

? 习题 **8-10**　[**] 下表中是两个判定人员基于某个信息需求对 12 个文档进行相关性判定的结果（0=不相关，1=相关）。假定我们开发了一个 IR 系统，针对该信息需求返回了文档集{4, 5, 6, 7, 8}。

docID	判断 1	判断 2
1	0	0
2	0	0
3	1	1
4	1	1
5	1	0
6	1	0
7	1	0
8	1	0
9	0	1
10	0	1
11	0	1
12	0	1

a. 计算两个判断之间的 *kappa* 统计量；

b. 当两个判断均认为是相关文档时才认为该文档相关，此时计算上述系统的正确率、召回率及 F_1 值；

c. 只要有一个判断认为是相关文档则认为该文档相关，此时计算上述系统的正确率、召回率及 F_1 值。

8.6　更广的视角看评价：系统质量及用户效用

形式化的评价指标和采用用户效用来度量的用户最终兴趣之间还是有些差距：每个用户提交信息需求后对系统返回的文档的满意程度到底如何？度量用户满意度的常规方法是进行各种各样的用户调查。用户调查中可能会包括定量的评价指标，这些指标或者是客观的，比如任务完成的时间，也或者是主观的，比如用户对搜索引擎的满意度评分。除此之外，用户调查也可能包含定性的评价指标，如用户在搜索界面上的评论结果。本节当中，我们将讨论其他有关系统定量评价的问题以及与用户效用相关的问题。

8.6.1　系统相关问题

存在很多实际的基准方法来对 IR 系统的检索质量进行排名，它们包括：

索引速度如何？对长度满足某种分布（参见第 4 章）的文档集构建索引，每小时能对多少篇文档构建索引？

检索速度如何？基于不同索引大小的响应函数如何？

查询语言的表达能力如何？复杂查询的处理速度多快？

文档集有多大？可以基于文档的数目来计算或者基于文档集中所包含的多个话题的分布信息来计算。

除查询语言的表达能力之外，上面提到的其他方面均可以直接度量。我们可以对速度或大小进行定量计算，可以通过功能列表的方法来半精确地度量查询语言的表达能力。

8.6.2 用户效用

我们真正希望的是，有一个方法能够根据系统的相关性、速度和用户界面等诸多因素，来定量地汇总计算出一个用户的满意程度。其中的一部分工作是理解系统所要满足的用户的分布情况，而这完全依赖于使用场景。对于 We 搜索来说，对搜索满意的用户是那些找到所需结果的人。一种间接找出此类用户的方法是，看他们是否再次访问同一搜索引擎。用户的回访率显然是一个非常有效的度量指标，当然，如果同时还能知道这些用户使用其他搜索引擎的频度，那就更好了。但是，广告商也是现代搜索引擎的用户，如果顾客点击了它们的网站并进行了交易，那么他们将会十分满意。对于电子商务网站而言，用户往往希望进行购物。因此，我们可以计算从开始到交易的时间或者搜索用户中的购物比例。在一个网上商店上，如果发生了交易，那么也许用户和店主的需求都得到了满足。然而，一般来说，我们要确定应该优化的目标是最终用户还是电子商务网站的拥有者。给我们付费的通常是店主，而不是最终用户。

对于企业级（公司、政府部门或学术机构）的内网搜索引擎来说，一个更相关的指标是用户生产率（user productivity），即用户用于查找所需信息所使用的时间。除此之外，这种场景下在实际中还有其他一些需要考虑的问题，比如 4.6 节中提到的信息的安全保密。

用户的满意度很难度量，这是为什么在标准评价中往往采用结果的相关性来进行度量的部分原因。常规的直接获得用户满意度的方法是进行用户调查，参与者的行为被观察记录，民族志式访谈[①]（ethnographic interview）技术被应用于获取定性的满意度信息。用户调查在系统设计中非常有用，但是非常费时费力。另外，想做好用户调查也很难，需要专家来设计用户调查并且对结果进行解释。因此，我们将不讨论这种基于人类可用性测试的细节。

8.6.3 对已有系统的改进

如果一个 IR 系统已经建好并正在被大量用户使用，系统的构建者可以对系统做些许改动，

① 民族志式访谈是社会科学特别是人类学和社会学中的一个属于，可以简单理解为一种进行用户调查研究的方法。——译者注

得到系统的多个改变版本，然后将其中一个版本和其他版本进行运行对比，通过记录能表示用户满意度的指标来对改动进行评价。这种方法常常被 Web 搜索引擎所采用。

该方法最常用的一个版本称为 A/B 测试（A/B testing），这是来自广告领域的一个术语。对于这样的一次测试来说，在现有系统和测试系统中只有一个因素不同，一小部分流量（1%～10%左右的用户）被随机导向测试系统，而大部分用户仍然使用现有系统。比如，如果我们想对排序算法的某个修改方案进行考察，我们会将部分用户导向测试系统，然后计算某个恰当的评价指标，如用户点击排名最高或第 1 页任意结果的频率。这种计算评价指标的方法称为点击日志分析（clickthrough log analysis）或点击流挖掘（clickstream mining）。关于这方面的内容，我们将在 9.1.7 节中介绍隐式相关反馈时再进一步讨论。

A/B 测试的基础是串行或并行地运行一系列单变量测试，即对每次测试，只有一个参数与现有运行系统不同。因此，很容易看出每个参数对系统的影响到底是正面的还是负面的。对运行系统进行这样的测试能够很容易也很廉价地测试出用户变化的效果，并且，在用户量基数足够大的情况下，即使是非常细微的正面或负面影响实际上也能够测量到。理论上说，同时随机变化多个因素并进行标准的多元统计分析（如多元线性回归）能获得更强的分析能力，但由于 A/B 测试更容易实现、更容易理解也更容易解释管理，它在实际中被广泛运用。

8.7　结果片段

对于某个查询，选择文档或对匹配上的文档进行排序以后，系统要将包含一定信息量的结果列表呈现给用户。很多情况下，用户并不对所有返回文档一一检查，因此，需要在结果列表中提供足够的信息，以便利用这些信息判断文档和自己需求的相关度，并根据这个相关度对结果进行最后的排序。[①]一个常规做法是提供结果片段（snippet），它是结果的一个短摘要，能够帮助用户确定结果的相关性。一般来说，结果片段包括文档的标题以及一段自动抽取的摘要。问题是如何设计这个摘要才能对用户提供最大的帮助。

常用的摘要包括两类：一类是静态（static）摘要，它永远保持不变，并不随查询变化而变化；另一类是动态（dynamic）摘要或基于查询（query dependent）的摘要，它主要根据查询所推导出的信息需求来进行个性化生成，并试图解释在给定查询下返回当前文档的原因。

静态摘要通常由文档的一部分文本或文档元数据单独或共同组成。一种最简单的摘要形式是取文档最开始的两句话或 50 个单词，或者是抽取文档的某些特定域（如标题或作者）。如果不抽取域信息，摘要也可以通过文档的元信息来得到。元信息是提供作者或日期的另一种有效方法，它也会包含设计摘要所需要的元素，比如 HTML Web 网页中 meta 元素的 description 元信息。静态摘要往往在索引构建时便已生成并放入缓存中，这样它们在检索时就能够快速返回

① 在强调召回率的场合存在例外情况。比如，在法律披露案例中，法律助理将会对基于关键词搜索出的所有文档进行一一浏览。

并显示，当然此时如果对实际的结果文档内容进行访问则相对耗时。

NLP（natural language processing，自然语言处理）领域中存在大量更好的文本摘要方法。大部分研究的目标仍然是从原始文档中选择部分句子，它们主要关注选择好句子的方法。相关的抽取模型通常将位置因素（比如重视文档的首段和末段以及段落中的首句和末句）和内容因素（强调包含关键词项的句子，这些词项在整个文档集中文档频率较低，但是在返回的单个文档中高频且具有良好的分布特性）综合在一起考虑。在更复杂的 NLP 方法中，系统可以通过自动全文生成方式或者对原文档中句子进行编辑或组合的方法来自动生成摘要句子。例如，可以删除一个比较从句或者将代词替换成它所代表的名词词组。这类通过句子生成的摘要方法仍然停留在研究阶段，很少用于搜索结果，实际上，从原始文档中抽取句子的方法更容易、更安全甚至更好。

动态摘要显示文档的一个或者多个“窗口”，其目的是想通过这些片段，能够让用户最方便地判断文档和信息需求是否相关。通常这些窗口会包含一个或者多个查询词项，因此也被称为基于关键词上下文（keyword-in-context，简称 KWIC）的结果片段（如图 8-5 所示）。当然有时候像静态摘要中的情况一样，该结果片段仍然可能是一段与查询无关的信息（如标题）。动态摘要往往通过评分来产生。如果查询就是一个短语，文档中该短语的多次出现将会显示在摘要中。如果查询不是一个短语，文档中包含多个查询词项的窗口将会被选出。一般地，选出的窗口往往是从查询词项左右两边抽取一些词来组成的。这时就可以使用 NLP 技术，这是因为用户希望看到阅读起来更通畅的包含完整短语的结果片段。

> … ***In recent years, Papua New Guinea has faced severe economic difficulties and*** economic growth has slowed, partly as a result of weak governance and civil war, and partly as a result of external factors such as the Bougainville civil war which led to the closure in 1989 of the Panguna mine (at that time the most important foreign exchange earner and contributor to Government finances), the Asian financial crisis, a decline in the prices of gold and copper, and a fall in the production of oil. ***PNG's economic development record over the past few years is evidence that*** governance issues underly many of the country's problems. Good governance, which may be defined as the transparent and accountable management of human, natural, economic and financial resources for the purposes of equitable and sustainable development, flows from proper public sector management, efficient fiscal and accounting mechanisms, and a willingness to make service delivery a priority in practice. …

图 8-5 一个动态结果片段产生的例子。该结果片段对应于查询 new guinea economic development，从文档中选出的结果片段在图中用粗斜字体表示

通常认为，动态摘要能够大大提高 IR 系统的可用性，但是随之而来也会增加系统设计的复杂性。动态摘要不能事先计算，而且，如果系统仅仅包含位置索引，那么便不容易从搜索引擎命中结果中抽取上下文来生成这样的动态摘要。这是使用静态摘要的原因之一。在目前大容量硬盘成本低廉的情况下，解决上述问题的一个常规做法是，在构建索引时利用这些磁盘在本地缓存所有的文档（尽管这种做法会引起包括法律、信息安全和控制方面的目前还远远无法解决的问题），参考图 7-5。于是，系统能够通过简单地扫描可能出现在结果列表中的文档来抽取包含查询词的

结果片段。除了简单地扫描文本之外，生成一段好的 KWIC 结果片段还需要注意一些细节。给定文档中关键词的一系列出现信息，生成动态摘要的目标是选出满足如下条件的片段：

(i) 在文档中最大限度地包括这些词项的信息；

(ii) 内容足够完整，方便用户阅读理解；

(iii) 足够短，满足摘要在空间上的严格限制。

由于系统要对每个查询生成多个结果片段，所以结果片段的生成速度一定要快。比缓存整篇文档更好的方法是，只缓存一段内容丰富但又有固定大小的文档前缀，比如前 10 000 个字符。对于大多数一般的短文档而言，这样实际上缓存了整篇文档，但是对大文档来说就节省了大量的本地存储资源。超过固定前缀长度的文档在产生动态摘要时只基于文档前缀来实现，而通常来说，该前缀对于查找文档摘要来说无论如何都是非常重要的域。

如果文档自上次爬虫和索引器处理之后已经被更新，那么这些变化既不会体现在缓存也不会体现在索引中。这种情况下，索引和摘要都不会精确地反映出当前文档的内容，但是摘要和实际文档内容之间的差异对用户感觉来说会更明显。

8.8　参考文献及补充读物

查询相关性的定义和实现从 1953 年一开始就困难重重。Swanson（1988）中报告了当年两个组之间的评价情况，对于 98 个问题，双方一致认为有 1 390 个文档是明显相关，但是有 1 577 篇文档双方的意见并不一致，而且这种不一致始终没有得到解决。

严格的 IR 形式化测试最早完成于 20 世纪 50 年代末开始的 Cranfield 实验。对 Cranfield 测试集及实验的回顾和讨论参见 Cleverdon（1991）。其他有影响的早期 IR 实验基于 Gerard Salton 及其同事开发的 SMART 系统（Salton 1971b, 1991）。有关 TREC 评价过程的详细介绍参见 Voorhees 和 Harman（2005）。也可以直接通过网站 http://trec.nist.gov/获得 TREC 的相关信息。一开始，很少有研究者会计算其实验结果的统计显著性，但是 IR 社区逐渐增加了对这一点的要求（Hull，1993）。利用用户调查评价 IR 系统的效果开始于最近几年（Saracevic 和 Kantor，1988, 1996）。

召回率和正确率的概念最早用于 Kent 等人（1955），但是当时并没有使用 precision 这个术语。F 值（或者它的补 $E = 1 - F$）由 van Rijsbergen（1979）引入，他不仅给出了一个详尽的理论介绍，而且还介绍了如何通过边缘相关性原则（到某个点之后，用户不愿意牺牲哪怕是一点点的正确率来获得召回率的一点点增长）导出“采用调和平均来组合正确率和召回率才合理”的结论（因此基于上述原则而不是采用最小值或者几何平均值）。

Buckley 和 Voorhees（2000）比较了多个评价指标，包括 P@k、MAP 和 R-precision，并且评估了它们的错误率。在 TREC HARD 任务（Allan，2005）中，采用 R-precision 作为正式的评价指标。Aslam 和 Yilmaz（2005）发现 R-precision 与 MAP 关系非常相近，在一些早些的研究中也提到这一点（Tague-Sutcliffe 和 Blustein 1995；Buckley 和 Voorhees，2000）。Chris Buckley

的 trec_eval 程序是 IR 系统的一个标准评价程序，它能够计算有序检索的多种效果评价指标，下载地址为 http://trec.nist.gov/trec_e val/。

Kekäläinen 和 Järvelin（2002）提出在大规模文档集中多级的相关度判断更具优势，然后在这种场景下，Järvelin 和 Kekäläinen（2002）引入了基于增益的 IR 系统的评价方法。Sakai（2007）研究了基于多级相关度判断的评价指标的稳定性和敏感性，最后的结论是 NDCG 是评价结果文档排序的最好的指标。

Schamber 等人（1990）考察了相关度的概念，强调它的多维性及上下文相关性，但同时也认为它能被有效测定。Voorhees（2000）是一篇有关 TREC Ad Hoc 任务中相关性判定差异及其对检索系统得分与排序影响的标准文献。Voorhees 的结论是，尽管得分值会有所变化，但是系统之间的排序却非常稳定。Hersh 等人（1994）给出了一个基于医学 IR 文档集的很类似的分析结果。与之相反，Kekäläinen（2005）分析了一些后来的 TREC 会议，它使用了 4 级的相关性判定并提出了累积增益的概念，最终认为所使用的相关性指标会充分地影响系统间的排名。有关内容也可以参考 Harter（1998）。Zobel（1998）主要考察采用 TREC 中使用的缓冲池方法得到的相关性评价结果是否公平可靠，最后的结论是肯定的。

kappa 统计量及其他在语言相关目标中的应用情况参见 Carletta（1996）。一些常规文献中（如 Siegel 和 Castellan，1988）采用缓冲池方法来计算随机一致性，但是 Di Eugenio 和 Glass（2004）采用了非缓冲区计算一致性的方法（尽管也许给出了多个指标）。对于一致性计算的其他可能实际上更好的指标参见文献 Lombard 等人（2002）和 Krippendorff（2003）。

文本摘要技术的研究很多年来一直都很活跃。有关句子选择的现代工作起始于 Kupiec 等人（1995）。最近的一些工作包括（Barzilay 和 Elhadad，1997）与（Jing，2000），还包括在每年的 DUC 及其他 NLP 相关会议上的一系列工作。Tombros 和 Sanderson（1998）指出了在 IR 场景下使用动态摘要的优点。Turpin 等人（2007）讨论了如何高效生成结果片段的方法。

点击日志分析相关的研究参见 Joachims（2002b）及 Joachims 等人（2005）。

在一系列论文中，Hersh、Turpin 及他们的同事指出，基于多个批处理实验进行评价时，采用形式化指标得到的检索系统效果的提高并不能完全直接对应用户体验的提高（Hersh 等人 2000a，2000b，2001；Turpin 和 Hersh，2001，2002）。

IR 界面及与人相关的一些因素（如用户信息查找的模型、可用性测试等）不在本书讨论范围之内。有关这些话题的介绍可以参见其他教材，包括 Baeza-Yates 和 Ribeiro-Neto（1999，第 10 章）、Korfhage（1997）以及与认知相关的文献（如 Spink 和 Cole，2005）。

第9章 相关反馈及查询扩展

在大多数文档集中，同一概念可以用不同的词来表达，这个现象称为一义多词（synonymy），它会对大部分信息检索系统的召回率产生影响。比如，输入查询aircraft时我们希望能找到包含plane的文档，当然，这里的plane指的是飞机（airplane），而不是木工刨（woodworking plane）。另外，我们也希望在查找thermodynamics时能够与特定环境下的heat匹配上。为了解决此类问题，用户常常通过手工对初始查询进行修改，这部分内容已在 1.4 节中有所讨论。本章主要讨论系统中进行查询优化（query refinement）[①]的各种方法，包括全自动的方法和用户参与的方法。查询优化的方法主要可以分成两类：全局方法和局部方法。全局方法指的是在不考虑查询及其返回文档情况下对初始查询进行扩展和重构的方法，因此，扩展后查询中的用词变化会使得该查询与其他语义相近的查询词项相匹配。这些全局方法包括：

- 基于同义词词典（thesaurus）[②]或WordNet的查询扩展或重构方法（参见 9.2.2 节）；
- 自动构造同义词词典并基于它进行查询扩展（参见 9.2.3 节）；
- 类似拼写校正的技术（参见第 3 章）。

而局部方法则通过查询的初始匹配文档对原始查询进行修改，基本方法包括：

- 相关反馈（参见 9.1 节）；
- 伪相关反馈，也称为盲相关反馈（blind relevance feedback）（参见 9.1.6 节）；
- （全局）间接相关反馈（参见 9.1.7 节）。

本章将会介绍上面提到的各种方法，但是主要关注相关反馈技术，它是最常用也最成功的方法之一。

9.1 相关反馈及伪相关反馈

RF（*relevance feedback*，相关反馈）的主要思想是，在信息检索的过程中通过用户交互来提高最终的检索效果。具体来说，用户会对初次检索结果的相关性给出反馈意见，其基本的过

① 也有人翻译成查询修正、查询改进、查询精化或查询细化等等。也称查询点移动(query point movement)。——译者注

② 也有人翻译为分类词典或类属词典等。在图书情报领域，thesaurus 也通常称为叙词表。这里采用同义词词典这个更常用的称呼。需要指出的是，thesaurus 中不仅包括同义词，也包括反义、近义及其他类属关系。——译者注

程包括：

- 用户提交一个简短的查询；
- 系统返回初次检索结果；
- 用户对部分结果进行标注，将它们标注为相关或不相关；
- 系统基于用户的反馈计算出一个更好的查询来表示信息需求；
- 利用新查询系统返回新的检索结果。

相关反馈过程可以按照上面的方式进行多次循环。上述过程主要利用了如下思想：当用户对文档集并不十分了解时，构造一个好的查询很困难，但是让用户来判断具体文档的相关性却是比较容易的。因此，按照上述方式进行查询的反复优化是非常有意义的。这种场景下，RF 对于跟踪用户信息需求的变化也是相当有效的，这是因为用户看到某些文档之后可能会使他们对原来所理解的信息需求进行修正。

图像搜索是一个使用相关反馈的很好的例子。这是因为在图像搜索中返回的结果非常直观，而且用户不容易用词语来表达其需求，但是却很容易标识相关和不相关的图像结果。在 http://nayana.ece.ucsb.edu/imsearch/imsearch.html 所给出的演示系统中，用户输入 bike 之后，系统会返回图 9-1a 所示的初始结果，然后用户可以从中标识出部分相关结果。这些信息可以用于修改原始的查询，而其他没有标识的结果对查询重构不起任何作用。图 9-1b 中给出了经过查询重构之后新的排名靠前的结果。

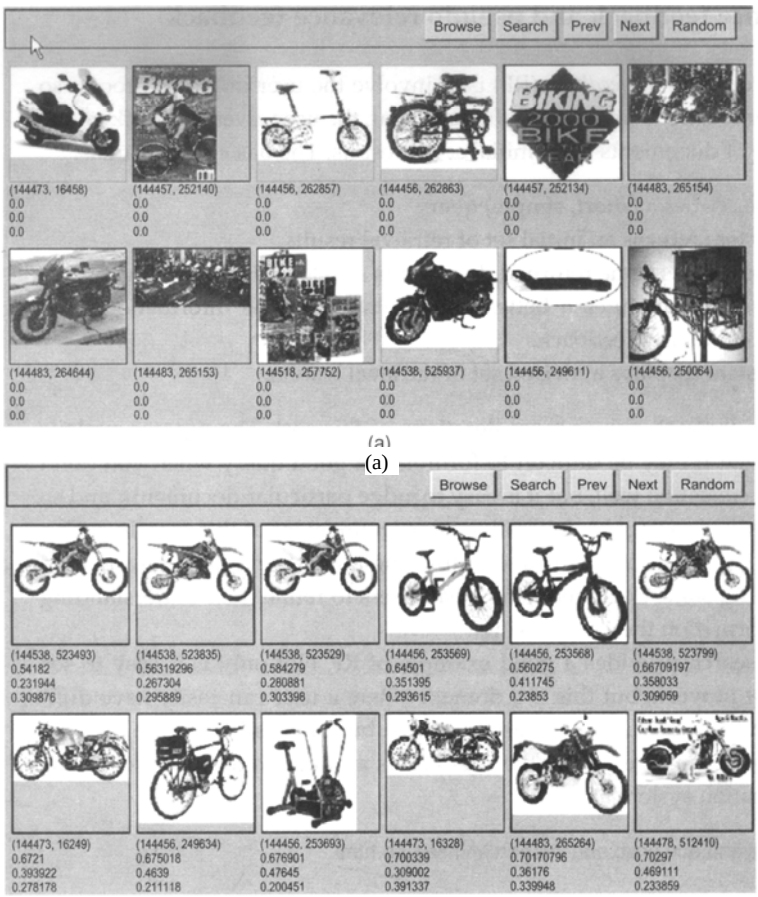


图 9-1　图像检索中的相关反馈过程。(a) 用户对查询 bike 的初始检索结果进行浏览，选择上面一行中的第 1、3、4 个图像及下面一行中的第 4 个图像作为相关图像，并提交该反馈信息。(b) 提交反馈后的检索结果，其正确率得到显著提高。来自 http://nayana.ece.ucsb.edu/imsearch/imsearch.html（Newsam 等人，2001）

图 9-2 给出了一个文本检索中相关反馈的例子，其中用户希望查找与 new applications of space satellites 相关的信息。

(a) Query: New space satellite applications

(b)
\+ 1. 0.539, 08/13/91, NASA Hasn't Scrapped Imaging Spectrometer
\+ 2. 0.533, 07/09/91, NASA Scratches Environment Gear From Satellite Plan
3. 0.528, 04/04/90, Science Panel Backs NASA Satellite Plan, But Urges Launches of Smaller Probes
4. 0.526, 09/09/91, A NASA Satellite Project Accomplishes Incredible Feat: Staying Within Budget
5. 0.525, 07/24/90, Scientist Who Exposed Global Warming Proposes Satellites for Climate Research
6. 0.524, 08/22/90, Report Provides Support for the Critics Of Using Big Satellites to Study Climate
7. 0.516, 04/13/87, Arianespace Receives Satellite Launch Pact From Telesat Canada
\+ 8. 0.509, 12/02/87, Telecommunications Tale of Two Companies

(c)
2.074 new 15.106 space
30.816 satellite 5.660 application
5.991 nasa 5.196 eos
4.196 launch 3.972 aster
3.516 instrument 3.446 arianespace
3.004 bundespost 2.806 ss
2.790 rocket 2.053 scientist
2.003 broadcast 1.172 earth
0.836 oil 0.646 measure

(d)
* 1. 0.513, 07/09/91, NASA Scratches Environment Gear From Satellite Plan
* 2. 0.500, 08/13/91, NASA Hasn't Scrapped Imaging Spectrometer
3. 0.493, 08/07/89, When the Pentagon Launches a Secret Satellite, Space Sleuths Do Some Spy Work of Their Own
4. 0.493, 07/31/89, NASA Uses 'Warm' Superconductors For Fast Circuit
* 5. 0.492, 12/02/87, Telecommunications Tale of Two Companies
6. 0.491, 07/09/91, Soviets May Adapt Parts of SS-20 Missile For Commercial Use
7. 0.490, 07/12/88, Gaping Gap: Pentagon Lags in Race To Match the Soviets In Rocket Launchers
8. 0.490, 06/14/90, Rescue of Satellite By Space Agency To Cost $90 Million

图 9-2 在某个文本集上进行相关反馈的例子。(a) 初始查询；(b) 用户用+号标识了一些相关文档；(c) 查询扩展成 18 个带权重的词项；(d) 再次检索的结果。*号标记的是那些在相关反馈时用户标识出的相关文档

9.1.1 Rocchio 相关反馈算法

Rocchio 算法是相关反馈实现中的一个经典算法，它提供了一种将相关反馈信息融入到向量空间模型（参见 6.3 节）的方法。

基本理论（underlying theory）。假定我们要找一个最优查询向量 $\vec{q}$，它与相关文档之间的相似度最大且同时又和不相关文档之间的相似度最小。若C_r表示相关文档集，C_{nr}表示不相关文

档集，那么我们希望找到的最优的 $\vec{q}$ 是[①]

$$\vec{q}_{opt} = \arg\max_{\vec{q}}[sim(\vec{q}, C_r) - sim(\vec{q}, C_{nr})]。 \quad (9\text{-}1)$$

其中，*sim* 函数采用公式(6-10)中的定义。采用余弦相似度计算时，能够将相关文档与不相关文档区分开的最优查询向量为

$$\vec{q}_{opt} = \frac{1}{|C_r|}\sum_{\vec{d}_j \in C_r}\vec{d}_j - \frac{1}{|C_{nr}|}\sum_{\vec{d}_j \in C_{nr}}\vec{d}_j 。 \quad (9\text{-}2)$$

这就是说，最优的查询向量等于相关文档的质心向量和不相关文档的质心向量的差（参见图 9-3）。然而，这个发现并没有什么意义，因为检索本来的目的就是要找相关文档，而所有的相关文档集事先却是未知的。

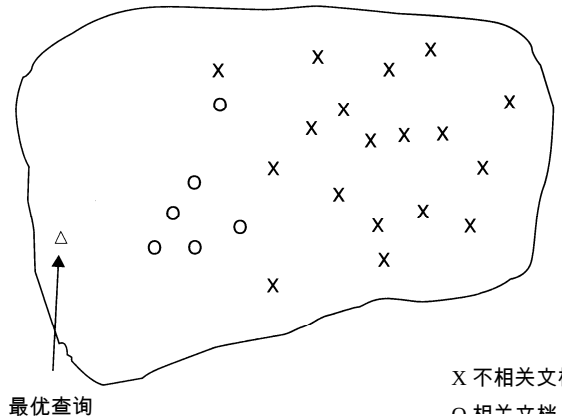


图 9-3　Rocchio 方法中将相关和不相关文档区分开的最优查询

Rocchio 算法。该算法（Rocchio，1971）是 20 世纪 70 年代左右在 Salton 的 SMART 系统中引入并广泛流传的一种相关反馈算法。在一个真实的信息检索场景中，假定我们有一个用户查询，并知道部分相关文档和不相关文档的信息，则可以通过如下公式得到修改后的查询向量 $\vec{q}_m$：

$$\vec{q}_m = \alpha\vec{q}_0 + \beta\frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}}\vec{d}_j 。 \quad (9\text{-}3)$$

其中，$\vec{q}_0$ 是原始的查询向量，D_r 和 D_{nr} 是已知的相关和不相关文档集合。α、β 及 γ 是上述三者的权重。这些权重能够控制判定结果和原始查询向量之间的平衡：如果存在大量已判断的文档，那么会给 β 及 γ 赋予较高的权重。修改后的新查询从 $\vec{q}_0$ 开始，向着相关文档的质心向量靠近了一段距离，而同时又与不相关文档的质心向量远离了一段距离。新查询可以采用常规的向量空间模型（参见 6.3 节）进行检索。通过减去不相关文档的向量，我们很容易保留向量空间的正值分量。在 Rocchio 算法中，文档向量中的权重分量如果为负值，那么该分量将

① 公式中，$\arg\max_x f(x)$ 返回使 $f(x)$ 取最大值时的 x，类似地，$\arg\min_x f(x)$ 返回使 $f(x)$ 取最小值时的 x。

会被忽略，也就是说，此时会将该分量权重设为 0。图 9-4 给出了应用相关反馈技术的效果示意图。

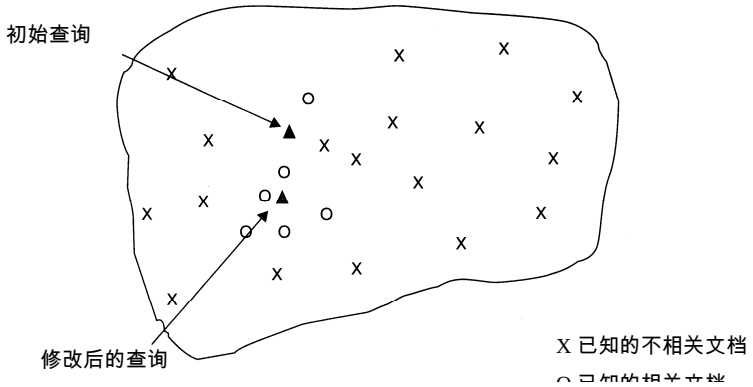


图 9-4 Rocchio 算法的一个应用示意图。一些文档被标记为相关或不相关，基于这些信息，初始查询向量会在反馈作用下移动

相关反馈可以同时提高召回率和正确率。然而，实际表明该技术在一些重召回率的场景下对于提高召回率非常有用。这其中的部分原因在于它对查询进行了扩展，另一个原因是应用的场景所带来的结果：在期望高召回率的情况下，可以预计用户可能会花更多时间来浏览结果并进行反复搜索。正反馈往往比负反馈更有价值，因此在很多IR系统中，会将参数设置成$\gamma < \beta$。一个合理的取值是$\alpha = 1$、$\beta = 0.75$ 及 $\gamma = 0.15$。实际上，很多系统，包括前面图 9-1 提到的图像搜索系统，都只允许进行正反馈，即相当于设置$\gamma = 0$。还有一种做法是，只取检索系统返回结果中排名最高的标记为不相关的文档进行负反馈，此时，公式（9-3）中的$|D_{nr}| = 1$。尽管上述相关反馈方法存在各种变形，并且很多比较实验也没有取得一致性的结论，但是一些研究却认为一种称为*Ide dec-hi* 的公式[①]最有效或至少在性能上表现最稳定。

习题 **9-1** 在什么情况下，公式（9-3）中修改后的查询 q_m 会等于原始查询 q_0? 在所有其他的情况下，q_m 是否比 q_0 更靠近相关文档的质心？

习题 **9-2** 为什么在 IR 系统中，正反馈可能比负反馈作用更大？为什么只用一篇不相关文档进行负反馈会比用多篇不相关文档的效果要好？

习题 **9-3** 假定用户的初始查询是 cheap CDs cheap DVDs extremely cheap CDs。用户查看了两篇文档 d_1 和 d_2，并对这两篇文档进行了判断：包含内容 CDs cheap software cheap CDs 的文档 d_1 为相关文档，而内容为 cheap thrills DVDs 的文档 d_2 为不相关文档。假设直接使用词项的频率作为权重（不进行归一化也不加上文档频率因子），也不对向量进行长度归一化。采用公式（9-3）进行 Rocchio 相关反馈，请问修改后的查询向量是多少？其中 $\alpha = 1$，$\beta = 0.75$，$\gamma = 0.25$。

① *Ide dec-hi* 公式为 $\vec{q}_m = \vec{q}_0 + \sum_{\vec{d}_j \in D_r} \vec{d}_j - \arg\max_{\vec{d} \in D_{nr}} sim(\vec{d}, \vec{q}_0)$ 。

习题 **9-4** [*]　Omar 实现了一个带相关反馈的 Web 搜索系统，并且为了提高效率，系统只基于返回网页的标题文本进行相关反馈。用户对结果进行判定，假定第一个用户 Jinxing 的查询是

banana slug

返回的前三个网页的标题分别是：

banana slug Ariolimax columbianus

Santa Cruz mountains banana slug

Santa Cruz Campus Mascot

Jinxing 认为前两篇文档相关，而第 3 篇文档不相关。假定 Omar 的搜索引擎只基于词项频率（不包括长度归一化因子和 IDF 因子）进行权重计算，并且假定使用 Rocchio 算法对原始查询进行修改，其中 $\alpha=\beta=\gamma=1$。请给出最终的查询向量（按照字母顺序依次列出每个词项所对应的分量）。

9.1.2　基于概率的相关反馈方法

如果用户告诉系统一些相关和不相关文档，那么此时我们就可以通过建立分类器而不是修改查询向量的权重来进行相关反馈。一种实现分类器的方法是采用朴素贝叶斯概率模型。假设 R 是一个布尔变量，用于表示文档的相关与否，那么我们就可以根据文档的相关性，来估计词项 t 出现在该文档中的概率，即

$$\begin{aligned}&\hat{P}(x_t=1\mid R=1)=|VR_t|/|VR|，\\&\hat{P}(x_t=0\mid R=0)=(\mathrm{d}f_t-|VR_t|)/(N-|VR|)。\end{aligned}\tag{9-4}$$

其中，N 是文档的总数，$\mathrm{d}f_t$ 是包含 t 的文档数目，VR 是已知的相关文档集，VR_t 是 VR 中包含 t 的文档子集。尽管已知的相关文档集可能只是所有的相关文档集的一个非常小的子集，但是如果假定所有相关文档集只是所有文档集的一个很小的子集，那么上述估计是合理的。这为词项权重的修改提供了另外一条途径。我们将在第 11 章和第 13 章进一步讨论这种基于概率的方法，特别地，我们将在 11.3.4 节介绍其在相关反馈中的应用。当前我们会看到，仅仅利用公式（9-4）来计算词项的权重很可能是不够的。因为它只考虑了词项在所有文档集上的统计信息以及其在已判定的相关文档中的分布信息，而并没有考虑原始查询的作用。

9.1.3　相关反馈的作用时机

相关反馈的成功依赖于某些假设。第一，用户必须要有足够的知识来建立一个不错的初始查询，该查询至少要在某种程度上接近需求文档。无论如何，这个假设对于基本的信息检索的成功来说都是必需的，但是，单独使用相关反馈技术并不能解决某些问题，认识到这一点非常重要。单独使用相关反馈技术并不足够的情况包括以下几种。

- 拼写错误。如果用户输入的查询词项的拼写方式和文档中不一致的话，那么相关反馈技术也很难取得好的效果。这种情况下，可以通过第 3 章提到的拼写校正技术来解决。
- 跨语言 IR。采用另外一种语言表达的文档在基于词项分布的向量空间上并不靠近原始查

询。实际上，同一语言的文档之间距离可能更近。

- 用户的词汇表和文档集的词汇表不匹配。比如用户想查找 laptop，但是所有文档都用另外一个说法 notebook computer，这样的话，原始查询将会失败，这种情况下相关反馈技术就很可能无效。

第二，相关反馈方法要求相关文档之间非常相似。也就是说，它们应该能聚成一团。理想情况下，所有相关文档中的词项分布应该与用户标出的相关文档中的词项分布相似，而同时所有不相关文档中的词项分布与相关文档中的词项分布差别很大。若所有相关文档都能紧密地聚在单个原型（prototype）周围时情况会更好，或至少存在多个不同原型，或者相关文档的词汇表之间的重合度很大，而与不相关文档的相似度较小。Rocchio 相关反馈模型通过计算簇质心向量，隐式地将相关文档看成单个簇。如果相关文档包括多个不同子类，即它们在向量空间中可以聚成多个簇，那么 Rocchio 方法效果会不太好。这种问题可能发生的情形如下：

- 文档子集之间使用不同的词汇表，如 Burma 与 Myanmar；
- 某个查询的答案集合本身就需要不同类的文档来组成，如 Pop stars who once worked at Burger King
- 通用概念的例子，它通常以多个具体概念的或关系来出现，如 felines。

文档集当中精心编辑的内容往往能够对上述问题提供解决方案。比如，一篇有关不同群体对缅甸（Burma）局势的态度的文章可能会引入不同群体所使用的不同术语，从而能把不同文档关联起来。

对于用户来说，相关反馈并不一定很受欢迎。用户往往不愿意进行显式反馈，或者通常来说，用户不希望延长搜索交互时间。更进一步来说，在应用相关反馈之后返回某个特定文档的原因往往更难理解[①]。

相关反馈可能还会存在一些实际中的问题。直接应用相关反馈技术会产生长查询，这对常规的 IR 系统来说会降低系统的效率。长查询会引起检索实现时更高的计算开销，从而导致系统对用户的应答时间更长。一个部分解决方案是只改变相关文档中某些关键词项（比如文档中词项频率最高的 20 个词项）的权重。一些实验结果也表明，像上面一样采用有限个词项可以获得更好的效果（参见 Harman，1992），但是也有些其他工作却认为，使用更多的词项会提高检索文档的质量（参见 Buckley 等人 1994b）。

9.1.4 Web 上的相关反馈

一些 Web 搜索引擎提供“相似或相关网页”（similar/related pages）的功能：用户可以从结果集中选出某个文档作为满足其信息需求的一个样例，并以此为起点寻找更多的与此相似的的

① 这段话可以这样来理解，用户输入初始查询时至少还知道输入了什么查询，所以他们对为什么返回初次检索结果还能理解。而修改后的查询往往对用户来说都是不可知的，因此，二次检索返回某个结果的原因可能更难以理解。——译者注

文档。这个功能可以看成相关反馈技术的一个简单应用。然而，一般而言，在 Web 搜索中很少使用相关反馈技术。Excite 搜索引擎是个例外，它一开始就提供了完整的相关反馈，不过，由于很少有人用，该功能后来及时被取消。在 Web 上，很少有人会用到高级搜索界面，而且大部分人都希望在一次交互中完成搜索任务。此外，相关反馈技术在 Web 上很少利用也可能反映出其他两个因素：一是相关反馈很难向普通用户解释清楚；二是相关反馈主要用来提高召回率，而 Web 搜索用户很少关注是否获得足够的召回率。

Spink 等人（2000）给出了在 Excite 搜索引擎中相关反馈技术使用的结果。在所有用户的查询会话（query session）中，只有 4%的会话使用了相关反馈功能，这大都是通过点击每个结果后面的“ More like this”链接来实现的。大约 70%的用户仅仅浏览了第一页的结果，对于使用相关反馈技术的用户来说，当时的检索效果大约提高了 2/3。

最近与此相关的重要工作是基于点击流数据（clickstream data，即用户点击的链接所构成的数据）来提供间接相关反馈。有关该数据使用的研究细节参见（Joachims，2002b；Joachims 等人，2005）。Web 上链接结构（参见第 21 章）的成功使用也可以看成是一种隐式相关反馈，只不过和用户点击数据不同的是，这里的反馈信息由网页的制作者而不是阅读者来提供（尽管在实际中，大部分制作者同时也是阅读者）。

9.1.5　相关反馈策略的评价

交互式相关反馈能够给检索性能带来实质性的提高。从经验上说，一轮相关反馈常常很有用，两轮反馈有时也会勉强有用。相关反馈的成功使用需要足够多的已判定文档，否则反馈过程会不稳定，甚至可能会偏离用户的信息需求。因此，一般建议至少需要有 5 篇已判定文档。

如何对相关反馈的效果进行合理而有效的评价，这里面存在一些玄机。一个明显的策略就是，首先计算出原始查询 q_0 的正确率–召回率曲线，一轮相关反馈之后，我们计算出修改后的查询 q_m 并再次计算出新的正确率–召回率曲线。这样，反馈前与反馈后我们都可以在所有文档上对结果进行评价，然后直接进行比较。如果这样做，我们会发现相关反馈会给检索效果带来令人瞩目的提升，以 MAP 指标为例，一般会有 50%左右的提高。但是，遗憾的是，这种结果极具欺骗性。由于用户判定了一部分相关文档，所以结果提高的部分原因是这些已知的相关文档的排名得到了提高。为了实现评价的公平性，我们只对用户没有看过的文档进行评价。

第二种思路是利用剩余文档集（residual collection，所有文档集中除去用户判定的相关文档后的文档集）对反馈后的结果进行评价。这种思路看上去更具现实性。不过，性能的度量结果往往低于原始查询的结果。如果相关文档数目本来就很少的话就更是如此，此时用户已经在第一次检索结束后对相当比例的相关文档进行了判断。不同相关反馈技术之间的相对效果可以采用剩余文档集的方法进行有效的比较，但是很难通过这种方法对是否采用相关反馈技术本身进行比较，这是因为文档集大小（此时指原始文档集变成剩余文档集）及相关文档数目在反馈前后会发生改变。

因此，上述的方法不能完全令人满意。第三种方法是给出两个文档集，一个用于初始查询和相关性判定，另一个用于比较和评价。因此，q_0 和 q_m 都可以在后一个文档集上进行有效对比。

对于相关反馈的作用，最好的评价方法或许是进行用户调查，特别是是采用一种基于时间的比较方法：和采用其他方法（如查询重构技术）相比，用户采用相关反馈技术找到相关文档的时间是否更短？或者说，在一个固定的时间内用户能否找到更多的相关文档？这些代表用户效用性的指标最公平，也更贴近真实的应用。

9.1.6 伪相关反馈

伪相关反馈（pseudo relevance），也称为盲相关反馈（blind relevance feedback），提供了一种自动局部分析的方法。它将相关反馈的人工操作部分自动化，因此用户不需要进行额外的交互就可以获得检索性能的提升。该方法首先进行正常的检索过程，返回最相关的文档构成初始集，然后假设排名靠前的 k 篇文档是相关的，最后在此假设上像以往一样进行相关反馈。

这种自动的伪相关反馈技术大部分情况下会起作用，有证据表明它比将要在 9.2 节提到的全局分析的效果要好。在 TREC Ad Hoc 任务中也发现它能提高检索的性能（参见图 9-5 中的例

	Precision at $k = 50$	
词项权重计算	无反馈	伪反馈
lnc.ltc	64.2%	72.7%
Lnu.ltu	74.2%	87.0%

图 9-5 通过伪相关反馈大幅度提高结果性能的例子。上述结果摘自康奈尔大学利用 SMART 系统在 TREC 4 上的实验（Buckley 等人 1995），其中同时对比了两种长度归一化方式的结果（L 方式和 l 方式，其定义参见图 6-15）。在进行伪相关反馈时每个查询增加 20 个词项

子）。当然，它不可能完全避免自动化操作所带来的风险。比如，查询关于“铜矿开采”（copper mines）的信息，返回前面的多篇文档都与“智利的开采”（mines in Chile）有关，那么进行伪相关反馈后查询会向“智利”（Chile）相关主题漂移。

9.1.7 间接相关反馈

在反馈过程中，我们也可以利用间接的资源而不是显式的反馈结果作为反馈的基础。这种方法也常常称为隐式相关反馈（implicit relevance feedback）。隐式反馈不如显式反馈可靠，但是会比没有任何用户判定信息的伪相关反馈更有用。此外，尽管用户往往不愿意提供显式相关反馈，但是在一个如 Web 搜索引擎一样的具有高访问量的系统中，收集用户的大量隐式反馈信息是十分容易的。

在 Web 上，DirectHit 引入一种文档排序的思路，即对于某文档，如果用户浏览的次数越多，那么它的排名也越高。换句话说，这里假设了用户对链接的点击能够反映出该页面的相关性。这种方法基于很多假设，比如结果列表中的文档摘要片段能够为用户判定文档的相关与否提供提示信息。在最初的 DirectHit 搜索引擎中，页面点击率数据的收集采用的是全局方法，而不是

基于用户或者查询来分别收集。这实际是点击流挖掘（clickstream mining）这个通用领域的一种形式。目前，一个非常相关的方法用于与 Web 查询相匹配的广告排序（参见第 19 章）。

9.1.8　小结

相关反馈技术能够有效提高检索结果的相关性，这一点已被证明。它的成功使用要求查询的相关文档数目中等或者很大。对于用户而言，直接的相关反馈常常费时费力，而且它在很多 IR 系统中的实现效率都不太高。很多情况下，存在很多其他的交互式检索方法，它们能够花更少的工作量来获得相当水平的检索相关性的提高。

除了在核心的 ad hoc 检索场景下使用，其他使用相关反馈技术的场景还包括：

- 跟踪信息需求的改变（比如，感兴趣的车型会随时间改变）；
- 维护信息过滤器（如新闻过滤器），这些过滤器将在第 13 章进一步讨论；
- 主动学习（即确定类别中的哪些样本最有用，以减少标注开销）。

? 习题 **9-5**　对应于搜索引擎中的“类似网页”（“Find pages like this one”功能），Rocchio 算法中的系数α、β、γ 应该如何设置？

习题 **9-6** [*]　给出相关反馈技术很少在 Web 搜索中使用的 3 个原因。

9.2　查询重构的全局方法

本节将简要介绍 3 种用于扩展查询的全局方法，它们分别是：简单辅助用户进行查询扩展、采用人工词典的方法及自动构建词典的方法。

9.2.1　查询重构的词汇表工具

在搜索过程中，有很多用户支持工具能够帮助用户来理解为什么他们的搜索成功或不成功。这包含停用词表中的查询词的省略信息、词干还原的结果、每个词项或短语的结果命中数目以及词语是否动态转换为短语的信息。信息检索系统也可以通过同义词典或者受控词汇表来推荐查询词项，或者可以允许用户浏览倒排索引中的词项列表从而找出文档集中的较好词项。

9.2.2　查询扩展

在相关反馈中，通过判定文档相关和不相关，用户会给文档以额外的输入信息，该信息可以用于对查询词项进行权重的重新调整。另一方面，在查询扩展（query expansion）中，用户会对查询词或短语给出额外输入信息，比如可能推荐额外的查询词项。一些搜索引擎特别是 Web 搜索引擎会给出相关的推荐查询，然后用户可以选择其中的某个推荐查询进行搜索。图 9-6 给出了 Yahoo! 搜索引擎中一个查询推荐的例子。这种形式的查询扩展的核心问题是如何生成或者扩展出新的查询。最普遍的查询扩展方法是通过某种形式的词典进行全局分析。对于查询中的

每个查询词项 t，可以通过在词典中找出 t 的同义词或者相关词对查询进行自动扩展。同义词词典的使用可以与词项的权重计算相结合，比如对增加的查询词项赋予一个低于原始查询词项的权重。

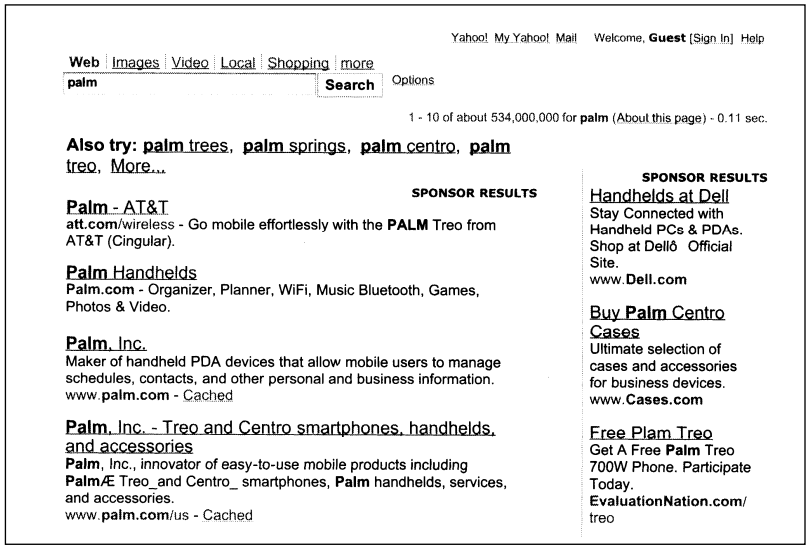


图 9-6 2008 年 Yahoo!搜索引擎界面上的一个查询扩展的例子，扩展的推荐查询出现在“ Search Results” 搜索条下

用于查询扩展的同义词词典的构建方法如下。

- 使用人工编辑的一部受控词汇表。这里，对每个概念都有一个规范的词项来表示。传统图书馆中主题索引中的主题标题，比如，美国国会图书馆分类法（Library of Congress Subject Headings） 或杜威十进分类（Dewey Decimal Classification）系统都是受控词汇表的例子。在资源充分的领域，受控词汇表的使用是非常普遍的。一个著名的例子是连同 Medline 一起用于查询生物医学研究文献的 UMLS 系统（Unified Medical Language System）。例如，在图 9-7 中，查询 cancer 时会被加上词 neoplasms。在 Medline 中的这种查询扩展也可以与图 9-6 中 Yahoo!的例子进行比较。Yahoo!提供的是一个交互式的查询扩展方法，而 PubMed 进行的是自动的查询扩展。如果用户不检查最终提交的查询，他们甚至不会知道已经进行了查询扩展。

- User query: cancer
- PubMed query: (“neoplasms”[TIAB] NOT Medline[SB]) OR “neoplasms”[MeSH Terms] OR cancer[Text Word]
- User query: skin itch
- PubMed query: (“skin”[MeSH Terms] OR (“integumentary system”[TIAB] NOT Medline[SB]) OR “integumentary system”[MeSH Terms] OR skin[Text Word]) AND ((“pruritus”[TIAB] NOT Medline[SB]) OR “pruritus”[MeSH Terms] OR itch[Text Word])

图 9-7 基于 PubMed 词典的查询扩展的例子。当用户在 PubMed 界面上提交查询 www.ncbi.nlm.nih.gov/entrez/）时，查询会映射成 Medline 词汇表中的词汇

- 使用人工编纂的同义词词典。这里，编辑人员建立了概念的同义词名称，而不是给定一个规范的词项。UMLS 元词典是此类词典的一个例子。加拿大统计局（Statistics Canada）维护了一部包含优先词项、同义词、上位词项（broader term）、下位词项（narrower term）的同义词词典，用于政府收集统计数据（如商品和服务的统计）时的内容描述。同时，该词典还是一部双语版本的词典（英语和法语）。
- 使用自动构建的同义词词典。在这里，某个领域文档集中的词共现统计信息可以用于导出该词典（参见 9.2.3 节）。
- 基于查询日志挖掘进行查询重构。这里，可以利用其他用户的人工查询重构信息来对新用户进行查询推荐。这需要很大的查询量，因此尤其适合在 Web 搜索中使用。

基于词典的查询扩展方法的优点是不需要用户输入任何信息。查询扩展通常可以提高召回率，它在很多科学及工程领域得到广泛应用。同全局分析技术一样，局部分析技术也可以进行查询扩展，比如，可以对检索的结果文档进行分析来进行查询扩展。当然此时往往需要用户的输入，但是用户是对文档还是对查询词项进行反馈这两者之间仍然具有很大的区别。

9.2.3　同义词词典的自动构建

人工构建同义词词典的代价很大，一种取代思路是通过分析文档集来自动构造这种词典。这主要有两种实现方法。一种方法是简单地使用词共现信息。我们可以认为同时出现在文档或段落中的词在某种意义上相似或者相关，这样就可以通过计算文本中的统计信息来找到最相似的词。另一种方法是采用浅层语法分析器来分析文本得到词汇之间的语法关系或语法依存性。比如，我们可以认为可生长、可烹调、可取食和可消化的实体很可能是食品。简单地采用词共现信息更具鲁棒性（它不可能会产生分析器出错所导致的错误），但是采用语法关系有可能会更精确。

最简单的计算共现同义词词典的方法是基于词项之间的相似度计算。首先我们给出词项–文档矩阵 A，矩阵中每个元素 $A_{t,d}$ 是词项 t 在文档 d 中加权计算后的数目 $w_{t,d}$，权重计算方法要保证 A 中每个行向量大小为 1。如果我们计算 $C = AA^T$，那么 $C_{u,v}$ 就是词项 u 和 v 的相似度得分，这个值越大越好。图 9-8 给出了一个采用这种方法导出的同义词词典的例子，当然该例子中还采用了隐性语义索引来进行额外的降维操作。关于隐性语义索引，我们将在第 18 章讨论。尽管这种方法产生的词典中的部分词项很好或者至少具有启发性，但其他词项的效果一般或者很差。在词典自动构建中，词项关联的质量通常是一个典型的问题。词项的歧义性很容易会引入那些统计上相关但是意义上并不相关的词项。比如，查询 Apple computer 可能会被扩展成 Apple red fruit computer。总的来说，这类词典会受假正例和假反例的错误干扰。另外，由于自动生成的同义词词典中的词项在文档集中高度相关，而且通常来说产生该词典的文档集就是建立索引的文档集，所以利用该词典进行查询扩展可能不会返回很多额外文档。

词	相似词
absolutely	whatsoever, totally, exactly, nothing
bottomed	dip, copper, drops, topped, slide, trimmed
captivating	shimmer, stunningly, superbly, plucky, witty
doghouse	dog, porch, crawling, beside, downstairs
makeup	repellent, lotion, glossy, sunscreen, skin, gel
mediating	reconciliation, negotiate, case, conciliation
keeping	hoping, bring, wiping, could, some, would
lithographs	drawings, Picasso, Dali, sculptures, Gauguin
pathogens	toxins, bacteria, organisms, bacterial, parasite
senses	grasp, psyche, truly, clumsy, naive, innate

图 9-8 一个自动构建的同义词词典的例子。本例基于 Schütze（1998）中的工作，其中采用了第 18 章提到的隐性语义索引方法

查询扩展在提高召回率方面往往很有效。然而，手工生成同义词词典后，不断对它更新需要付出很高的代价。一般而言，我们更需要与领域相关的词典，通用词典对大多数科学领域中丰富的领域特定词汇的覆盖率很低。但是，有时候特别是当查询中含有歧义词时，查询扩展可能会明显降低正确率。比如，用户搜索 interest rate，将查询扩展成 interest rate fascinate evaluate 可能没用。尽管查询扩展有时和伪相关反馈的效果相当，但是总的来说查询扩展不如相关反馈技术成功。当然，它的优点是更容易为用户所理解。

? 习题 **9-7** 如果词项–文档矩阵 A 是一个布尔共现矩阵，那么矩阵 C 中的元素是什么？

9.3 参考文献及补充读物

IR 工作很快面临表达不同的问题，也就是说，尽管一篇文档与用户输入的查询相关，但是用户输入的查询词并不在该文档中出现。Swanson（1988）所引用的一个 20 世纪 60 年代的实验结果表明，在主题 toxicity 所标引的 23 篇文档中仅有 11 篇包含了词干 toxi 的某种词汇形式。这里也存在“翻译转换”的问题，即用户如何知道文档使用哪些词项。Blair 和 Maron（1985）得出结论，用户不可能精确估计出在所有（或大部分）相关文档中所使用且仅仅（或主要）在这些文档所使用的词、词组或短语。

使用向量空间模型进行相关反馈的主要的早期文献请参见 Salton（1971b），包括 Rocchio 方法（Rocchio，1971）的介绍、Ide dec-hi 公式及多个变种公式之间的评价（Ide，1971）。另外一个变形公式是，将所有文档集中判断为相关文档之外的文档全部看成不相关文档，而不是将那些显式地判断为不相关的文档看成不相关文档。然而，Schütze 等人（1995）和 Singhal 等人（1997）的工作表明，就信息路由（routing）任务而言，只利用那些与查询兴趣相近的文档会比使用所有文档做为不相关文档的效果要好。后来的一些其他工作有 Salton 和 Buckley（1990）、Riezler 等人（2007）（其中提出了一个基于统计 NLP 的相关反馈方法）以及一篇综述性文章

Ruthven 和 Lalmas （2003）。

交互式相关反馈系统的效果在（Salton，1989；Harman，1992；Buckley 等人，1994b）中讨论。Koenemann 和 Belkin（1996）采用了用户调查的方法来对相关反馈的效果进行评价。

在传统意义上，罗杰斯同义词词典（Roget's Thesaurus）是最出名的英语同义词词典（Roget，1946）。近年来的工作中，人们常常使用 WordNet （Fellbaum，1998），这不仅是因为它是免费的，而且还因为 WordNet 中具有丰富的链接结构。WordNet 可以通过地址 http://wordnet.princeton.edu 得到。

Qiu 和 Frei（1993）和 Schütze（1998）讨论了词典的自动生成方法。Xu 和 Croft（1996）探索了局部分析和全局分析相结合的查询扩展方法。

第10章 XML检索

人们通常拿IR与关系数据库进行对比。传统意义上说，IR系统是从无结构的文本（unstructured text）中检索出信息，这里的无结构信息指的是不带标记（markup）的原始文本。而数据库主要是用于查询RDB（relational data，关系型数据），即由预先定义的属性的值所构成的一系列记录，这些属性包括员工的编号、头衔和工资等等。不论在检索模型、数据结构还是查询语言等方面，IR和数据库系统之间都存在着根本的差异，这种差异如表10-1所示[①]。

表10-1 RDB搜索、非结构化IR及结构化IR。对于结构化检索来说，尽管很多学者都认为Xquery（10.5节）将会成为结构化查询的标准，但是关于这一点目前还没有最后定论

	RDB搜索	非结构化检索	结构化检索
对象	记录	非结构化文档	以文本为叶节点的树
模型	关系模型	向量空间或其他	?
主要数据结构	表格	倒排索引	?
查询语言	SQL查询	自由文本查询	?

对某些高度结构化的文本搜索问题而言，采用关系数据库的处理效率最高。比如，如果员工表中有一个文本类属性用于对工作的简短描述，而我们想寻找与财务结算（invoicing）有关的所有员工。这种情况下，利用如下SQL查询足以得到具有高正确率和召回率的令人满意的结果：

```
select lastname from employees where job_desc like 'invoic%';
```

但是，很多包含文本的结构化数据源最好通过结构化文档（structured document）而不是关型系数据来建模。基于这种结构化文档的搜索称为结构化检索（structured retrieval）。我们假定本章中给出的文档集仅仅由结构化文档组成，当然结构化检索中的查询既可以是结构化查询也可以是非结构化查询。结构化检索的应用场景或对象包括数字图书馆、专利数据库、博客、包含已标注命名实体（如人名、地名）的文本以及诸如Open Office办公套件输出的带标记的文本等等。在所有这些应用中，我们都希望能够处理将内容需求和结构需求相结合的查询。比如，数字图书馆中的查询give me a full-length article on fast fourier transforms（查找关于快速傅立叶变换的完整论文）、专利搜索中的查询give me patents whose claims mention RSA public key

① 在大部分现代数据库系统中，都允许对文本字段进行全文搜索。这通常意味着系统中建立了倒排索引，也可以允许布尔检索或者向量空间检索。这样可以将核心数据库和IR技术高效地结合起来。

encryption and that cite US patent 4,405,829（在专利权利要求中提到 RSA 公钥密码并且引用了专利号为 4 405 829 的美国专利的专利）及针对已标注命名实体的文本所进行的查询 give me articles about sightseeing tours of the Vatican and the Coliseum（查找关于梵蒂冈和古罗马竞技场观光旅游的文章）等等。这 3 个查询都是结构化查询，它们很难通过一个无序检索系统来得到好的应答结果。正如我们在例 1-1 中提到的那样，像布尔模型这样的无序检索模型的召回率不高。比如，一个无序检索系统可能会返回大量的提到 Vatican、Coliseum 及 sightseeing tours 的文档，而不将最相关的查询结果放在最前面。大部分用户都很难精确描述结构化的限制条件。比如，用户可能并不知道搜索系统支持对哪些结构化元素的查询。在上例中，用户可能不能确认到底是提交查询 sightseeing AND COUNTRY:Vatican AND LANDMARK：Coliseum、sightseeing AND STATE:Vatican AND BUILDING:Coliseum 还是其他形式的查询。用户可能对结构化搜索和高级搜索一无所知，或者他们压根就不想用这些搜索功能。本章当中，我们考察如何将排序检索模型用于结构化文档搜索来解决上述问题。

下面只考察目前使用最广泛的一种结构化文档编码标准：XML（Extensible markup language，扩展标记语言）。至于 XML 与其他标记语言（如 HTML 和 SGML）间具体的差异细节我们并不做过多介绍，但是本章所讨论的大部分内容对一般标记语言来说都具有通用性。

在 IR 领域，我们仅仅关注 XML 对文本和文档进行编码的功能。当然，XML 更广泛地被应用于非文本数据的编码。例如，我们可能想从一个 ERP（Enterprise Resource Planning，企业资源规划）系统中以 XML 格式导出数据，并将它们读入到一个数据分析系统中来生成演示图表。之所以称这种的 XML 应用为“以数据为中心”（data-centric），是因为整个数据当中，数值型及其他非文本型数据占了大部分，而文本数据只是其中的一小部分。大部分以数据为中心的 XML 数据通过数据库来存储，与之形成鲜明对照的是，本章介绍的以文本为中心的 XML 数据则主要基于倒排索引方法来存储。

本章中，我们将 XML 检索称为结构化检索（structured retrieval），而有些学者更喜欢采用术语半结构化检索（semistructured retrieval）来将 XML 检索与数据库检索区别开来。这里我们采用了前者，这主要是因为它在 XML 检索领域的使用更广泛。比如，谈到 XML 查询时，标准的说法是结构化查询（structured query）而不是半结构化查询（semistructured query）。而术语“结构化检索”很少在数据库查询中使用，因此在本书中“结构化检索”均指 XML 检索。

在非结构化检索和数据库查询之间，还存在另外一种类型的 IR 问题，即我们在 6.1 节所讨论的参数化搜索及域搜索。在这类搜索的数据模型中，存在参数化字段（如 date 或 file-size 之类的关系属性）和以非结构化文本为属性值的文本属性——域（如图 6-1 中的 author 和 title）。这些数据模型属于扁平结构，即不存在属性之间的嵌套关系，并且属性的数目也比较少。与之相反，XML 文档包含非常复杂的树型结构（参考图 10-2），属性之间还存在嵌套关系。另外，XML 检索中的属性数目也高于参数化搜索和域搜索。

下面，我们将首先在 10.1 节给出 XML 的一些基本概念；接着在 10.2 节讨论 XML 检索面

对的一些挑战性问题；然后在 10.3 节介绍一个用于 XML 检索的向量空间模型；10.4 节将介绍 INEX 公共评测会议，该会议已经举办多年，并且已经成为 XML 检索研究的最重要的平台；最后，我们将在 10.5 节介绍以数据为中心和以文本为中心的 XML 在处理上的差异。

10.1 XML 的基本概念

一篇 XML 文档是一个有序的带标签树，树上的每个节点都是一个 XML 元素（XML element），它由起始标签（tag）和结束标签来界定。一个 XML 元素可以有一个或多个 XML 属性（XML attribute）。在图 10-1 所示的 XML 文档中，scene 元素的起始标签和结束标签分别是 <scene ...> 和 </scene>。它包含一个属性 number，其对应的属性值为 vii，另外还有两个子元素，分别是 title 和 verse。

```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number="I">
<scene number="vii">
<title>Macbeth's castle</title>
<verse>Will I with wine and wassail ...</verse>
</scene>
</act>
</play>
```

图 10-1 一篇 XML 文档的例子

图 10-2 给出了图 10-1 中文档的树型表示。树的叶节点（leaf node）中包含了一些文本，如 Shakespeare、Macbeth 及 Macbeth's castle 等等。而树的内部节点（internal node）对文档的结构信息（title、act 和 scene）或元信息（如 author）进行编码。

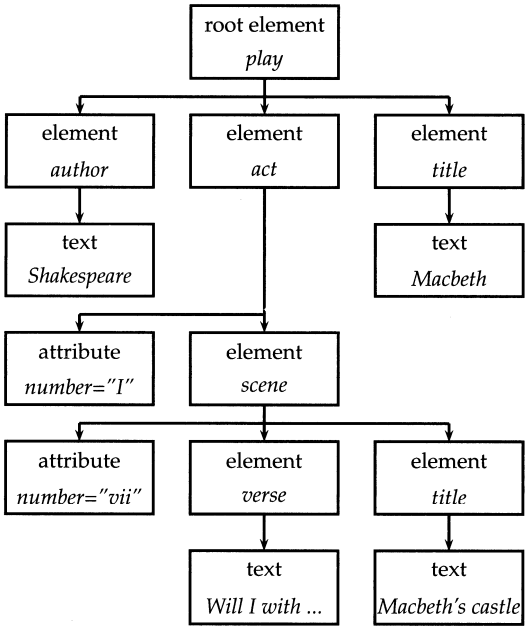


图 10-2　图 10-1 中 XML 文档的简化 DOM 对象

访问和处理XML文档的标准是XML DOM*document object model*，文档对象模型）。DOM将元素、属性及元素内部的文本表示成树中的节点。图 10-2 是对图 10-1 所示XML文档的一个简化的DOM树表示[①]。使用DOM API，我们可以从根元素出发，依据元素间的父子关系，自上而下对整个XML文档进行处理。

XPath 是 XML 文档集中的路径表达式描述标准。本章中，我们也将路径称为 XML 上下文（XML context）或者就直接简称上下文（context）。实际上，本章只需要 XPath 的一个较小的子集就足以说明问题。XPath 表达式中的 node 表示选择满足该表达式的所有节点，路径上前后元素间用斜杠“ /”来分隔，所以 act/scene 表示选择所有父节点为 act 元素的 scene 元素。双斜杠表示路径中可以插入任意多个元素，如 plan//scene 表示选择出现在 play 元素下的所有 scene 元素。在图 10-2 中，满足该表达式的集合仅仅包含一个 scene 元素，它是从根节点出发经 play → act→ scene 而到达的一条路径。路径若以斜杠开始则表示从该路径起始于根元素，比如在图 10-1 中，/play/title 表示选择剧本的标题，/play//title 则会选出 2 个元素（剧本的标题及场景的标题），而/scene/title 不会选出任何元素。为便于表示，我们允许路径上的最后的元素是词汇表中的一个词项，并利用“ #”号将其与前面的路径分隔开来。比如，title#“ Macbeth”表示选择标题中包含词项 Macbeth 的文档。需要指出的是，这种表示只是为了方便，它并不严格满足 XPath 的标准。

① 该表示在多个方面进行了简化。比如，没有显示根（root）节点（很多 XML 表示中，根结点实际上是一个虚拟的节点，在此图中并没显示），并且文本没有嵌在 text 节点中。参考 www.w3.org/DOM/。

本章中另一个必须提到的概念是schema[①]。一个schema给出了某个具体应用中的XML文档所允许的结构限制条件。比如，莎士比亚剧本的schema规定，场（scene）只能以幕（act）的子节点方式出现，而只有幕和场才具有number属性。XML文档的两个schema标准分别是 XML DTD（document type definition，文档类型定义）和 XML schema。用户只有对文档集的schema有起码的了解后才能构造并提交结构化查询。

XML 查询的常用格式为 NEXI（Narrowed Extended XPath I），图 10-3 中给出了一个例子。为了排版的方便，我们将查询表示成 4 行，但是实际上它们是一个完整的单位，之间并没有行间隔。其中，该查询中的//section 嵌套在//article 之下。

```
//article
[.//yr = 2001 or .//yr = 2002]
//section
[about(.,summer holidays)]
```

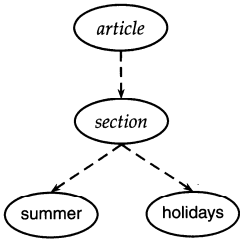


图 10-3 采用 NEXI 形式表达的一个 XML 查询及其部分树型表示

图 10-3 给出的查询搜索的是 2001 年到 2002 年间有关暑假的文章小节（section）。在 XPath 中，双斜杠表示路径中可以插入任意多个元素，而方括号中的句点表示的是该查询子句（clause）所修饰的元素。比如，子句[.//yr = 2001 or .//yr = 2002] 修饰的是 //article，因此，此时句点代表的是//article。类似地，[about（., summer holidays）]中的句点代表的是该查询子句所修饰的小节。

上述限制条件中，两个 yr 条件是属性限制条件。只有 yr 属性为 2001 或 2002（或者包含某个 yr 属性为 2001 或 2002 的元素）的文章才会被考虑。而 about 子句是排序限制条件：符合属性限制条件的文档中，各小节会按照它们与 summer holidays 这个主题的相关度进行排序。

通常可以通过预过滤或者后过滤来处理属性限制条件，比如简单地从结果集中排除那些不满足属性限制的元素即可。本章并不介绍这类过滤操作的高效处理方法，而主要关注 XML 检索中的核心信息检索问题，也就是说，如何按照 NEXI 查询中 about 子句所表达的相关性限制条件对文档进行排序。

如果去掉 XML 文档中的属性，也就是说去掉 XML 文档中所有属性节点（如图 10-1 中的 number 属性），就可以将 XML 文档表示成只包含元素这一种类型节点的树结构。图 10-4 给出了图 10-1 例子文档的一棵只包含元素型节点的子树（标记为 d_1）。

① 有人将 schema 译成模式，有人译成构架，为避免不必要的误解，这里我们直接采用原术语，这也是在很多 XML 相关教程中的普遍做法。——译者注

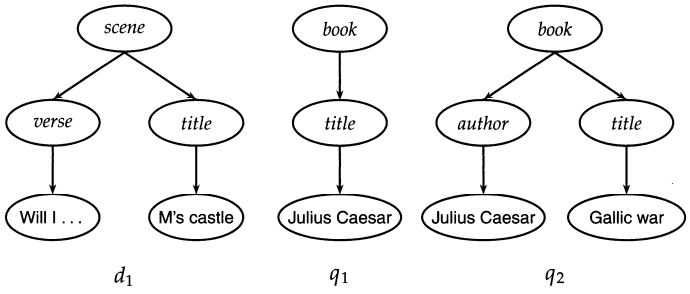


图 10-4　XML 文档及查询的树型表示

采用同样的办法也可以将查询表示成树。由于用户构建的查询满足和文档一样的形式化描述，所以这种查询语言设计是一种基于样例的查询（query-by-example）方法。图 10-4 中，q_1 查找的是书名与关键词Julius Caesar高度相关的书，而q_2 查找的是作者名与Julius Caesar高度相关且书名与Gallic war[①]高度相关的书[②]。

10.2　XML 检索中的挑战性问题

本节将讨论结构化检索中的一系列挑战性问题，这些问题也使得结构化检索比非结构化检索更加困难。回忆一下，本章一开始我们就假定结构化检索中的基本配置是：文档集包含的是结构化文档，而查询既可以是结构化的（如图 10-3 所示）也可以是非结构化的（如 summer holidays）。

结构化检索中的第一个挑战是用户希望返回文档的一部分（即 XML 元素），而不像非结构化检索那样往往返回整个文档。如果在莎士比亚剧本中查找 Macbeth's castle，那么到底应该返回场（scene）、幕（act）还是图 10-2 所示的整个剧本呢？在本例中，用户可能要找的是场。但对于另一个没有具体指定返回节点的查询 Macbeth，应该返回剧本的名称而不是某个子单位。

选择最合适的文档部分的一个准则是结构化文档检索原理（structured document retrieval principle）：系统应该总是检索出回答查询的最明确最具体的文档部分。

上述原理会引发这样一种检索策略，即返回包含信息需求的最小单位。但是，要在算法上实现这种原理是非常困难的。考虑图 10-2 上的一个查询 title#"Macbeth"，整个剧本的标题 *Maccbeth* 以及第一幕第七场的标题 *Macbeth's castle* 都是包含匹配词项 Macbeth 的较好的命中结果。但是在这个例子中，剧本的标题这个位于更高层的节点作为答案却更合适。可见，确定查

① Gallic war 即《高卢战记》，是凯撒大帝的一部传世之作，记录了公元前 58~前 51 年间古罗马征服山北高卢(大致相当于今天的法国、比利时和卢森堡地区)的战争。——译者注

② 为了完整表达 NEXI 查询的语义，我们还需要指定树中的一个节点为“目标节点”，比如，图 10-3 中的 section 节点。如果不指定目标节点的话，图 10-3 所示的树就不是搜索嵌在文章中的小节（由 NEXI 所规定），而是包含相关小节的文章。

询应答的正确层次是非常困难的。

与“返回文档的哪些部分给用户”这个问题对应的是“对文档的哪些部分进行索引”。在 2.1.2 节中，我们讨论了在索引和检索中确定文档单位或者说索引单位(indexing unit)的必要性。在非结构化检索中，合适的文档单位往往比较明显，如 PC 上的文档、邮件、Web 上的网页等等。而在结构化检索中，却有定义索引单位的一系列不同的方法。

一种方法是将节点分组，形成多个互不重叠的伪文档(pseudodocument，如图 10-5 所示)。该例子中，书(book)、章(chapter)及小节(section)都被指定为索引单位，但是它们之间并不重叠。比如，最左边虚线框中的索引单位仅仅包含树中受 book 节点所支配的部分，且这部分并不包含在其他索引单位中。这种方法的缺点是，由于这些伪文档的内容并不连贯，所以它们对用户而言可能没有什么意义。比如，图 10-5 中的最左边的索引单位合并了 class、author 和 title 这 3 个完全不同的元素。

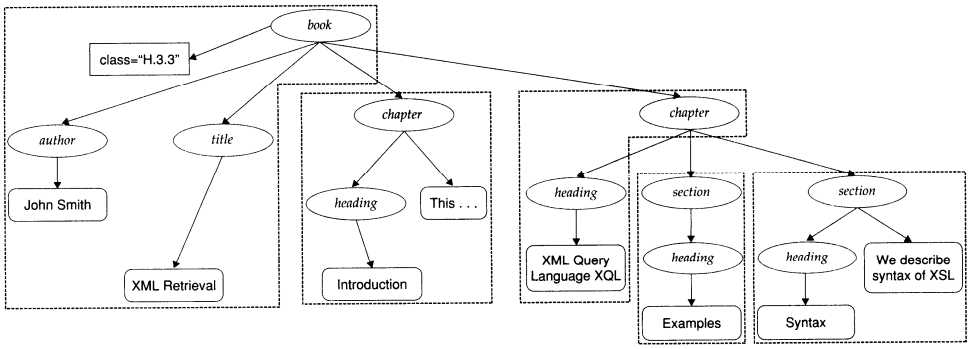


图 10-5 将 XML 文档分割成不重叠的索引单位

我们也可以使用最大的一个元素作为索引单位，比如，在一个书的集合中以元素 book 为索引单位，而莎士比亚的剧本则以元素 play 为索引单位。然后通过对结果进行后处理，从而在每本书或每个剧本中找到更适合作为答案的子元素。例如，查询 Macbeth's castle 可能会返回 *Macbeth* 这个剧本，经后处理就会发现第一幕第七场可能是匹配最佳的子元素。遗憾的是，对很多查询来说，这种“分两步走”的做法往往并不能返回最佳匹配子元素，这是因为整本书与查询的相关性并不能代表其子元素与查询的相关性。

与上述先返回大单位然后识别子元素（自顶向下）的方法不同的是，可以直接对所有叶节点进行搜索并返回最相关的一些节点，然后通过后处理将它们扩展成更大的单位（自底向上）。比如对于图 10-1 所示的查询 Macbeth's castle，第一遍我们可能返回的是标题(title)*Macbeth's castle*，然后在后处理中确定到底是返回标题(title)、幕(scene)、场(act)，还是整个剧本(play)。这种做法与自顶向下的方法存在类似的问题，即一个叶节点的相关性往往不能代表包含该叶节点的上层元素的相关性。

限制最少的方法是对所有元素建立索引，然而这种做法也有问题。很多 XML 元素并不是

有意义的搜索结果，比如，排版相关的元素（如definitely）或者不能脱离上下文进行单独解释的 ISBN 书号等。另外，对所有元素建立索引也意味着搜索结果将会有高度的冗余性。比如，对于图 10-1 给出的查询 Macbeth's castle 和文档，我们会返回从根节点到 Macbeth's castle 路径上的所有 play、act、scene 和 title 元素。此时，该叶节点会在结果集合中会出现 4 次，其中 1 次作为索引对象直接出现，而其他 3 次则作为其他元素的一部分出现。元素之间互相包含的关系称为嵌套（nested）。对用户而言，在返回结果中包含冗余的嵌套元素则显得不太友好。

针对元素嵌套所造成的冗余性，普遍的做法是对返回元素进行限制。这些限制策略包括：

- 忽略所有的小元素；
- 忽略用户不会浏览的所有元素类型（这需要记录当前 XML 检索系统的运行日志信息）；
- 忽略通常被评估者判定为不相关性的元素类型（如果有相关性判定的话）；
- 只保留系统设计人员或图书馆员认定为有用的检索结果所对应的的元素类型。

在大部分上述方法中，结果集中仍然包含嵌套元素。因此，我们可以通过一个后处理过程来去掉某些元素，从而降低检索结果的冗余性。一种做法是，在结果列表中将多个嵌套元素折叠起来，并通过对查询词项进行高亮显示（highlighting）来吸引用户关注相关段落。如果加亮查询词项，那么对中等规模元素（如 section）的扫描时间会比对其子元素（如 paragraph）的扫描时间稍长。因此，如果 section 和 paragraph 都出现在结果列表中，那么显示 section 就足够了。这种做法的另一个好处是 paragraph 会和它的上下文（包含该 paragraph 的 section）一起显示。因此，即使 paragraph 本身就可以满足查询的需求，但这种上下文表示的方法仍然对 paragraph 有很好的解释作用。

如果用户了解文档集的 schema，能够指定所需的元素类型，那么由于具有同样类型的嵌套元素很少，冗余的问题就可以得到缓解。但是正如我们在前面讨论的那样，用户往往并不知道文档集中元素的名称（比如不知道 Vatican 到底是国家还是城市的名字），或者他们对如何构造结构化查询根本就一无所知。

嵌套问题也会给 XML 检索带来另一个挑战。由于元素之间嵌套关系的存在，在结果排序中计算词项统计信息（比如 6.2.1 中定义的 idf 信息）时，必须要区分词项的不同上下文。比如，出现在 author 节点下的词项 Gates 与出现在内容节点（如 section）下的 gates （当然此时 gates 代表的是 gate 的复数）毫不相干。因此，在这类例子中，为 Gates 计算一个单独的文档频率 df 意义不大。

一种解决的办法是为 XML 的每个上下文-词项对计算 idf 值，比如分别计算 author#"Gates" 和 section#"Gates"的 idf 值。遗憾的是，这样做会导致数据稀疏问题（参考 13.2 节有关数据稀疏性的讨论），即许多上下文-词项对出现过少从而导致对其文档频率估计的不足。一个折衷方案是在区分上下文时只考虑词项的父节点 x，而不考虑从根节点到 x 路径上的其他部分。但是，这种做法仍然存在将不同上下文混在一起所带来的危害性。比如，如果作者（author）名字和公司（corporation）名字都有父节点名字（name），那么上述做法很难将它们区分开来。当然，这时

候一些非常重要的区别，比如 author#"Gates" 和 section#"Gates"之间的区别倒是会被考虑到。

很多情况下，由于 IR 应用中的 XML 文档往往有多个来源，所以在文档集中可能存在多个不同的 XML schema。这个现象称为 schema 异构性（schema heterogeneity）或 schema 多样性（schema diversity），这也对 XML 检索提出了另外一个挑战。异构性的一个体现是有可比性的元素可能具有不同的名称，如图 10-6 d_2 中的 creator 与 d_3 中的 author。另一种情况是，不同的 schema 可能采用了不一样的结构化组织方式。比如，图 10-6 查询 q_3 中的作者名是节点 author 的直接后代，但是在 d_3 中，节点 author 和最后的名字之间还有两个中间节点 firstname 和 lastname。如果我们进行严格的树匹配，那么输入查询 q_3 将不会返回文档 d_2 和 d_3，而尽管它们都与 q_3 相关。此时，如果对元素名进行某种形式的近似匹配，再结合不同文档结构之间的半自动匹配，那么对解决上述问题会有所帮助。当然，在不同 schema 的元素之间建立对应关系，人工方法往往要好于自动方法。

schema 异构性是造成类似 q_3/d_2 和 q_3/d_3 查询–文档失配的一个原因。另一个原因我们在前面也提到过，即用户往往对文档集 schema 中的结构及元素名称并不熟悉。这对 XML 检索中的界面设计提出了挑战。理想情况下，用户界面应该展示出文档集的树型结构并允许用户指定要查询的元素。如果采用这种方法的话，那么结构化检索中的查询界面设计就要比非结构检索中关键词查询搜索框的方式要复杂很多。

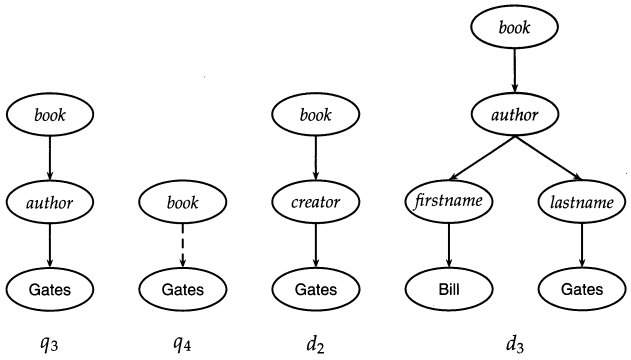


图 10-6 一个 schema 异构性的例子：存在中间节点及名字失配（mismatch）现象

我们也可以支持用户将查询中的所有父子关系解释成包含多个中介节点的后裔关系，这类查询称为扩展查询（extended query）。图 10-3 所示的树和图 10-6 中的 q_4 是两个扩展查询的例子。其中后裔关系表示成虚线箭头。在 q_4 中，虚线箭头链接了 book 和 Gates。我们可以采用一种伪 XPath 符号将该查询表示为 book//#"Gates"，即表示查询的对象是一本书，该书中的某个地方包含单词 Gates，而且从 book 节点到 Gates 的路径可以任意长。一个表示 Gates 出现在 book 的 section 内的伪 XPath 扩展查询表示为 book//section//#"Gates"。这样的话，用户就很方便来提交这种扩展查询，而并不需要精确指定查询词项出现的结构配置信息，当然，这可能是由于用户本身对精确的结构配置并不关心，也有可能是他们对文档集的 schema 了解不够而不足以给出这类信息。

图 10-7 中，用户要查找标题为 FFT（q_5）的一章。假定某书中不存在这样的一章，但是存在有关 FFT 的参考文献（d_4）。虽然，该书参考文献并不是用户所想要的最好结果，但是返回它总比什么都不返回要强。此时，扩展查询也起不到什么作用，比如扩展查询 q_6 不会返回任何结果。这种情况下，我们希望将查询中指定的结构限制条件解释成一些提示线索而不是严格的限制条件。正如我们将要在 10.4 节讨论的那样，用户更希望对结构限制条件进行非严格的解释，即不完全满足结构限制条件的元素会排名较低，但是它们不应该从结果中剔除。

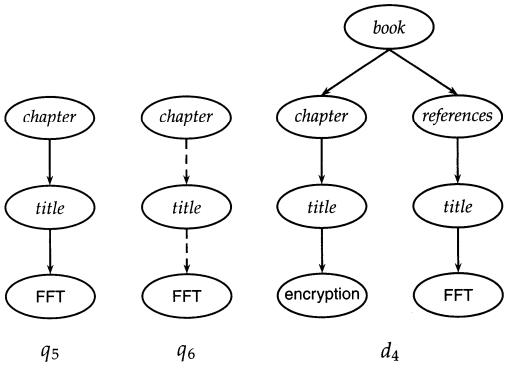


图 10-7　查询和文档间结构失配的两个例子

10.3　基于向量空间模型的 XML 检索

本节给出了一个应用于 XML 检索的简单向量空间模型，我们的目的并不是完整而全面地介绍当前最新的 XML 检索系统。实际上，通过这一节的介绍，我们只希望读者对 XML 检索中文档的表示和检索有所了解。

考虑到图 10-4 检索中的结构信息，我们希望书 *Julius Caesar* 和查询 q_1 匹配而和 q_2 不匹配或匹配得分很低。在非结构化检索中，词项 Caesar 本身就会对应单独的一维。而在 XML 检索中，要将 *title* 中的 Caesar 和 *author* 中的 Caesar 区分开来。一种实现的方法是对向量空间中的每一维都同时考虑单词及其在 XML 树中的位置信息。

图 10-8 给出了这种表示方法的示意图。首先我们考虑每个文本节点（在本章中永远为叶节点）并将它们分裂成多个节点，每个节点对应一个词。因此，叶节点 Bill Gates 会被分裂成两个叶节点 Bill 和 Gates。下一步，我们将向量空间的每一维定义为文档的词汇化子树（lexicalized subtree），这些子树至少包含词汇表中的一个词项。这些词汇化子树的一个可能的子集如图 10-8 所示。实际上，还存在一些其他的子树，比如，整个文档去除 Gates 叶节点后剩余的子树。现在我们就可以将查询和文档表示成这些词汇化子树空间上的向量，并进行相似度计算。这也意味着我们可以利用第 6 章介绍的向量空间方法来进行 XML 检索。当然，第 6 章介绍的是面向非结构化检索的向量空间模型，其向量的每一维都是词汇表中的词项，而在这里的每一维都是词

汇化子树。

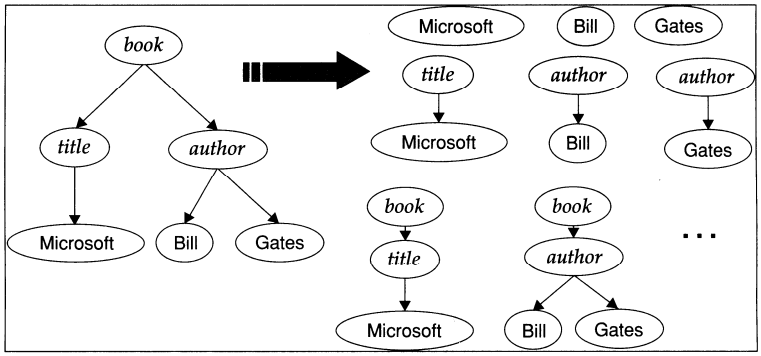


图 10-8 XML 文档（左部）到词汇化子树集合（右部）的一个映射

在向量空间的维数和查询结果的精度之间存在一个折衷。一方面，如果我们将每一维限制到词汇表词项上，那么就得到一个常规的向量空间系统，该系统会返回很多和查询的结构并不匹配的文档（比如 Gates 在标题中出现而不是作者中出现）。而另一方面，如果将文档集中出现的每个词汇化子树都看成一维，那么整个空间的维数就会很大。一个折衷的方法是对所有的最终以单个词项结束的路径建立索引，换句话说，对所有的 XML 上下文-词项对建立索引。这种 XML 上下文-词项对被称为结构化词项（structural term），记为<*c*,*t*>，其中 *c* 是 XML 上下文，*t* 是词项。图 10-8 中的文档有 9 个结构化词项，其中 7 个已在图中显示（如“Bill”和 Author#"Bill"），另外 2 个则没有显示（/Book/Author#"Bill" 和/Book/Author#"Gates"）。以 Bill 和 Gates 作为叶节点的子树是一棵词汇化子树，但它并不是结构化词项。我们可以使用前面介绍过的伪 XPath 语言来表示结构化词项。

在上一节我们提到，用户很难记得 schema 的细节，也很难构造符合 schema 的查询。因此，我们将所有的查询都解释为扩展查询，即对于查询中的父子节点，文档中可以有任意多个中间节点介于它们之间。比如，我们将图 10-7 中的 q_5 解释成 q_6。

当然，我们仍然希望优先考虑那些与查询结构相匹配且中间节点数量较少的文档。为了保证检索结果遵守这种优先级，我们要对每个匹配计算一个权重。一个简单的度量查询中路径 c_q 和文档中路径 c_d 相似度的指标是上下文相似度（context resemblance）函数 C_R，其定义如下：

$$C_R(C_q,C_d)=\begin{cases}\dfrac{1+|c_q|}{1+|c_d|} & \text{如果}c_q\text{和}c_d\text{匹配上，}\\ 0 & \text{如果}c_q\text{和}c_d\text{不匹配。}\end{cases} \qquad (10\text{-}1)$$

其中$|c_q|$和$|c_d|$分别是查询路径和文档路径中的节点数目，并且当且仅当可以通过插入额外的节点使 c_q 转换成 c_d 时，c_q 和 c_d 才能匹配。图 10-6 中的两个例子是 $C_R(c_{q_4},c_{d_2})=3/4=0.75$ 及 $C_R(c_{q_4},c_{d_3})=3/5=0.6$，其中 c_{q_4}、c_{d_2} 及 c_{d_3} 分别是从根节点到 q_4、d_2 和 d_3 中叶节点的路径。如果 q 和 d 相等，那么 $C_R(c_q,c_d)$ 为 1.0。

最终的文档得分计算可以看成是公式（6-10）余弦相似度计算方法的一个变形。我们将这个计算公式称为 SIMNOMERGE，取这个名字的原因我们很快就会在后面介绍。SIMNOMERGE 的定义如下：

$$\begin{aligned}&\text{SIMNOMERGE}(q,d)\\&=\sum_{c_k\in B}\sum_{c_l\in B}\text{CR}(c_k,c_l)\sum_{t\in V}weight(q,t,c_k)\frac{weight(d,t,c_l)}{\sum_{c\in B,t\in V}weight^2(d,t,c)}\end{aligned}\quad,\qquad(10\text{-}2)$$

其中，V 是非结构化词项的词汇表，B 是所有 XML 上下文的集合。weight(q,t,c)和 weight(d,t,c)分别是词项 t 在查询 q 和文档 d 的上下文 c 中的权重。我们可以采用第 6 章的某种权重机制来计算这个权重，比如，$\text{idf}_t\times\text{wf}_{t,d}$。逆文档频率 idf_t 的值取决于 df_t 计算时所利用的元素，关于这一点在 10.2 节已经有所讨论。相似度度量函数 $\text{SIMNOMERGE}(q,d)$ 并不是一个真正的余弦相似度计算函数，因为它的值可能会超过 1.0（参考习题 10-11）。除以 $\sqrt{\sum_{c\in B,t\in V}\text{weight}^2(d,t,c)}$ 的目的是为了进行文档长度归一化（参考 6.3.1 节）。而为了简化计算，我们忽略了查询长度的归一化因子。事实上，对于给定的查询，查询长度对检索结果排序并没有影响，也就是说，归一化因子 $\sqrt{\sum_{c\in B,t\in V}\text{weight}^2(q,t,c)}$ 对所有文档都相同。

给定查询下，计算文档集中所有文档的 SIMNOMERGE 值的算法如图 10-9 所示。其中，normalizer 数组中保存的是每篇文档的归一化因子 $\sqrt{\sum_{c\in B,t\in V}\text{weight}^2(d,t,c)}$。

```
SCOREDOCUMENTSWITHSIMNOMERGE(q, B, V, N, normalizer)
 1  for n ← 1 to N
 2  do score[n] ← 0
 3  for each ⟨c_q, t⟩ ∈ q
 4  do w_q ← WEIGHT(q, t, c_q)
 5     for each c ∈ B
 6     do if CR(c_q, c) > 0
 7           then postings ← GETPOSTINGS(⟨c, t⟩)
 8                for each posting ∈ postings
 9                do x ← CR(c_q, c) * w_q * weight(posting)
10                   score[docID(posting)] += x
11  for n ← 1 to N
12  do score[n] ← score[n]/normalizer[n]
13  return score
```

图 10-9　利用 SIMNOMERGE 计算文档得分的算法

下面我们在图 10-10 中给出一个计算查询和文档间 SIMNOMERGE 值的例子。其中，$<c_1,t>$是查询中的一个结构化词项。我们先后返回具有相同词项 t 的结构化词项$<c',t>$的所有倒排记录表，图中给出了 3 个倒排记录表的例子。对于第一个倒排记录表，由于两个上下文完全一样，所以 $C_{\text{R}}(c_1,c_1)$=1.0。而上下文 c_2 中不包含与 c_1 类似的上下文，因此 $C_{\text{R}}(c_1,c_2)=0$，其对应的倒排记录表被忽略。c_1 和 c_3 的上下文匹配结果是 0.63>0，因此需要被处理。在本例中，排名最高的文档是 d_9，其相似度为 1.0 × 0.2 + 0.63 × 0.6 = 0.578。为了简化计算过程，本例中假定$<c_1,t>$

的查询权重为 1.0。

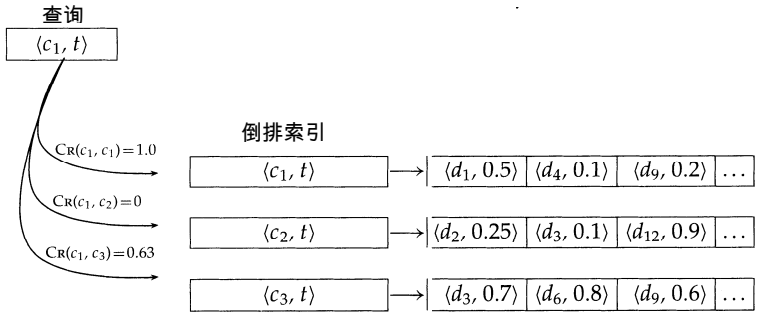


图 10-10 某个单结构化词项查询的 SIMNOMERGE 评分示意图

图 10-9 中的查询–文档相似度函数之所以称为 SIMNOMERGE ，是因为不同的 XML 上下文在权重计算时会分开处理。而另一种相似度计算函数称为 SIMMERGE ，它放宽了查询和文档的匹配条件，具体的做法有如下 3 种。

- 计算 weight (*q*,*t*,*c*) 和 weight (*d*,*t*,*c*) 所需的统计信息时基于所有与 *c* 有非零相似度的上下文的（而在 SIMNOMERGE 中只选择 *c* ）。比如，为了计算结构化词项 atl#"recognition"的文档频率，我们也对 fm/atl、article//atl 等上下文中 recognition 的出现次数进行统计。
- 给定查询结构化词项，我们将文档中所有与之有非零相似度的上下文结构化词项进行合并，并以此修改公式(10-2)。比如，当与查询词项/play/title#"Macbeth"进行匹配时，文档中的/play/act/scene/title 及/play/title 上下文将会被合并。
- 上下文相似函数进一步放宽条件：即使按照公式(10-1)中的定义计算得到的 C_R 值为 0，在很多情况下上下文相似度并不为 0。

对上述做法的具体细节感兴趣的读者可以参考 10.6 节给出的相关参考文献。

上面 3 种做法能够缓解 10.2 节所讨论的词项统计信息的稀疏性问题，并且在面对低质量的结构化查询时能够增加匹配函数的鲁棒性。10.4 节对 SIMNOMERGE 和 SIMMERGE 的评价结果表明，SIMMERGE 中匹配条件的放宽能够提高 XML 检索的效果。

? 习题 **10-1** 结构化词项的文档频率可以定义为该结构化词项在某个具体父节点下出现的次数。假设：结构化词项<*c*,*t*>=author#"Herbert" 作为 squib 节点的子节点出现了 1 次，而在文档集中总共有 10 个 squib 节点；<*c*,*t*>作为 article 节点的子节点出现 1 000 次，而整个文档集中总共有 1 000 000 个 article 节点。因此，如果利用 squib 节点进行计算，那么<*c*,*t*>的 idf 权重为 $\log_2 10/1 \approx 3.3$；如果利用 article 节点进行计算，结果为 $\log_2 1\,000\,000/1000 \approx 10.0$。

(i) 解释这种计算<*c*,*t*>权重的方法的不合理性。为什么<*c*,*t*>在 article 节点下获得的权重不应该是在 squib 节点下权重的 3 倍？

(ii) 提出一种更好的计算 idf 的方法。

习题 **10-2** 写出图 10-8 所示 XML 文档的所有结构化词项。

习题 **10-3**　图 10-1 中的文档会产生多少结构化词项？

10.4　XML 检索的评价

INEX（INitiative for the Evaluation of XML retrieval）计划是 XML 检索研究中的首要评测平台，它通过大家的协作产生参考文档集、查询集及相关性判断。在每年一度的 INEX 会议上，研究人员展示并讨论交流各自的研究结果。INEX 2002 文档集包含大概 12 000 篇来自 IEEE 期刊的文章。在表 10-2 中我们给出了文档集的统计信息，并在图 10-11 中给出了部分的 schema。2005 年，IEEE 期刊文档集进行了扩充。从 2006 年开始，INEX 利用大得多的英文 Wikipedia 数据作为测试集。文档的相关性判定主要采用 8.1 节介绍的方法通过人工判断来完成，并且针对结构化文档进行了适当修改。关于这一点，我们很快会在后面讲到。

表10-2　INEX 2002文档集的统计信息

12 107	文档数目
494 MB	数据集大小
1995 ~ 2002	发表时间
1 532	每篇文档的平均XML节点数目
6.9	节点的平均深度
30	CAS主题的数目
30	CO主题的数目

INEX 中有两种信息需求或者说是主题类型，它们是 CO（content-only，仅基于内容的）主题和 CAS（content-and-structure，内容结构相结合的）主题。CO 主题即是常规的关键词查询，这与非结构化信息检索中的查询一样。CAS 主题在关键词基础上增加了结构化限制。在图 10-3 中，我们已经给出了一个 CAS 主题的例子。该例子中的关键词为 summer 和 holidays，其结构化限制条件为 :关键词出现在文档的某一节中、该文档具有 year 属性，且其属性值为 2001 或 2002。

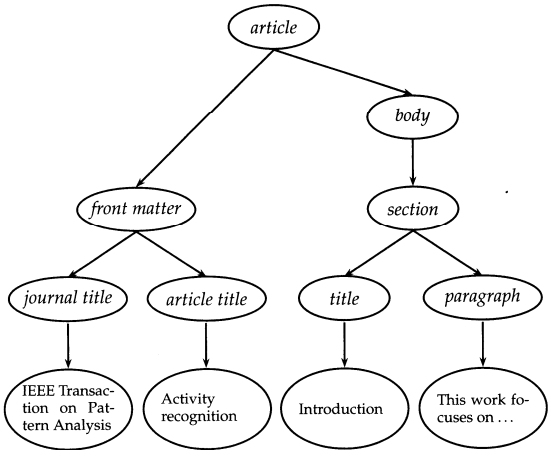


图 10-11 INEX 文档集中简化后的文档 schema 示意图

由于 CAS 查询同时包含结构信息和内容信息，其相关性判断就比非结构化中的相关性判断要复杂得多。INEX 2002 定义了部件覆盖度和主题相关性作为相关性判断的两个方面。部件覆盖度（component coverage）评价的是返回元素在结构上是否正确，也就是说，其在树中的层次既不太高也不太低。部件覆盖度分为以下 4 种情况。

- 精确覆盖（E）。所需求的信息是部件的主要主题，并且该部件是一个有意义的信息单位。
- 覆盖度太小（S）。所需求的信息是部件的主要主题，但是该部件不是一个有意义（自包含）的信息单位。
- 覆盖度太大（L）。所需求的信息在部件中，但不是主要主题。
- 无覆盖（N）。所需求的信息不是部件的主题。

主题相关性也有 4 个层次：强相关（3）、较相关（2）、弱相关（1）和不相关（0）。每个部件在覆盖度和主题相关性两个方面都要进行判断，然后将判断结果组合成一个数字-字母编码。2S 表示一个比较相关的部件，但是其覆盖度太小。而 3E 表示高度相关并具有精确覆盖的一个部件。理论上说，4 个等级的覆盖度和 4 个等级的相关性相组合，则对一个部件的评价有 16 种可能，但是实际评价中很多组合并不会出现。比如，一个不相关的部件不可能具有精确覆盖度，所以，编码为 3N 的组合是不可能的。

相关度–覆盖度组合可以采用如下量化方法：

$$\mathrm{Q}(rel,\mathrm{cov})=\begin{cases}1.00 & \text{if } (rel,\mathrm{cov})=3\mathrm{E}\\ 0.75 & \text{if } (rel,\mathrm{cov})\in\{2\mathrm{E},3\mathrm{L}\}\\ 0.50 & \text{if } (rel,\mathrm{cov})\in\{1\mathrm{E},2\mathrm{L},2\mathrm{S}\}\\ 0.25 & \text{if } (rel,\mathrm{cov})\in\{1\mathrm{S},1\mathrm{L}\}\\ 0.00 & \text{if } (rel,\mathrm{cov})=0\mathrm{N}\end{cases}$$

上述评估机制实际上考虑到了这样一个事实，传统的非结构化检索（参考 8.5.1 节）的二值相关性判断对于 XML 检索来说是不合适的。一个 2S 部件提供的信息尽管不完整，而且如果没

有更多的上下文将很难进行解释，但是它却能部分地回答查询。量化函数 Q 并不强制使用二值相关性（相关或不相关），而是通过对部件划分等级来处理部分相关性。

于是，检索结果集合 A 中相关部件的数目可以定义为

$$\#(relevant \text{ items retrieved}) = \sum_{c \in A} Q(rel(c), \text{cov}(c))。$$

第 8 章中介绍的正确率、召回率及F值的标准定义都可以近似地应用到上式所示的相关部件数目上来。和前面的细微差别是，这里计算的是评分等级之和而不是二值相关性之和。关于这一点的进一步讨论，可以参考 10.6 节给出的有关焦点检索（focused retrieval）[①]的参考文献。

上述衡量相关性做法的一个缺点是没有考虑到重合现象。8.5.1 节中讨论了非结构化检索中的边缘相关性概念，在 XML 检索中，由于搜索结果中的元素会存在多重嵌套问题，所以边缘相关性问题更加严重。INEX 近年来的关注焦点在于，提出结果无冗余的检索算法和评估指标并对结果进行恰当评价。有关这一点，参考 10.6 节给出的参考文献。

表 10-3 给出了 INEX 2002 上两个向量空间系统（参考 10.3 节的描述）的运行结果。其中 SIMMERGE 的运行结果更好一些，它融合的结构化限制很少而主要依赖于关键词匹配。然而，SIMMERGE 的平均准确率的中位数也仅仅为 0.147，这里的中位数指的是所有主题的平均准确率的中位数。由于 XML 检索更难，所以它的效果往往低于非结构化检索。XML 检索的目标不仅仅是寻找一篇文档，还必须寻找与查询非常相关的文档片段。另外，当采用这里介绍的衡量指标时，对 XML 检索效果的评分可能会低于非结构化检索，这是因为分级评价方法会降低评价的分值。考虑一个系统，假定其返回结果中的第一篇文档的分级相关度为 0.6，而二值相关度为 1。那么对于后者，召回率为 0.00 时对应的插值正确率为 1，而对于前者而言，同一点上插值的正确率为 0.6。

表10-3　10.3节的向量空间模型在INEX 2002上的运行结果（使用CAS主题和量化函数Q）

算　法	平均正确率
SIMNOMERGE	0.242
SIMMERGE	0.271

表 10-3 也让我们对 XML 检索的典型效果有所了解，但是它并没有将结构化检索与非结构化检索进行比较。表 10-4 直接给出了使用结构对检索效果的影响，这些结果是由基于语言模型的检索系统（参考第 12 章）在 INEX 2003、2004 中 CAS 主题的子集上所得到的。其评价指标是前 k 个结果的准确率（参考 8.4 节）。这里，评价时通过离散化函数将强相关元素（大概相当于量化函数 Q 为 3E 的元素）映射为 1，而将其他元素映射为 0。只基于内容的系统将查询和文档视为非结构化的词袋（bag of words），而全结构模型将满足结构限制的元素排名在前面。比如，对于图 10-3 中的查询，section 中包含短语 summer holidays 的元素将会比 abstract 中包含该

① 和传统检索只是返回文档不同，焦点检索返回更细粒度的检索单位。目前的焦点检索任务包括问答(question answering)系统、段落检索(passage retrieval)和 XML 中的元素检索(element retrieval)等等。——译者注

短语的元素排名更靠前。

表10-4 INEX 2003/2004中基于内容和基于全部结构的搜索结果比较

	仅考虑内容	考虑全结构	提高幅度
P@5	0.200 0	0.326 5	63.3%
P@10	0.182 0	0.253 1	39.1%
P@20	0.170 0	0.179 6	5.6%
P@30	0.152 7	0.153 1	0.3%

表 10-4 表明，结构信息能够帮助提高返回结果中排名靠前的元素的准确率，其中 P@5 和 P@10 都有较大的提高，而 P@30 几乎没有提高。这些结果展示了结构化检索的好处。结构化检索对返回结果增加了一些额外的限制，能够通过结构化限制条件的文档更可能相关。当然，由于有些相关文档被过滤掉，这种做法可能会损害召回率，但是在面向正确率的任务中，结构化检索更具优势。

10.5 XML 检索：以文本为中心与以数据为中心的对比

在本章介绍的结构化检索中，XML 的结构作为一个框架存在，在该框架下，可以对查询中的文本和文档中的文本进行匹配。这实际上是对以文本为中心的 XML（text-centric XML）检索系统进行优化的例子。尽管文本和结构都很重要，我们仍然给文本赋予更高的优先级。具体的实现方式是通过对非结构化检索方法进行调整，使之能够处理额外的结构化限制。这里的 XML 文档检索方法存在若干前提假设：(i) 长文本字段(如文档的 section)；(ii) 非精确匹配；(iii) 按相关度对结果排序。关系数据库不能很好地处理上述假设。

与之相反，以数据为中心的 XML（data-centric XML）主要对数值型和非文本的属性值数据进行编码。当对这类 XML 数据进行查询时，很多情况下都需要给出精确的匹配条件。这也将重点转移到了 XML 文档及查询的结构方面，以下是一个例子。

找出工资和 12 个月以前一样的员工。

此时并不需要对查询结果进行排名，它是一个纯结构化的查询，对两个时间段的工资进行精确匹配就可以充分满足该用户需求。

以文本为中心的方法对那些本质上是文本文档、而只是通过 XML 来标记文档结构的数据更加合适。由于绝大部分文本文档，包括汇编手册、出版的期刊、莎士比亚全集以及新闻报道等等，都具有某种形式的有趣的结构（如段落、章节、脚注等等），XML 也正在成为发布文本数据库的事实标准。

以数据为中心的方法通常用于具有复杂结构并且主要包含非文本数据的数据集。而当面对生物信息学中的蛋白组数据，或者导航数据库中的城市地图表示数据（包括街道名称和其他文

本描述）时，以文本为中心的检索引擎处理起来将会遇到很大的困难。

在以文本为中心的结构化检索模型中，还有两种类型的查询是比较难以处理的，它们是包含连接（join）或次序限制（ordering）的查询。上面查询工资不变的员工需要一个连接操作，而如下的查询则给出了一个次序限制。

返回书 Introduction to algorithms 中，Binomial heaps 这一章的下一章。

该查询需要对 XML 的元素进行排序，本例中就需要对 book 节点下的 chapter 元素进行排序。目前，能够处理数值型属性、连接和次序限制的 XML 查询语言有很多，其中最著名的是 W3C 为标准化而提出的 XQuery 语言，它被设计成可以广泛应用于所有使用 XML 的领域的查询语言。由于 XQuery 语言的复杂性，实现一个基于 XQuery 的排序式 IR 系统，并且希望该系统满足用户对 IR 系统性能的期望，是一项非常具有挑战性的工作，这也是目前 XML 检索中最热门的研究领域之一。

关系数据库能够更好地处理很多结构化限制，特别是连接限制（次序限制在数据库框架下也是很难处理的，因为关系演算中的关系元组并没有排序）。因此，大部分以数据为中心的 XML 检索系统实际上是关系数据库的扩展（参考 10.6 节的相关参考文献）。如果文本字段很短，精确匹配就能满足用户的需求，并且结果以无序集合的方式返回也可以接受的话，那么利用关系数据库进行 XML 检索也是合适的。

习题 **10-4**　在 Web 上找到一个规模适度的 XML 文档集（或者采用不同于 XML 的标记语言的文档集，如 HTML）并下载。Jon Bosak 的有关莎士比亚及多个宗教作品的 XML 版本（www.ibiblio.org/bosak/）或者 Wikipedia 的前 10 000 篇文档都是不错的选择。 按照图 10-3 所示的例子，构造 3 个 CAS 主题，对于这些主题，你期望利用它们进行检索的效果会好于同类的 CO 主题。请解释为什么 XML 检索系统中，可以通过利用文档的 XML 结构来获得比非结构化检索系统更好的检索结果。

习题 **10-5**　对于习题 10-4 中的文档集和查询，(i) 是否存在两个元素 e_1 和 e_2，其中 e_2 是 e_1 的子元素，且它们两个都是某个主题的答案？(ii) 给出一个例子，其中 e_1 是主题的更好答案。(iii) 给出另外一个例子，其中 e_2 是主题的更好答案。

习题 **10-6**　实现 10.3 节介绍的 (i) SimMerge 及 (ii) SimNoMerge 算法，并采用习题 10-4 中的文档和主题进行实验。(iii) 对每个算法的 3 个检索结果的前 5 个文档进行二值相关性判断，基于这些判断来比较这两个算法。

习题 **10-7**　习题 10-4 中的返回元素是否都满足用户的需求或者说是否存在一些需要排除的元素（就像 10.2 节中提到的definitely的例子）？

习题 **10-8**　我们在 10.3 节讨论过结果精度和向量空间维数之间的折衷问题。给出一个需求的例子，如果对所有词汇化子树建立索引则可以正确回答该需求，而如果仅仅索引那些结构化词项则不能回答该需求。

习题 **10-9** 如果对所有的结构化词项进行索引，那么索引大小和文本大小的函数关系怎样？

习题 **10-10** 如果对所有的词汇化子树进行索引，那么索引的大小和文本的大小的函数关系又怎样？

习题 **10-11** 给出一个查询-文档对的例子，其 $\text{SimNoMerge}(q,d)$ 值大于 1.0。

10.6 参考文献及补充读物

有很多很好的有关 XML 的介绍材料，包括（Harold 和 Means 2004）。表 10-1 的灵感来自于（van Rijsbergen 1979）中的一张类似的表。10.4 节主要基于 Gövert 和 Kazai 关于 INEX 2002 的综述（2003），它发表在该会议的论文集上（Fuhr 等人 2003a）。接下来四年的 INEX 会议论文集分别发表在 Fuhr 等人（2003b）、Fuhr 等人（2005）、Fuhr 等人（2006）和 Fuhr 等人（2007）上。在写作本书之际，一篇最新的综述文章来自 Fuhr 和 Lalmas （2007）。表 10-4 的结果来自（Kamps 等人 2006）。Chu-Carroll 等人（2006） 给出的证据也表明，XML 查询能够比非结构化查询提供更高的准确率。与以往的覆盖度和相关度不同，INEX 现在从两个相关但又不同的维度来评价，它们分别是详尽性（exhaustivity）和特定性（specificity）（Lalmas 和 Tombros 2007）。Trotman 等人（2006）将 INEX 中考察的任务和现实世界对结构化检索的使用联系起来，比如在网上书店中进行结构化书籍搜索。

结构化文档检索原则归功于 Chiaramella 等人（1996）。图 10-5 来源于（Fuhr 和 GroBjohann 2004）。Rahm 和 Bernstein（2001） 给出了一篇 XML schema 自动匹配的综述。

10.3 节给出的基于向量空间的 XML 检索方法实际上源自 IBM Haifa 的 JuruXML 系统，它在 Mass 等人（2003）和 Carmel 等人（2003）中有所介绍。Schlieder 和 Meuss（2002）及 Grabs 和 Schek（2002）也给出了类似的方法。Carmel 等人（2003）将查询表示为 XML 片段（XML fragment）。本章中所有表示 XML 查询的树都是 XML 片段，但是 XML 片段还可以允许在内容节点上使用+、−和 phrase 等操作符。

本章只选择向量空间模型来进行 XML 检索，只是因为它非常简单，而且能很自然地从第 6 章非结构化向量空间模型扩展过来。 但是，很多其他的非结构化检索方法也已经应用到 XML 检索，而且至少取得了和向量空间模型一样的成功。这些方法包括语言模型（参考第 12 章，相关文献包括 Kamps 等人（2004）、List 等人（2005）及 Ogilvie 和 Callan（2005））、以关系数据库为后端的系统（如 Mihajlović 等人 2005; Theobald 等人 2005, 2008）、基于概率的权重计算机制（如 Lu 等人 2007）以及融合方法（如 Larson 2005）等等。目前为止，在 XML 检索中到底哪种方法最好还没有定论。

大部分有关 XML 检索的早期工作主要是在相关性排序中重点关注每个独立的词项，包括它们在查询和文档中的结构化上下文。像非结构化 IR 一样，近年来的工作存在这样一种趋势，即把对相关性排序的建模视为对多种不同证据的融合，这些证据来自查询、文档及它们之间的

匹配。融合函数可以通过人工来调优（Arvola 等人 2005; Sigurbjörnsson 等人 2004）或使用机器学习方法来训练（Vittaut 和 Gallinari（2006），参考 15.4.1 节）。

XML 检索研究的一个活跃领域是 focused retrieval（Trotman 等人 2007），它的目标是避免返回共享一个或多个公共子元素的嵌套元素（参考 10.2 节的讨论）。有证据表明，用户不喜欢元素嵌套带来的冗余性（Betsi 等人 2006）。focused retrieval 的评价指标需要对冗余结果进行惩罚（Kazai 和 Lalmas 2006；Lalmas 等人 2007）。Trotman 和 Geva（2006）认为 XML 检索是段落检索（passage retrieval）的一种形式。在段落检索中（Salton 等人 1993；Hearst 和 Plaunt 1993；Zobel 等人 1995；Hearst 1997；Kaszkiel 和 Zobel 1997），检索系统以短段落而不是文档方式来回答用户的查询请求。尽管 XML 文档中的元素边界可能对段落分段有很好的提示作用，但是相关度最高的段落往往并不与 XML 元素相吻合。

过去的数年中，INEX 中的查询格式已经成为 Trotman 和 Sigurbjörnsson（2004）所提出的 NEXI 标准，图 10-3 就来自这篇文章。O'Keefe 和 Trotman（2004）给出的证据表明，用户并不能很可靠地区分子元素和后代元素（child and descendant axes），这也证明了在 NEXI（和 XML 片段）中只允许后代元素是很正当的。在最近的 INEX 会议中，这些结构化的限制仅仅被视为“线索”（hint），也就是说，即使某个元素与 NEXI 查询的某个结构化限制条件相抵触，它也可能被人判为高度相关。

在使用像 NEXI 一样的结构化查询语言之外，还可以使用一种更复杂的支持查询重构的用户界面（Tannier 和 Geva 2005; van Zwol 等人 2006; Woodley 和 Geva 2006）。

一个更全面的 XML 检索介绍参见 Amer-Yahia 和 Lalmas（2006），其中既包含了数据库方法也包含了 IR 方法。有关这个主题的一个详细参考文献列表可以参考（Amer-Yahia 等人 2005）。Grossman 和 Frieder（2004）的第 6 章是一个从数据库角度出发的介绍结构化文本检索的很好的材料。XQuery 的提交标准可以参见 www.w3.org/TR/xquery/，其中包含全文本查询的一个扩展（Amer-Yahia 等人 2006）：www.w3.org/TR/xquery-full-text/。将关系数据库和非结构化 IR 方法融合在一起的工作可以参考文献（Fuhr 和 Rölleke 1997）、（Navarro 和 Baeza-Yates 1997）、（Cohen 1998）和（Chaudhuri 等人 2006）。

第 11 章

概率检索模型

在 9.1.2 节讨论相关反馈时，我们已经注意到：如果已知一些相关文档和不相关文档，那么就可以直接估计词项 t 在相关文档中的出现概率 $P(t|R=1)$，基于这些概率可以构造一个分类器来判定某文档是否相关。本章将更系统地介绍这种基于概率的 IR 方法，它为检索模型提供了一个与以往不同的形式化基础，并且基于它可以导出不同的词项权重计算方法。

用户起始于信息需求(information need),然后将这些需求转换成查询表示(query representation)。类似地，文档（document）也可以转换成文档表示（document Representation）[①]。转换后的文档表示和原始文档相比，至少会由于经过词条化处理而不同，有时其包含的信息量还会明显偏少，比如在使用无位置信息的索引时。基于上述两种表示结果，系统试图来确定文档对信息需求的满足程度。在布尔模型和向量空间模型中，这种满足程度通过索引词来计算，但是上述两种模型中的计算方式虽有形式化定义，但在语义上并不精确。在仅仅给定查询的情况下，IR难以精确理解其背后的信息需求。即给定查询表示和文档表示，系统只能给出文档内容和需求是否相关的一个非确定性推测。而概率论可以为这种非确定性推理提供一个基本的理论。本章介绍的就是基于该理论来估计文档和需求的相关程度的方法。

目前存在多种基于概率的 IR 模型。本章中，我们首先介绍概率论的基础知识以及概率排序原理（参见 11.1 节和 11.2 节）。然后主要关注二值独立模型（binary independence model），它是最早出现并至今仍有很大影响的模型（参见 11.3 节）。最后，我们将介绍对基本模型的扩展方法，这些方法中使用词项出现的次数，其中包括著名的在经验上非常成功的 Okapi BM25 权重计算方法。另外我们还将介绍基于贝叶斯网络的 IR 模型（参见 11.4 节）。在第 12 章中，我们还将介绍近年来取得成功的基于概率语言建模的信息检索模型。

11.1 概率论基础知识

我们希望读者已经了解了概率论的一些基本知识，这里只做个简单回顾，进一步的概率论知识可以参考本章末尾给出的相关文献。假定变量A代表一个事件，它是所有可能的结果构成的空间[②]上的一个子集[①]。同样，我们可以通过随机变量（random variable）来代表该子集，它

① 从文档到文档表示的转换至少会因为词条化处理的不同而不同。也有可能因为处理而使得转换后的结果包含更少的信息，比如采用无位置信息的索引时就少了词的位置信息。

② 这个空间往往称为样本空间、结果空间或者基本事件空间。——译者注

是从结果到实数的一个映射函数。对应该子集，随机变量A会取一个具体的值。通常我们并不能确信某个事件是否为真，此时，我们想知道事件A发生的概率P(A)，它满足$0 \leqslant P(A) \leqslant 1$。对于两个事件$A$、$B$，它们的联合事件发生的可能性通过联合概率$P(A,B)$来描述。条件概率$P(A|B)$表示在事件$B$发生的条件下$A$发生的概率。联合概率和条件概率的关系可以通过如下的链式法则（chain rule）来体现：

$$P(AB) = P(A \cap B) = P(A \mid B)P(B) = P(B \mid A)P(A)。 \quad (11\text{-}1)$$

上式表明，在不做任何假设的条件下，两个事件的联合概率等于其中一个事件的概率乘上在该事件发生的条件下另一个事件发生的条件概率。

事件A的补集的概率记为$P(\overline{A})$，同样有

$$P(\overline{A}B) = P(B \mid \overline{A})P(\overline{A})。 \quad (11\text{-}2)$$

概率论中的全概率定理（partition rule）为：如果事件B可以划分成互不兼容（即互斥）的多个子事件，那么B的概率将是所有子事件概率的和。下式给出的是该定理的一个特例：

$$P(B) = P(AB) + P(\overline{A}B)。 \quad (11\text{-}3)$$

基于上述结果可以推导出贝叶斯定理（Bayes's rule）：

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} = \left[\frac{P(B \mid A)}{\sum_{X \in \{A, \overline{A}\}} P(B \mid X)P(X)} \right] P(A)。 \quad (11\text{-}4)$$

上述公式也可以理解成一种对概率进行修正的方法。首先，在没有任何其它信息的情况下我们对事件A的可能性给出一个初始估计$P(A)$，这称为先验概率（prior probability）。贝叶斯定理可以在看到B的情况下，基于B出现在A或$\overline{A}$两种情况下的似然（likelihood）[②]来计算后验概率（posterior probability）$P(A|B)$。

最后值得一提的是，另外一个常用的概念是事件的优势率（odds），它提供了一种反映概率如何变化的“放大器”（multiplier）：

$$\text{优势率：} O(A) = \frac{P(A)}{P(\overline{A})} = \frac{P(A)}{1 - P(A)}。 \quad (11\text{-}5)$$

11.2 概率排序原理

11.2.1 1/0 风险的情况

① 比如掷一个骰子，可能出现的朝上的点数集合为{1，2，3，4，5，6}，某个事件为{有奇数个点朝上}={1，3，5}。——译者注

② 似然是概率的同义词，它指的是根据某个模型得到的事件或数据的概率。在人们考虑模型变化而数据固定时，常常使用似然这个术语。

假定像 6.3 节一样，我们给出一个排序式检索系统所需要的基本配置：包括文档集、用户输入的查询以及返回的有序文档结果。像第 8 章一样，我们仍然假设相关度是二值的，即要么相关，要么不相关。对查询 q 和文档集中的一篇文档 d，假定变量 $R_{d,q}$ 代表 d 和查询 q 是否相关，也就是说，当文档 d 和查询 q 相关时 $R_{d,q}$ 的取值为 1，不相关时 $R_{d,q}$ 的取值为 0。在不造成上下文歧义的情况下，我们将 $R_{d,q}$ 简记为 R。

于是，可以利用概率模型来估计每篇文档和需求的相关概率 $P(R=1|d,q)$，然后对结果进行次序。这是 PRP（*probability ranking principle*，概率排序原理），参见 van Rijsbergen 1979，113～114 页）：

如果某个参照检索系统对每个需求进行应答时，会按照文档和需求的相关性概率从大到小排序，其中相关性概率是基于系统能得到的所有数据来尽可能精确估计而得到的，那么该系统是基于已知数据的可以获得的总体效果最优的系统。

最简单的 PRP 情况是，检索没有任何代价因子，或者说不会对不同行为或错误采用不同的权重因子。在返回一篇不相关文档或者返回一篇相关文档不成功的情况下，将失去 1 分（在计算精确率时这种基于二值的情形也往往称为 1/0 风险）。而检索的目标是对于用户任意给定的 k 值，返回可能性最高的文档前 k 篇作为结果输出。也就是说，PRP 希望可以按照 $P(R=1|d,q)$值的降序来排列所有文档。当返回一个无序文档集而不是排序的结果时，可以通过贝叶斯最优决策原理（Bayes optimal decision rule）来基于最小损失风险作出决策，即返回相关的可能性大于不相关的可能性的文档：

当且仅当 $P(R=1|d,q) > P(R=0|d,q)$时，d 相关。　　（11-6）

定理 11-1　在 1/0 损失的情况下，PRP 对于最小化期望损失（也称为贝叶斯风险）而言是最优的。

相关的证明可以参见 Ripley（1996）。然而，这需要知道所有概率的正确值，这在实际当中是不可能的。尽管如此，PRP 仍然为发展 IR 模型提供了一个非常有益的基础。

11.2.2　基于检索代价的概率排序原理

我们假定检索中存在检索代价，在此基础上来给出一个检索模型。令C_1表示一篇相关文档未返回所发生的代价，而C_0表示返回一篇不相关文档发生的代价。PRP认为，如果对于一篇特定的文档d及所有其他未返回的文档d'都满足[①]

$$C_0 \cdot P(R=1|d) - C_1 \cdot P(R=0|d) \leq C_0 \cdot P(R=1|d') - C_1 \cdot P(R=0|d')，\qquad (11\text{-}7)$$

那么 d 就应该是下一篇被返回的文档。这种模型给出一个形式化的框架，便于在建模阶段就将假正例和假反例的不同代价甚至系统性能问题考虑在内，而不是像以前只在评价阶段考虑

① 这个式子的左部和右部可以分别看成是 d 和 d' 的函数，每个函数表示一篇文档的代价。它由两部分组成：一个是当它不相关时却返回的代价 $C_0 \cdot P(R=1|d)$，另一个是相关时却没有被返回所造成的代价 $C_1 \cdot P(R=0|d)$。两者相减表示返回文档 d 的代价函数，也即此时前者越低越好，后者越高越好。公式(11-7)的意思是我们选代价函数最小的文档。——译者注

这些因素（参见 8.6 节）。然而，本章将不再考虑基于损失或代价的模型。

11.3 二值独立模型

这节要介绍的 BIM（binary independence model，二值独立模型）是在传统上随同 PRP 一起使用的一种模型。为了能够在实际中对概率函数 $P(R|d,q)$ 进行估计，该模型中引入了一些简单的假设。在这里，二值等价于布尔值：文档和查询都表示为词项出现与否的布尔向量。也就是说，文档 d 表示为向量 $\vec{x}=(x_1,\cdots,x_M)$，其中当词项 t 出现在文档 d 中时，x_t=1，否则 x_t=0。在这种表示下，由于不考虑词项出现的次数及顺序，许多不同的文档可能都有相同的向量表示。类似地，我们将查询 q 表示成词项出现向量 $\vec{q}$（由于查询 q 通常就是采用一系列词的集合来表示，所以 q 和 $\vec{q}$ 的之间的区别并不十分重要）。“独立性”指的是词项在文档中的出现是互相独立的，BIM 并不识别词项之间的关联。独立性假设和实际情况很不相符，但在实际中常常却能给出令人满意的结果。这种假设实际上就是后面我们要在 13.4 节介绍的朴素贝叶斯模型中的“朴素”假设。本质上说，BIM 模型就是 13.3 节中要介绍的多元贝努利朴素贝叶斯模型。在某种意义上说，这个假设等价于在向量空间模型中将每个词项作为一维并和其他词项正交的假设。

首先，我们给出一个在用户仅有单步（sing-step）[①]信息需求下的模型。我们在第 9 章提到，用户在见到一系列结果之后可以对信息需求进行更新。幸而，正如在第 9 章所见的那样，我们可以直接对BIM进行扩展，扩展后的模型将能够提供一个相关反馈的框架。我们将在 11.3.4 节介绍这个模型。

为保证概率检索策略的计算精度，我们需要估计文档中的词项如何对相关性计算产生影响。具体地，我们想知道词项频率、文档频率、文档长度和其它能够计算出的统计特性如何影响文档相关性的判断结果，同时也想知道如何组合这些因素来估计文档相关的概率。这样的话，我们就能按照相关概率从高到低对文档进行排序。

这里我们假设每篇文档的相关性与其它文档的相关性无关。正如在 8.5.1 节指出的那样，这个假设显然是错误的，当允许系统返回冗余或者近似冗余文档时，这个假设在实际当中尤其有害。在 BIM 模型下，我们基于词项出现向量的概率 $P(R|\vec{x},\vec{q})$ 对概率 $P(R|d,q)$ 建模。然后，利用贝叶斯定理，有

$$P(R=1|\vec{x},\vec{q})=\frac{P(\vec{x}|R=1,\vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})},$$
$$P(R=0|\vec{x},\vec{q})=\frac{P(\vec{x}|R=0,\vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})}。\quad (11\text{-}8)$$

其中，$P(\vec{x}|R=1,\vec{q})$ 和 $P(\vec{x}|R=0,\vec{q})$ 分别表示当返回一篇相关或不相关文档时文档表示为 $\vec{x}$ 的

① 这里的单步指的是不需要进行查询扩展。——译者注

概率，我们应该将上述概率想象成定义在某个领域所有可能的文档空间上的量[①]。那么，如何来计算这些可能的概率值？实际上我们不可能知道这些概率的精确值，所以要进行估计。通常我们用实际文档集分布的统计特性来估计这些概率。$P(R=1|\vec{q})$ 和 $P(R=0|\vec{q})$ 分别表示对于查询 $\vec{q}$ 返回一篇相关和不相关文档的先验概率。当然，如果我们知道文档集中相关文档的百分比，那么就可以用这个百分比来估计 $P(R=1|\vec{q})$ 和 $P(R=0|\vec{q})$。另外，对于某个查询，一篇文档要么和它相关，要么和它不相关，所以有

$$P(R=1|\vec{x},\vec{q})+P(R=0|\vec{x},\vec{q})=1。 \quad (11\text{-}9)$$

11.3.1　排序函数的推导

给定查询q，我们将按照$P(R=1|d,q)$从高到低将所有文档排序。在BIM模型下，也就是要按照 $P(R=1|\vec{x},\vec{q})$ 来排序。由于检索系统关心的只是文档的相对次序，所以这里并不需要直接估计出这个概率值，而是采用其它的更容易计算的排序函数，这中间只需要保证采用排序函数和直接计算概率所得到的文档次序一致即可。具体地，我们可以根据文档相关性的优势率来对文档排序，它是相关性概率的单调递增函数[②]，这样的话就可以忽略公式(11-8)中的公共分母，使得计算起来更容易。文档相关性的优势率定义如下：

$$O(R|\vec{x},\vec{q})=\frac{P(R=1|\vec{x},\vec{q})}{P(R=0|\vec{x},\vec{q})}=\frac{\frac{P(R=1|\vec{q})P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|\vec{q})}}{\frac{P(R=0|\vec{q})P(\vec{x}|R=0,\vec{q})}{P(\vec{x}|\vec{q})}}=\frac{P(R=1|\vec{q})}{P(R=0|\vec{q})}\cdot\frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})}。 \quad (11\text{-}10)$$

对于给定查询来说，上述公式中的 $\frac{P(R=1|\vec{q})}{P(R=0|\vec{q})}$ 是个常数。由于我们只关注文档排序，所以没有必要估计这个常数。因此，我们只需要估计 $\frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})}$，但是这个问题看上去很难，因为我们要精确估计整个词项出现向量的概率。这里我们引入了朴素贝叶斯条件独立性假设（Naive Bayes conditional independence assumption），即在给定查询的情况下，认为一个词的出现与否与任意一个其他词的出现与否是互相独立的，即

$$\frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})}=\prod_{t=1}^{M}\frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}。 \quad (11\text{-}11)$$

因此有

$$O(R|\vec{x},\vec{q})=O(R|\vec{q})\cdot\prod_{t=1}^{M}\frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}。 \quad (11\text{-}12)$$

由于每个 x_t 的取值要么为 0 要么为 1，所以有

$$O(R|\vec{x},\vec{q})=O(R|\vec{q})\cdot\prod_{t:x_t=1}\frac{P(x_t=1|R=1,\vec{q})}{P(x_t=1|R=0,\vec{q})}\cdot\prod_{t:x_t=0}\frac{P(x_t=0|R=1,\vec{q})}{P(x_t=0|R=0,\vec{q})}。 \quad (11\text{-}13)$$

---

① 也就是说，对任意一篇文档 $\vec{x}$，都可以求它的 $P(\vec{x}|R=1,\vec{q})$和$\vec{P}(\vec{x}|R=1,\vec{q})$。——译者注

② 当 0≤P<1 时，很容易证明 P/(1-P)是 P 的单调递增函数。——译者注

简便起见，记 $p_t = P(x_t = 1 | R = 1, \vec{q})$，即词项出现在一篇相关文档中的概率，同样令 $u_t = P(x_t = 1 | R = 0, \vec{q})$，即词项出现在一篇不相关文档中的概率[①]。这些值之间的关系展示在如下的列联表中，其中每列值的和为 1。

	文档	相关（R=1）	不相关（R=0）
词项出现	x_t=1	p_t	u_t
词项不出现	x_t=0	$1-p_t$	$1-u_t$

（11-14）

可以对上述式子进行进一步的简化，假定没有在查询中出现的词项在相关和不相关文档中出现的概率相等，即当 q_t=0 时，p_t=u_t（当然，我们也可以像 11.3.4 节进行相关反馈时那样对此假设进行修改）。因此，我们只需要考虑在查询中出现的词项的概率的乘积，于是有

$$O(R | \vec{x}, \vec{q}) = O(R | \vec{q}) \cdot \prod_{t: x_t = q_t = 1} \frac{p_t}{u_t} \cdot \prod_{t: x_t = 0, q_t = 1} \frac{1 - p_t}{1 - u_t} \tag{11-15}$$

其中，$\prod_{t: x_t = q_t = 1} \frac{p_t}{u_t}$ 计算的是出现在文档中的查询词项的概率乘积，而第 3 个因子计算的是不出现在文档中的查询词项的概率乘积。

对上述公式可以进一步转换，有

$$O(R | \vec{x}, \vec{q}) = O(R | \vec{q}) \cdot \prod_{t: x_t = q_t = 1} \frac{p_t(1 - u_t)}{u_t(1 - p_t)} \cdot \prod_{t: q_t = 1} \frac{1 - p_t}{1 - u_t} 。 \tag{11-16}$$

此时，公式中因子 $\prod_{t: x_t = q_t = 1} \frac{p_t(1 - u_t)}{u_t(1 - p_t)}$ 仍然基于出现在文档中的查询词项来计算，而此时 $\prod_{t: q_t = 1} \frac{1 - p_t}{1 - u_t}$ 考虑的是所有查询词项。显然，对于给定的查询而言，第 3 个因子就像 $O(R | \vec{q})$ 一样是个常数。因此文档排序中唯一需要估计的量是 $\prod_{t: x_t = q_t = 1} \frac{p_t(1 - u_t)}{u_t(1 - p_t)}$。该模型中，最后用于排序的量称为 RSV（retrieval status value，检索状态值）：

$$RSV_d = \log \prod_{t: x_t = q_t = 1} \frac{p_t(1 - u_t)}{u_t(1 - p_t)} = \sum_{t: x_t = q_t = 1} \log \frac{p_t(1 - u_t)}{u_t(1 - p_t)} 。 \tag{11-17}$$

因此，所有的计算最后都归结于 RSV 值，定义 c_t：

$$c_t = \log \frac{p_t(1 - u_t)}{u_t(1 - p_t)} = \log \frac{p_t}{1 - p_t} + \log \frac{1 - u_t}{u_t} 。 \tag{11-18}$$

c_t 是查询词项的优势率比率（odds ratio）的对数值。当查询词项出现在相关文档时，优势率为 p_t/（1-p_t）；当查询词项出现在不相关文档时，优势率为 u_t/(1-u_t)。优势率比率是上述两个优势率的比值，最后对这个值取对数。如果词项在相关和不相关文档中的优势率相等，c_t 值为 0。如果词项更可能出现在相关文档中，那么该值为正。c_t 实际上给出的是模型中词项的权重，而

① 这里的概率是指一旦文档相关（或不相关），那么出现该词项的可能性。所有词项的这个概率值之和不为 1，比如两个词项的概率值都可以是 1。不要和词项在文档内的出现概率相混淆。——译者注

查询文档的得分是 $RSV_d = \sum_{x_t=q_t=1} c_t$。运算时，我们通过一个累加器对当前文档中的查询词项的权重进行累加求和，这和 7.1 节中讨论向量空间模型的计算一样。下面我们将介绍在给定文档集和查询的情况下如何估计 c_t。

11.3.2　理论上的概率估计方法

对于每个词项 t，在整个文档集中 c_t 的值到底应该怎样计算？公式(11-19)给出了一个在文档集中不同类型的文档数目的列联表，其中 df_t 是包含 t 的文档数目。

	文档	相关	不相关	总计
词项出现	x_t=1	s	df_t–s	df_t
词项不出现	x_t=0	S–s	(N–df_t)–(S–s)	N–df_t
	总计	S	N–S	N

（11-19）

基于上述表格中的数值进行计算，有 $p_t=s/S$，$u_t=(\mathrm{df}_t-s)/(N-S)$，因此有

$$c_t = K(N, df_t, S, s) = \log \frac{s/(S-s)}{(df_t - s)/((N-df_t)-(S-s))}。\quad (11\text{-}20)$$

为了避免可能出现的 0 概率（比如所有的相关文档都包含或不包含某个特定的词项），一种很常规的做法是在公式(11-19)中的每个量的基础上都加上 $\frac{1}{2}$，因此总数也做相应改变（比如上述表格的右下角的总数也会变成 N+2）。于是有

$$\hat{c}_t = K(N, df_t, S, s) = \log \frac{(s+\frac{1}{2})/(S-s+\frac{1}{2})}{(df_t - s+\frac{1}{2})/(N-df_t-S+s+\frac{1}{2})}。\quad (11\text{-}21)$$

加上 $\frac{1}{2}$ 是一种简单的平滑方法。对于输出结果为类别型（比如词项出现或不出现两个类别）的试验来说，往往可以通过事件发生的次数除以试验的总次数来从数据中估计出事件的概率。这被称为事件的相对频率（relative frequency）。由于相对频率使得观察数据出现的概率最大，所以这种估计称为MLE（maximum likelihood estimate，最大似然估计）。但是，如果我们只是简单地使用MLE，那么在观察数据中出现过的事件的概率估计值总是非常高，而那些没有在观察数据中出现的事件的相对频率则为 0，这不仅低估了这些事件的概率值，往往也损害了模型本身，因为 0 乘以任何数得 0。在减少出现事件的概率估计值的同时提高未出现事件的概率估计值的方法称为平滑（smoothing）[①]。一种最简单的平滑方法就是对每个观察到的事件的数目都加上一个数α。这样得到的伪数目（pseudocount）相当于在所有词汇表上使用了均匀分布作为一个贝叶斯先验（Bayesian prior，参见公式 11-4）。一开始我们假设事件的分布是均匀的，α的大

① 严格地说，采用平滑技术时，不一定所有出现的事件的概率估计值都会下降。一个具体的例子参考公式(12-10)，当某个出现词项的 MLE 估计 $\hat{P}_{\mathrm{mle}}(t|M_d)$ 比整个文档集中的出现频度 $\hat{P}_{\mathrm{mle}}(t|M_c)$ 还要小时，平滑之后的估计值可能会比 $\hat{P}_{\mathrm{mle}}(t|M_d)$ 还大。——译者注

小表示我们对均匀分布的信心强度，然后我们根据观察到的事件来更新概率。由于对均匀分布的信心强度较弱，我们令$\alpha=\frac{1}{2}$。这是MAP（maximum a posteriori，最大后验估计）的一种形式，相当于在先验知识和观察到的证据的基础上按照公式(11-4)得到的最有可能的概率值。12.2.2 节中将会进一步介绍概率模型中的其他平滑估计方法，目前我们只是简单地对每个观察到的数目加上$\frac{1}{2}$。

11.3.3 实际中的概率估计方法

假设相关文档只占所有文档的极小一部分，那么就可以通过整个文档集的统计数字来计算与不相关文档有关的量。在该假设下，在某查询下不相关文档中出现词项 t 的概率 $u_t=df_t/N$，于是有

$$\log[(1-u_t)/u_t]=\log[(N-df_t)/df_t]\approx\log N/df_t。 \quad (11\text{-}22)$$

换句话说，我们可以为最常使用的 idf 权重计算方法（参见 6.2.1 节）给出一个理论上的合理解释。

当然，公式(11-22)中的估计方法很难推广到相关文档中概率值 p_t 的计算上去。而对于 p_t，可以采用如下几种方式来估计。

1. 如果我们知道某些相关文档，那么可以利用这些已知相关文档中的词项出现频率[①]来对 p_t 进行估计。这是基于概率的相关反馈机制中每次反馈循环的基础，我们将在 11.3.4 节对此展开讨论。

2. Croft 和 Harper（1979）在他们的组合匹配模型（combination match model）中提出了利用常数来估计 p_t 的方法。比如，我们可以假设所有查询中的词项的 p_t 是个常数 $p_t=0.5$。这意味着在相关文档中出现的每个词项有一个等值的优势率，所以 p_t 和 $(1-p_t)$ 因子可以在 RSV 的计算中约掉。这种估计比较弱，但是我们只是期望搜索词项出现在很多但并非所有的相关文档中，而上述估计与该期望并没有剧烈冲突。将这种 p_t 的估计方法与前面 u_t 的估计方法结合在一起就可以得到最终的文档排序函数，它实际上等于所有出现在文档中的查询词项的 idf 权重之和。在不想对短文档（标题或摘要）进行反复迭代搜索的情况下，单独利用上述词项权重计算方法可以取得令人满意的结果。当然对于其他的情况，我们想做得更好。

3. Greiff（1998）指出，Croft 和 Harper（1979）中 p_t 的常数估计方法不仅理论上存在缺陷，在经验上观察结果与期望结果也不相符。可以预料，p_t 会随 df_t 的增长而增长。基于对数据的分析，文章提出了一个自认为更合理的计算方法 $p_t=\frac{1}{3}+\frac{2}{3}\cdot\frac{df_t}{N}$。

下一节，我们将介绍结合了上述思想的迭代式估计方法。

11.3.4 基于概率的相关反馈方法

① 这里实际上指的是出现 t 的相关文档的数目，而并不是指 t 在某篇文档中的出现次数。——译者注

可以通过（伪）相关反馈技术，不断迭代估计过程来获得 p_t 的更精确的估计结果。基于概率的相关反馈方法如下。

(1) 给出 p_t 和 u_t 的初始估计。这可以通过前面提到的概率估计方法来实现，比如，假设所有查询中的词项的 p_t 是个常数，具体地可以取 p_t=0.5。

(2) 利用当前的 p_t 和 u_t 的估计值对相关文档集合 $R = \{d : R_{d,q} = 1\}$ 进行最佳的猜测。利用该模型返回候选的相关文档集并呈现给用户。

(3) 利用用户交互对上述模型进行修正，这是通过用户对某个文档子集 V 的相关性判断来实现的。基于相关性判断结果，V 可以划分成两个子集：$VR = \{d \in V, R_{d,q} = 1\} \subset R$ 和 $VNR = \{d \in V, R_{d,q} = 0\}$，后者与 R 没有交集。

(4) 利用已知的相关文档和不相关文档对 p_t 和 u_t 进行重新估计。如果 VR 和 VNR 足够大的话，可以直接通过集合中的文档数目来进行最大似然估计：

$$p_t=|VR_t|/|VR|。 \tag{11-23}$$

其中，VR_t 是 VR 中包含词项 x_t 的文档子集。实际中往往要对上述估计进行平滑，此时可以对包含和不包含词项的文档数目都加上 $\frac{1}{2}$，从而得到

$$p_t = \frac{|VR_t| + \frac{1}{2}}{|VR| + 1}。 \tag{11-24}$$

但是，用户判断的文档往往都比较少，也就是 V 较小，因此即使是在平滑的情况下，上述估计也非常不可靠。因此，将新信息和原有估计通过贝叶斯更新方法组合在一起往往更好。这种情况下，我们有

$$p_t^{(k+1)} = \frac{|VR_t| + \kappa p_t^{(k)}}{|VR| + \kappa}。 \tag{11-25}$$

其中，$p_t^{(k)}$ 是第 k 次迭代以后得到的更新值，在下一次迭代中作为贝叶斯先验值并结合权重 κ 使用。将这个式子和公式(11-4)联系起来还需用到比现在介绍的更多的概率论知识，比如我们需要使用贝塔先验分布，它是贝努利随机变量 Xt 的共轭分布。当然，这里给出的计算结果非常直观：对伪数目我们不是基于均匀分布，而是按照上次的估计值来分布 κ 个伪数目，后者可以作为下一次估计的先验分布。在缺乏其他信息的情况下（假定用户或许只是粗略地给出了 5 篇相关或不相关文档），κ 取 5 比较合适。也就是说，在只有一小部分文档信息的情况下更强调先验知识，估计值因此不会发生太大的改变。

(5) 重复第 2 步～第 4 步，不断产生 R 和 p_t 的估计值，直到用户满意为止。

同样，很直观地，也可以基于伪相关反馈的方法来实现上述算法。比如假定 $VR=V$，整个过程简述如下。

(1) 同前面一样给出 p_t 和 u_t 的初始估计。

(2) 预测相关文档集的大小。如果不能肯定，最好采用比较保守的估计（认为相关文档集很小）。这种思路往往导致使用固定大小的高排名的文档集 V。

(3) 对 p_t 和 u_t 的估计进行修正。这里我们仍然沿用了公式(11-23)和公式(11-25)的思路来估计 p_t，不同的是这里采用集合 V 来代替集合 VR。记 V 中包含 x_t 的文档集为 V_t 并采用加 1/2 的平滑方法，则有

$$p_t = \frac{|V_t| + \frac{1}{2}}{|V| + 1}。 \tag{11-26}$$

此时假定所有的未返回文档都是不相关的，则可以按照如下方式更新 u_t：

$$u_t = \frac{df_t - |V_t| + \frac{1}{2}}{N - |V| + 1}。 \tag{11-27}$$

(4) 重复第 2 步 ~ 第 3 步，直到返回结果的序收敛。

一旦对 p_t 得到实际的估计值，RSV 计算中用到的 c_t 看上去就像 tf-idf 值。比如，利用公式(11-18)、公式(11-22)和公式(11-26)，我们有

$$c_t = \log\left[\frac{p_t}{1-p_t}\cdot\frac{1-u_t}{u_t}\right] \approx \log\left[\frac{|V_t| + \frac{1}{2}}{|V| - |V_t| + 1}\cdot\frac{N}{df_t}\right]。 \tag{11-28}$$

但是，事情并不完全一样：$p_t/(1-p_t)$ 计算的是词项 t 出现在相关文档的比例，而不是词项频率。更进一步来讲，如果使用 log 函数，我们有

$$c_t = \log\frac{|V_t| + \frac{1}{2}}{|V| - |V_t| + 1} + \log\frac{N}{df_t}。 \tag{11-29}$$

我们会看到我们是对两个对数化的数值求和，而不是求积。

? 习题 **11-1** 根据公式(11-18)和公式(11-19)推导出公式(11-20)。

习题 **11-2** 当不提供任何文档相关信息时，向量模型中的标准 tf-idf 权重计算方法和 BIM 概率检索模型有什么不同？

习题 **11-3** [**] 令 X_t 表示词项 t 在文档中出现与否的随机变量。假定文档集中有$|R|$篇相关文档，所有文档中有 s 篇文档包含词项 t，即在这 s 篇文档中 X_t=1。假定所观察到的数据就是这些 X_t 在文档中的分布情况。请证明采用 MLE 估计方法对参数 $p_t = (X_t = 1 | R = 1, \vec{q})$ 进行估计的结果，即使得观察数据概率最大化的参数值为 $p_t = s/|R|$。

习题 **11-4** 试描述向量空间相关反馈方法和概率相关反馈方法的区别。

11.4 概率模型的相关评论及扩展

11.4.1 概率模型的评论

概率模型是最早的形式化 IR 模型之一。上世纪 70 年代概率模型就有机会将 IR 置于一个更

坚固的理论基础之上，但是这种愿望最终却没有实现。随着上世纪 90 年代概率方法在计算语言学领域的复苏，这种愿望又重新回归，概率的方法再次成为当前最热门的 IR 话题之一。传统上说，概率检索方法的思想很好但是在性能上却从来没有优势。对于一个概率检索模型来说，对所需概率的合理近似是完全可能的，但是它往往需要一些重要的假设。在 BIM 模型中，这些假设包括：

- 文档、查询及相关性的布尔表示；
- 词项的独立性；
- 查询中不出现的词项不会影响最后的结果；
- 不同文档的相关性之间是互相独立的。

或许因为这些假设过于苛刻，检索性能很难达到较好的水平。一个普遍问题是，概率模型需要部分的相关性信息，否则只能得到一个较差的由词项权重计算方式构成的模型。

20 世纪 90 年代，在 BM25 权重计算机制表现出非常好的检索性能之后，上述问题才有所转变，BM25 也被多个研究队伍作为权重计算公式。关于 BM25，我们将在后面介绍。基于向量空间模型和概率模型的 IR 系统之间的差别已经不是很大。两种模型下，我们都会像第 7 章介绍的那样建立一个信息检索机制。所不同的是，概率 IR 系统最后不是通过向量空间模型中所用的余弦相似度和 tf-idf 来对文档打分，而是通过一个从概率论推导出的稍微有点区别的公式来打分。实际上，有时候人们已经通过简单引入概率模型中的词项权重计算方法，而将现存的基于向量空间模型的 IR 系统改造成一个高效的概率检索系统。本节将简单介绍传统概率模型的 3 种扩展形式，在第 12 章，我们将介绍另外一种基于概率语言模型的信息检索方法。

11.4.2　词项之间的树型依赖

BIM 中的一些假设可以去掉。比如，可以把词项的独立性假设给去掉。该假设远远不符合实际情况，一个实际的例子是类似 Hong 和 Kong 的词项对，这些词项之间存在很强的依赖性。但是，这些依赖可能有多种不同的复杂方式，比如在集合{New, York, England, City, Stock, Exchange, University}中的各个词项之间就存在非常复杂的依赖关系。van Rijsbergen（1979）给出了一个非常简单的、但看起来很合理的允许词项之间存在树型依赖的模型（参见图 11-1）。该模型中，每个词项可能直接依赖于一个其他词项，最后得到一个树型依赖结构。该模型提出于 20 世纪 70 年代，但模型估计问题阻碍了该模型在实际中的成功应用。但是，上述模型的思路再次在 Friedman 和 Goldszmidt（1996）提出的树增强朴素贝叶斯模型（tree-augmented Naive Bayes model）中得到应用，作者在多个机器学习数据集上都获得了一定的成功。

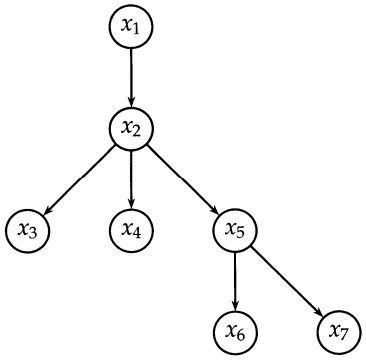


图 11-1 词项之间的树型依赖。在上图模型表示中，如果存在箭头 $x_k \rightarrow x_i$，那么词项 x_i 直接依赖于词项 x_k

11.4.3 Okapi BM25: 一个非二值的模型

BIM 模型最初主要为较短的编目记录（catalog record）和长度大致相当的摘要所设计，在这些环境下它用起来也比较合适。但是对现在的全文搜索文档集来说，很显然应该像第 6 章提到的那样，模型应该重视词项频率和文档长度。一种称为 BM25 权重计算机制（BM25 weighting scheme）或 Okapi 权重计算机制（Okapi weighting）的方法第一次在某个检索系统实施之后，便发展成为基于词项频率、文档长度等因子来建立概率模型的一种方法，并且它不会引入过多的模型参数（Spärck Jones 等人 2000）。这里不讨论 Okapi 模型背后的全部理论，而只给出目前所用的一些常规文档评分的形式。对于文档 d，最简单的文档评分方法是给文档中的每个查询词项仅仅赋予一个 idf 权重（参见公式 11-22）。

$$RSV_d = \sum_{t \in q} \log \frac{N}{df_t} 。 \qquad (11\text{-}30)$$

有时会采用另外一种 idf 的计算方法。此时，我们基于公式（11-21）来计算，但是并不考虑相关反馈信息，即 $S=s=0$，那么可以得到另外一种 idf 的计算方法：

$$RSV_d = \sum_{t \in q} \log \frac{N - df_t + \frac{1}{2}}{df_t + \frac{1}{2}} 。 \qquad (11\text{-}31)$$

上述计算公式的表现有点奇怪：如果词项在文档集中出现的文档数目超过一半，那么该公式得到的权重是个负值，这可能不是事先想要的结果。不过，在使用停用词表的情况下，这种情况往往不会发生，我们可以将每个求和量的最小值设为 0。

通过引入词项的频率和文档的长度，公式(11-30)可以进一步修改为

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{df_t} \right] \cdot \frac{(k_1 + 1) tf_{td}}{k_1[(1-b) + b \times (L_d / L_{\text{ave}})] + tf_{td}} 。 \qquad (11\text{-}32)$$

其中，tf_{td} 是词项 t 在文档 d 中的权重，L_d 和 L_{ave} 分别是文档 d 的长度及整个文档集中文档的平

均长度。k_1 是一个取正值的调优参数，用于对文档中的词项频率进行缩放控制。如果 k_1 取 0，则对应 BIM 模型；如果 k_1 取较大的值，那么对应于使用原始词项频率。b 是另外一个调节参数（$0 \leq b \leq 1$），决定文档长度的缩放程度：$b=1$ 表示基于文档长度对词项权重进行完全的缩放，$b=0$ 表示归一化时不考虑文档长度因素。

如果查询很长，那么对于查询词项也可以采用类似的权重计算方法。此时的计算公式如（11-33）所示。特别地，当查询为采用段落来表示的信息需求时，上述做法是非常合适的，但是对于短查询，就没有必要这样做。

$$RSV_d = \sum_{t \in q} \log\left[\frac{N}{df_t}\right] \cdot \frac{(k_1+1)tf_{td}}{k_1[(1-b)+b\times(L_d/L_{ave})]+tf_{td}} \cdot \frac{(k_3+1)tf_{td}}{k_3+tf_{td}}。 \tag{11-33}$$

其中，tf_{tq}是词项t在查询q中的权重。这里k_3 是另一个取正值的调优参数，用于对查询中的词项频率进行缩放控制。在上述公式中，对于查询的长度并没有进行归一化（相当于对应于查询归一化的b参数取值为 0）。由于检索是在单个固定的查询上进行的，所以没有必要对查询的长度进行归一化。理想情况下，整个公式中的参数可以通过在某个开发测试集上进行结果优化而得到。也就是说，可以在一个单独的开发测试集上搜索最优参数来最大化检索的性能。搜索过程可以通过手工方法来实现，或者采用某些优化策略，比如网格搜索方法（grid search）[①]或者一些更高级的方法。搜索得到的最优参数可以用于实际的测试集。如果没有采用上述开发测试集进行优化，那么现有的实验结果表明，这些参数的一个合理的取值范围是：k_1 和k_3 的取值区间为 1.2~2，b取 0.75。

如果存在相关性判断结果，那么可以使用公式(11-21)的完整计算形式来代替公式(11-22)中的 $\log(N/df_t)$近似计算形式，从而得到

$$RSV_d = \sum_{t \in q}\left[\log\left[\frac{\left(|VR_t|+\frac{1}{2}\right)/\left(|VNR_t|+\frac{1}{2}\right)}{\left(df_t-|VR_t|+\frac{1}{2}\right)/\left(N-df_t-|VR|+|VR_t|+\frac{1}{2}\right)}\right] \times \frac{(k_1+1)tf_{td}}{k_1((1-b)+b\times(L_d/L_{\text{ave}}))+tf_{td}} \times \frac{(k_3+1)tf_{tq}}{k_3+tf_{tq}}\right]。 \tag{11-34}$$

这里，VR_t 、NVR_t、VR 的含义参见 11.3.4 节。上述表达式的第 1 部分反映的是相关反馈（如果没有相关反馈信息的话反映的就是 idf 权重计算方法），第 2 部分是考虑文档的词项频率及文档长度归一化的实现方法，第 3 部分考虑的是查询的词项频率。

除了对用户查询提供词项权重计算方法外，在相关反馈中还可以考虑查询扩展（采用自动方法或手工方法），可以在已知的相关文档利用公式（11-21）中的 $\hat{c}_t$ 因子对词项进行排序，并选取最靠前的多个词项构成新的查询，最后基于新查询采用公式（11-34）进行计算。

① 网格搜索的思想很简单，就是把要选择的参数当成坐标格子上的点，然后通过遍历来选择合适的参数。

——译者注

BM25 词项权重计算公式广泛使用在多个文档集和多个搜索任务中并获得了成功。尤其是在 TREC 评测会议上，BM25 的性能表现很好并被多个团队所使用。有关 BM25 实验结果的详细分析和讨论参见文献 Spärck Jones 等人（2000）。

11.4.4 IR 中的贝叶斯网络方法

Turtle 和 Croft （1989, 1991）将贝叶斯网络（Jensen 和 Jensen 2001）引入到信息检索中。贝叶斯网络是一种概率图网络模型。由于详细介绍贝叶斯网络需要花费很大的篇幅，这里我们不介绍该模型的细节，而只是在概念上进行简单介绍。贝叶斯网络是通过有向图来表示不同随机变量之间的概率依赖关系（参见图 11-1），对于任意有向无环图，已经发展出多种在节点之间传递影响度的复杂算法，可以基于任意知识在图上面进行学习和推理。Turtle 和 Croft 利用了一个复杂的图模型来对文档和信息需求之间的复杂依赖关系进行建模。

该模型分解成两部分：文档集网络及查询网络。文档集网络非常大，但是它可以事先计算，它从文档映射成词项和概念。这里的概念指的是文档中出现的词项的某种基于同义词的扩展形式。查询网络规模相对较小，但是每个查询进入的时候都需要建立一个新的网络，然后将它附着在文档网络上。查询网络将查询词项映射成查询子表达式（利用概率方式或者AND和OR系列操作符的“噪音”版本[①]）和用户需求。

上述操作的结果是产生了一个概率模型，能够概括更简单的布尔和概率模型。实际上，这是一个能够很自然地支持结构化查询操作的统计排序模型的一个特例。基于贝叶斯网络的系统能够支持大规模信息的高效检索，是麻省大学建立的 InQuery 文本检索系统的基础。在 TREC 评测中，该系统也取得了很好的效果，并且一度商业化。另一方面，为了进行参数估计和计算，上述模型仍然使用了许多近似和独立性假设。该模型的后续工作已经很多，但是需要指出的是，该模型实际上建立于现代贝叶斯网络理论的早期阶段，目前的贝叶斯网络理论已经取得了很多进展，也许现在正是利用新理论建立新一代基于贝叶斯网络的信息检索系统的最佳时机。

11.5 参考文献及补充读物

有关概率论的更详细的讨论可以参见很多概率统计的入门书籍，如（Grinstead 和 Snell 1997；Rice 2006；Ross 2006）。有关贝叶斯效用理论（Bayesian utility theory）的一个介绍参见（Ripley 1996）。

信息检索中的概率方法最早于 20 世纪 50 年代起源于英国。概率模型的第一篇重要论述是 Maron 和 Kuhns（1960）。Robertson 和 Jones（1976）给出了 BIM 的主要基础理论，van Rijsbergen（1979）给出了经典 BIM 模型的细节。PRP 的思想来自于很多人的贡献，包括 S. E. Robertson、M. E. Maron 和 W. S. Cooper，最早在 Robertson 和 Jones（1976）使用的术语为 probabilistic ordering

① 这里指对原始的 AND 和 OR 操作进行扩充或者修改得到的表示方法。——译者注

principle，后来的工作中大都用 PRP 这个术语。Fuhr（1992）给出了一个概率 IR 模型的更新的介绍，其中还介绍了一些其它的概率模型，如概率逻辑模型和贝叶斯网络模型等。Crestani 等人（1998）是另外一篇有关概率检索模型的综述。Spärck Jones 等人（2000）是伦敦学院（London school）所进行的概率 IR 实验的一篇权威文献。Robertson （2005）给出了他所在的队伍参加 TREC 评测的总结和回顾，其中包括有关 Okapi BM25 评分函数的详细介绍及其发展结果。Robertson 等人（2004）将 BM25 扩展成一种能够处理不同权重字段的形式。

开源的 Indri 搜索引擎随同 Lemur 工具集（www.lemurproject.org/ ）一起分发，它融合了包括概率推理网络、统计语言模型（参见第 12 章）等在内的多个信息检索模型，特别是能够支持概率推理网络模型的结构化查询操作。

第 12 章

基于语言建模的信息检索模型

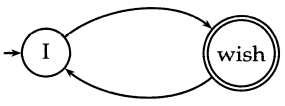
用户要提出好的查询，一个通常的建议是，应该尽量采用那些最有可能出现在相关文档中的词来构成查询。信息检索中的语言建模方法直接对上述思路建模：给定查询，如果某篇文档所对应的文档模型可能生成该查询，那么这篇文档是一个好的匹配文档。此时，查询词项也往往在该文档中频繁出现。虽然语言建模方法也遵循 6.2 节所讨论的文档排序的概率思路，但是它却提供了一个与一般概率模型完全不同的文档排序实现方法。在第 11 章传统的概率模型中，需要对文档 d 与查询 q 的相关概率 $P(R=1|q, d)$进行显式建模，而基本的统计建模方法则首先对每篇文档 d 建模得到文档的概率语言模型 M_d，然后按照模型生成查询的概率 $P(q|M_d)$的高低来对文档进行排序。

本章首先在 12.1 节中介绍语言模型的概念，然后介绍信息检索中常用的基本统计语言建模方法–查询似然模型（12.2 节）。在比较了不同的语言建模方法和其他的信息检索方法之后（12.3 节），本章最后将介绍 IR 中统计语言建模的一些扩展方法（12.4 节）。

12.1 语言模型

121.1 有穷自动机和语言模型

上面我们提到从文档模型生成查询，那么这种说法的具体含义如何？和为人所熟知的形式语言理论（formal language theory）类似，一个传统的语言生成模型（generative model）可以用于识别或生成字符串。比如，图 12-1 中的有穷自动机可以生成例子中的字符串。所有可能的字符串的全集称为该自动机的语言 ①。



I wish
I wish I wish
I wish I wish I wish
I wish I wish I wish I wish I wish I wish
…

CANNOT GENERATE: wish I wish

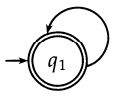
① 有穷自动机的输出可以基于状态或者边，由于前者直接与概率自动机常用的形式化方法相对应，所以这里我们基于的是状态。

图 12-1　一个简单的有穷自动机及其生成语言中的一些字符串。→ 指向的是自动机的初始状态，而双圈节点对应的是（可能的）终止状态

如果每个节点都有一个生成不同词项的概率分布，便可以得到一个语言模型。语言模型的概念本质上也是基于概率的。一个语言模型（language model）是从某词汇表上抽取的字符串到概率的一个映射函数。也就是说，对于字母表∑上的语言模型 M 有

$$\sum_{s\in\Sigma^*} P(s) = 1。 \qquad (12\text{-}1)$$

一类最简单的语言模型与一个概率有穷自动机等价，该自动机仅仅包含一个节点，它也只有一个生成不同词项的概率分布（参见图 12-2），因此有$\sum_{t\in V} P(t)$=1。生成每个词之后，要决定是停止还是继续循环生成另外一个词，因此，模型还需要一个在终止状态停止的概率。该模型会给任意可能的词项序列都赋予一个概率，通过造句，它也能提供一个按照概率分布来生成文本的模型。



$P(\text{STOP}|q_1) = 0.2$

the	0.2
a	0.1
frog	0.01
toad	0.01
said	0.03
likes	0.02
that	0.04
…	…

图 12-2　对应于一元语言模型的单状态有穷自动机。图中只给出了部分的状态发射概率

例 12-1　为了获得一个词序列的概率，对序列中所有词，我们将模型中赋予每个词的概率相乘，同时乘上生成每个词之后继续前进或停止的概率。例如：

$$\begin{aligned} P(\text{frog said that toad likes frog}) &= (0.01\times0.03\times0.04\times0.01\times0.02\times0.01) \\ &\quad \times (0.8\times0.8\times0.8\times0.8\times0.8\times0.8\times0.2) \\ &\approx 0.000\,000\,000\,001\,573。 \end{aligned} \qquad (12\text{-}2)$$

我们看到，一个具体的字符串或文档的概率往往非常小。而且这里我们还只是在第二次生成frog时就停止。公式（12-2)中的第一行是词项发射概率，而第二行是生成每个词后继续前进或停止的概率。按照公式（12-1)，一个有穷自动机要转变成一个良构(well-formed)[①]的语言模型，必须要有一个显式的停止概率。然而，大部分情况下，我们都像大部分作者一样，不考虑STOP概率和(1-STOP)概率。在一个数据集上比较两个模型，可以计算似然比（likelihood ratio），即将其中一个模型的数据生成概率除以另外一个模型的数据生成概率。假定停止概率是固定的，那么考虑它并不会改变似然比的顺序，也

① 这里的良构指的是符合某种语法规则。——译者注

不会改变两个语言模型生成同一字符串的概率大小次序。因此，它不会影响文档的排序[①]。尽管如此，形式上得到的结果将不再是概率，而只是概率的某个比例项。关于这一点请参考习题 12-4。

例 12-2 现在假设有两个语言模型 M_1 和 M_2，模型的部分概率值参见图 12-3。每个模型都可以像 12-1 一样估计一个词项序列的概率。给词项序列赋予更大概率的那个模型更可能是生成该词项序列的模型。假定我们在这次计算中忽略停止概率，于是有

s	frog	said	that	toad	likes	that	dog
M_1	0.01	0.03	0.04	0.01	0.02	0.04	0.005
M_2	0.0002	0.03	0.04	0.0001	0.04	0.04	0.01

（12-3）

$P(s|M_1) = 0.000\ 000\ 000\ 000\ 48$，

$P(s|M_2) = 0.000\ 000\ 000\ 000\ 384$。

我们发现 $P(s \mid M_1) > P(s \mid M_2)$。这里我们是对概率求积，但是通常在概率应用中，实际上往往采用对数求和的计算方法。

模式 M_1		模式 M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
…	…	…	…

图 12-3 两个一元语言模型的部分概率赋值

12.1.2 语言模型的种类

对于词项序列如何求解其生成的概率值？我们往往通过公式（11-1）所示的链式规则将一系列事件的概率分解成多个连续的事件概率之积，每个概率是每个事件基于其历史事件的条件概率。具体计算公式如下：

$$P(t_1t_2t_3t_4) = P(t_1)P(t_2|t_1)P(t_3|t_1t_2)P(t_4|t_1t_2t_3)。 \quad (12\text{-}4)$$

最简单的语言模型形式是：去掉所有条件概率中的条件来独立地估计每个词项的概率。这

① 在 IR 场景中，不同模型中停止概率取固定值是合理的。这是因为查询的长度分布是固定的，与用于产生语言模型的文档无关。

种模型称为一元语言模型（unigram language model）：

$$P_{\text{uni}}(t_1t_2t_3t_4) = P(t_1)P(t_2)P(t_3)P(t_4) \quad (12\text{-}5)$$

还有很多更复杂的语言模型，比如二元语言模型（bigram language model），即计算条件概率时只考虑前一个词项的出现情况：

$$P_{\text{bi}}(t_1t_2t_3t_4) = P(t_1)P(t_2|t_1)P(t_3|t_2)P(t_4|t_3) \quad (12\text{-}6)$$

还有更复杂的基于文法的语言模型，如概率上下文无关文法等。这类模型对于诸如语音识别、拼写校对、机器翻译等需要根据上下文来求词项条件概率的应用非常重要。但是，将统计语言模型用于信息检索领域的大部分工作都只使用了一元模型。IR 并不是一个必须使用复杂语言模型的领域，这是因为它并不像语音识别等应用那样直接依赖于句子的结构。对于判断文本的主题来说，一元模型往往已经足够。另外，我们即将看到，IR 的语言模型往往基于单篇文档来估计，因此训练数据是否足够用于模型估计往往值得怀疑。数据稀疏性造成的损失（参见 13.2 节）会超过更复杂模型所带来的利益。这是偏差–方差折–准则（bias-variance tradeoff，参见 14.6 节）的一个例子，该准则具体内容是：在训练数据有限的情况下，受限越多的模型越倾向于获得更好的性能。除此之外，与更高阶模型相比，一元模型的参数估计和运用也更加高效。尽管如此，IR 中的短语查询和邻近查询的重要性都表明，未来的工作中应该引入更复杂的语言模型，有一些工作已经在这方面进行了尝试（参见 12.5 节）。实际上，这些做法的结果和第 11 章 van Rijsbergen 提出的树型依赖模型很类似。

12.1.3　词的多项式分布

在一元语言模型中，词出现的先后次序无关紧要，因此，这类模型也往往称为词袋模型（参见第 6 章）。尽管不基于前面的事件求条件概率，一元语言模型仍然会给有序的词项序列赋予概率。当然，这些词项按照其他次序出现的概率也等于这个概率。因此，实际上这相当于词项存在一个多项式分布。只要我们坚持采用一元模型，那么语言模型的名称和动机就可以看成是沿袭历史的结果而不是必须的。此时，我们也可以将上述模型称为多项式模型。基于这种观点来看的话，上面的公式中并没有给出词袋的多项式概率，这是因为它们没有对这些词项的所有可能出现序列的概率求和。在多项式模型中，上述求和通过一个多项式系数来实现(即下式中右边的因子 $\frac{L_d!}{tf_{t_1,d}!tf_{t_2,d}!\ldots tf_{t_M,d}!}$)：

$$P(d) = \frac{L_d!}{tf_{t_1,d}!tf_{t_2,d}!\cdots tf_{t_M,d}!}P(t_1)^{tf_{t_1,d}}P(t_2)^{tf_{t_2,d}}\cdots P(t_M)^{tf_{t_M,d}}。 \quad (12\text{-}7)$$

这里，$L_d = \sum_{1\le i\le M} tf_{t_i,d}$ 是文档的长度（即词条的总个数），M 是词项词典的大小。上式对词典中的所有词项求积，而不是对文档中的每个位置求积。然而，就像前面的 STOP 概率一样，在实际中往往也可以省去多项式系数的计算，因为，对于某个特定的词项集来说，它是一个常数，对生成同一词袋的两个不同模型的似然比并没有任何影响。有关多项式模型的介绍也可以参见

13.2 节。

设计语言模型的一个基本问题就是,我们不知道到底应该用什么来估计语言模型 M_d。但是，通常我们都有一个能够代表模型的文本样本。在语言模型最初的主要应用当中，上述问题很有道理。例如，在语音识别中，我们有一些训练样本（即观察数据）。但是我们必须预料到，用户在将来会使用样本中没有出现过的词或者序列。因此，必须对模型在训练数据之外进行推广，使之能够处理未知的词和序列。在 IR 领域，文档往往大小有限且比较固定，因此上述语音识别领域中的解释并不十分明显。IR 中采取的策略如下：我们假设文档 d 仅仅是来自满足某个词分布模型的抽样样本，该词分布代表了一个细粒度的话题；然后，基于这个样本估计出一个语言模型并利用该模型计算观察中的词序列的概率；最后，我们按照不同模型生成查询的概率大小来对文档进行排序。

? 习题 **12-1 [*]** 如果在计算中考虑 STOP 概率，那么语言中所有长度为 1（以词为单位）的字符串的估计概率之和是多少？假定生成了一个词，然后决定是否停止（也就是说空串不属于该语言）。

习题 **12-2 [*]** 如果在计算中忽略 STOP 概率，那么语言中所有长度为 1（以词为单位）的字符串的估计概率之和又是多少？

习题 **12-3 [*]** 例 12-2 中按照 M_1 和 M_2 算出的文档的似然比是多少？

习题 **12-4 [*]** 例 12-2 中没有给出显式的 STOP 概率，假定每个模型的 STOP 概率都是 0.1，请问这会不会改变两个模型下文档的似然比？

习题 **12-5 [**]** 在拼写校对系统中，语言模型应该怎么用？特别地，考虑上下文敏感的拼写校对情况，对词的错误使用进行修正，如句子 Are you their?中的 their。可以参考 3.5 节来获得该主题的更多相关文献。

12.2 查询似然模型

12.2.1 IR 中的查询似然模型

语言建模是 IR 中一种非常通用的形式化方法，它在实现时有多个变形。IR 中最早使用也是最基本的语言模型是查询似然模型（query likelihood model）。在这个模型中，我们对文档集中的每篇文档 d 构建其对应的语言模型 M_d。我们的目标是将文档按照其与查询相关的似然 $P(d|q)$ 排序。利用 11.1 节介绍的贝叶斯公式，有

$$P(d|q) = P(q|d)P(d)/P(q)。$$

$P(q)$对所有的文档都一样，因此可以被忽略。文档的先验概率 $P(d)$往往可以视为均匀分布，因此也可以被省略。当然，实际中也可以使用真实的先验概率，比如可以包括权威度、长度、类型、新鲜度和以前阅读过该文档的用户数目这些因素。这里我们暂时不考虑 $P(d)$，这样的话，最后我们会按照 $P(q|d)$进行排序，它是在文档 d 对应的语言模型下生成 q 的概率。因此，IR 中

的语言建模方法实际上是在对查询的生成过程进行建模：首先每篇文档对应一个文档模型，然后计算查询被视为每个文档模型的随机抽样样本的概率，最后根据这些概率对文档排序。

最普遍的计算 $P(d|q)$的方法是使用多项式一元语言模型，该模型等价于多项式朴素贝叶斯模型，其中这里的文档相当于后者中的类别，每篇文档在估计中都是一门独立的“ 语言” 。在这个模型下，有

$$P(q \mid M_d) = K_q \prod_{t \in V} P(t \mid M_d)^{tf_{t,d}} 。 \quad (12\text{-}8)$$

其中，$K_q = L_d !/ (tf_{t_1,d} ! tf_{t_2,d} ! \cdots tf_{t_M,d} !)$ 是查询 q 的多项式系数，对于某个特定的查询，它是一个常数，因此可以被忽略。

在基于语言模型（简记为 LM）的检索中，可以将查询的生成看成一个随机过程。具体的方法是：

(1) 对每篇文档推导出其 LM；

(2) 估计查询在每个文档 d_i 的 LM 下的生成概率 $P(q \mid M_{d_i})$ ；

(3) 按照上述概率对文档进行排序。

上述模型的直观意义是，用户脑子里有一篇原型文档，然后按照该文档中的词语用法来生成查询。通常，用户对感兴趣的文档中可能出现的词项有一些合理的想法，然后他们会选择那些能够区分其他文档的查询项构成查询 ①。在其他检索模型中，文档集的统计信息往往启发式地被使用，但是在语言模型中，它是不可分割的、很自然的一部分。

12.2.2　查询生成概率的估计

本节主要介绍如何估计概率 $P(q \mid M_{d_i})$ ，在 MLE（maximum likelihood estimation，最大似然估计）及一元语言模型假设的情况下，给定文档 d 的 LM M_d 的情况下生成查询 q 的概率为：

$$\hat{P}(q \mid M_d) = \prod_{t \in q} \hat{P}_{\text{mle}}(t \mid M_d) = \prod_{t \in q} \frac{tf_{t,d}}{L_d} 。 \quad (12\text{-}9)$$

其中，M_d 是文档 d 的 LM，$tf_{t,d}$ 是词项 t 在文档 d 中出现的原始频率，L_d 是文档 d 中的词条数目。也就是说，我们只是将词在文档中出现的次数除以文档中出现的所有的词的总数，这和 11.3.2 节中看到的使用 MLE 计算方法是一样的，但是这里使用的是基于词的数目的多项式分布。

LM 使用中的一个经典问题是参数估计（公式中 P 上面的^符号强调模型是估计出来的）的稀疏性问题，即词项在文档中的出现非常稀疏。特别是有些词根本不会在文档中出现，但是这些词很有可能构成信息需求并可能在查询中使用。如果将不出现在文档中的词项 t 的概率 $\hat{P}(t \mid M_d)$ 估计成 0，那么就会得到一个非常严格的“ 与” 语义：一篇文档只会给所有查询项都出现在文档中的查询赋予一个非零的概率。在 LM 的其他应用中，零概率显然是一个问题。比如在语音识别中，由于很多词在训练数据上存在稀疏性，所以在预测下一个词时可能会遇到零概率问题。在 IR 领域，零概率是不是一个重要问题看上去不是那么明显。这可以想成一个人机

① 当然，其他情况下，用户并不会这么做。语言建模体系下的解决该问题的方法参见 12.4 节的翻译 LM。

接口问题：基于向量空间模型的系统采用的是非常宽松的匹配方法，但是近年以来的 Web 搜索引擎更倾向于采用上述的“与”语义。不管上面采用何种方法，LM 的估计当中还有一个更一般的问题就是，对文档中出现的词的估计也存在很大问题，特别是那些在文档中只出现一次的词往往会被过度估计，这是因为它们仅有的一次出现在一定程度上出于偶然性。解决这种估计不当的方法是平滑（参见 11.3.2 节）。但是，随着人们对 LM 的深入理解，有一点越来越明显：模型中平滑的角色不仅仅是避免零概率，平滑实际上实现了词项权重计算的主要部分（参考习题 12-8）。未平滑的模型不仅仅存在“与”语义问题，而且由于缺乏权重计算的多个部分而导致效果很差。

因此，这里我们需要对文档LM的概率进行平滑，即对非零的概率结果进行折扣，并对未出现的词的概率赋予一定的值。对概率分布进行平滑方法的有很多。11.3.2 节中我们已经讨论过在事件的观察数目上加某个数字（1、1/2 或一个很小的α）然后对概率分布重新进行归一化的平滑方法 ①。本节中，我们将介绍一些其他的平滑方法，其中包括将观察数目和一个更一般的参照概率分布相结合的方法。在一般的参照概率分布中，文档中未出现的查询项都可能在查询中出现，它的概率在某种程度上接近但不可能超过在整个文档集中偶然出现的概率。也就是说，如果$tf_{t,d}=0$，那么有

$$\hat{P}(t \mid M_d) \leqslant \mathrm{cf}_t / T \text{。}$$

其中，cf_t是 t 在整个文档集的出现次数，T 是所有文档集中词条的个数。一个实际效果较好的简单方法是，将基于文档的多项式分布和基于全部文档集估计出的多项式分布相混合，即

$$\hat{P}(t \mid d) = \lambda \hat{P}_{\mathrm{mle}}(t \mid M_d) + (1-\lambda)\hat{P}_{\mathrm{mle}}(t \mid M_c) \text{。} \qquad (12\text{-}10)$$

其中，$0 < \lambda < 1$，M_c是基于全部文档集构造的LM。上述公式混合了来自单个文档的概率词在整个文档集的出现频率，这类模型称为线性插值LM（linear interpolation LM）②。该模型中如何设置正确的λ是获得良好性能的关键。

另外一种方法是，将从全部文档集中获得的语言模型（不是像 11.3.2 节讨论的那样看成均匀分布）看成贝叶斯更新过程（Bayesian updating process）的一个先验分布。于是有

$$\hat{P}(t \mid d) = \frac{tf_{t,d} + \alpha \hat{P}(t \mid M_c)}{L_d + \alpha} \text{。} \qquad (12\text{-}11)$$

在 IR 实验中，上面的两种平滑方法都体现出很好的效果，因此，在剩下部分我们主要使用线性插值的平滑方法。尽管这两种方法细节有所不同，但是在概念上是很相近的。两种方法中，对文档中出现的词，它的概率估计都是将一个折损的 MLE 估计和它在整个文档集中的流行度因子相结合而得到的。而对于文档中没有出现的词，则直接采用词在整个文档集中的流行度因子

① 在概率论中，归一化指的是对事件空间上的每个事件数目求和，并除以所有事件的总数，从而保证最后得到的是一个概率分布，即各概率之和等于 1。这个归一化的概念和第 2 章提到的词项归一化及第 6 章提到的采用 $L2$ 范式进行的长度归一化概念是不一样的。

② 线性插值平滑也称为 Jelinek-Mercer 平滑。

来计算。

IR中的LM平滑方法不只是或者说主要不是用来避免估计问题。将LM应用于IR领域的早期，上述结论还不是很清晰。但是，现在我们已经认识到，平滑对于模型的良好性能具有实质性的影响，这其中的原因可以参考习题 12-8。上述两种方法的平滑程度主要受参数λ及α的控制：较小的λ及较大的α意味着更平滑。这些参数可以通过一维搜索（line search）[①]的方法进行调优来获得最佳性能（或者在线性插值模型中，可以通过其他方法进行参数估计，比如 16.5 节采用的EM算法）。参数的值也不必是个常数，一种做法就是将参数定义为查询长度的一个函数，这是因为对于短查询来说，轻度平滑（类似于“ 与” 搜索）更为合适，而长查询则需要加重平滑的权重。

总结一下，在我们讨论的基本语言建模的 IR 模型下，查询 q 的检索排序函数定义如下：

$$P(d \mid q) \propto P(d)\prod_{t \in q}[(1-\lambda)P(t \mid M_c)+\lambda P(t \mid M_d)]。 \qquad (12\text{-}12)$$

上述公式给出了用户心目中的文档就是 d 的概率。

例 12-3　假定某文档集中包含如下两篇文档：

- d_1: Xyzzy reports a profit but revenue is down
- d_2: Quorus narrows quarter loss but revenue decreases further

模型采用的是利用 MLE 估计的两个一元模型的混合，其中一个来自文档而另一个来自文档集，混合参数 $\lambda = 1/2$。假定查询为 revenue down，则有

$$\begin{aligned} P(q|d_1) &= [(1/8 + 2/16)/2] \times [(1/8 + 1/16)/2] \\ &= 1/8 \times 3/32 = 3/256， \\ P(q|d_2) &= [(1/8 + 2/16)/2] \times [(0/8 + 1/16)/2] \\ &= 1/8 \times 1/32 = 1/256， \end{aligned} \qquad (12\text{-}13)$$

因此，d_1 的排名高于 d_2。

12.2.3　Ponte 和 Croft 进行的实验

Ponte 和 Croft（1998）首次将语言建模的方法引入到IR领域并进行了实验。他们的基本思路和我们目前为止所介绍的模型是一致的。但是，在前面介绍的模型中，我们采用了两个多项式分布混合的方法，这更像（Miller 等人 1999；Hiemstra 2000）中所提到的模型，而不是Ronte和Croft提出的多元贝努利模型。在大部分后来的有关语言模型IR的工作中都采用了多项式模型[②]这种方法，在文本分类领域也存在这类模型表现更优越的证据。Ponte和Croft强调，采用语

① 一维搜索也称为线性搜索，就是指单变量函数的最优化。它是多变量函数最优化的基础。——译者注

② 假定词典的大小是 L，某篇文档 d 的长度为 $|d|$。则在多希式模型下，文档 d 的生成可以看成是一个不规则 L 面体的过程，每个面对应词典中的一个词。连续掷 $|d|$ 次的结果得到文档 d。而多元贝努利模型相当于独立地掷 L 个不规则硬币，其中每个硬币对应词典中的一个词。每个硬币抛一次，然后将所有朝上的硬币对应的词集中起

言建模方法得到的词项权重计算方法优于传统的tf-idf方法。这里给出了他们的部分实验结果，实验中使用了TREC的第 202~250 号主题，文档集由TREC disk 2 和disk 3 组成。所有的查询都采用自然语言来描述，且长度和一个句子长度差不多。利用语言建模的方法获得的结果明显好于tf-idf方法。并且，实际上在后来的工作中利用LM所带来的好处比现在看到的更大。

	precision			
Rec.	tf-idf	LM	%chg	
0.0	0.7439	0.7590	+2.0	
0.1	0.4521	0.4910	+8.6	
0.2	0.3514	0.4045	+15.1	*
0.3	0.2761	0.3342	+21.0	*
0.4	0.2093	0.2572	+22.9	*
0.5	0.1558	0.2061	+32.3	*
0.6	0.1024	0.1405	+37.1	*
0.7	0.0451	0.0760	+68.7	*
0.8	0.0160	0.0432	+169.6	*
0.9	0.0033	0.0063	+89.3	
1.0	0.0028	0.0050	+76.9	
Ave	0.1868	0.2233	+19.55	*

图 12-4 tf-idf及LM词项权重计算方法(Ponte 和 Croft 1998)的比较结果。tf-idf采用了INQUERY检索中的版本，它考虑了tf的长度归一化。表中采用了 11 点平均正确率来评价，利用Wilcoxon符号秩检验 ①得到的具有统计显著性的结果用*号标识。大部分情况下，LM比tf-idf效果更好，特别是在召回率水平越高的情况下，结果的优越性越显著

? 习题 **12-6 [*]** 考虑从如下训练文本中构造 LM：

the martian has landed on the latin pop sensation ricky martin

请问：

a. 在采用 MLE 估计的一元概率模型中，P(the)和 P(martian)分别是多少？

b. 在采用 MLE 估计的一元概率模型中，P(sensation|pop)和 P(pop|the)的概率是多少？

习题 **12-7 [**]** 假定某文档集由如下 4 篇文档组成：

文档ID	文档文本
1	click go the shears boys chick click click
2	click click
3	metal here
4	metal shears click here

来便得到文档 d。很显然，后者会考虑在文档 d 中不出现的词但不考虑词频。而前者只考虑出现在 d 中的词，但却考虑词频。——译者注

① Wilcoxon 符号秩检验（signed rank test）是非参数检验（nonparametric test）的一种，简单地说，它是检验差值的中位数是否为 0 的一种检验。具体请参考非参数检验相关的书籍。——译者注

为该文档集建立一个查询似然模型。假定采用文档语言模型和文档集语言模型的混合模型，权重均为 0.5。采用 MLE 来估计两个一元模型。计算在查询 `click`、`shears` 以及 `click shears` 下每篇文档模型对应的概率，并利用这些概率来对返回的文档排序。将这些概率填在下表中。

query	doc1	doc2	doc3	doc4
click shears click shears				

对于查询 `click shears` 来说，最后得到的文档次序如何？

习题 **12-8 [**]**　以习题 12-7 的计算结果为出发点，当然需要的时候也可以举例说明。基于公式 (12-10)对以下每个量的处理给出一句话说明，包括该量是否包含在模型中以及它采用的是原始值还是尺度变换后的值：

a. 文档中的词项频率；

b. 文档集中的词项频率；

c. 词项的文档频率；

d. 词项的长度归一化。

习题 **12-9 [**]**　在公式（12-12）所示的采用混合模型的查询似然模型中，某个词项的概率计算既考虑了词项在文档中的词频，又考虑了词在整个文档集中的频率。这样做当然能够使查询中的每个词项在每个文档模型下都有一个非零值。然而更重要的是，它对于词项权重的计算（参考第 6 章中的词项权重计算）而言，虽然表面上不明显，但是实际上体现出很重要的影响效果，请解释其中的原因。特别地，请给出一个具体的例子来说明其中的词项权重计算方法。

12.3　语言建模的方法与其他检索方法的比较

语言建模的方法给出了一个全新的看待文本检索的视角，同时它也把文本检索和语音及语言处理的一些近期的工作联系起来。正如 Ponte 和 Croft (1998)强调的那样，IR 中的 LM 方法提供了另外一种计算查询与文档匹配得分的方式，其带来的希望在于，概率语言建模能够优化已有的权重计算方法从而升高检索的性能。语言建模 IR 中最主要的问题是文档模型的估计，包括如何选择有效的平滑方法。基于语言建模的 IR 模型已经获得了非常好的检索结果，与其他概率模型的方法相比（如第 11 章提到的 BIM 模型），最主要的区别似乎是语言模型的方法不再对相关性进行显式建模（而在 BIM 模型中，相关性作为一个最主要的变量来建模）。但是这不是考虑事情的正确方法，关于这一点在 12.5 节给出的一些论文中将有进一步的讨论。在 LM 中，要求文档和信息需求的表达式都是同一类型的对象，并且通过引入语音及自然语言处理中的统计语言建模工具和方法来计算匹配度。最后得到的模型在数学上很精确、概念上简洁、计算上可行、直观上也非常吸引人。这看上去与第 10 章介绍的 XML 检索的情形非常相似，在第 10 章中，假定查询和文档属于同一类型的做法同样是最成功的一类做法。

另一方面，同其他 IR 模型一样，对基于语言建模的 IR 模型人们也有很多不同意见。文档

和信息需求表示之间的对象同类假设显然不符合实际情况。当前的 LM 方法采用了非常简单的语言模型，比如常常采用一元模型。由于没有定义一个显式的相关性概念，相关反馈技术和用户偏好信息很难集成到模型中。另外，将一元模型扩展到更复杂的、能够容纳短语或段落匹配或布尔操作符的模型看上去也很有必要。一些后续的有关 LM 的工作已经对上述关注点进行了研究，包括将相关反馈放回到模型中，并且允许查询和文档的语言不匹配。

语言建模的 IR 模型与传统的 tf-idf 模型具有很显著的关联。在 tf-idf 模型中，词项频率会直接进行表示，最近的一些工作也意识到文档长度归一化的重要性。将文档生成概率和文档集生成概率二者相混合的效果有点像 idf，那些在整个文档集中出现较少而在某些文档中出现较多的词项将对文档的最后排序起到较大作用。在很多具体的模型实现中，所有模型基本上都将不同词项看成是相互独立的。另一方面，统计建模 IR 模型和 tf-idf 模型相比，在直觉上一个来自概率论而另一个来自几何学，在数学模型上一个更具理论性而另一个更基于启发式知识，在诸如词项频率和文档长度等统计因素的使用细节上也有区别。如果我们主要关心性能数字的话，那么近年来的工作表明，LM 方法的检索效果非常好，超过了 tf-idf 和 BM25 方法。尽管如此，还没有足够的证据来证明 LM 方法的效果能够大大超过一个经过精心调节参数后的传统向量空间模型，因此也无法证明改变现有的实现系统是很有必要的。

12.4 扩展的 LM 方法

本节简单介绍一些对基本的 LM 方法进行扩展的工作。

在 IR 环境下，还有其他使用 LM 的思路，其中很多都在后续的工作中进行了尝试。比如我们可以不考虑文档语言模型 M_d 生成查询的概率，而考虑查询语言模型 M_q 生成文档的概率。一方面，由于查询文本提供的用于估计其 LM 的信息太少，因此沿着上述思路建立一个文档似然模型的方法并不太吸引人，并且由于信息缺乏导致模型估计的效果很差，所以必须要利用其他 LM 进行平滑。另一方面，很容易知道如何将相关反馈技术引入到该模型，并且可以像以往一样利用相关文档抽取的词项对原始查询进行扩展，因此可以对语言模型 M_q 进行更新（Zhai 和 Lafferty 2001a）。实际上，通过合适的建模选项，这种方法能够推出第 11 章的 BIM 模型。Lavrenko 和 Croft（2001）的相关性模型实际上是文档似然模型的一个例子，它将伪相关反馈结合到语言模型中，获得了非常有说服力的经验性结果。

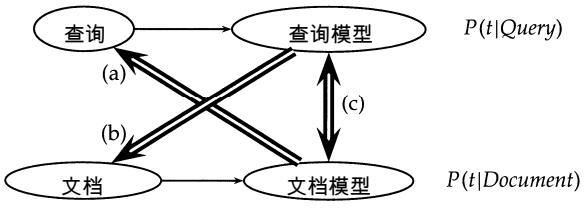


图 12-5 IR 中使用统计语言建模的 3 种方式：(a) 查询似然 (b) 文档似然 (c) 模型比较

另一种做法不是从单个方向来直接生成对象，而是从查询和文档两个方向同时生成两个语言模型，然后比较这两个模型之间的差别。Lafferty 和 Zhai（2001）同时将考虑问题的 3 个不同方法展示出来（参见图 12-5），并提出了一个通用的风险最小化的文档检索方法。比如，给定查询q，可以通过计算查询模型和文档模型的KL距离（Kullback-Leibler divergence）[①]来对返回文档d的风险进行建模：

$$R(d;q) = LK(M_d \,||\, M_q) = \sum_{t \in V} P(t \mid M_q) \log \frac{P(t \mid M_q)}{P(t \mid M_d)} 。 \qquad (12\text{-}14)$$

KL 距离是源自信息论的一个非对称距离度量方法，主要度量的是概率分布 M_q 对 M_q 建模的无效程度（参见 Cover 和 Thomas 1991 及 Manning 和 Schütze 1999）。Lafferty 和 Zhai（2001）给出的结果表明基于模型对比的方法比查询似然和文档似然的方法都好。使用 KL 距离作为排序函数的一个缺点是最后的得分在查询之间没有可比性。这对于 ad hoc 检索来说没有关系，但是对于一些其他应用（如话题跟踪）却影响很大。Kraaij 和 Spitters（2003）给出了另外一种方法，它将相似度当成一个归一化的对数似然比（与交叉熵的差值等价）来进行计算。

基本的 LM 方法没有考虑表达方式不同的问题，比如一义多词或者查询语言和文档语言间的偏差问题。Berger 和 Lafferty（1999）引入了翻译模型来弥补查询和文档之间的语言差距。翻译模型可以通过翻译的方法，让不在文档中的词项转变为其近义词项后出现在查询中。这种方法也为跨语言 IR 提供了基础。假定翻译模型可以通过一个词项间的条件概率分布 T(.|.)来表示，则基于翻译的查询生成模型可以定义为：

$$P(q \mid M_d) = \prod_{t \in q} \sum_{v \in V} P(v \mid M_d) T(t \mid v) 。 \qquad (12\text{-}15)$$

其中，$P(v|M_d)$是基本的文档语言模型，$T(t|v)$表示的是翻译概率。很明显该模型的计算开销很大，并且需要建立一个翻译模型。翻译模型往往基于单独的资源来构建（比如传统的同义词词典或者双语词典或某个统计机器翻译系统的翻译词典）。当然，如果存在一些文本片段很自然地解释或者概括了其他文本，那么也可以基于文档集来构建翻译模型。一些可能的例子包括文档和它们的标题或摘要，或者在超文本环境下的文档及指向该文档的锚文本。

对 LM 方法进行扩展仍然是研究上的一个热点。一般来说，翻译模型、相关反馈模型及基于模型对比的方法的表现都优于基本的查询似然模型。

12.5　参考文献及补充读物

如果要了解更多的有关概率 LM 的概念及相关的平滑技术，可以参考 Manning and Schütze 1999 的第 6 章及 Jurafsky 和 Martin 2008 的第 4 章。

一些重要的有关 IR 中统计语言建模方法的早期论文包括：Ponte 和 Croft 1998、Hiemstra

①也称为交叉熵。——译者注

1998、 Berger 和 Lafferty 1999 及 Miller 等人 1999。其他相关的论文可以在随后几年的 SIGIR 论文集中找到。Croft 和 Lafferty 2003 包含了从一个有关语言模型的研讨会上收集的一系列相关论文，Hiemstra 和 Kraaij 2005 对 TREC 任务中使用 LM 的一系列重要工作进行了综述。Zhai 和 Lafferty 2001b 阐明了语言建模 IR 模型中平滑方法的角色，并给出了不同平滑方法的经验性比较结果。Zaragoza 等人 2003 倡导采用全贝叶斯预测分布而不是 MAP 点估计，他们发现，尽管这种做法好于贝叶斯平滑方法，但是却比线性插值方法差。Zhai 和 Lafferty 2002 给出了一个两步的平滑模型，首先进行贝叶斯平滑然后进行线性插值，文章指出两步的平滑方法性能上优于单个平滑方法，而且表现更稳定。LM 的方法的一个优点是为融入各种先验知识提供了一个方便的、理论上的方法。Kraaij 等人 2002 的工作表明，将链接信息作为先验知识融入到模型中可以提高 Web 入口网页检索的性能。Liu 和 Croft 2004 的工作表明，相似文档簇可以对文档模型的平滑带来益处，这个工作还会在第 16 章讨论聚类时简要介绍。Tao 等人 2006 指出，基于文档相似度的平滑方法还能带来更大的好处。

Hiemstra 和 Kraaij 2005 给出的 TREC 结果表明，LM 超过了 BM25 方法。一些近期的工作不再限于一元模型，而是涉及高阶模型，在实现时高阶模型往往通过低阶模型来进行平滑。尽管到目前为止的提高幅度有限，但是高阶模型的方法还是取得了一定的成果（参见 Gao 等人 2004 及 Cao 等人 2005）。Spärck Jones 2004 对语言建模方法的基本原理提出了一个批评的观点，但是 Lafferty 和 Zhai 2003 指出，不论是本章提出的 LM 方法还是第 11 章给出的经典概率检索方法，都可以给出一个统一的概率语义上的解释。Lemur 工具集（www.lemurproject.org/）为考察基于 LM 的 IR 方法提供了一个灵活的开源框架。

第13章 文本分类及朴素贝叶斯方法

迄今为止，本书主要讨论的是 ad hoc 检索过程，其中用户的信息需求是瞬时的、短暂的，而用户则通过向搜索引擎提交一个或者多个查询的方式来处理需求。但是，很多用户拥有固定的信息需求。比如，有人可能需要跟踪多核计算机芯片（multicore computer chips）的相关进展，一种做法是每天一早从最新的新闻报道中查询 multicore AND computer AND chip。在本章以及接下来的两章当中，我们将主要回答这样一个问题：如何才能自动完成上述例子中每天的重复性工作？ 为此，许多系统都支持固定查询（standing query），它和其他类型的查询的唯一不同之处在于：它会在一个不断增量式更新的文档集上得到定期的处理。

如果用户的固定查询仅仅是 multicore AND computer AND chip，那么用户可能会丢失很多新的使用了其他术语的相关文档，比如 multicore processors。为了达到较高的召回率，固定查询本身也要随时间不断改进从而逐渐变得越来越复杂。在刚才的例子当中，如果使用一个带有词干还原功能的布尔搜索引擎，那么最后得到的查询很可能类似于（multicore OR multi-core）AND（chip OR processor OR microprocessor）。

不失一般性，并且为了获得固定查询所属问题的空间范围，下面我们引入分类（classification）这个一般性的概念。给定一系列类别，分类是指将给定对象归入一个或者多个类别的过程。上例中，基于固定查询，可以将新的新闻文章分类两类：和 multicore computer chips 相关的文档及和 multicore computer chips 不相关的文档。 基于固定查询进行分类也称为信息路由（routing）或者过滤（filtering），这个话题将在 15.3.1 节中进一步讨论。

一个类别不一定就像前面提到的固定查询 multicore computer chips 那样关注范围那么窄。通常来说，类别往往是一个更一般的主题领域，比如 China 或者 coffee。这种更一般意义上的类别往往也称为主题（topic），面向文本的分类任务则称为文本分类（text classification， text categorization, topic classification）或主题发现（topic spotting）。图 13-1 给出了一个 China 类的例子。尽管固定查询和主题在具体程度上有所不同，但是解决信息路由、过滤和文本分类的方法在本质上是一致的。因此，本章及以后章节都会将信息路由和过滤归入到文本分类任务。

分类是一个极其一般性的概念，在信息检索及其他领域都有很多应用。比如，在计算机视觉领域，分类器可以将图像分成风景类、肖像类及其它类。这里，我们主要关注来自信息检索的分类的例子。

- 第 2 章介绍的索引构建过程中几个必要的预处理问题：文档编码（ASCII、Unicode UTF-8 等）的识别、分词（两个字母之间的空格是否代表词之间的间隔？参见 2.2.1 节）、真实

的大小写处理（truecasing，参见 2.2.3 节）及文档语言类型的判定（参见 2.5 节）。

- 垃圾网页的自动判定（这些垃圾网页不会被搜索引擎索引）。
- 色情淫秽内容的自动判定（只有当用户关闭了类似 SafeSearch 的选项之后这些内容才会出现在搜索引擎的结果中）。

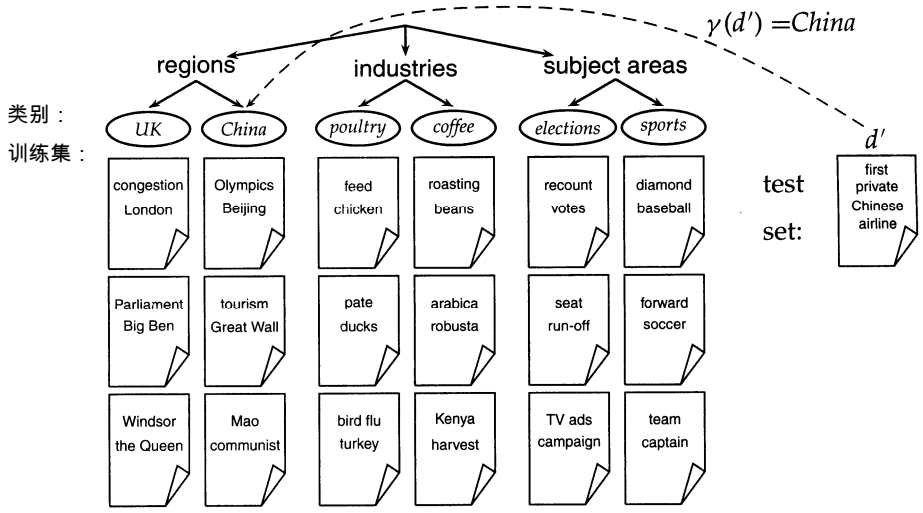


图 13-1　文本分类中的类别、训练集及测试集

- 情感发现（Sentiment detection）或者自动将电影或产品评论按照褒贬性分类。一个褒贬性分类应用的例子是，用户在购买一款相机前，会先搜索相关的贬义评论以确保相机没有自己不想要的功能并且不存在质量问题。
- 个人的邮件组织和整理。用户可能有诸如 talk announcements、electronic bills、email from family and friends 之类的邮件夹，他可能希望能够将到来的邮件自动放到合适的邮件夹下。这样的话，在这些邮件夹下寻找信息就比在整个大收件箱中查找要容易得多。这类应用中，最普遍的使用场景是建立一个垃圾邮件夹来存放所有可能的垃圾邮件。
- 面向主题的搜索或者说垂直搜索。垂直搜索引擎（vertical search engine）将搜索限定于某个特定的主题或领域。比如，在一个有关 China 主题的垂直搜索引擎中输入查询 computer science，一个通用搜索引擎则将以比搜索查询 computer science China 更高的正确率和召回率返回中国的计算机科学系的列表。造成这种结果的原因在于，该垂直搜索引擎并不包含那些另外的含义的 china（比如，表示瓷器意义的 china）所在的网页，然而它会包含那些即使不明显包含 China 但却相关的网页。
- 最后，ad hoc IR 中的排序函数也可以基于文档分类器来构建，这点将在 15.4 节中讨论。

上面这个列表表明了 IR 中分类在各方面的重要性。实际上，当今大部分检索系统中的多个部件中都使用了某种形式的分类器。本书中主要以文本分类为例来介绍分类任务。

分类并不一定要使用计算机。通常来说，很多分类任务都是通过人工来完成的。图书馆管

理员将美国国会图书馆（Library of Congress）分类号分配给每本图书。但是，人工分类的方法一旦要规模化则开销会很大。对于上面提到的 multicore computer chips 的例子，可以采用另外一种思路，即直接利用固定查询将其想象成某种规则来进行分类，这些规则一般由人来编写。上述例子中，一个可能的规则是(multicore OR multi-core) AND (chip OR processor OR microprocessor)。有时规则即等价于布尔表达式。

这些规则通过关键词的某种组合来代表一个类别。人工编写的规则具有很好的扩展性，但是创建和长时间维护这些规则需要很高的人力成本。高水平的专业人士（比如，一个善于撰写正则表达式规则的领域专家）能够建立很好的规则集，这些规则集能够与我们马上要介绍的自动分类器的精度相当甚至超过后者，但是，找到具有这种专长的人才非常困难。

除了手工分类和人工编写规则之外，还存在第 3 种文本分类的方法，即基于机器学习的方法。在接下来的章节中，我们主要关注这种方法。在机器学习中，规则集（更通用的说法是分类决策准则）是从训练数据中自动学习得到的。当学习方法基于统计时，这种方法也称为统计文本分类（statistical text classification）。在统计文本分类中，对于每个类别我们需要一些好的文档样例（或者称为训练文档）。由于需要人来标注训练文档，所以对人工分类的需求仍然存在。这里的标注（labeling）指的是对每篇文档赋予类别标签的过程。但是可以证明，标注工作会比撰写规则更容易。几乎每个人都可以浏览一篇文档然后确定它是否和 China 相关。有时，人们的标注过程还会隐含在现有的工作流程中（即此时并不需要人们进行特意的、显式的标注）。比如，针对某个固定查询，人们可能每天早晨都会浏览系统返回的新闻文章，然后对这些文章进行隐式相关性反馈（参考第 9 章），比如将相关的文章移到一个诸如 multicore-processors 的指定文件夹下。

本章首先将给出文本分类问题的一般性介绍，其中包括它的一个形式化定义（见 13.1 节），然后介绍一种具体的简单高效的分类方法–朴素贝叶斯方法（13.2 节~13.4 节）。我们考察的所有分类方法都将文档表示在高维空间。为了提高这些方法的效率，通常要考虑进行空间降维。因此，一种称为特征选择（feature selection）的技术在文本分类中被普遍使用，有关内容将在 13.5 节介绍。13.6 节主要介绍文本分类的评价。接下来的第 14 章和第 15 章，我们将介绍另外两种分类方法：基于向量空间的方法和基于支持向量机的方法。

13.1 文本分类问题

文本分类中，给定文档 $d \in$ □ 和一个固定的类别集合∐= $\{c_1, c_2, \ldots, c_J\}$，其中□表示文档空间（document space），类别（class）也通常称为类（category）或类标签（label）。通常来说，文档空间□是某种类型的高维空间，而类别通常由人们根据自己的应用需求来定义，比如上面例子中的 China 类及有关 multicore computer chips 的文档类。给定已经标注好类别的训练集（training set）$\lambda = \langle d, c\rangle$，其中$\langle d, c\rangle \in$□ ×∐，比如，

<Beijing joins the World Trade Organization, *China*> = $\langle d, c\rangle$

表示的是单句文档 Beijing joins the World Trade Organization 被标记为 China 类。

利用某种学习方法（learning method）或学习算法（learning algorithm），我们希望学到某个分类函数（classification function）γ，它可以将文档映射到类别：

$$\gamma : \square \rightarrow \amalg \qquad (13\text{-}1)$$

由于监督者（定义类别体系并标注训练集的人）在学习过程中起到类似教师的作用，所以这种类型的学习称为有监督的学习（supervised learning）。我们将有监督学习方法记为 Γ，于是有 $\Gamma(\lambda) = \gamma$。Γ 以训练文档集 λ 为输入，返回学习到的分类函数 γ。

许多学习方法 Γ 的名字也常常被用于分类器 γ。在谈到“朴素贝叶斯（Naive Bayes，简称 NB）很鲁棒”时，我们指的是朴素贝叶斯学习方法 Γ，也就是说这种方法能用应用于很多不同的学习问题并且不太可能产生效果极差的分类器。但是当我们提到“朴素贝叶斯的错误率是 20%”的时候，指的是一个具体的 NB 分类器 γ（该分类器由 NB 学习方法产生）在某个应用中的错误率是 20%。

图 13-1 给出了一个基于 Reuters-RCV1 语料（参见 4.2 节的介绍）的文本分类的例子，其中给出了 6 个类（UK、China、sports 等等），且每个类别都有 3 篇训练文档。为了便于记忆，我们对每篇文档的内容给出了一些代表词。训练集为每个类别提供了一些代表性的文档，因此可以基于这些文档学到分类函数 γ。一旦学到分类函数 γ 之后，就可以将它应用到测试集（test set 或 test data）。比如，可以对一篇类别未知的新文档 first private Chinese airline 进行分类。图 13-1 中，分类函数将新文档的类别分到 China 类，即 $\gamma(d)$ = China，这个分类结果是正确的。

文本分类中的类别体系往往有一些很有趣的结构，比如图 13-1 中显示的层次结构。地域（region）、产业（industry）及主题（subject）这 3 个类别中每个类别下面都有 2 个子类别。类别的层次结构能够为解决分类问题提供重要的辅助作用，关于这一点将在 15.3.2 节中做进一步的讨论。在那之前，我们都只假设文本分类中的类别之间不存在层次关系。

定义（13-1）规定每篇文档只能属于一类。对于图 13-1 所示的具有层次结构的类别体系来说，这种规定显然不合适。比如，一篇有关 2008 奥运会的文档应该同时属于两个类：China 类及 sports 类。这种分类问题称为多标签（any-of）问题，将会在 14.5 节讨论。现在我们只考虑单标签（one-of）问题，即一篇文档只属于一个类别。

文本分类的目标是在测试数据或新数据（new data）上获得高精确率的结果。比如，上面提到的次日早晨阅读新闻文章以获取与 multicore chip 相关信息的那个例子。在训练集上获得高精确率是非常容易的事（比如，一种简单的方法就是记忆所有训练集文档的类别标签），但是这并不意味着该分类器就能在新数据上获得好的效果。当使用训练集来学习用于测试数据的分类器时，我们通常假设训练数据和测试数据类似或者说来自同一分布（the same distribution）。14.6 节将会给这个概念一个精确的定义。

13.2 朴素贝叶斯文本分类

我们介绍的第一个有监督的学习方法是多项式朴素贝叶斯（multinomial Naive Bayes）或者多项式 NB（multinomial NB）模型, 它是一种基于概率的学习方法。该方法中，文档 d 属于类别 c 的概率的计算方法如下：

$$P(c \mid d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k \mid c) \text{。} \qquad (13\text{-}2)$$

其中，$P(t_k|c)$是t_k出现在类c文档中的条件概率[①]，也可以把$P(t_k|c)$视为当正确类为c时t_k的贡献程度。$P(c)$是文档出现在类c中的先验概率。如果根据文档的词项并不能清晰地区分它属于哪一类时，我们就选择先验概率最大的那个类。$<t_1, t_2, \ldots, t_{nd}>$是d 中的词条，它们是分类所用词汇表的一部分，$n_d$是$d$中所有词条的数目。例如，对于单句文档Beijing and Taipei join the WTO，如果将and和the视为停用词过滤掉的话，那么这里的$<t_1, t_2, \ldots, t_{nd}>$就可以是<Beijing，Taipei, join, WTO>，其中$n_d = 4$。

在文本分类中，我们的目标是找出文档最可能属于的类别。对于 NB 分类来说，最可能的类是具有 MAP（maximum a posteriori，最大后验概率）估计值的结果 c_{map}：

$$c_{map} = \arg\max_{c \in C} \hat{P}(c \mid d) = \arg\max_{c \in C} \hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(t_k \mid c) \qquad (13\text{-}3)$$

由于我们不知道参数的真实值，所以上述公式中采用了从训练集中得到的估计值 $\hat{P}$ 来代替 P。

公式（13-3）中会对所有的$1 \le k \le n_d$（给校对者，上次的公式有误!），计算其对应的条件概率的乘积，这可能会导致浮点数下界溢出。因此，更好的方法是引入对数，从而将原公式的计算转变成多个概率的对数之和。由于 $\log(xy) = \log(x) + \log(y)$，且 log 是单调递增函数，因此具有较高概率对数值的类别就是最有可能的类别。因此，大多数 NB 在实现时所求的最大值实际是

$$c_{map} = \arg\max_{c \in C} \hat{P}(c \mid d) = \arg\max_{c \in C} [\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k \mid c)] \text{。} \qquad (13\text{-}4)$$

对于上式，有个简单的解释。条件参数 $\log \hat{P}(t_k \mid c)$ 表示的是 t_k 在类别 c 中的权重，而对数先验值 $\log \hat{P}(c)$ 表示的是类别 c 的相对频率的一个权重值。相对于低频类而言，高频类更可能是正确类。类别的对数先验值和词项在类别中权重累加求和之后就得到了文档属于类别的可能程度，公式（13-4）选择最可能的类别作为最终的类别。

我们先基于这种直观解释来使用上述公式，这实际上是多项式 NB 模型的一个解释，我们将在 13.4 节中给出一个形式化的推导。

如何估计参数 $\hat{P}(c)$ 及 $\hat{P}(t_k \mid c)$？首先我们使用最大似然估计（MLE，参见 11.3.2 节），它实际最后算出的是相对频率值，这些值能使训练数据的出现概率最大。MLE 估计下的类别先验概

① 下一节将解释为什么这里用的是正比关系($\propto$)而不是相等。

率为

$$\hat{P}(c)=\frac{N_c}{N}。\qquad(13\text{-}5)$$

其中，N_c是训练集合中 c 类所包含的文档数目，而 N 是训练集合中的文档总数。

条件概率 $\hat{P}(t_k\mid c)$ 的估计值为 t 在 c 类文档中出现的相对频率：

$$\hat{P}(t\mid c)=\frac{T_{ct}}{\sum_{t'\in V}T_{ct'}}\qquad(13\text{-}6)$$

其中，T_{ct}是 t 在训练集合 c 类文档中出现的次数，在对每篇文档计算时用的是其在文档中多次出现的词频。这里我们引入了位置独立性假设（positional independence assumption），在该假设下，T_{ct}是 t 在训练集某类文档中所有位置 k 上的出现次数之和。这样，对于不同位置上的概率值都采用相同的估计办法，比如，如果某词在一篇文档中出现过两次，分别在 k_1和 k_2的位置上，那么我们假定 $\hat{P}(t_{k_1}\mid c)=\hat{P}(t_{k_2}\mid c)$。关于这个假设，我们在下一节还要详细讨论，这里就不再赘述。

MLE 估计的一个问题是：对没有在训练集中出现的<词项，类别>对来说，其 MLE 估计值为 0。比如，如果在训练集上，WTO 仅仅在 China 类文档中出现，那么对于其他类（如 UK），采用 MLE 估计的概率值就会为 0，即

$$\hat{P}(\mathrm{WTO}\mid UK)=0。$$

现在，假定有一篇单句文档为 Britain is a member of the WTO，那么按照公式（13-2）来计算其属于 UK 类的条件概率值就为 0。很显然，由于文档中包含 Britain，此时应该为其属于 UK 类的条件概率赋予一个较高的值。也就是说，此时不能对 WTO 属于 UK 类的概率值赋 0，因为一旦出现 0 值，其他词项的概率再高也没有意义。出现零概率的主要原因来自于数据的稀疏性（sparseness），即训练集合永远都不可能大到所有罕见事件都能出现，这样就难以计算这些事件的频率。比如，上面要计算的 WTO 出现在 UK 类文档中的频率。

为了去掉零概率，一个简单的方法是采用加一平滑（add-one smoothing）或称拉普拉斯平滑（Laplace smoothing），即在每个数字上加 1（参考 11.3.2 节）：

$$\hat{P}(t\mid c)=\frac{T_{ct}+1}{\sum_{t'\in V}(T_{ct'}+1)}=\frac{T_{ct}+1}{\sum_{t'\in V}T_{ct'}+B}。\qquad(13\text{-}7)$$

其中，$B=|V|$是词汇表中所有词项的数目。加一平滑可以认为是采用均匀分布作为先验分布（每个词项在每个类中出现一次）然后根据训练数据进行更新得到的结果。需要指出的是，这里的先验分布是词项的先验分布，而不是前面公式 13-5 中提到的基于文档数目来计算的类别的先验分布。

到现在为止，我们已经给出了训练和应用 NB 分类器的所有环节，完整的算法描述参见图 13-2。

```
TrainMultinomialNB(ℂ, 𝔻)
 1  V ← ExtractVocabulary(𝔻)
 2  N ← CountDocs(𝔻)
 3  for each c ∈ ℂ
 4  do N_c ← CountDocsInClass(𝔻, c)
 5     prior[c] ← N_c/N
 6     text_c ← ConcatenateTextOfAllDocsInClass(𝔻, c)
 7     for each t ∈ V
 8     do T_ct ← CountTokensOfTerm(text_c, t)
 9     for each t ∈ V
10     do condprob[t][c] ← (T_ct+1)/(Σ_t'(T_ct'+1))
11  return V, prior, condprob

ApplyMultinomialNB(ℂ, V, prior, condprob, d)
 1  W ← ExtractTokensFromDoc(V, d)
 2  for each c ∈ ℂ
 3  do score[c] ← log prior[c]
 4     for each t ∈ W
 5     do score[c] += log condprob[t][c]
 6  return arg max_{c∈ℂ} score[c]
```

图 13-2 多项式 NB 的训练和分类算法

例 13-1 对于表 13-1 中的例子，文档分类所需要的多项式参数包括先验概率 $\hat{P}(c) = 3/4$，$\hat{P}(\overline{c}) = 1/4$。其他参数为：

$$\hat{P}(\text{Chinese}|c) = (5+1)/(8+6) = 6/14 = 3/7$$

$$\hat{P}(\text{Tokyo}|c) = \hat{P}(\text{Japan}|c) = (0+1)/(8+6) = 1/14$$

$$\hat{P}(\text{Chinese}|\overline{c}) = (1+1)/(3+6) = 2/9$$

$$\hat{P}(\text{Tokyo}+\overline{c}) = \hat{\text{P}}(\text{Japan}|\overline{c}) = (1+1)/(3+6) = 2/9$$

表13-1 用于参数估计的数据

	文档ID	文档中的词	属于*c=China*类?
训练集	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
测试集	5	Chinese Chinese Chinese Tokyo Japan	?

上述计算中的分母分别是(8+6)和(3+6)，这是因为 c 类文档的总长度为 8，而非 c 类文档的总长度为 3，而按照公式（13-7)，常数 B 为词汇表大小 6。因此，我们有

$$\hat{P}(c \mid d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003,$$

$$\hat{P}(\overline{c} \mid d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001。$$

于是，分类器会将测试文档归于 c=China 类。这是因为 d_5 中的正特征 Chinese 的权重比负特征 Japan 和 Tokyo 的权重都要大。

接下来我们讨论 NB 算法的复杂度。由于所要估计的参数集合包括$|\coprod||V|$个条件概率和$|\coprod|$个先验概率，所以参数估计的时间复杂度为 $\Theta(|\coprod||V|)$。参数计算所需的预处理过程（词汇表的抽取、词项的计数等）可以在对训练数据的单遍扫描中完成，因此，这部分的时间开销是 $\Theta(|\daleth|L_{\text{ave}})$，其中$|\daleth|$是训练集合中文档的数目，$L_{\text{ave}}$是文档的平均长度。

这里我们采用 $\Theta(|\daleth|L_{\text{ave}})$来表示 $\Theta(T)$，其中 T 是训练集的长度。应该说这不是一种常规表示，因为 $\Theta(.)$ 并不定义在平均意义上。这里之所以这样来表示，主要是为了突出$|\daleth|$ 和 L_{ave}这两个常用于刻画训练集的统计量。

图 13-2 中APPLYMULTINOMIALNB的时间复杂度为$\Theta(|\coprod|L_{\text{a}})$，$L_a$和$M_a$分别是测试文档中词条及词条类的数目。可以对APPLYMULTINOMIALNB进行修改，修改后的复杂度为$\Theta(L_{\text{a}}+|\coprod|M_{\text{a}})$（参见习题 13-8）。最后，假定测试文档长度有界的情况下[①]，$\Theta(L_{\text{a}}+|\coprod|M_{\text{a}})=\Theta(|\coprod|M_{\text{a}})$，这是因为对于某个固定的常数$b$，有$L_{\text{a}}<b|\coprod|M_{\text{a}}$。

表 13-2 对时间复杂度进行了总结。一般来说，我们有$|\coprod||V|<|\daleth|L_{\text{ave}}$，因此不论是训练还是测试，扫描数据的复杂度都是线性的。由于至少需要一次数据扫描，因此可以说 NB 具有最优的复杂度，而它的高效性也是其在文本分类中被广泛使用的原因。

表13-2　NB的训练和测试的时间复杂度

阶段	时间复杂度
训练	$\Theta(\mathbb{D}\|L_{\text{ave}}+\square\|V\|)$
测试	$\Theta(L_{\text{a}}+\square\|M_{\text{a}})=\Theta(\square\|M_{\text{a}})$

与多项式语言模型的关系

多项式 NB 模型形式上等价于 12.2.1 节中介绍的多项式一元 LM。特别地，公式（13-2）是公式（12-12）中 $\lambda=1$ 时的一个特例，这种情况下公式（12-12）可以重写为

$$P(d\mid q)\propto P(d)\prod_{t\in q}P(t\mid M_d)\text{。}\qquad(13\text{-}8)$$

公式（13-2）中的文档 d 相当于公式（13-8）的 LM 中查询的角色，而（13-2）的类 c 相当于公式（13-8）中的文档 d。按照公式（13-8），可以根据文档和查询 q 相关的概率对文档进行排序。同样，在分类中可以根据类别和文档的相似度对类别进行排序。在 NB 分类中，我们往往只关注排名最高的类。

在 12.2.2 节中，我们同样使用了 MLE 估计，由于数据的稀疏性导致了零概率问题。但是，在那里我们并没有使用加一平滑方法，而是采用了两个分布的混合。加一平滑与 11.3.4 节中的加$\frac{1}{2}$平滑非常类似。

? 习题 **13-1**　对于表 13-2，为什么在绝大部分文本集中$|\coprod||V|<|\daleth|L_{\text{ave}}$都成立？

① 这里假设测试文档的长度是有界的，对于某些特别长的测试文档来说，L_{a}有可能超过 $b|\coprod|M_{\text{a}}$。

13.3 贝努利模型

建立 NB 分类器有两种不同的方法。上节介绍的是基于多项式的方法，它基于一个生成式模型（详细讨论参见 13.4 节）在文档的每个位置上生成词表中的一个词项。

另外一种方法是多元贝努利模型（multivariate Bernoulli model）或者直接称为贝努利模型（Bernoulli model）。它等价于 11.3 节的二值独立模型，对于词汇表中的每个词项都对应一个二值变量，1 和 0 分别表示词项在文档中出现和不出现。图 13-3 给出了基于贝努利模型的 NB 分类器的训练和测试算法。贝努利模型和多项式模型具有一样的时间复杂度。

```
TRAINBERNOULLINB(ℂ, 𝔻)
1  V ← EXTRACTVOCABULARY(𝔻)
2  N ← COUNTDOCS(𝔻)
3  for each c ∈ ℂ
4  do N_c ← COUNTDOCSINCLASS(𝔻, c)
5     prior[c] ← N_c/N
6     for each t ∈ V
7     do N_ct ← COUNTDOCSINCLASSCONTAININGTERM(𝔻, c, t)
8        condprob[t][c] ← (N_ct + 1)/(N_c + 2)
9  return V, prior, condprob

APPLYBERNOULLINB(ℂ, V, prior, condprob, d)
1  V_d ← EXTRACTTERMSFROMDOC(V, d)
2  for each c ∈ ℂ
3  do score[c] ← log prior[c]
4     for each t ∈ V
5     do if t ∈ V_d
6           then score[c] += log condprob[t][c]
7           else score[c] += log(1 − condprob[t][c])
8  return arg max_{c∈ℂ} score[c]
```

图 13-3 基于贝努利模型的 NB 算法的训练及分类过程。函数 TRAINBERNOULLINB 第 8 行的加一平滑类似于在公式（13-7）中取 $\boldsymbol{B}=2$

不同的生成模型也意味着不同的参数估计策略和分类规则。贝努利模型中 $\hat{P}(t\mid c)$ 利用类 c 文档中包含 t 的文档数的比率来计算（参见图 13-3，TRAINBERNOULLINB，第 8 行）。而与之形成鲜明对比的是，多项式模型中计算的是 t 出现的次数占类 c 文档中所有词条数目的比率（参见公式（13-7））。当对测试文档进行分类时，贝努利模型只考虑词项的出现或不出现（即二值），并不考虑出现的次数，而多项式模型中则要考虑出现次数。这样做的结果是，当对长文档进行分类时，采用贝努利模型往往会犯很多错误。比如，可能会因为 China 在书中的一次出现而将整本书归于 China 类。

两种模型对于未出现词项在分类中的使用也不相同：未出现的词项在多项式模型中并不影响分类效果，但是在贝努利模型中计算 $P(c|d)$ 时要以一个因子来参与计算（参见图 13-3 中 APPLYBERNOULLINB 函数的第 7 行），其主要原因是，贝努利模型对词项的未出现也要显式建模。

例 13-2　对于表 13-1 中的例子采用贝努利模型进行计算，对于先验概率，我们同多项式模型中一样估计，即 $\hat{P}(c)=3/4$，$\hat{P}(\overline{c})=1/4$。条件概率为：

$$\hat{P}(\text{Chinese}|c)=(3+1)/(3+2)=4/5$$
$$\hat{P}(\text{Japan}\,|\,c)=\hat{P}(\text{Tokyo}|c)=(0+1)/(3+2)=1/5$$
$$\hat{P}(\text{Beijing}\,|\,c)=\hat{P}(\text{Macao}|c)=\hat{P}(\text{Shanghai}|c)=(1+1)(3+2)=2/5$$
$$\hat{P}(\text{Chinese}|\overline{c})=(1+1)/(1+2)=2/3$$
$$\hat{P}(\text{Japan}\,|\,\overline{c})=\hat{P}(\text{Tokyo}\,|\,\overline{c})=(1+1)/(1+2)=2/3$$
$$\hat{P}(\text{Beijing}\,|\,\overline{c})=\hat{P}(\text{Macao}\,|\,\overline{c})=\hat{P}(\text{Shanghai}|\overline{c})=(0+1)/(1+2)=1/3$$

这个问题中有 3 篇文档属于 c 类，1 篇文档属于非 c 类，另外由于对每个词项都只考虑出现与不出现两种情形，因此公式（13-7）中的常数 B 为 2。

因此，测试文档分别属于两个类别的得分为

$$\begin{aligned}\hat{P}(c\,|\,d_5)&\propto\hat{P}(c)\cdot\hat{P}(\text{Chinese}\,|\,c)\cdot\hat{P}(\text{Japan}\,|\,c)\cdot\hat{P}(\text{Tokyo}\,|\,c)\\&\quad\cdot(1-\hat{P}(\text{Beijing}\,|\,c))\cdot(1-\hat{P}(\text{Shanghai}\,|\,c))\cdot(1-\hat{P}(\text{Macao}\,|\,c))\\&=3/4\cdot4/5\cdot1/5\cdot1/5\cdot(1-2/5)\cdot(1-2/5)\cdot(1-2/5)\\&\approx0.005\end{aligned}$$

类似地，有

$$\begin{aligned}\hat{P}(\overline{c}\,|\,d_5)&\propto1/4\cdot2/3\cdot2/3\cdot2/3\cdot(1-1/3)\cdot(1-1/3)\cdot(1-1/3)\\&\approx0.022\end{aligned}$$

因此，根据上述结果，分类器最终会将测试文档归为非 c 类。当只关注词项出现与否而不考虑词项频率时，Japan 和 Tokyo 对于 $\overline{c}$ 来说是正向标志特征（2/3 > 1/5），而 Chinese 属于 c 类和非 c 类的条件概率的差异还不足以影响分类的结果。

13.4　NB 的性质

为了对两种不同模型以及它们给出的假设有更深的理解，我们回顾一下在第 11 章和第 12 章分类规则的导出过程。我们将文本归入后验概率最大的那个类别（参考 11.3.2 节），计算方式如下：

$$\begin{aligned}c_{\text{map}}&=\arg\max_{c\in C}P(c\,|\,d)\\&=\arg\max_{c\in C}\frac{P(d\,|\,c)P(c)}{P(d)}&(13\text{-}9)\\&=\arg\max_{c\in C}P(d\,|\,c)P(c)\ 。&(13\text{-}10)\end{aligned}$$

其中公式（13-9）应用了贝叶斯定理（参见公式（11-4））, 由于 $p(d)$对任何类别来说取值都一样，

不会影响 argmax 函数的结果，所以在推导中可以去掉。

对于公式（13-10），我们可以用文本的生成过程来解释，并将该过程作为假设用于贝叶斯文本分类中。为了生成一篇文档，首先基于概率 $p(c)$选择 c 类（图 13-4 和图 13-5 中的顶层节点）。两个模型在第二步的形式化上有所不同，即在给定类别的情况下文档生成的条件概率计算有所不同：

$$\text{多项式模型 } P(d|c) = P(<t_1, \ldots, t_k, \ldots, t_{n_d}>|c)， \tag{13-11}$$

$$\text{贝努利模型 } P(d|c) = P(<e_1, \ldots, e_i, \ldots, e_M>|c)。 \tag{13-12}$$

其中，$<t_1,\ldots, t_{nd}>$是在 d 中出现的词项序列（当然要去掉那些从词汇表中去掉的词，如停用词），$<e_1,\ldots, e_i, \ldots, e_M>$是一个 M 维的布尔向量，表示每个词项在文档 d 中存在与否。

现在我们应该明白了为什么在定义分类问题时在公式（13-1）中引入文档空间□。解决文本分类问题的一个关键步骤是选择文档的表示方法，而$< t_1, \ldots, t_{nd} >$和$<e_1, \ldots, e_i, \ldots, e_M>$正好是两种不同的文档表示方法。第一种表示方法中，□是所有词项序列的集合，或者更精确地说，是所有词条序列的集合[①]；在第二种表示方法中，□是$\{0,1\}^M$。

我们不能把公式（13-11）和公式（13-12）直接用于文本分类中。因为，对于贝努利模型，必须要估计 $2^M|\coprod|$个不同的参数，每个参数都是M个e_i取值和一个类别取值的组合。多项式模型和贝努利模型具有相同数量级的参数个数[②]。由于参数空间巨大，对这些参数进行可靠估计是不可行的。

为了减少参数的数目，我们引入了朴素贝叶斯的条件独立性假设（conditional independence assumption），即给定类别时，假设属性值之间是相互独立的：

$$\text{多项式模型 } P(d \mid c) = P(< t_1, \cdots, t_{n_d} > |c) = \prod_{1 \le k \le n_d} P(X_k = t_k \mid c)， \tag{13-13}$$

$$\text{贝努利模型 } P(d \mid c) = P(< e_1, \cdots, e_M > |c) = \prod_{1 \le i \le M} P(U_i = e_i \mid c)。 \tag{13-14}$$

上面公式中引入了两类随机变量 X_k 和 U_i，这样的话两个不同的文本生成模型就更清晰。X_k 是文档在位置 k 上的随机变量，$P(X_k = t \mid c)$表示的是一篇 c 类文档中词项 t 出现在位置 k 上的概率。随机变量 U_i对应词项 i，当词项在文档中不出现时取 0，出现时取 1。$P(U_i = 1 \mid c)$ 表示的是 t_i出现在 c 类文档中的概率，这时可以是在任意位置上出现任意多次。

在图 13-4 和图 13-5 中我们给出了条件独立性假设的一个示意图。对 5 个词项属性（对应多项式模型）和 6 个二值属性（对应贝努利模型），China 类到它们都有一个生成概率值。一篇 China 类文档中包含 Taipei 的事实并不会增加或者减少该文档包含 Beijing 的可能性。

① 也就是说，不同 t_i之间可能有重复，此时是文档每个位置上的词项的列举。——译者注

② 实际上，当文档长度没有限制时，多项式模型中的参数个数是无穷的。

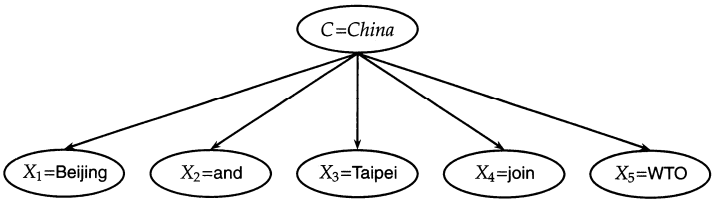


图 13-4　多项式 NB 模型

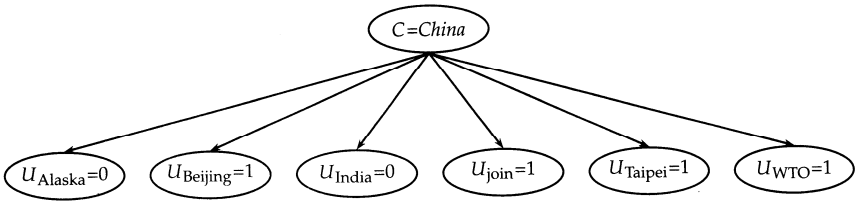


图 13-5　贝努利 NB 模型

现实当中，文本数据上的条件独立性假设并不成立，词项之间存在条件依赖。但是我们很快就会看到，尽管采用了条件独立性假设，NB 模型也表现出很好的性能。

即使是采用条件独立性假设，但假如在文档中每个位置 k 上的概率分布不同的话，那么对于多项式模型来说仍然有太多的参数需要估计。词项在文档中的出现位置本身并不包含任何对分类有用的信息。尽管 China sues France 和 France sues China 不同，但是 China 是在文档中第 1 个位置还是在第 3 个位置出现对于 NB 分类来说毫无分别，这是因为 NB 独立看待每个词项。条件独立性假设对上述处理方法提供了有效支持。

另一方面，如果假设在不同位置 k 上的词项分布不一样的话，那么就要估计针对每个 k 的一系列参数。比如，bean 出现在 coffee 类文档的第一个位置和出现在其第二个位置的概率是不同的，其他位置可以依次类推。这会再次导致数据估计中的稀疏性问题。

基于上述原因，我们在多项式模型中引入第二个独立性假设—位置独立性假设（positional independence），即词项在文档中每个位置的出现概率是一样的，也就是对于任意位置 k_1、k_2、词项 t 和类别 c，有

$$P(X_{k_1}=t\mid c)=P(X_{k_2}=t\mid c)。$$

因此，对所有位置k_i，我们都有单一的词项分布，这时每个位置上的随机变量[①]可以统一写成X。位置独立性假设等价于在第 6 章介绍ad hoc 检索时所采用的词袋模型。

基于条件独立性和位置独立性假设，我们只需要估计 $\Theta(M|\coprod|)$个多项式模型下的参数 $P(t_k|c)$ 或贝努利模型下的参数 $P(e_i|c)$，其中每个参数对应一个词项和类别的组合。也就是说，这里的参数数目远远小于前面提到的值，而那个值至少具有词汇表大小 M 的指数级大小。上述独立性

① 这里我们使用的术语可能不是特别规范。随机变量 X 是一个类别型变量而不是多项式变量。相应的 NB 模型应该称为序列模型（sequence model）。由于在计算上完全相同，在 13.4.1 节中我们已经将这个序列模型和多项式模型视为等价模型。

假设使得需要考虑的参数的数目减小了多个数量级。

概括一下，在多项式模型（参见图 13-4）中，首先以概率 $P(c)$来选择一个类别 $C = c$ ，其中 C 是一个从$\coprod$中取值的随机变量，然后根据模型生成一篇文档。接着，对于文档的 n_d 个位置，在每个位置 k 上以概率 $P(X_k = t_k | c)$生成词项 t_k。并且对于给定的 c ，每个 X_k.的分布是一样的。在图 13-4 所示的例子中，我们给出了单句文档 Beijing and Taipei join WTO 的生成过程，其中$<t_1, t_2, t_3, t_4, t_5> = <$Beijing , and, Taipei, join , WTO$>$。

对于一个完全确定的文档生成模型而言，还需要对 $P(n_d | c)$这个长度分布进行定义。如果没有这个分布，那么该多项式分布就是一个词条的生成模型而不是一个文档的生成模型。

在贝努利模型（图 13-5）文档的生成过程中，首先以概率 $P(c)$来选择一个类别 $C = c$，然后对词典中的每个词项 t_i （$1 \leq i \leq M$），都产生一个对应的二值变量 e_i。图 13-5 的例子中，我们仍然以单句文档 Beijing and Taipei join WTO 为例，介绍了$<e_1, e_2, e_3, e_4, e_5, e_6> = <0, 1, 0, 1, 1, 1>$的生成过程，其中 and 被看成停用词。

在表 13-3 中，我们给出了两个模型之间的比较结果，其中包括计算公式和决策规则的比较。

表13-3 多项式模型和贝努利模型的比较

	多项式模型	贝努利模型
事件模型	词条生成模型	文档生成模型
随机变量	$X = t$，当且仅当 t 出现在给定位置	U_t=1，当且仅当 t 出现在文档中
文档表示	$d = <t_1, \ldots, t_k, \ldots, t_{nd}>, t_k \in V$	$d = <e_1, \ldots, e_i, \ldots, e_M>, e_i \in \{0,1\}$
参数估计	$\hat{P}(X = t \mid c)$	$\hat{P}(U_i = e \mid c)$
决策规则：最大化	$\hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(X = t_k \mid c)$	$\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i \mid c)$
词项多次出现	考虑	不考虑
文档长度	能处理更长文档	最好处理短文档
特征数目	能够处理更多特征	特征数目较少效果更好
词项the的估计	$\hat{P}(X = \text{the} \mid c) \approx 0.05$	$\hat{P}(U_{\text{the}} \mid c) \approx 1.0$

朴素贝叶斯之所以得名，在于刚才给出的独立性假设对于自然语言建模来说确实非常朴素。条件独立性假设声称在给定类别的情况下特征之间相互独立，这对于实际文档中的词项来说几乎不可能成立。很多情况下，词项之间明显存在着相互关联。比如，图 13-7 中的 hong 和 kong、london 和 english 之间就存在高度的相关性。另外，多项式模型中还给出了位置独立性假设。而由于贝努利模型中只考虑词项出现或不出现，所以它忽略了所有的位置信息。这种词袋模型忽略了自然语言句子中词序相关的信息，所以 NB 对自然语言的建模做了非常大的简化，从这个意义上讲，如何能保证 NB 方法的分类效果？

问题的答案是，即使 NB 的概率估计效果很差，但是其分类决策的效果也却出人意料地好。考虑如表 13-4 所示的文档 d，其真实的概率是 $P(c_1|d) = 0.6$、$P(c_2|d) = 0.4$。假设 d 包含了很多 c_1 的正特征和 c_2 的负特征，因此当使用公式（13-3）的多项式模型计算时，$\hat{P}(c_1)\prod_{1 \le k \le n_d} \hat{P}(t_k \mid c_1)$

会远远大于 $\hat{P}(c_2)\prod_{1\le k\le n_d}\hat{P}(t_k \mid c_2)$（表中它们的值分别为 0.000 99 和 0.000 01）。如果两个值都除以 0.001 来得到合理的 $P(c|d)$概率值的话，那么前者接近 1，而后者几乎为 0。这种现象很普遍：NB 分类中胜出的那一类往往会比其他类得到的概率估计值大很多，并且其估计值会和真实值相差非常大。然而，分类决策取决于哪个类别得分最高，它并不关注得分本身的精确性。尽管概率估计效果很差，但是 NB 会给 c_1 一个很高的分数，因此最后会将 d 归到正确的类别中（参见表 13-4 中的例子）。正确的参数估计意味着精确的预测，但是精确的预测不一定意味着正确的参数估计。NB 分类器的估计效果很差，但是往往分类效果不错。

表13-4　正确的参数估计意味着精确的预测，但是精确的预测不一定意味着正确的参数估计

	c_1	c_2	选择的类别
真实概率$P(c \mid d)$	0.6	0.4	c_1
$\hat{P}(c)\prod_{1\le k\le n_d}\hat{P}(t_k \mid c)$（公式（13-13））	0.000 99	0.00 001	
NB估计 $\hat{P}(c \mid d)$	0.99	0.01	c_1

虽然分类效果不是最好，但是 NB 仍然有很多优点，足以令它在文本分类中占有一席之地。首先，当有多个同等重要的特征联合起来对分类决策起作用时，NB 能够表现出很好的效果。其次，NB 能够对噪音特征（参见 13.5 节的定义）和概念漂移（concept drift）现象表现出一定程度的鲁棒性。其中，概念漂移指的是随时间的推移类别的概念发生变化的现象，比如 US president 这个类的概念已从 Bill Clinton 变成 George W. Bush（参见 13.7 节）。而像 kNN 一样的分类器（参见 14.3 节）可以通过精细的调优来适应某个特定时期数据的特点，但是一旦下一段时期的文档稍有不同，便会影响分类的效果。

贝努利模型尤其对概念漂移具有鲁棒性。从图 13-8 中可以看到，当使用不到十二个词项特征的时候模型便取得了不错的效果。某个类别最重要的那些特征一般不太可能变化，因此在概念漂移时，仅仅基于这些特征的模型更可能获得一定精度的结果。

NB 的主要优势是其速度快，不论是训练还是分类过程都可以在数据的一遍扫描中完成。速度的优势再加上不低的精确率，使得 NB 往往作为文本分类研究的一个基准分类器来使用。在下列情况下可以考虑选择 NB 来分类：

(i) 不值得为提高一丁点的精确率而在文本分类中引入更繁琐的策略；

(ii) 当有大规模的训练数据时，NB 可以从这么多数据中获得更多的信息，而不是采用工作在小规模训练数据上的更好的分类器；

(iii) 要考虑概念漂移处理的鲁棒性。

本书中我们主要将 NB 作为文本分类器，但实际上，文本中的独立性假设并不成立。然而，可以证明当数据满足独立性假设时，从新数据上的最小错误率这个意义上说，NB 是最优分类器（optimal classifier）。

多项式模型的一个变形

多项式模型的另外一种形式化表示中，将文档 d 表示成 M 维的数字向量$<tf_{t_1,d}, \ldots, tf_{t_M,d}>$，其中 $tf_{t_i,d}$ 是词项 t_i 出现在 d 中的词项频率。$P(d|c)$可以按照如下公式计算：

$$P(d \mid c) = P(< tf_{t_1,d}, \cdots, tf_{t_M,d} > \mid c) \propto \prod_{1 \le i \le M} P(X = t_i \mid c)^{tf_{t_i,d}}。 \quad (13\text{-}15)$$

需要指出的是，上述计算中略去了多项式因子。具体请参考公式（12-8）。

因为当词项在 d 中不出现时（此时 $tf_{t_i,d}$=0），有 $P(X = t_i \mid c)^{tf_{t_i,d}} = 1$，而当词项的出现次数 $tf_{t_i,d} \geq 1$ 时，两个公式中 $tf_{t_i,d}$ 都作为幂指数出现，所以公式（13-15）和公式（13-2）所示的序列模型是等价的。

? 习题 **13-2 [*]** 表 13-5 中的文档中，对于如下的两种模型表示，哪些文档具有相同的模型表示？哪些文档具有不同的模型表示？对于不同的表示进行描述。

(i) 贝努利模型，(ii) 多项式模型。

表13-5 NB独立性假设存在问题的几个文档例子

(1)	He moved from London, Ontario, to London, England.
(2)	He moved from London, England, to London, Ontario.
(3)	He moved from England to London, Ontario.

习题 **13-3** 位置独立性假设的基本原则是，词项在文档的位置 k 上出现这个事实并没有什么有用的信息。请给出这个假设的反例。提示：可以考虑那些套用固定文档结构的文档。

习题 **13-4** 表 13-3 给出了单词 the 的贝努利和多项式估计，请解释它们的不同之处。

13.5 特征选择

特征选择（feature selection）是从训练集合出现的词项中选出一部分子集的过程。在文本分类过程也仅仅使用这个子集作为特征。特征选择有两个主要目的：第一，通过减少有效的词汇空间来提高分类器训练和应用的效率。这对于除 NB 之外其他的训练开销较大的分类器来说尤为重要。第二，特征选择能够去除噪音特征，从而提高分类的精度。噪音特征（noise feature）指的是那些加入文本表示之后反而会增加新数据上的分类错误率的特征。假定某个罕见词项（如 arachnocentric）对某个类别（如 China）不提供任何信息，但是在训练集中所有的 arachnocentric 恰好都出现在 China 类中，那么通过学习后可能会产生一个分类器，它会将包含 arachnocentric 的测试文档误分到 China 类中去。这种由于训练集的偶然性导出的不正确的泛化结果称为过学习（overfitting）。

我们可以把特征选择看成将复杂分类器替换成简单分类器的过程，其中复杂分类器使用全

部特征，而简单分类器只使用部分特征。乍一看这与我们的直觉相矛盾，似乎在统计文本分类中弱分类器更具有优势。但是，当我们在 14.6 节讨论到偏差–方差折衷准则时，我们就会明白，当训练数据有限时往往倾向于使用弱分类模型。

图 13-6 展示了基本的特征选择算法。给定类别 c，对词汇表中的每个词项 t，我们计算效用指标 $A(t,c)$，然后从中选择 k 个具有最高值的词项作为最后的特征，其他的词项则在分类中都被忽略。本节我们将介绍 3 种不同的效用指标：互信息 $A(t, c) = I(U_t; C_c)$、χ^2 统计量 $A(t, c) = \mathrm{X}^2(t, c)$及词项频率 $A(t, c) = N(t, c)$。

```
SELECTFEATURES(𝔻, c, k)
1  V ← EXTRACTVOCABULARY(𝔻)
2  L ← []
3  for each t ∈ V
4  do A(t, c) ← COMPUTEFEATUREUTILITY(𝔻, t, c)
5     APPEND(L, ⟨A(t, c), t⟩)
6  return FEATURESWITHLARGESTVALUES(L, k)
```

图 13-6　选择 k 个最佳特征的基本特征选择算法

在两种 NB 模型当中，贝努利模型对噪音特征特别敏感。对于贝努利 NB 分类器，必须进行某种形式的特征选择，否则它的精度会很低。

本节主要介绍二类问题（如 China 类和非 China 类）的特征选择方法，13.5.5 节将会对多类问题系统特征选择的优化问题做个简要介绍。

13.5.1　互信息

一个常用的特征选择方法是计算词项t和类别c的 MI（expected mutual information，期望互信息）[①]作为$A(t,c)$。MI度量的是词项的存在与否给类别c的正确判断所带来的信息量。MI的形式化定义如下：

$$I(U;C) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} P(U = e_t, C = e_c) \log_2 \frac{P(U = e_t, C = e_c)}{P(U = e_t)P(C = e_c)} \quad , \qquad (13\text{-}16)$$

其中，U 是一个二值随机变量，当文档包含词项 t 时，它取值为 e_t=1，否则取值为 e_t=0。而 C 也是个二值随机变量，当文档属于类别 c 时，它取值为 e_c=1，否则取值为 e_c=0。当在某个上下文中具体的 e 和 c 并不确定时，我们笼统地用 U_t 和 C_c 来表示。

当采用 MLE 来估计概率参数时，公式（13-16）等价于公式（13-17）：

$$I(U;C) = \frac{N_{11}}{N} \log_2 \frac{NN_{11}}{N_{1.}N_{.1}} + \frac{N_{01}}{N} \log_2 \frac{NN_{01}}{N_{0.}N_{.1}} + \frac{N_{10}}{N} \log_2 \frac{NN_{10}}{N_{1.}N_{.0}} + \frac{N_{00}}{N} \log_2 \frac{NN_{00}}{N_{0.}N_{.0}} \text{。} \qquad (13\text{-}17)$$

① 注意不要将期望互信息和点互信息（pointwise mutual information）混淆，后者定义为 $\log N_{11}/E_{11}$，其中 N_{11}、E_{11} 在公式（13-17）中定义。期望互信息和点互信息具有不同的性质，具体参见 13.7 节。

其中，每个 N_{xy} 表示的是 $x=e_t$ 和 $y=e_c$ 情况下所对应的文档数目。比如 N_{10} 表示的是就是包含词项 t (此时 $e_t=1$)但是不属于类别 c（此时 $e_c=0$）。$N_{1.}=N_{10}+N_{11}$ 是所有包含词项 t 的文档数目。$N=N_{00}+N_{01}+N_{10}+N_{11}$ 是所有文档的数目。从公式（13-16）转换到公式（13-17）的一个 MLE 估计的例子是 $P(U=1, C=1)=N_{11}/N$。

例 3-3 考虑 Reuters-RCV1 语料中的一个类别 poultry 及词项 export。类别和词项四种组合的文档数目如下：

	$e_c=e_{\text{poultry}}=1$	$e_c=e_{\text{poultry}}=0$
$e_t=e_{\text{export}}=1$	$N_{11}=49$	$N_{10}=27\,652$
$e_t=e_{\text{export}}=0$	$N_{01}=141$	$N_{00}=774\,106$

将以上数值代入公式（13-17），有：

$$\begin{aligned}I(U;C)=&\frac{49}{801\,948}\log_2\frac{801\,948\times 49}{(49+27\,652)(49+141)}\\&+\frac{141}{801\,948}\log_2\frac{801\,948\times 141}{(141+774\,106)(49+141)}\\&+\frac{27\,652}{801\,948}\log_2\frac{801\,948\times 27,652}{(49+27\,652)(27\,652+774\,106)}\\&+\frac{774\,106}{801\,948}\log_2\frac{801\,948\times 774\,106}{(141+774\,106)(27\,652+774\,106)}\\\approx&\ 0.000\,110\,5。\end{aligned}$$

为从给定的类别中选出 k 个词项，我们采用图 13-6 所示的特征选择算法：先计算每个词项的效用指标 $A(t, c)=I(U_t, C_c)$，然后选择值最大的 k 个词项。

从信息论的角度来讲，互信息度量的是词项是否被类别包含所带来的信息量。如果某个词项在类别中的分布等同于其在所有文档集上的分布，那么 $I(U;C)=0$。当词项是判定类别归属的最佳特征时，互信息达到最大值。此时的词项应满足：当且仅当某篇文档属于当前类别时，词项出现在该文档中。

图 13-7 给出了图 13-1 中的 6 个类别中互信息得分最高的词项[①]。很显然，选出的词项（如对应类别*UK*有london、uk、british等词项）能对相应类别的判定起作用。在类别*UK*的词项表的末尾，我们还发现了诸如peripherals和tonight之类的明显对于分类没有意义的词项（这些词项在图中没有列出）。正如我们所期望的那样，保留那些具有信息含量的词项同时去掉那些没有信息含量的词项往往能够去除噪音从而提高分类的精确度。

① 除 poultry 这个罕见类之外，对其他类，我们都在所使用的 800 000 篇文档中，选择前 100 000 篇文档来计算特征的得分。在前十个特征的列表中，我们去掉了数字和一些其他的特殊词。

UK

london	0.1925
uk	0.0755
british	0.0596
stg	0.0555
britain	0.0469
plc	0.0357
england	0.0238
pence	0.0212
pounds	0.0149
english	0.0126

China

china	0.0997
chinese	0.0523
beijing	0.0444
yuan	0.0344
shanghai	0.0292
hong	0.0198
kong	0.0195
xinhua	0.0155
province	0.0117
taiwan	0.0108

poultry

poultry	0.0013
meat	0.0008
chicken	0.0006
agriculture	0.0005
avian	0.0004
broiler	0.0003
veterinary	0.0003
birds	0.0003
inspection	0.0003
pathogenic	0.0003

coffee

coffee	0.0111
bags	0.0042
growers	0.0025
kg	0.0019
colombia	0.0018
brazil	0.0016
export	0.0014
exporters	0.0013
exports	0.0013
crop	0.0012

elections

election	0.0519
elections	0.0342
polls	0.0339
voters	0.0315
party	0.0303
vote	0.0299
poll	0.0225
candidate	0.0202
campaign	0.0202
democratic	0.0198

sports

soccer	0.0681
cup	0.0515
match	0.0441
matches	0.0408
played	0.0388
league	0.0386
beat	0.0301
game	0.0299
games	0.0284
team	0.0264

图 13-7　Reuters-RCV1 语料 6 个类别中互信息值较高的部分词项列表

在图 13-8 中我们可以观察到特征选择后分类精确度提高的现象，图中的横坐标是 Reuters-RCV1 语料在特征选择后词汇表的大小，纵坐标是分类的F_1值 ①。在使用全部 132,776 个特征和采用MI进行特征选择后选用 10~100 个特征的情况下，我们发现，在采用多项式模型时，后一种情况下F_1值大约提高了 0.1，而采用贝努利模型时，提高值超过 0.2。贝努利模型下，F_1值很早就达到最高点（选择 10 个特征），此时，贝努利模型也要超过多项式模型。当采用很少特征对分类进行判定时，那么仅仅考虑特征的出现与否即可。而对于多项式模型，最高点来得较晚（当选择 100 个特征时），当使用所有特征时，分类的精度则有所恢复。这其中的原因在于，多项式模型在参数估计和分类时都考虑了特征的出现次数，因此当使用大量特征时性能会优于贝努利模型。需要指出的是，无论采用哪种模型，认真挑选出部分特征而获得的分类效果都会优于使用全部特征。

① 我们使用前 100 000 文档训练出分类器，然后在后 100 000 篇文档上得到 F_1 值。图中对 5 个类别进行了平均。

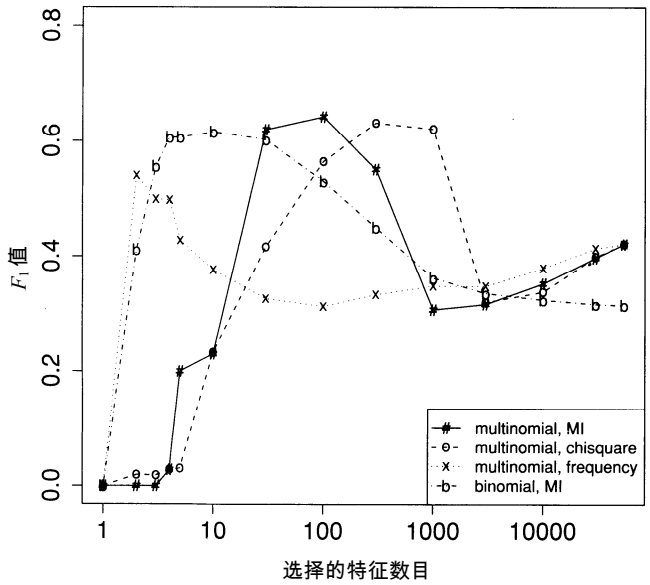


图 13-8 不同特征数目下多项式模型和贝努利模型的分类效果

13.5.2 χ^2统计量

另一个常用的特征选择方法是 χ^2，在统计学中，χ^2 统计量常常用于检测两个事件的独立性。两个事件 A 和 B 独立，是指两个事件 A、B 的概率满足 P(AB)=P(A)P(B)或者 P($A|B$)=P(A)且 P($B|A$)=P(B)。在特征选择中，两个事件分别是指词项的出现和类别的出现。此时，我们按照如下公式计算：

$$X^2(\mathrm{D},t,c)=\sum_{e_t\in\{0,1\}}\sum_{e_c\in\{0,1\}}\frac{(N_{e_te_c}-E_{e_te_c})^2}{E_{e_te_c}}。\qquad(13\text{-}18)$$

其中，e_t、e_c 的定义参见公式（13-16），字母 N 表示的是文中观察到的频率，而 E 则是期望频率。比如，E_{11} 表示在假定词项 t 和类别 c 独立的情况下它们在一篇文档中期望的共现频率。

例 13-4 对于例 13-3 中的数据，我们首先计算 E_{11}：

$$E_{11}=N\times P(t)\times P(c)=N\times\frac{N_{11}+N_{10}}{N}\times\frac{N_{11}+N_{01}}{N}$$

$$=N\times\frac{49+141}{N}\times\frac{49+27\,652}{N}\approx 6.6。$$

其中，和前面一样，N 是所有文档的数目。

同样，我们可以计算得到其他的 E 值：

	$e_c=e_{\text{poultry}}=1$		$e_c=e_{\text{poultry}}=0$	
$e_t=e_{\text{export}}=1$	$N_{11}=49$	$E_{11}\approx 6.6$	$N_{01}=27\,652$	$E_{01}\approx 27\,694.4$
$e_t=e_{\text{export}}=0$	$N_{10}=141$	$E_{10}\approx 183.4$	$N_{00}=774\,106$	$E_{00}\approx 774\,063.6$

将上述结果代入公式（13-18），得到 χ^2 的值为 284：

$$X^2(\mathrm{D},t,c)=\sum_{e_t\in\{0,1\}}\sum_{e_c\in\{0,1\}}\frac{(N_{e_te_c}-E_{e_te_c})^2}{E_{e_te_c}}\approx 284。$$

X^2 度量的是期望值*E*和观察值*N*的偏离程度。X^2 值大则意味着独立性假设不成立，此时期望值和观察值相差不大。在上例中，$X^2\approx 284>10.83$，基于表 13-6，我们可以拒绝*poultry*和export互相独立的假设，并且此时的错误发生概率仅有 0.001[①]，即在 0.001 这个水平上，结果 $X^2\approx 284>10.83$ 是统计显著的（statistically significant）。如果两个事件互相依赖，那么词项的出现也会使某个类别的出现更有可能或更不可能，因此它适合于作为特征被选出来。这就是 χ^2 特征选择方法的基本原理。

一个算术上更简单的 X^2 计算公式如下：

$$X^2(\mathrm{D},t,c)=\frac{(N_{11}+N_{10}+N_{01}+N_{00})\times(N_{11}N_{00}-N_{10}N_{01})^2}{(N_{11}+N_{01})(N_{11}+N_{10})(N_{10}+N_{00})(N_{01}+N_{00})}。\qquad(13\text{-}19)$$

这个公式实际上与公式（13-18）是等价的（参考习题 13-14）。

表13-6　自由度为1的 χ^2 分布的临界值。比如，如果两个事件独立，那么$P(X^2>6.63)<0.01$，也就是说如果$X^2>6.63$，那么有99%的置信度来拒绝两个事件的独立性假设

P	χ^2 临界值
0.1	2.71
0.05	3.84
0.01	6.63
0.005	7.88
0.001	10.83

对 χ^2 特征选择方法的讨论

从统计的观点来看，χ^2 特征选择方法存在着很多问题。对于自由度为 1 的测试来说，应该要使用一个称为 Yates 修正的技术（参见 13.7 节），该技术会导致很难达到统计显著性。另外，任何时候一个统计测试方法多次使用的话，出现至少一次错误的可能性就会增大。如果 1 000 个假设被拒绝，而每次拒绝的错误率是 0.05 的话，那么平均来说 1 000 次测试中会发生 50 次错误。然而，在文本分类当中，特征集合上加入或去除几个词项一般都无关紧要。实际上，特征的相对重要性比较关键。只要 χ^2 仅仅被用作计算词项的作用并排序，而不是对变量的独立性和非独立性进行判定，那么就不用过度考虑其是否严格满足统计论的要求。

13.5.3　基于频率的特征选择方法

① 我们能够这样推理的原因在于：如果两个事件是独立的，那么 $X^2\sim\chi^2$，其中 χ^2 表示 χ^2 分布，参见 Rice (2006)。

第 3 种特征选择方法是基于频率的特征选择方法（frequency-based feature selection），即选择那些在类别中频率较高的词项作为特征。这里的频率可以定义为文档频率（类别 c 中包含某个词项 t 的文档数目）或文档集频率（c 类别所有文档中 t 出现的总次数）。文档频率更适合于贝努利模型，而文档集频率更适合于多项式模型。

基于频率的方法会选择那些不包含类别相关的特定信息的高频词项，如一周中的每一天（Monday, Tuesday,…）等等，这些词项在新闻文本的多个类别中都会出现。当选出数千特征后，基于频率的特征选择方法往往会取得很好的效果。因此，如果能够接受次优分类精度的话，基于频率的方法可以作为更复杂方法的替代品使用。然而，图 13-8 所示的例子中，基于频率方法的分类效果大大低于基于互信息和 χ^2 的方法，因此在这个例子中它并不适用。

13.5.4 多类问题的特征选择方法

在一个拥有大量分类器的实际运行系统当中，往往需要选择单个特征集，而不是对每个分类器选择自己的特征集。一种处理方法是基于一张 $n\times2$ 的表格来计算 X^2 统计量，表格中每列分别表示词项出现或不出现，而每行对应一个类别。最后我们可以像前面一样，选择 X^2 得分最高的 k 个词项。

更普遍的做法是，首先基于二类分类问题（c 和 $\overline{c}$）对每个类计算特征选择指标，然后将这些指标进行组合。一种组合的方法是对每个特征计算一个单一指标，比如，对 $A(t,c)$ 值基于类别求平均，然后选择得分最高的 k 个特征。另外一种组合方法是，对 n 个分类器中每个分类器都选出 k/n 个特征，然后将所有选出的特征都放到全局特征集中去。

相对于 n 个分类器中每个分类器都可能有不同 k 值的特征集而言，对一个系统采用统一的 k 个特征往往会降低分类的精度。尽管如此，由于后者采用了统一文档表示，能带来分类效率的提升，所以在分类效果上有些损失也是值得的。

13.5.5 不同特征选择方法的比较

MI 和 χ^2 是完全不同的两种特征选择方法。即使词项 t 几乎不携带任何有关文档归属类别 c 的信息，t 和 c 的独立性假设有时也可能在置信度很高的情况下被拒绝。对于罕见词项尤其如此。如果某个词项仅在文档集的 *poultry* 类中出现了一次，那么这就具有统计显著性。然而，根据信息论，一次出现所携带的信息量是不够的。由于 χ^2 基于显著统计性来选择特征，因此它会比 MI 选出更多的罕见词项，而这些词项对于分类是不太可靠的。当然，MI 也不一定就能选出使得分类精度最大化的词项。

尽管MI和 χ^2 有很多不同之处，基于两者的分类精度看上去并没有系统上的太大不同。在很多文本分类问题中，只有很少的强指示特征（简称强特征）[①]，大部分都是弱指示特征（简称弱特征）。只要所有的强特征和很多弱特征被选出，那么分类的期望精度就不错。而上述两种

① 指那些对确定类别具有很强导向性的特征。——译者注

特征选择方法都能做到这一点。

图 13-8 基于多项式模型比较了 MI 和 χ^2 特征选择方法，两者的最佳效果也基本相当。对于 χ^2 特征选择方法，最优值来得稍迟一些（在选择 300 个特征时），这可能是因为 χ^2 开始选择的具有统计显著性的罕见词没有覆盖类别中的所有文档。然而，在这之后的特征选择区间（100～300 个特征）上，χ^2 表现出比 MI 更好的效果。

不论是 MI、χ^2 还是基于频率的方法，都是基于贪心的策略。它们所选择的特征中，相对于前面选出的特征，后面选出的特征可能并没有提供新的增量信息。在图 13-7 中，kong 在特征选择中排名第 7，但是它和前面选出的 hong 高度相关，因此它是冗余信息。尽管这种冗余会对分类精度造成负面影响，非贪心策略（参考 13.7 节）由于其计算开销很大而很少在文本分类中使用。

? 习题 **13-5**　考虑 Reuters-RCV1 语料前 100 000 篇文档中 4 个词项在类别 *coffee* 中的出现频率。

词项	N_{00}	N_{01}	N_{10}	N_{11}
brail	98 012	102	1 835	51
council	96 332	133	3 525	20
producers	98 524	119	1 118	34
roasted	99 824	143	23	10

根据(i) χ^2 (ii) 互信息及 (iii) 频率的值，从上述 4 个词项中选出 2 个词项。

13.6　文本分类的评价

历史上，经典的 Reuters-21578 语料库是文本分类评价的基准测试语料。该语料库包含 21 578 篇新闻，最早是由 Carnegie Group, Inc.和 Reuters，Ltol.在开发 CONSTRUE 文本分类系统的过程中收集和标注的。Reuters-21578 远远小于我们在第 4 章所讨论的 Reuters-RCV1 语料，当然前者出现的时间也要早很多。它将新闻分配到 118 个主题类当中去，某篇文章可能同时属于多个类，也可能不属于任何一个类，当然大多数情况下，一篇文章属于一个类（至少属于一个类的文档的所属类别的平均数是 1.24）。解决这种多类问题的常规方法是学习到 118 个两类问题的分类器，对于每个类 c 都有一个分类器，它将文档分到 c 类或者它的补类 $\bar{c}$ 中去。

对于上面的每个分类器，我们都可以计算其召回率、正确率和精确率。近年来，大家往往采用 Reuters-21578 语料库的一个称为 ModeApte split 的版本，它仅仅包括那些经过人工浏览和评判的文档，包括 9 603 篇训练文档和 3 299 篇测试文档。不同类别中的文档分布很不均衡，有些研究工作仅仅对其中最大的 10 个类别进行评估。这些类别的相关统计数字列在表 13-7 中。Reuters-21578 语料中一个典型的文档及其主题的例子参见图 13-9。

表13-7　Reuters-21578语料中10个最大类的训练和测试文档数目

类别	训练文档数目	测试文档数目	类别	训练文档数目	测试文档数目
earn	2 877	1 087	trade	369	119

acquisitions	1 650	179	interest	347	131
money-fx	538	179	ship	197	89
grain	433	149	wheat	212	71
crude	389	189	corn	182	56

在 13.1 节中，我们指出文本分类的目标是使得测试数据上的分类错误率最小。分类错误率等于 1 减去精确率，其中精确率等于所有判断中正确的比率。这个指标在 8.3 节中曾经介绍过，当某类别上的文档数目占所有文档数目的百分比不是太低时，比如 10%到 20%或者更高，这个指标是适用的。但是正如我们在 8.3 节所讨论的那样，对于小类来说，精确率不是一个好的衡量指标。这是因为绝大部分文档都不属于该类，一个将所有文档都不归于该类的分类器会获得很高的精确率，但是这显然不符合分类的宗旨。具体来说，该分类器对于只有 1%正例的小类来说能获得 99%的精确率。因此，对于小类来说，正确率、召回率和 F_1 值是更好的衡量指标。

我们使用效果（effectiveness）这个术语来统称那些分类结果质量的评价指标，包括正确率、召回率和 F_1 值。同时，本书中性能（performance）这个术语主要指的是分类或 IR 系统的计算效率。然而，许多研究使用 performance 这个术语时指的也是分类的效果而不是效率。

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN"
CGISPLIT="TRAINING-SET" OLDID="12981" NEWID="798">
<DATE> 2-MAR-1987 16:51:43.42</DATE>
<TOPICS><D>livestock</D><D>hog</D></TOPICS>
<TITLE>AMERICAN PORK CONGRESS KICKS OFF TOMORROW</TITLE>
<DATELINE> CHICAGO, March 2 - </DATELINE><BODY>The American Pork
Congress kicks off tomorrow, March 3, in Indianapolis with 160
of the nations pork producers from 44 member states determining
industry positions on a number of issues, according to the
National Pork Producers Council, NPPC.
Delegates to the three day Congress will be considering 26
resolutions concerning various issues, including the future
direction of farm policy and the tax law as it applies to the
agriculture sector. The delegates will also debate whether to
endorse concepts of a national PRV (pseudorabies virus) control
and eradication program, the NPPC said. A large
trade show, in conjunction with the congress, will feature
the latest in technology in all areas of the industry, the NPPC
added. Reuter
&#3;</BODY></TEXT></REUTERS>
```

图 13-9 Reuters-21578 语料中的一篇文档

当对具有多个分类器的文档集（如包含 118 个类别的 Reuters-21578 语料）进行处理时，我们往往需要计算出一个融合了每个分类器指标的综合指标。为实现这个目的，通常有宏平均和微平均两种做法：其中宏平均（macroaveraging）是在类别之间求平均值，而微平均（microaveraging）则是将每篇文档在每个类别上的判定放入一个缓冲池，然后基于这个缓冲的

列联表（13-8 中简称缓冲表）计算效果指标。表 13-8 给出了一个例子。

表13-8　宏平均和微平均的计算

类别1			类别2			缓冲表		
	实际 yes	实际 no		实际 yes	实际 no		实际 yes	实际 no
判定yes	10	10	判定 yes	90	10	判定yes	100	20
判定 no	10	970	判定 no	10	890	判定 no	20	1860

注：“实际”表示实际上属于该类，“判定”表示的是分类器的判定情况。下例中，宏平均正确率为$[10/(10+10)+90/(10+90)]/2=(0.5+0.9)/2=0.7$，而微平均正确率为 $100/(100+20)\approx 0.83$。

宏平均和微平均的计算结果可能会相差很大。宏平均对每个类别同等对待，而微平均则对每篇文档的判定结果同等对待。由于 F_1 值忽略判断正确的负例（上表中每个小表的右下角），所以它的大小主要由判断正确的正例数目所决定，所以在微平均计算中大类起支配作用。在表 13-8 的例子中，系统的微平均正确率(0.83)更接近 c_2 类的正确率(0.9)，而与 c_1 类的正确率(0.5)相差较大，这是因为 c_2 的大小是 c_1 的 5 倍。因此，微平均实际上是文档集中大类上的一个效果度量指标。如果要度量小类上的效果，往往需要计算宏平均指标。

在两类问题中（参见 14.5 节），微平均 F_1 值实际上就是精确率（参考习题 13-6）。

表 13-9 中给出了 NB 分类器在 Reuters-21578 的 ModApte split 版本下的微平均和宏平均指标。为了能对 NB 的相对效果有所认识，这里将它和线性 SVM 分类器（表中的最后一列，参见第 15 章）进行了比较，SVM 分类器是效果最好的分类器之一，当然它的训练复杂度也高于 NB。NB 的微平均 F_1 值是 80%，比 SVM 的值（89%）小 0.09，相对低大约 10%的比例（参考标记为“micro-avg-L（90 classes）”的那行）。因此，尽管 NB 简单高效，但由此带来的效果降低幅度却不大，这一点有些出乎意料。但是，在一些训练集仅包含数十篇正例的小类上，NB 的效果就要差很多。其宏平均 F_1 值比 SVM 小 13%，相对低大约 22%的比例（参考标记为“macro-avg（90classes）”的那行）。

表13-9　Reuters-21578语料上文本分类的F_1值（采用百分比表示）

(a)		NB	Rocchio	KNN		SVM
	micro-avg-L(90 classes)	80	85	86		89
	macro-avg(90 classes)	47	59	60		60
(b)		NB	Rocchio	KNN	trees	SVM
	earn	96	93	97	98	98
	acq	88	65	92	90	94
	money-fx	57	47	78	66	75
	grain	79	68	82	85	95
	crude	80	70	86	85	89
	trade	64	65	77	73	76
	interest	65	63	74	67	78

ship	85	49	79	74	86
wheat	70	69	77	93	92
corn	65	48	78	92	90
micro-avg(rop 10)	82	65	82	88	92
micro-avg-D(118 classes)	75	62	n/a	n/a	87

注：这些结果来自 Li 和 Yang (2003) (a), Joachims (1998) (b: kNN) 及 Dumais 等人 (1998) (b: NB, Rocchio, trees, SVM)。

上表也给出了除 NB 之外其他一些分类器的结果，包括 Rocchio 和 kNN 分类器，这些分类器将在后面作介绍。另外，我们还给出了本书没有介绍的另外一种重要的分类方法——决策树（decision tree）方法的分类效果。上表的下半部分表明：采用不同分类方法时类别之间的效果差异还是很大的。比如，在 *ship* 类上，NB 的效果要优于 kNN，但是在 *money-fx* 类上，NB 的效果却要差很多。

将上表中的(a)部分和(b)部分进行比较就会发现，不同参考文献中的结果差异极大。造成这种现象的部分原因是 (b)部分的数字采用的是 118 个类上等值点的平均值，而(a)中的结果是不知道测试集相关知识的情况下在 90 个类上计算出的真实 F_1 值。遗憾的是，在文本分类中对不同方法的结果进行比较时常常会出现这种不一致的情况。实验设置或者评价方法的经常性不同加大了结果解释的复杂性。

这里给出的结果再加上其他地方的一些结果都表明，当训练数据和测试数据满足独立同分布（independent and identically distributed，简称 i.i.d）时，NB 的平均效果无法和诸如 SVM 的分类器相提并论。这些数据分布一致，并满足统计抽样的所有优点。但是，当面对真实数据时，这种差别并不明显甚至会出现相反的结果。此时，训练数据往往是从即将应用分类器的数据集上抽取而得到的，而数据在本质上会随时间发生漂移而不会始终保持静态不变（可以参考我们在前面提到的概念漂移），同时数据中还肯定会存在错误。很多实践者都有过这种体验，即对某个问题而言，很难建立一个在效果上能够稳定地超过 NB 的更好的分类器。

从表 13-9 也可以得到另外一条结论，即尽管大多数学者都相信 SVM 优于 kNN、而 kNN 又优于 NB，但是分类器的实际好坏根本上还是取决于类别、文档集及实验的设置。在文本分类中，往往需要知道更多的信息而不只是知道使用了哪些机器学习方法，关于这一点，我们在 15.3 节中还会进一步讨论。

当采用类似表 13-9 的方法进行分类效果评估时，必须要严格区分训练集和测试集。如果使用从测试集上获得的信息（比如基于测试集的结果选择某个词项，该词项对于测试集的预测大有帮助，当然此时并不要求它在训练集上也能取得同样的效果），那么我们可以很容易地对测试集中的文档进行正确分类。一个更投机取巧的方法就是，在测试集中遍历各种可能的参数取值（比如特征的选择数目）并取使得分类结果最优的参数值。通常在对真实应用当中的新数据进行分类时，其分类效果往往会大大低于经过调优的测试集上的分类效果。在 8.1 中介绍 ad hoc 检索时，我们已讨论过这个问题。

在一个不掺假的统计文本分类实验当中，在开发文本分类系统的时候，我们不应该在测试

集上运行任何程序，甚至都不能看一眼测试集。当然，我们可以基于开发集（development set）来对文本分类方法进行测试 ①。当这个集合用于寻找最优的参数（如特征的选择数目）时，也称为留存数据（held-out data）。在训练集中除去留存数据后的其他数据上采用不同参数进行训练，然后选择使得留存数据上分类效果最优的参数值。理想情况下，当所有的参数及分类方法已经确定时，才能在测试集上运行一遍程序并报告结果。采用这种做法的情况下，由于在分类器开发过程当中没有使用任何有关测试集的知识，因此实验的结果基本上可以代表实际中的真实性能。

然而，这种理想的情况在实际中往往很少遇到，研究人员往往在数年间基于同一测试集对多个系统进行评价。但是，在开发分类器时尽量不要看测试集并且要尽可能少地在上面运行程序，这一点仍然相当重要。初学者往往违反这条规则，由于他们通常通过显式地在测试集上反复运行并调节到最优参数，所以他们获得的结果往往缺乏可信度。

? 习题 **13-6** [**]　假定测试集中的每篇文档都只属于一个类别，并且分类器也只给一篇文档赋予一个类别。这类问题称为单标签问题（参见 14.5 节）。请证明在单标签问题中：

(i) 假正例（false positive）的个数等于假负例（false negative）的个数；

(ii) 微平均 F_1 值等于精确率。

习题 **13-7**　图 13-2 中的类先验概率按照类别中的文档数目（而不是词条数目）占所有文档数目的比例计算，为什么？

习题 **13-8**　图 13-2 中的 APPLYMULTINOMIALNB 函数的时间复杂度是 $\Theta(L_a + |\mathbb{V}|L_a)$，如何对之进行改进使得它的时间复杂度降低到 $\Theta(L_a + |\mathbb{V}|M_a)$？

习题 **13-9**　基于表 13-10 中的数据，进行如下计算：

(i) 估计多项式 NB 分类器的参数；

(ii) 将(i)中的分类器应用到测试集；

(iii) 估计贝努利 NB 分类器的参数；

(iv) 将(iii)中分类器应用到测试集。

不要估计那些在对测试集文档进行分类时不会用到的参数值。

表13-10　用于参数估计的数据

	文档ID	文档中的词	属于*c=China*类?
训练集	1	Taipei Taiwan	yes
	2	Macao Taiwan Shanghai	yes
	3	Japan Sapporo	no
	4	Sapporo Osaka Taiwan	no
测试集	5	Taiwan Taiwan Sapporo	?

① 即将训练集分成两部分，一部分仍然用于训练，而另一部分称为开发集，用于测试。分类方法或参数可以在开发集上调优。由于没有利用真正的测试集，因此这种做法是允许的。通常还有一种做法是将训练集分成 k 等份，其中 k-1 份用于训练，剩余那份用于测试。则该过程可以循环 k 次，每次选择不同的那份文档用于测试。这种做法称为 k 交叉验证（k cross validation）。——译者注

习题 **13-10** 比如有一个任务要将单词分成英语类或非英语类。这些单词的产生来自如下分布：

事件	词语	英语？	概率
1	ozb	no	4/9
2	uzu	no	4/9
3	zoo	yes	1/18
4	bun	yes	1/18

(i) 计算多项式 NB 分类器的参数（含先验概率和条件概率），分类器使用字母 b、n、o、u 和 z 作为特征。假设训练集能完美反应表中的概率分布，使用多项式分类器中常用的特征独立性假设。在计算参数时使用平滑方法，零概率平滑成 0.01，而非零概率不做改变。（这种简单的平滑方法显然会使 $P(A)+P(\bar{A})>1$），但是这里不需要解决这个问题。）

(ii) 上述分类器对单词 zoo 的分类结果是什么？

(iii) 使用(i)中的多项式分类器对 zoo 进行分类，但是此时不使用位置独立性假设，也就是说，对单词中的每个位置上都要估计独立的参数，当然你只需要估计对 zoo 分类时所需要的参数。

习题 **13-11** 如果 t 和 c 完全独立，$I(U_t;C_c)$ 及 $X^2(\mathfrak{D}, t, c)$的值各是多少？如果 t 和 c 完全依赖，上述值又分别是多少？

习题 **13-12** 公式（13-16)中的特征选择方法适合于贝努利模型，为什么？如果要让它适用于多项式模型，应该如何修改？

习题 **13-13** 特征选择也可以采用信息增益（information gain）的方法，它的定义如下：

$$\mathrm{IG}(\mathrm{D},t,c)=H(p_{\mathrm{D}})-\sum_{x\in\{\mathrm{D}_{t+},\mathrm{D}_{t-}\}}\frac{|x|}{|\mathrm{D}|}H(p_x)\ 。$$

其中，H 是熵函数，$\mathfrak{D}$是训练集，$\mathfrak{D}_{t+}$、$\mathfrak{D}_{t-}$分别表示包含 t 和不包含 t 的文档子集。p_A 表示文档集 A 中的类别分布，比如 $p_A(c)=0.25$，$p_A(\bar{c})=0.75$，表示 A 中 1/4 的文档属于类别 c。试证明互信息和信息增益等价。

习题 **13-14** 证明 χ^2 计算的两个公式（13-18）和（13-19）等价。

习题 **13-15** 在例 13-4 中，我们假定$|N_{11}-E_{11}|=|N_{10}-E_{10}|=|N_{01}-E_{01}|=|N_{00}-E_{00}|$，试证明该假设在通常情况下成立。

习题 **13-16** χ^2 和互信息并不区分正相关特征和负相关特征。由于大多数好的文本分类特征都是正相关（比如，它们在 c 中的出现次数大大多于在 $\bar{c}$ 中的出现次数），我们可能要显式地剔除负相关特征，请问如何实现这一点？

13.7 参考文献及补充读物

对统计分类和机器学习的一般性介绍可以参考（Hastie 等人 2001）、（Mitchell 1997）及（Duda 等人 2000），它们包括了很多我们没有介绍的重要方法（如决策树和 boosting 方法）。一个有关文本分类方法及结果的全面性介绍可以参考（Sebastiani 2002）。Manning and Schütze（1999）的第 16 章也给出了包含决策树、感知机及最大熵模型在内的文本分类方法的简要介绍。有关具有超线性时间复杂度比 NB 更精确的学习方法的更多信息参考（Perkins 等人 2003）和

（Joachims 2006a）。

Maron 和 Kuhns（1960）给出了一个最早的 NB 分类器。Lewis（1998）集中关注 NB 分类器的历史。贝努利模型和多项式模型及它们在不同文档集上的精确度的讨论参见 McCallum 和 Nigam（1998）。Eyheramendy 等人（2003）给出了其他的 NB 模型。Domingos 和 Pazzani（1997）、Friedman（1997）及 Hand 和 Yu（2001）分析了 NB 在参数估计很差的情况下效果却不错的原因。Domingos 和 Pazzani（1997）还讨论了当数据满足独立性假设时 NB 的最优性。Pavlov 等人（2004）提出了一个改进的文档表示方法，它能够部分解决独立性假设不恰当的问题。Bennett（2000）将 NB 的概率估计接近 0 或者 1 这种趋势归结于文档长度的影响。Ng 和 Jordan（2001）的工作表明，由于 NB 很快能够达到它的最优错误率，所以它有时候（尽管这种情况不多）会优于判别式方法。本章中的基本 NB 模型可以调节出更好的效果（参考 Rennie 等人 2003；Kołcz 和 Yih 2007）。有关概念漂移问题以及现有最好方法在实际中并不总能取得很好结果的原因在 Forman（2006）和 Hand（2006）中讨论。

将互信息和 χ^2 用于文本分类特征选择的早期工作可以分别参见 Lewis 和 Ringuette（1994）和 Schätze 等人（1995）。Yang 和 Pedersen（1997）总结了多个特征选择方法及它们对分类效果的影响。他们发现点互信息（pointwise mutual information）的效果比不上其它方法。Yang 和 Pedersen 将公式（13-16）所示的期望互信息称为信息增益（参考习题 13-13）。（Snedecor 和 Cochran 1989）是有关 χ^2 统计测试的一本较好的参考书，书中还包括对 2×2 表格的 Yates 修正技术。Dunning（1993）讨论了当数字较小时 χ^2 测试的问题。非贪心的特征选择技术在 Hastie 等人（2001）中有介绍。Cohen（1995）讨论了多次使用统计测试带来的缺陷以及如何避免这些缺陷。Forman（2004）在多个分类器上对不同的特征选择方法进行了评价。

David D. Lewis 基于 Apté 等人（1994）的工作在 www.daviddlewis.com/resources/ testcollections/ reuters21578/readme.txt 上定义了 ModApte split 版本的含义。Lewis（1995）描述了文本分类评估中的效用指标（utility measure）。Yang 和 Liu（1999）在文本分类的评估中开发了统计显著性测试方法。

Lewis 等人（2004）发现在 Reuters-RCV1 语料上，SVM（参见第 15 章）的性能要优于 kNN 及 Rocchio 方法（参见第 14 章）。

第 14 章

基于向量空间模型的文本分类

在 NB 分类当中，文档表示成词项序列或者一个布尔向量$\{e_1, \ldots, e_M\} \in \{0, 1\}^{|V|}$。本章将采用一种不同的文档表示方法。具体地，我们将采用第 6 章介绍的向量空间模型来表示文档。向量空间模型将每篇文档表示成实数型分量所构成的向量，每个分量对应一个词项，并往往采用 tf-idf 权重来计算。因此，在这种表示下，分类函数 γ 的输入文档空间实际上是$\mathbb{R}^{|V|}$。本章主要介绍基于这种实数向量表示的文本分类方法。

利用向量空间模型进行文本分类的思路主要基于邻近假设（contiguity hypothesis）。

邻近假设：同一类的文档会构成一个邻近区域，而不同类的邻近区域之间是互不重叠的。

很多种分类任务，尤其是我们在第 13 章见到的那种分类任务，可以通过词的出现模式来区别不同的类别。比如，*China* 类中的文档倾向于在诸如 Chinese、Beijing 或 Mao 之类的词所对应的维度上取较大值，而 *UK* 类中的文档在诸如 London、British 或 Queen 之类的词上取较大值。因此，上述两个类会形成截然不同的邻近区域（见图 14-1），这样就可以画出两个邻近区域的边界从而对新文档进行分类。上述过程的具体实现是本章的主题。

文档集是否会映射成邻近区域取决于在文档表示中的很多选项，例如权重计算方法、停用词表等。为了理解文档表示的重要性，考虑两个类“一组人写的文档”（*written by a group*）和“一个人写的文档”（*written by a single person*）。第一人称代词 I 的出现频度可以作为第二个类的判定依据。然而，如果我们使用了停用词表的话，这些代词就有可能已经被删除。也就是说，如果选择了不合适的文档表示方法，那么邻近假设将不成立，此时基于向量空间的分类方法也不可能成功。

基于同样的考虑，在进行文档表示时往往要考虑带权重的表示方法，特别是在第 6 章和第 7 章所介绍的基于长度归一化的 tf-idf 权重表示方法也会在这儿使用。比如在一篇文档中，出现 5 次的词项显然比只出现 1 次的词项具有更高的权重。但是，如果赋予前者的权重正好就是后者的 5 倍，那么就会过度强调前者的作用。无权重计算和无归一化的数字将不会在基于向量空间模型的分类方法中使用。

本章将介绍两种基于向量空间模型的分类方法，它们分别是 Rocchio 方法和 kNN 方法。前者将在 14.2 节进行介绍，它基于质心或原型(*prototype*)将整个向量空间划分成多个区域。每个质心或原型代表一类，通常采用该类中所有文档向量的平均来计算。Rocchio 方法非常简单、时空消耗也小，但是当某类的内部文档并不近似分布在半径相近的球体之内时，其分类精度并不高。

kNN 或 k 近邻分类方法将在 14.3 节中介绍。它将 k 个最近邻文档所属的主类别赋给测试文档。kNN 不需要显式的训练过程，它可以直接使用未处理过的训练集进行分类。相对于其他分类方法，kNN 的分类效率较低。但如果训练集很大，kNN 在处理非球体型类别以及其他复杂的类别时优于 Rocchio 方法。

14.4 节将介绍线性分类器，它是指基于特征的简单线性组合就可以对文档进行分类的分类器，许多文本分类器都可以看成是线性分类器。这种分类器通过线性决策超平面（decision hyperplane）将整个特征空间划分成多个区域，具体的划分细节将在后面详细介绍。基于 14.6 节中介绍的偏差–方差折衷准则，更复杂的非线性分类模型并不会系统地优于线性模型。非线性模型需要在有限的训练数据上拟合出更多的参数，因此对于小规模的有噪音数据集则更有可能犯错误。

当将二类分类器应用到多类问题上时，通常有两种任务，第一种任务称为单标签（one-of）任务，该任务中要求每篇文档只能分到多个互斥类别当中的某一个类别中去。第二种任务称为多标签（any-of）任务，即一篇文档可以分到任意数目的类别当中去。我们将在 14.5 节介绍后一种任务。二类分类器能够解决多标签问题，通过组合也可以解决单标签问题。

14.1 文档表示及向量空间中的关联度计算

同第 6 章一样，本章将文档表示成 $\mathbb{R}^{|V|}$ 上的向量。为了说明在向量分类当中文档向量的特点，我们将这些文档向量表示成平面上的点（参见图 14-1）。现实当中，文档向量都是指向超球体表面的经长度归一化后的单位向量。我们可以将图 14-1 中的二维平面看成是图 14-2 中（超）球体表面在平面上的投影。如果只考虑球体表面的小块区域并且采用恰当的投影方法（参考习题 14-1）的话，那么球体表面两点之间的距离与这两点在投影平面上投影之间的距离基本上相等。

很多基于向量空间的分类器在分类决策时用到距离的概念，比如在 kNN 分类当中要计算最近邻时就要计算距离。本章当中我们主要使用欧氏距离（Euclidean distance）来计算。在前面我们就注意到（参见习题 6-18），余弦相似度计算方法和长度归一化向量的欧氏距离计算之间具有直接的对应关系（即最后得到的结果排名一致）。在基于向量空间模型的分类当中，文档之间的关联度无论是采用相似度来计算还是采用距离来计算一般都无关紧要。

然而，除文档之外，多个向量的质心或者平均向量在基于向量空间的分类当中也扮演着十分重要的角色。质心向量并不是长度归一化向量[①]。对于未归一化的向量，内积、余弦相似度和欧氏距离通常会有不同的表现（参考习题 14-6）。在计算文档和质心之间的相似度时，我们主要关注较小的局部区域，区域越小，以上 3 种计算方法表现也越相似。

① 如果每个向量根据向量长度进行了归一化，那么它们的和向量显然没有做向量长度归一化。——译者注

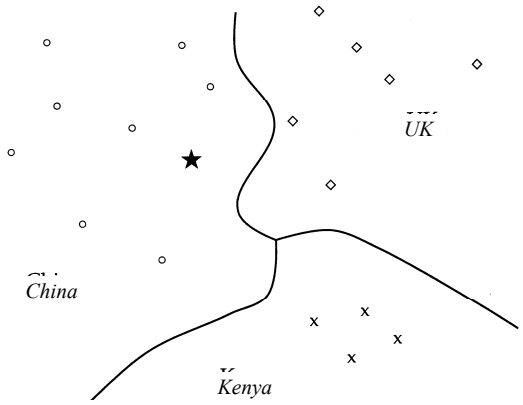


图 14-1　基于向量空间的方法将所有点划分成 3 类的例子

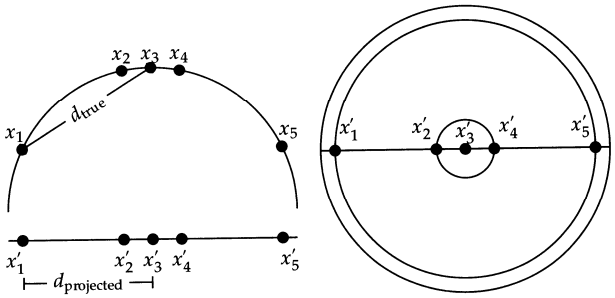


图 14-2　能够保持距离的单位球面内小块区域的映射示意图。左图：从二维空间的半圆映射到一维直线上。点 x_1、x_2、x_3、x_4、x_5 的 X 轴坐标分别是−0.9、−0.2、0、0.2 和 0.9，距离 $|x_2x_3| \approx 0.201$，和 $|x'_2x'_3| = 0.2$ 只有 0.5%的差异，但是当对较大的区域进行投影的话，比如 $|x_1x_3| / |x'_1x'_3| = d_{\text{true}}/d_{\text{projected}} \approx 1.06/0.9 \approx 1.18$ 却会产生较大的差异（18%）。右图：相应的从三维的半球面到二维平面上的投影

习题 14-1　对于较小的区域而言，球表面的距离可以通过其投影距离来近似计算，由于对一个比较小的角 α 来说有 $\alpha \approx \sin\alpha$，因此上述计算的结果较准确。那么 α 的角度多大时其失真度(distortion) $\alpha/\sin(\alpha)$ 分别等于：

(i) 1.01, (ii) 1.05 及 (iii) 1.1?

14.2　Rocchio 分类方法

图 14-1 在二维平面上给出了 3 个类 *China*、*UK* 及 *Kenya* 的分布情况，在这 3 个类中文档分别以圆形点、菱形点和叉号标识。图中的边界称为决策边界（decision boundary），用来将平面分成 3 个类，但是很显然这种决策边界可以是任意形状。对新文档（图中用五角星来表示）进行分类时，我们根据其出现的区域将它分到某一类当中去。对于图 14-1 中的新文档来说，它属

于 *China* 类。在基于向量空间的分类当中，我们的目标是设计出能够计算出“好”边界的算法，这里的“好”指的是对训练集中没有出现的新文档也能取得很高的分类精确率。

在基于向量空间的分类中，我们必须要做的一项主要工作是定义类别之间的边界，因为它们决定了分类的决策结果。或许这当中最出名的方法或许是 Rocchio 分类（Rocchio classification）方法，它利用质心（centroid）来定义分类边界。一个类别 c 的质心可以通过类中文档向量的平均向量或者质心向量来计算，即

$$\vec{\mu}(c)=\frac{1}{|D_c|}\sum_{d\in D_c}\vec{v}(d)\ 。 \qquad (14\text{-}1)$$

其中，D_c 是文档集 λ 中属于类别 c 的文档子集：$D_c=\{d: <d,c>\in \lambda\}$。这里将归一化的文档向量记为 $\vec{v}(d)$（参考公式(6-11)）。在图 14-3 中，3 个质心用实心圆点标记。

在 Rocchio 分类当中，两类的边界由那些到两个类质心等距的点集组成。例如，在图 14-3 中，有 $|a_1|=|a_2|$、$|b_1|=|b_2|$ 和 $|c_1|=|c_2|$。二维平面上的一条直线在 M 维空间中可以推广成一个超平面，该超平面上的点 $\vec{x}$ 满足

$$\vec{w}^T\vec{x}=b。 \qquad (14\text{-}2)$$

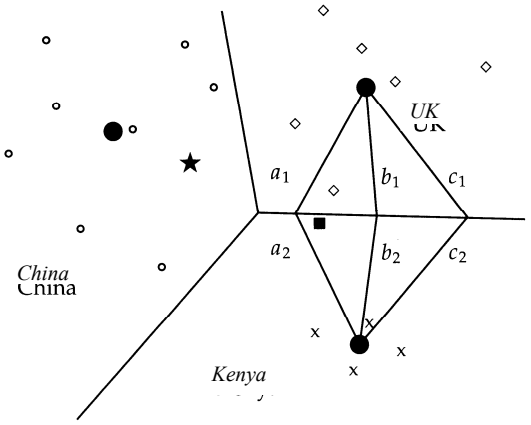


图 14-3 Rocchio 分类方法示意图

其中 $\vec{w}$ 称为超平面上的 M 维法向量（normal vector）[①]，b 是一个常数。这个超平面的定义中也涵盖了直线（二维空间中的任一直线都可以定义为 $w_1x_1+w_2x_2=b$）和二维平面（三维空间中的任一二维平面都可以定义为 $w_1x_1+w_2x_2+w_3x_3=b$）。一条直线将二维平面一分为二，一个平面将三维空间一分为二，同样，超平面将更高维的空间一分为二。

因此，Rocchio 分类方法中的类边界是超平面。Rocchio 方法中的分类规则是，按照点所属的区域将之划分到相应的类别中去。也就是说，一个点距离哪个质心向量 $\vec{\mu}(c)$ 最近，就将它分配到其对应的类别中去。考虑图 14-3 中的新文档例子（以五角星标记的那个点），由于它属于 *China* 类所在的区域，所以 Rocchio 算法将它归于 *China* 类。在图 14-4 中，我们给出了 Rocchio

① 在基本的线性代数中有 $\vec{v}\cdot\vec{w}=\vec{v}^T\vec{w}$，也就是说 $\vec{v}$ 和 $\vec{w}$ 的内积等于 $\vec{v}$ 的转置矩阵和 $\vec{w}$ 的乘积。

方法的伪代码。

```
TrainRocchio(ℂ, 𝔻)
1  for each c_j ∈ ℂ
2  do D_j ← {d : ⟨d, c_j⟩ ∈ 𝔻}
3     μ⃗_j ← (1/|D_j|) Σ_{d∈D_j} v⃗(d)
4  return {μ⃗_1, ..., μ⃗_J}

ApplyRocchio({μ⃗_1, ..., μ⃗_J}, d)
1  return arg min_j |μ⃗_j − v⃗(d)|
```

图 14-4　Rocchio 分类方法中的训练和测试

例 **14-1**　表 14-1 给出了表 13-1 中的 5 篇文档的 tf-idf 向量表示，具体的权重计算方法是，如果 $tf_{t,d} > 0$，则权重为$(1 + \ln tf_{t,d}) \ln(4/df_t)$（参考公式(6-14)）。两个类别的质心分别是 $\vec{\mu}(c) = 1/3 \cdot (\vec{d}_1 + \vec{d}_2 + \vec{d}_3)$ 及 $\vec{\mu}(\overline{c}) = 1/1 \cdot (\vec{d}_4)$。测试文档和两个类别质心向量的距离分别是 $|\vec{\mu}(c) - \vec{d}_5| \approx 1.15$ 及 $|\vec{\mu}(\overline{c}) - \vec{d}_5| = 0.0$。因此，Rocchio 方法将 d_5 分到 $\overline{c}$ 类。

该例对应的分类超平面的参数为

$$\vec{w} \approx (0 \ -0.71 \ -0.71 \ 1/3 \ 1/3 \ 1/3)^{\mathrm{T}},$$

$$b = -1/3。$$

表14-1　表13-1中数据对应的文档向量及类别质心向量

向量	词项权重					
	Chinese	Japan	Tokyo	Macao	Bejing	Shanghai
$\vec{d}_1$	0	0	0	0	1.0	0
$\vec{d}_2$	0	0	0	0	0	1.0
$\vec{d}_3$	0	0	0	1.0	0	0
$\vec{d}_4$	0	0.71	0.71	0	0	0
$\vec{d}_5$	0	0.71	0.71	0	0	0
$\vec{\mu}_c$	0	0	0	0.33	0.33	0.33
$\vec{\mu}_{\overline{c}}$	0	0.71	0.71	0	0	0

至于上述参数的计算方法，可以参考习题14-15。很容易验证该超平面能够按照我们的要求区分两类文档：

$$\vec{w}^{\mathrm{T}}\vec{d}_1 \approx 0 \times 0 + (-0.71) \times 0 + (-0.71) \times 0 + 1/3 \times 0 + 1/3 \times 1.0 + 1/3 \times 0 = 1/3 > b$$

同样，对于 $2 \leq i \leq 3$，有 $\vec{w}^{\mathrm{T}}\vec{d}_i > b$，而 $\vec{w}^{\mathrm{T}}\vec{d}_4 = -1 < b$。因此，$c$ 类中所有的文档出现在超平面之上（$\vec{w}^{\mathrm{T}}\vec{d} > b$），而 $\overline{c}$ 类中的所有文档出现在超平面之下（$\vec{w}^{\mathrm{T}}\vec{d} < b$）。

图 14-4 中的类别分配准则是欧氏距离，另外一种方法是采用余弦相似度，即将 d 分配到类别 $c = \arg\max_{c'} \cos(\vec{\mu}(c'), \vec{v}(d))$ 中去。正如 14.1 节讨论的那样，两种类别分配准则有时会产生不同

的分类决策。这里我们采用了基于欧氏距离的 Rocchio 分类方法，这主要是因为它强调了 Rocchio 方法和 *K*–均值聚类方法的紧密联系（参见 16.4 节）。

Rocchio 分类是 Rocchio 相关反馈（参见 9.1.1 节）的一种形式。相关文档的平均值就是相关文档所属类别的质心向量，它对应于 Rocchio 相关反馈公式（参见公式(9-3)）中最重要的组成部分。在文本分类中不存在原始查询，因此 Rocchio 相关反馈公式中的原始查询向量被忽略。另外，Rocchio 分类方法还可以应用到多类问题中，而 Rocchio 相关反馈只区分相关和不相关两类文档。

为了遵循邻近性的要求，Rocchio 分类中的每个类别一定要近似球形，并且它们之间具有相似的球半径。在图 14-3 中，*UK* 和 *Kenya* 两类边界下面的实心正方形所代表的文档可能更适合 *UK* 类，这是因为 *UK* 类比 *Kenya* 类的文档分布更分散。然而，对于 Rocchio 分类方法来说，由于它忽略了类别内部文档的分布细节，而仅仅根据文档和类别质心之间的距离来分类，因此它会把该文档分到 *Kenya* 类中。

球体分布的假设在图 14-5 中也不成立。对于“a”类来说，由于它包含两个簇，因此我们不能用单个原型来描述它。Rocchio 方法常常对这种多模态类别（multimodal class）问题误分类。文本分类中多模态类别的一个例子是更改了国名的国家，比如缅甸，就在 1989 年将国名从 Burma 改成了 Myanmar。在修改名称前后的两个簇之间可能并不接近。在相关反馈中，我们也遇到过多模态问题（参见 9.1.2 节）。

二类问题是另外一种很难满足等半径球体分布假设的例子。绝大部分二类分类器将某个只占据较小区域的类（如 *China* 类）和它的补类区分开，而补类往往很大，并且文档的分布很分散。如果假定这两个类别等半径的话则会出现大量假正例。因此，对于绝大多数二类问题来说需要按照如下方式修改分类规则：

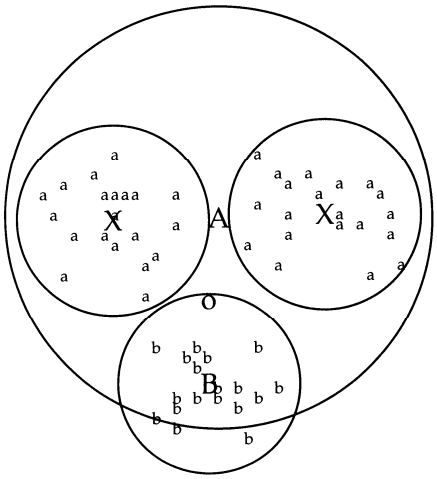


图 14-5 多模态类别“a”由两个不同簇（分别是以 X 为中心的两个小圆）组成。由于“O”更接近“a”的中心 A，因此，Rocchio 分类会将其错分到“a”类

当且仅当 $|\vec{\mu}(c)-\vec{v}(d)|<|\vec{\mu}(\overline{c})-\vec{v}(d)|-b$ 时，将 d 分配到 c 中。

其中 b 是一个正常数。同 Rocchio 相关反馈一样，通常不会使用反例文档的质心质量，因此，上述规则可以简化成：对于某个正常数 b'，有 $|\vec{\mu}(c)-\vec{v}(d)|<b'$。

表 14-2 给出了Rocchio分类的时间复杂度[①]。对所有文档向量求和的时间复杂度是$\Theta(|\lambda|L_{ave})$而不是$\Theta(|\lambda||V|)$，这是因为只需要考虑向量中的非零分量即可。计算每个类别质心向量的时间复杂度是$\Theta(|V|)$。因此，Rocchio分类方法训练的时间复杂度是文档集大小的线性函数（参考习题 13-1），也就是说，Rocchio方法和NB方法在训练上具有相同的时间复杂度。

表 14-2　Rocchio 分类方法的训练和测试时间复杂度

阶段	时间复杂度
训练	$\Theta(\lvert\lambda\rvert L_{ave})+\lvert\amalg\rvert\lvert V\rvert$
分类	$\Theta(L_a+\lvert\amalg\rvert M_a)=\Theta(\amalg)M_a$

注：L_{ave} 是测试文档的平均词条数目，$L_{\amalg}$ 和 $M_{\amalg}$ 分别是词项数目和类别数目。计算类别质心与文档的欧氏距离的时间复杂度是 $\Theta(|\amalg|M_a)$。

下一节将介绍另外一种基于向量空间的分类方法 kNN，它能更好地处理非球体、不连通或其他非规则形状的类别。

习题 14-2[*]　试举例说明，如果采用 Rocchio 分类方法对训练文档分类，那么分类结果有可能会不同于训练集上的结果。

14.3　k 近邻分类器

和 Rocchio 不一样 kNN（k nearest neighbor，k 近邻）方法通过局部信息来确定类别边界。对于 1NN 分类方法，我们将距离某文档最近的文档的类别赋给该文档。而对于 kNN 方法，我们将与测试文档最近的 k 篇文档所属的主类别赋给该文档，其中 k 是一个参数。kNN 的基本依据在于：根据邻近假设，一篇测试文档 d 将和其邻域中的训练文档应该具有相同的类别。

1NN分类器的判别边界是Voronoi剖分（Voronoi tessellation）[②]形成的多个线段的连接（参见图 14-6）。Voronoi剖分会将一个对象集进行划分，得到多个Voronoi网格，每个网格中都包含了所有与本地对象更相近的点。Voronoi剖分会将整个平面分成$|\lambda|$个凸多边形，每个多边形都仅包含其对应的文档（参见图 14-6），而每个凸多边形是在二维空间中通过直线围成的凸区域。

① 这里我们将 Θ(T)记为 Θ (|　|Lave)，并且假设测试文档的长度有界。参见 13.2 节。

② Voronoi 剖分也称为 Theissen 剖分或 Dirichlet 剖分，给定二维平面上的 N 个点，它会将这些点分到 N 个多边形区域中。每个边界为两个点连线的中垂线。——译者注

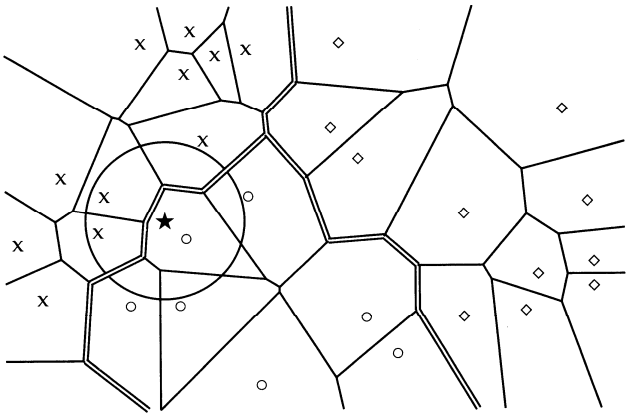


图 14-6 1NN 分类中的 Voronoi 剖分及分类边界(双线表示)。3 个类别分别采用 x、圆圈和菱形表示

对于$k\in$.·̈的一般kNN分类来说，考虑k个最近邻的区域的方法同前面一样。这里会再次得到一个凸多边形，整个空间也会划分为多个凸多边形，每个凸多边形中的k个近邻组成的集合是不变的（参考习题 14-11）[①]。

1NN 分类方法的鲁棒性并不很高。这是因为对每篇测试文档类别的判定仅仅依赖于单个训练文档的类别，而该训练文档可能被误标识或者根本没有代表性。$k>1$ 时的 kNN 分类方法则更具鲁棒性。它将 k 个近邻所属的主类别分配给文档，如果出现多个可能的主类时（多个类在前 k 篇文档中占据的文档数目一样），则随机选择其中一个作为主类。

上述 kNN 分类算法还有一个基于概率的版本，它将属于类别 c 的概率估计为 k 个近邻中属于类别 c 的文档比例。图 14-6 给出了 k=3 时的一个例子。星号所代表的文档属于各个类别的概率分别为：$\hat{P}(\text{圆圈类}\mid\text{文档})=1/3$，$\hat{P}(\text{X类}\mid\text{文档})=2/3$，$\hat{P}(\text{菱形类}\mid\text{文档})=0$。需要指出的是，在 3NN 中，$\hat{P}(\text{圆圈类}\mid\text{文档})=1/3$，而在 1NN 中，$\hat{P}(\text{圆圈类}\mid\text{文档})=1$，因此，3NN 会将该文档分到 X 类，而 1NN 却将该文档分到圆圈类。也就是说，不同 k 值情况下的分类结果有可能不同。

kNN 中 k 的取值往往取决于经验或者分类问题本身的有关知识。k 一般取奇数来减少多个主类同时存在的可能性。k=3 和 k=5 是两组常用的取值，但是，k 也常取 50 到 100 之间的更大的值。另外一种选取 k 值的方法是，取在训练集的留存数据（held-out data）上效果最好的 k 值。

我们也可以将 k 个近邻基于其余弦相似度进行加权。这种情况下，文档 d 属于某个类别 c 的得分计算如下：

$$\text{score}(c,d)=\sum_{d'\in S_k} I_c(d')\cos(\vec{v}(d'),\vec{v}(d))\text{。}$$

其中，S_k 表示的是文档 d '的 k 个近邻文档组成的集合，如果 d '属于类别 c 则 $I_c(d')$=1，否则

① 多边形扩展到高一维空间就是多面体。M 维空间下的多面体是由 $M-1$ 维超平面围成的区域。在 M 维空间中，kNN 的分类边界由多段 $M-1$ 维的超平面组成，这些超平面通过 Voronoi 剖分将训练文档集分成多个凸多面体。将文档按照其 k 个近邻所属的主类别进行分类决策准则在 M=2（Voronoi 剖分成多边形）及 $M>2$（Voronoi 剖分成多面体）的情况下都是一样的。

$I_c(d')=0$。最后将得分最高的类别赋予文档 d'。这种基于相似度加权“ 投票” 的 kNN 方法的精度往往要高于简单的投票方法。比如在 k 篇近邻文档中，如果两个类具有相同的最大近邻数目，那么此时拥有更相似近邻的类别会最后胜出。

图 14-7 概括了 kNN 算法的流程。

```
TRAIN-KNN(ℂ, 𝔻)
1  𝔻′ ← PREPROCESS(𝔻)
2  k ← SELECT-K(ℂ, 𝔻′)
3  return 𝔻′, k

APPLY-KNN(ℂ, 𝔻′, k, d)
1  S_k ← COMPUTENEARESTNEIGHBORS(𝔻′, k, d)
2  for each c_j ∈ ℂ
3  do p_j ← |S_k ∩ c_j|/k
4  return arg max_j p_j
```

图 14-7　kNN 的训练 (包括预处理) 和分类过程。p_j 是概率 $P(c_j|S_k) = P(c_j|d)$的估计值，c_j 表示的是类别 c_j 中的所有文档

例 14-2　表 14-1 中测试文档和 4 篇训练文档的距离分别是 $|\vec{d}_1-\vec{d}_5| = |\vec{d}_2-\vec{d}_5| = |\vec{d}_3-\vec{d}_5| \approx 1.41$，而 $|\vec{d}_4-\vec{d}_5|=0.0$，因此 d_5 的最近邻是 d_4，按照 1NN 进行分类的话，d_5 属于 d_4 所在的类 $\bar{c}$。

kNN 的时间复杂度及最优性

表 14-3 给出了 kNN 的时间复杂度。kNN 具有很多绝大多数其他分类器所不具备的特性。kNN 的训练过程只是确定 k 值和对文档的预处理。实际上，如果预先选择 k 值并且不进行预处理，那么 kNN 就不需要任何训练过程。在实践当中，我们必须要对训练文档进行诸如词条化的预处理。将所有训练文档进行预处理，就可以避免在对每篇测试文档进行分类时反复进行预处理，因此，对所有训练文档进行一次性预处理还是很有意义的。

表 14-3　kNN 分类器的训练和测试时间复杂度，M_{ave} 是文档集中每篇文档的平均词汇量大小（即平均词项个数），L_{ave} 是文档的平均长度，L_a 和 M_a 分别是测试文档中词条及词条类（即不同词项）的数目

对训练集进行预处理的 kNN	
训练	$\Theta(\|\lambda\|L_{ave})$
测试	$\Theta(L_a+\|\lambda\|M_{ave}M_a)= \Theta(\|\lambda\|M_{ave}M_a)$
对训练集不进行预处理的 kNN	
训练	$\Theta(1)$
测试	$\Theta(L_a+\|\lambda\|L_{ave}M_a)= \Theta(\|\lambda\|L_{ave}M_a)$

kNN 测试过程的时间复杂度为 $\Theta(|\lambda|M_{ave}M_a)$，是训练集大小的线性函数，在分类过程中必须要计算测试文档和每篇训练文档的距离。测试时间与类别的个数 J 没有关系，因此对于大规模的分类体系(即 J 比较大)来说，kNN 具有一定的潜在优势。

在 kNN 分类器中，我们不需要像在 Rocchio 分类（估计质心）或 NB 分类（估计先验概率

和条件概率）中一样估计参数。kNN 要做的是记忆所有的训练文档并将测试文档与它们比较。基于这个原因，kNN 也被称为基于记忆的学习（memory-based learning）或基于实例的学习（instance-based learning）。在机器学习中，一般都要求有尽可能多的训练文档。但是对于 kNN 来说，过多的训练文档会对分类的效率造成严重影响。

那么，kNN 分类的时间能否比 $\Theta(|\lambda|M_{ave}M_a)$ 更快？或者说在不考虑文档长度的情况下比 $\Theta(|\lambda|)$ 更好？在向量空间维数 M 较小的情况下存在着快速 kNN 方法（参见习题 14-12），对于较大的 M 也存在 kNN 的近似实现方法，它能在满足一定错误上界的条件下提高算法的效率（参考 14.7 节）。但是，在文本分类的应用中，这些近似实现方法并没有受到广泛的测试，所以它们能否在不显著降低分类精确率的情况下获得远远优于 $\Theta(|\lambda|)$ 的时间复杂度还不得而知。

细心的读者可能还会注意到，寻找测试文档的近邻的问题和 ad hoc 检索中寻找与查询相似度最高的多篇文档（参见 6.3.2 节）的问题非常类似。实际上，这两个问题都是 k 近邻问题，只不过 kNN 分类中的测试文档向量相对密集一些（存在几十或者几百个非零分量），而 ad hoc 检索中的查询向量则要稀疏得多（通常少于 10 个非零分量）。为了提高 ad hoc 检索的效率，在 1.1 节中我们引入了倒排索引技术。因此，一个很直观的问题就是，能否也在 kNN 中引入倒排索引来提高分类的效率？

倒排索引可以将搜索限制在那些与查询至少包含一个公共词项的文档当中。因此，在 kNN 分类当中，当测试文档同大多数训练文档都没有公共词项时，采用倒排索引能够提高分类的效率。当然，能否出现这种情况取决于分类问题本身。如果文档很长且保留停用词的话，那么节省不了太多的时间。但是对于短文档并且使用大停用词表去除停用词的情况，那么采用倒排索引可能会将平均测试时间降至 1/10 或更短。

基于倒排索引的搜索时间复杂度是查询词项对应的倒排记录表长度的函数。由于词汇表的增长满足Heaps定律，而该定律表明，如果某些词项的出现概率增大的话，那么其他词项的出现概率必然会降低，所以倒排记录表的数目会随着文档集长度的增长而呈亚线性增长[①]。然而，大部分新词项的出现频度都不高。因此，可以将倒排索引的搜索复杂度看成 $\Theta(T)$（参见 2.4.2 节的讨论），进一步而言，如果假定平均文档长度不随时间的改变而改变，那么则有 $\Theta(T)=\Theta(|\lambda|)$。

我们将在第 15 章看到，kNN 的效果可以与精度最高的文本分类器相媲美（参见表 15-2）。衡量学习方法质量的一个指标是它的贝叶斯错误率（Bayes error rate），即对于一个特定问题学习到的多个分类器的平均错误率。kNN 对于一个具有非零贝叶斯错误率的问题来说不是最优的，也就是说，对于这些问题即使最好的分类器也会存在一个非零分类错误。随着训练集的增长，1NN 的渐近错误率以两倍的贝叶斯错误率为上界。也就是说，如果最优分类器的错误率为 x 的话，1NN 将以 $2x$ 为其渐近错误率的上界。这个结果源于噪音的影响，在 13.5 节中我们已经以噪音特征为例介绍过噪音的存在，实际上，噪音可能会以其他形式存在，下一节将会讨论到这一点。噪音会影响 kNN 分类器的两部分：测试文档及最邻近的训练文档（即最近邻）。两个来源的噪音可以叠加，因此 1NN 的总错误率会是最优错误率的两倍。对于贝叶斯错误率为 0 的分

① Heaps 定律表明，词汇增长的速度比不上文档增长的速度，因此倒排记录表的数目会随文档集长度的增长而呈亚线性增长。而由根据 Hepas 定律，如果文档大小增加 K 倍，那么文档集中的词项增加 $K\beta$ $(0<\beta<1)$ 倍，因此每个词项的平均倒排记录表长度将增加 $K1-\beta$ 倍，该倍数显然小于 K，因此如果某些词项概率增大(如长度增加超过 K 倍)，那么另外一些词项的出现概率肯定会减小。——译者注

类问题，当训练集增长时，1NN 的错误率也接近 0。

? 习题 **14-3** 请解释为什么对于多模态类别问题，kNN 能比 Rocchio 方法处理得更好？

14.4　线性及非线性分类器

本节将会指出，NB 和 Rocchio 分类器都是线性分类器的实例，而线性分类器可能是文本分类当中最重要的一类分类器。我们将它们与非线性分类器进行比较。为讨论简便起见，这里仅考虑二类分类器。根据特征的线性组合和某个阈值的比较结果来确定类别归属的二类分类器叫做线性分类器（linear classifier）。

在二维平面上，线性分类器就是一条直线。在图 14-8 中给出了 5 条直线的例子。这些直线的方程形式都是 $w_1x_1 + w_2x_2 = b$。线性分类器的分类规则是：当 $w_1x_1 + w_2x_2 > b$ 时，则将文档归于 c 类，否则（即当 $w_1x_1 + w_2x_2 \le b$ 时）归于 $\overline{c}$ 类。这里 $(x_1,x_2)^T$ 是文档的二维向量表示，$(w_1,w_2)^T$ 是参数向量，它和 b 一起确定了分类边界。线性分类器的另一个几何解释参见图 15-7。

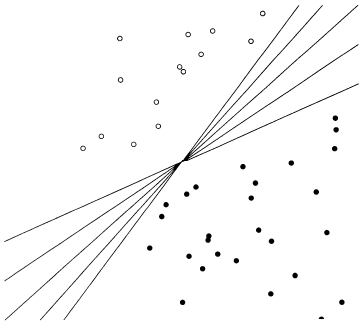


图 14-8　对于二类线性可分问题存在无穷多个能区分两类的超平面

我们可以将二维平面上的线性分类器推广到高维空间上去，这时的分类器就是一个在公式（14-2）中定义的超平面，这里我们将这个公式重写为（14-3）：

$$\vec{w}^T\vec{x} = b。 \qquad (14\text{-}3)$$

此时，类别判定准则就是：如果 $\vec{w}^T\vec{x} > b$，则将文档归于 c 类，而当 $\vec{w}^T\vec{x} \le b$ 时，归于 $\overline{c}$ 类。我们将代表线性分类器的超平面称为决策超平面（decision hyperplane）。

M 维空间中的相应线性分类器算法参见图 14-9。由于算法非常简单，乍看上去线性分类器无须多说。但是，线性分类器的困难主要来自训练，即基于训练集来确定参数 $\vec{w}$ 和 b。通常来说，某些学习算法能够比其他算法学到更好的参数，这里所谓的“更好”是指学到的分类器在新数据上能够获得更好的效果。

```
APPLYLINEARCLASSIFIER(w⃗, b, x⃗)
1  score ← Σ_{i=1}^{M} w_i x_i
2  if score > b
3     then return 1
4     else return 0
```

图 14-9 线性分类算法

下面我们来说明 Rocchio 方法和 NB 都是线性分类器。对于 Rocchio 方法，如果向量 $\vec{x}$ 到两个类中心向量的距离相等，即满足

$$|\vec{\mu}(c_1)-\vec{x}|=|\vec{\mu}(c_2)-\vec{x}|\text{。} \qquad (14\text{-}4)$$

则 $\vec{x}$ 处于两个类的决策边界上。

通过一些基本的算术操作就可以证明，这对应于某个具有法向量 $\vec{w}=\vec{\mu}(c_1)-\vec{\mu}(c_2)$ 及 $b=0.5\times(|\vec{\mu}(c_1)|^2-|\vec{\mu}(c_2)|^2)$ 的线性分类器（参考习题 14-15）。

从 NB 的决策规则出发也可以推出它的线性特点，NB 的决策规则是：选择具有最大 $\hat{P}(c|d)$ 值的类别 c 作为文档的类别，其中

$$\hat{P}(c|d)\propto\hat{P}(c)\prod_{1\le k\le n_d}\hat{P}(t_k|c)\ ,$$

n_d 是文档当中词条的数目。将类别 c 的补类记为 $\bar{c}$，则可以获得如下对数优势率：

$$\log\frac{\hat{P}(c|d)}{\hat{P}(\bar{c}|d)}=\log\frac{\hat{P}(c)}{\hat{P}(\bar{c})}+\sum_{1\le k\le n_d}\log\frac{\hat{P}(t_k|c)}{\hat{P}(t_k|\bar{c})}\text{。} \qquad (14\text{-}5)$$

如果优势率大于 1 或者说优势率对数大于 0，则将文档归于 c 类。很容易发现，如果将公式（14-5）中的 $\log\frac{\hat{P}(t_i|c)}{\hat{P}(t_i|\bar{c})}$ 定义为 w_i、将 $-\log\frac{\hat{P}(c)}{\hat{P}(\bar{c})}$ 定义为 b，再将 x_i 定义为 t_i 在 d 中的出现次数，则公式（14-5）就可以转换成公式（14-3）的形式。这里，下标 i 满足 $1\le i\le M$，指的是词汇表中的所有词项，而不是文档 d 中的每个位置，即不是公式（14-5）中 k 的含义(参见 13.4.1 节)。$\vec{x}$ 和 $\vec{w}$ 都是 M 维向量。因此，在对数空间中，NB 是一个线性分类器。

表 14-4 一个线性分类器的例子。每一维对应的词项为 t_i，w_i 是 Reuters-21578 中 *interest* 类对应的线性分类器的参数，阈值 $b=0$。由于词项 dlr 和 world 更倾向于代表竞争类 *currency*，因此它们在 *interest* 类中的权重为负值

t_i	w_i	d_{1i}	d_{2i}	t_i	w_i	d_{1i}	d_{2i}
prime	0.70	0	1	dlrs	–0.71	1	1
rate	0.67	1	0	world	–0.35	1	0
interest	0.63	0	0	sees	–0.33	0	0
rates	0.60	0	0	year	–0.25	0	0
discount	0.46	1	0	group	–0.24	0	0
bundesbank	0.43	0	0	dlr	–0.24	0	0

例 14-3 表 14-4 定义了 Reuters-21578 语料（参考 13.6 节）中 *interest* 类的一个线性

分类器。由于文档 $\vec{d}_1$“ rate discount dlrs world” 满足 $\vec{w}^T\vec{d}_1 = 0.67\times1 + 0.46\times1 + (-0.71)\times1 + (-0.35)\times1 = 0.07 > 0 = b$,所以它被分到 *interest* 类。而文档 $\vec{d}_2$“ prime dlrs”” 满足 $\vec{w}^T\vec{d}_2 = -0.01 \leq b$, 所以被分到 *interest* 类的补类。为简单起见，这里采用了布尔向量的文档表示方法，即 1 表示词项出现，0 表示不出现。

图 14-10 给出的是线性问题的一个图形化例子，所谓“ 线性问题”（linear problem），指的是两个类别的分布 $P(d|c)$和 $P(d|\bar{c})$能够被一条直线分开。这条直线称作类边界（class boundary）。这是两个类之间的真实边界，不要与学习策略所产生的决策边界相混淆，后果只是对类别边界求近似的计算结果。

文本分类中的一个典型情况是，在图 14-10 中存在一些噪音文档（noise document，在图中用箭头标记），它们和类别的总体分布不太吻合。在 13.5 节中，我们曾经定义过噪音特征，即那些如果包含在文档表示中就会起误导作用从而增大平均分类错误的特征。类似地，噪音文档是那些如果包含在训练集中就会对学习过程产生误导从而增加分类错误的文档。直观上，总体分布（underlying distribution）会将文档表示空间划分成多个区域，每个区域主要由同类的文档组成。而在本区域中与主要类别不同的类别的文档就是噪音文档。

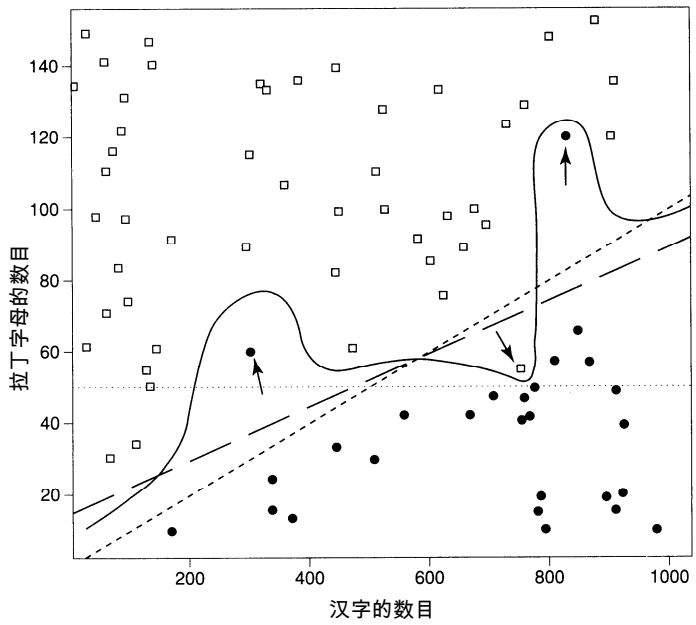


图 14-10　一个带噪音的线性问题。在这个假想的 Web 网页分类下，仅包含中文的网页用实心圆表示，而中英文混合网页用小方块表示。除了 3 篇噪音文档（用箭头标记）外，这两个类可以被一个线性类别边界（用短破折号虚线表示）分开

噪音文档的存在是造成线性分类器训练困难的原因之一。如果在选择分类器的决策超平面时过度关注噪音文档，那么得到的分类器会在新数据上表现不佳。更根本的问题是，通常我们

很难判断哪些文档才是会误导分类的噪音文档。

如果存在一个超平面能将两类完全分开，那么就称这两个类是线性可分的（*l*inearly separable）。实际上，如果问题线性可分，那么就存在无穷多个线性分类器或分隔器（linear separator）可将两类分开（参考习题 14-4）。图 14-8 给出了无穷多个分类超平面的例子。

图 14-8 描述的是训练线性分类器的另一个难题，即如果要处理好线性可分问题，我们必须找到一个能准确衡量决策超平面能否完美分类训练数据的标准。一般来说，其中有些超平面在新数据上分类效果会好，而有些却会差。

一个非线性分类器（nonlinear classifier）的例子是 kNN。只要看看图 14-6 中的例子，就会对 kNN 的非线性特征确信无疑。kNN 的决策边界由多段局部线段构成，但是通常都会呈现出复杂的形状，不会在二维空间上表现为直线，在高维空间上也不会表现为超平面。

图 14-11 是非线性问题的另一个例子：由于左上角存在一个圆形圈住的内嵌区域，在分布 $P(d|c)$和 $P(d|\bar{c})$之间不存在好的线性分类器。线性分类器会误分这个区域内的文档，但此时如果采用诸如 kNN 的非线性分类器，在训练数据足够大的情况下会取得高精度的结果。

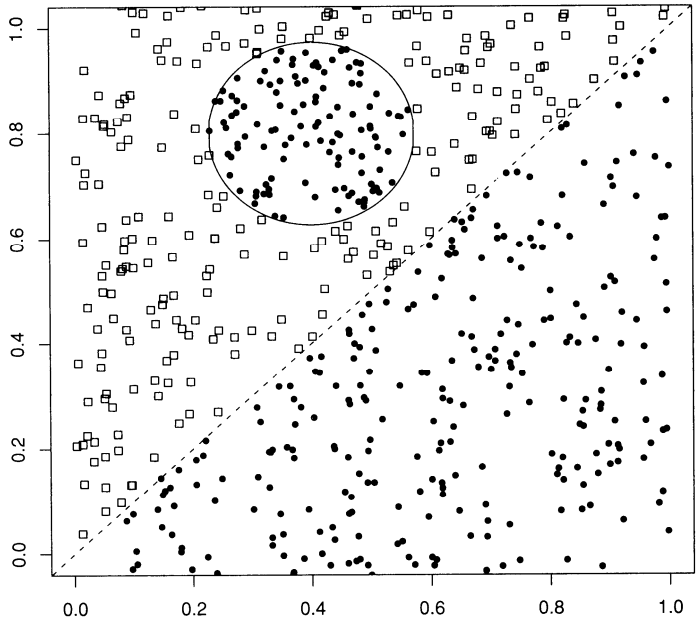


图 14-11 一个非线性问题的例子

如果一个问题是非线性的，即它的类别边界不能通过线性超平面来近似，那么此时使用非线性分类器的分类结果往往会好于使用线性分类器；但是如果一个问题是线性的，那么最好用简单的线性分类器。

? 习题 **14-4** 试证明两个类别间的线性分类器的数目要么是无穷的，要么是 0。

14.5　多类问题的分类

可以把二类的线性分类器推广到多类问题中去，即分类中类别的数目 $J>2$。推广的方法取决于类别之间是否互斥。

在非互斥类别上的多类分类问题称为多标签（any-of 或 multilabel）或多值分类（multivalue classification）问题。在这种情况下，一篇文档可以同时属于多个类、只属于一个类、或者不属于任何类。在某个类别上的分类决策并不影响它在其他类别上的决策。有时，我们说类别之间是相互独立（independent）的，但是这种说话会造成误解，这是因为很少有类别能满足我们在 13.5.2 节所定义的统计上的独立性概念。按照公式（13-1）所给出的分类的形式化定义，在多标签问题分类中，我们会学到 J 个不同的分类器 γ_j，对于每篇文档 d，γ_j 返回的类别结果要么是 c_j，要么是 $\overline{c}_j$，即 $\gamma_j(d) \in \{c_j, \overline{c}_j\}$。

通过线性分类器来解决多标签问题的步骤非常简单：

(1) 对每个类别建立一个分类器，此时训练集包含所有属于该类的文档（正例）和所有不属于该类的文档（反例）；

(2) 给定测试文档，分别使用每个分类器进行分类，每个分类器的分类结果并不影响其他分类器的结果。

第二种存在多类别的分类问题称为单标签（one-of或single-label）问题。这里的类别是互斥的，一篇文档只能属于其中的一类。单标签分类也称为多项式（multinomial）分类或多类（polytomous[①]或multiclass）分类。形式化地，在这种分类中存在单个分类函数 γ，其值域为 $\coprod$，也就是说 $\gamma(d) \in \{c_1,\ldots,c_J\}$。kNN是一个非线性的单标签分类器。

在实际当中，真正的单标签问题并不如多标签问题普遍。例如，对于像 *UK*、*China*、*poultry* 或 *coffee* 这样的类别而言，一篇文档可以同时与多个主题相关。比如，*UK* 的首相访问 China 时谈到有关 coffee 和 poultry 的话题时，产生的文档就会同时和上述多个主题相关。

然而，即使类别之间并不真正互斥，正如我们在图 14-1 中所做的那样，在实际当中我们往往使用单标签假设。在文档的语种判断这个分类问题中，单标签假设往往是实际情况的一个很好的近似，这是因为大部分文本都只使用一种语言来书写。在这类情况下，利用单标签假设的限制能够提高分类器的效果，因为这避免了像多标签分类器那样给文档分配不止一个类别或者不分配任何类别而导致错误。

J 个超平面并不会将空间 $\square^{|V|}$ 划分成 J 个不相交的区域（参见图 14-12）。因此，使用二类分类器来处理单标签问题时需要使用组合策略。最简单的方法是对所有类别排序然后选出排名最高的类。从几何上来说，排序可以通过计算到 J 个线性分隔边界的距离来实现。靠近类别分隔边界的文档被错分的可能性更大，因此文档离分隔边界的距离越远，那么越有可能属于这个类。

① Polytomous 的一个同义词为 polychotomous。

另外一种方法是，我们可以使用一个直接的置信度指标来对类别排序，比如，类别归属的概率。于是，这种方式下利用线性分类器来处理单标签问题的算法可以描述如下：

(1) 对每个类别建立一个分类器，此时训练集包含所有属于该类的文档（正例）和所有不属于该类的文档（反例）；

(2) 给定测试文档，分别使用每个分类器进行分类；

(3) 将文档分配给得分最高的类、置信度最高的类或概率最大的类。

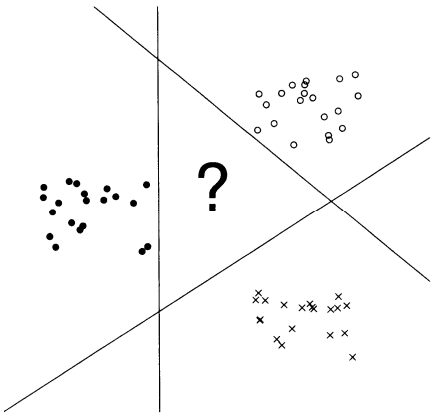


图 14-12 J 个超平面并不会将空间分成 J 个不相交的区域

对于J类（$J > 2$）情况下的分类器性能进行分析的一个重要工具是混淆矩阵（*confusion matrix*）。对于每个类别对$<c_1, c_2>$，混淆矩阵给出了c_1类文档错分到类别c_2中的文档数目。在表14-5中，分类器试图将3个经济类别*money-fx*、*trade*、*interest*与3个农业类别*wheat*、*corn*、*grai*区分开来，但是产生了很多错误。混淆矩阵能够帮助查明问题的细节从而提高系统的精确率。比如，为解决表14-5中的第二大错误，可以考虑引入区分*wheat*类文档和*grain*类文档的特征。

表 14-5 Reuters-21578 语料上的一个混淆矩阵。比如，14 篇属于 grain 类的文档被误分到 wheat 类中。选自 Picca 等人（2006）

实际类别	分配类别	money-fx	trade	interest	wheat	corn	grain
money-fx		95	0	10	0	0	0
trade		1	1	90	0	1	0
interest		13	0	0	0	0	0
wheat		0	0	1	34	3	7
corn		1	0	2	13	26	5
grain		0	0	2	14	5	10

? 习题 **14-5** 建立一个 300 篇文档构成的训练集，其中每 100 篇属于一种语言，共 3 种语言（如英语、法语和西班牙语）。采用同样的方式来构造测试集，并增加 100 篇采用第 4 种语言的文档。在上述训练集上训练出一个 (i) 单标签分类器和 (ii) 一个多标签分类器并将它们在测试文档上测试 (iii) 在这个问题上两个分类器的表现有没有什么有趣的区别？

14.6　偏差–方差折中准则

非线性分类器比线性分类器更强大。对有些问题来说，存在着零分类错误的非线性分类器，而并不存在这样的线性分类器。这是否意味着为了获得最优的分类效果，在统计文本分类中我们应该总是优先选择非线性分类器呢？

为了回答这个问题，本节当中我们将介绍机器学习中一个非常重要的概念——偏差–方差折衷准则。折衷准则可以解释为什么不存在普遍最优的学习方法。因此，在解决一个文本分类问题时，选择合适的学习方法是不可避免的一个环节。

在本节当中，我们分别用线性和非线性分类器来作为“弱”和“强”学习方法的实例。这种做法做了相当的简化，这是因为：第一，在很多非线性模型当中，线性模型是它们的一个特例，比如，像 kNN 这种非线性学习方法在某些情况下会产生一个线性分类器。第二，存在着一些比线性模型更简单的非线性模型，比如，一个包含两个参数的二项式分类器并不比一个 10 000 维的线性分类器强。第三，学习的复杂度并不真正是分类器的真正特性，这是因为在学习中还存在很多其他的因素(如 13.5 节提到的特征选择以及第 15 章将要提到的正则化和诸如间隔最大化的限制条件等)会使学习方法要么更强要么更弱，而并不对学习方法最终学出的分类器类型有任何影响，即不管分类器到底是线性还是非线性的。在 14.7 节当中我们为读者列出了很多参考文献，它们在研究偏差—方差准则时考虑了上述这些复杂因素。本节当中，线性和非线性分类器只是简单地作为文本分类当中的弱学习和强学习方法的代名词。

首先，我们要把文本分类的目标表达得更精确。在 14.6 偏差–方差折中准则 类的目标是使在测试集上的分类错误最小化。这其中隐含的假设是训练文档和测试文档都产生自同一总体分布。我们将这个分布记为 $P(<d, c>)$，其中 d 是文档、c 是类别或标签。图 13-4 和图 13-5 给出了将 $P(<d, c>)$分解成 $P(c)$和 $P(d|c)$的生成模型的例子。而图 14-10 和图 14-11 则描述了当 $d \in \square^2$ 且 $c \in$ {正方形，实心圆}时$<d, c>$的生成模型。

本节当中，我们不采用测试文档中正确分类的数目（或者，一种等价的方式是采用相当于测试文档的错误率指标）作为评价指标，而是采用一个能够体现分类内在的不确定性的评价指标。在很多文本分类问题中，一个给定的文档表示可能来自属于不同类别的多篇文档，这是因为不同类别的文档可以映射成同一文档表示。比如，在词袋模型中，文档 China sues France 及 France sues China 都会映射成文档表示 d'={China, France, sues}。但是只有后面那篇文档才与类别 c'=legal actions brought by France（这个类别可以以一个国际贸易律师输入固定查询的方式来定义）相关。

本节中为了简化计算，我们在评价分类器时并不计算它在测试集上的错误分类数目，而是计算分类器估计条件概率 $P(c|d)$的好坏程度。在上述例子当中，$P(c'|d')$可能等于 0.5。

在文本分类中，我们的目标是寻找一个分类器 γ，在所有 d 的平均意义上说，$\gamma(d)$要尽可能地接近其真实概率值 $P(c|d)$。采用均方误差来衡量的话，就是

$$\mathrm{MSE}(\gamma) = E_d[\gamma(d) - P(c \mid d)]^2。 \qquad (14\text{-}6)$$

其中，E_d是基于 $P(d)$的数学期望。均方误差会给那些虽不完全正确但是已经非常接近真实值的决策 γ 也赋予一定的得分。

对于分布 $P(<d,c>)$来说，如果某个分类器 γ 使得 MSE(γ)最小，那么我们称该分类器最优。

对于分类器而言，最迫切的需求就是如何最小化MSE。而对于学习方法我们也需要一个准则[①]。前面我们将学习方法Γ定义为以标注好的训练集λ为输入、以分类器γ为输出的一个函数。

对于学习方法而言，我们的目标是寻找到最优的方法 Γ，能够在所有训练集的平均意义上说，学到具有最小 MSE 的分类器。形式地，我们将这种最小的学习误差定义为

$$\text{learing-error}(\Gamma) = E_{\mathrm{D}}[\text{MSE}(\Gamma(\mathrm{D}))]。 \tag{14-7}$$

其中，E_{λ}是基于训练集上的期望。为简单起见，可以假设不同训练集的大小（即训练文档数目）固定，于是基于分布 $P(<d, c>)$ 就可以定义出训练集上的分布 $P(\lambda)$。

在统计文本分类中，可以使用学习误差来作为选择学习方法的准则。对于概率分布 $P(\lambda)$，如果学习方法的学习误差最小，那么它是最优的。

为了便于阅读，我们将 $\Gamma(\lambda)$简记为 Γ_{λ}，于是可以将公式（14-7）转换为

$$\begin{aligned}
\text{learing-error}(\Gamma) &= E_{\mathrm{D}}[\text{MSE}(\Gamma(\mathrm{D}))] \\
&= E_{\mathrm{D}}E_d[\Gamma_{\mathrm{D}}(d) - P(c|d)]^2 && (14\text{-}8)\\
&= E_d[\text{bias}(\Gamma,d)+\text{variance}(\Gamma,d)] && (14\text{-}9)\\
\text{bias}(\Gamma,d) &= [P(c|d) - E_{\mathrm{D}}\Gamma_D(d)]^2 && (14\text{-}10)\\
\text{variance}(\Gamma,d) &= E_{\mathrm{D}}[\Gamma_D(d) - E_{\mathrm{D}}\Gamma_D(d)]^2 && (14\text{-}11)
\end{aligned}$$

其中公式（14-8）和公式（14-9）的等价性可以参见图 14-13 中的公式（14-13）。需要指出的是，d 和λ之间是相互独立的。一般来说，对一篇随机的文档 d 和一个随机的训练集λ，λ一般不会包含带类别标签的 d，也就说测试文档 d 一般不会在训练集上出现。

$$\begin{aligned}
E[x-\alpha]^2 &= Ex^2 - 2Ex\alpha + \alpha^2 \\
&= (Ex)^2 - 2Ex\alpha + \alpha^2 \\
&\quad + Ex^2 - 2(Ex)^2 + (Ex)^2 \\
&= [Ex-\alpha]^2 && (14\text{-}12)\\
&\quad + Ex^2 - E2x(Ex) + E(Ex)^2 \\
&= [Ex-\alpha]^2 + E[x-Ex]^2
\end{aligned}$$

$$\begin{aligned}
E_{\mathrm{D}}E_d[\Gamma_{\mathrm{D}}(d) - P(c|d)]^2 &= E_dE_{\mathrm{D}}[\Gamma_{\mathrm{D}}(d) - P(c|d)]^2 && (14\text{-}13)\\
&= E_d[[E_{\mathrm{D}}[\Gamma_{\mathrm{D}}(d)] - P(c|d)]^2 \\
&\quad + E_{\mathrm{D}}[\Gamma_{\mathrm{D}}(d) - E_{\mathrm{D}}\Gamma_{\mathrm{D}}(d)]^2
\end{aligned}$$

图 14-13 偏差–方差分解的算术转换公式。在（14-12）中，令 $\alpha = P(c \mid d)$，$x = \Gamma_{\lambda}(d)$，即可推导出公式（14-13）

① 可以这样理解，整个错误分成来两部分，一个是分类的错误，一个是训练(学习方法)的错误。前者发生在测试集上，后者发生在训练集上。——译者注

偏差是$P(c|d)$和$E_{\lambda}\Gamma_{\lambda}(d)$的差平方，前者是$d$属于$c$的真实条件概率值，后者是在不同训练集上训练出的分类器对d的平均预测值。如果学习方法产生的分类器始终出错，那么偏差就会很大。在以下几种情况下，偏差较小：(i) 分类器始终正确；(ii)不同的训练集导致在不同文档上出错[①]；(iii) 不同的训练集导致在同一文档产生正向或者负向误差，但是平均起来接近 0。如果上述条件之一成立，那么$E_{\lambda}\Gamma_{\lambda}(d)$就会接近$P(c|d)$。

对于像 Rocchio 和 NB 一样的线性方法来说，对于非线性问题它们的偏差就比较大，这是因为它们只能对线性超平面这一种类型的类边界建模。如果生成模型 $P(<d,c>)$有复杂的非线性边界，那么很多点始终都会被分错，因此公式（14-9）中的偏差部分就会很大。比如，图 14-11 的圆圈内的点都不符合线性模型，因此始终会被线性分类器错分。

我们也可以将偏差想成是在建立分类器时融入了领域知识(或者说缺乏领域知识)所带来的后果。如果知道类别之间的真实边界是线性的话，那么采用能够生成线性分类器的学习算法会比采用非线性的方法更有可能取得成功。但是如果真实的分类边界不是线性的，而我们却错误地偏向线性分类器，那么分类的平均精确率会很低。

像 kNN 一样的非线性方法的偏差较小。从图 14-6 中可以看出，kNN 的决策边界是可变的，它取决于训练集中的文档分布，不同训练集学到的决策边界可能会相差很大。因此，对某些训练集来说，每个文档都有被正确分类的机会。因此，此时的平均预测结果 $E_{\lambda}\Gamma_{\lambda}(d)$就会接近 $P(c|d)$，于是 kNN 方法的偏差小于线性的学习方法。

方差（variance）指的是学到的分类器的预测结果的变异程度，即 $\Gamma_{\lambda}(d)$和 $E_{\lambda}\Gamma_{\lambda}(d)$的差平方在所有文档上的平均值。如果不同的训练集$\lambda$得到的分类器 Γ_{λ}相差很大，那么方差会较大。如果不同训练集对 Γ_{λ}所做的分类决策结果的影响不大，那么不管决策结果是对还是错，此时的方差就较小。方差主要度量的是决策的不一致程度，而不管决策的正确与否。

像 kNN 一样的非线性学习方法的方差较大。从图 14-6 中可以很明显地看出，kNN 能够对两类之间非常复杂的边界进行建模。因此它对图 14-10 所示的那类噪音文档非常敏感。于是，对 kNN 而言，公式（14-9）中的方差部分较大：测试文档如果正好靠近训练集中的噪音文档，那么它可能会被误分类；而如果在训练集中它的附近不存在噪音文档，那么它可能会被正确分类。这会导致在不同的训练集上的结果差异很大。

高方差的学习方法很容易造成对训练数据的过学习（overfitting，也称过拟合）。分类的目的是在一定程度上拟合训练数据，从而希望能够获取总体分布 $P(<d, c>)$的真实特性。如果过学习的话，那么学习方法也能从噪音中学习到结果。过学习会增大 MSE，这也是高方差学习方法的普遍性问题。

也可以将方差想象成模型的复杂度（model complexity），或者说学习方法的记忆能力（memory capacity），即能够记住训练集的特性细节并把它们应用到新数据的程度。这种记忆能力对应于拟合训练集的独立参数的数目。kNN 的每组邻居 S_k都可以独立进行分类决策，也就是说这里的参数实际上是 $\hat{P}(c \mid S_k)$（参见图 14-7）。因此，kNN 的记忆能力仅受训练集的大小所限制，它可以记忆任意大小的训练集。而对于 Rocchio 方法来说，它的参数个数是固定的：不管训练集多大，其空间的每一维上都有 J 个参数（对应 J 个类别），多维一起表示一个类别质心。

① 也就是说，在不同的训练集下训练出的分类器，对一篇固定的文档 d 出错的可能性是最小的，在这篇文档的累积错误进行平均以后会较小。——译者注

Rocchio 方法（采用质心向量定义）不能“记住”训练集中文档的分布细节。

按照公式（14-7），在选择学习方法时，我们的目标是学习误差最小化。公式（14-9）所表现的原则可以简单总结为：学习误差 = 偏差 + 方差，也就是说学习误差由偏差和方差两部分构成。通常情况下，这两个部分不会同时最小。当我们比较两个学习方法 Γ_1 和 Γ_2 时，大部分情况下最后的结果都是，其中一个方法偏差高方差低而另一个方法偏差低方差高。因此，从两个学习方法中选择一个时，我们不是简单地选择能够在不同训练集上产生好的分类器的学习方法（方差小），也不是选择那些能学出复杂决策边界的学习方法（偏差小）。实际的做法是，根据应用的需要，选择不同的权重对偏差和方差进行加权求和。这种折衷称为偏差-方差折衷准则（bias–variance tradeoff）。

图 14-10 给出了一个例子，虽然有点人为构造的嫌疑，但是有利于对上述折衷准则的理解。在一些中文文本中，存在用拉丁字母写的英文词，如 CPU、ONLINE 和 GPS 等等。考虑中文及中英文混合两类网页的分类问题。搜索引擎可能会给没有英文知识的中文用户（当然，他能理解像 CPU 这种借用词）一个过滤混合网页的选项。对于该分类问题，我们可以引入两个特征，分别是网页中的拉丁字母数目和中文字符数目。如前所述，生成式模型 $P(\langle d,c\rangle)$生成了大部分混合网页和中文网页，在图 14-10 它们分别在短虚线的上面和下面，但是也存在一些噪音文档。在图 14-10 中，我们可以看到 3 个分类器。

- 单特征分类器。图中以点横线表示。该分类器只使用了一个特征，即拉丁字母的数目。假定一个学习方法能够将训练集上的错误个数降至最小，那么横线所表示的决策边界的位置就不太受训练集（比如噪音文档）差异的影响。因此，产生这种分类器的学习方法的方差小，但是偏差大，这是因为它会一直错分左下角的正方形文档及超过实心圆点中超过 50 个拉丁字母的文档。
- 线性分类器。图中以长虚线表示。线性分类器的学习偏差较小，只有噪音文档和边界附近的文档被错分。与单特征分类器相比，它的方差要大一些，但是仍然很小：长虚线和真实的分类边界（图中以短虚线表示）偏差不大，而几乎所有从训练集学到的线性分类器的决策边界也都满足这个性质。因此，只有很少的文档（距离类边界比较近的文档）会被一直分错。
- “完美拟合训练集”的分类器。图中以实线来表示。在这里，学习方法构建了一个能够将训练文档完美分开的决策边界。由于不存在一篇文档始终会被错分，所以该方法的偏差最小，有时甚至对于测试文档集中的噪音文档也能分对。但是这种方法的方差较大。由于噪音文档能够将决策边界任意移动，因为靠近训练集中噪音文档的测试文档会被误分，但线性学习方法不太可能会出现这种情形。

很多知名的文本分类方法都是线性的，这一点或许有点奇怪。这其中的一些分类方法，特别是线性 SVM、正则化的 logistic 回归及线性回归方法等等，都是当今最有效的方法之一。偏差–方差折衷准则能够解释这些方法的成功。一般情况下，文本分类中的类别比较复杂，似乎很难通过线性的方法来建模。但是，这种直觉常常会在文本分类经常遇到的高维空间常常会起到误导作用。当空间维度增大时，线性可分的可能性也会迅速增加（参考习题 14-17）。因此，尽

管高维空间下的线性模型是线性的，但是它们非常强大。即使更强大的非线性学习方法能够对复杂度远超过超平面的决策边界建模，但是它们同时会对训练集中的噪音文档极其敏感。在训练集很大的情况下，非线性学习方法有时效果会更好，但绝不是在所有情况下都会如此。

? 习题 **14-6**　在图 14-14 中，3 个向量 $\vec{a}$、$\vec{b}$ 及 $\vec{c}$ 中哪一个满足：(i) 采用内积计算的情况下与 $\vec{x}$ 最近？(ii) 采用余弦相似度计算的情况下与 $\vec{x}$ 最近？(iii) 采用欧氏距离计算的情况下与 $\vec{x}$ 最近？

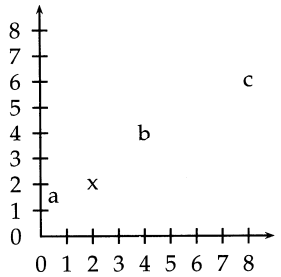


图 14-14　一个表明欧氏距离、内积相似度及余弦相似度计算不同的例子。其中 $\vec{a}$ = (0.5 1.5)^T，$\vec{x}$ = (2 2)^T，$\vec{b}$ = (4 4)^T 及 $\vec{c}$ = (8 6)^T

习题 **14-7**　下载 Reuters-21578 语料，在类别 *acquisitions*、*corn*、*crude*、*earn*、*grain*、*interest*、*money-fx*、*ship*、*trade* 及 *wheat* 上训练出 Rocchio 及 kNN 分类器并进行测试。整个过程中采用语料的 ModApte split 版本，同时可以使用某个包含 Rocchio 及 kNN 分类器实现的软件包，如 Bow 工具集（McCallum 1996）。

习题 **14-8**　下载 20 个新闻组（Newgroups）语料，在其 20 个类别上训练和测试 Rocchio 及 kNN 分类器。

习题 **14-9**　说明 Rocchio 分类器的决策边界和 kNN 一样都是 Voronoi 剖分的结果。

习题 **14-10*]**　在计算一个密集的质心向量和一个稀疏向量之间的距离时，一个很原始的实现方法是在 M 个维度上进行反复的迭代计算，因此其时间复杂度是 $\Theta(M)$。基于等式 $\sum(x_i-\mu_i)^2 = 1.0 + \sum\mu_i^2 - 2\sum x_i\mu_i$ 并假定 $\sum\mu_i^2$ 可以预先计算，请给出一个时间复杂度是 $\Theta(M_a)$的算法，其中 M_a 是测试文档中不同词项的个数。

习题 **14-11[***]**　证明由所有具有同样 k 个最近邻的点组成的平面区域是凸多边形。

习题 **14-12**　设计一个在一维空间的高效率的 1NN 搜索方法（复杂度基于文档数目 N 来计算），该算法的时间复杂度是多少？

习题 **14-13** [***]　设计一个在二维空间上高效率的 1NN 搜索方法，预处理最多需要 N 的多项式时间就可以完成。

习题 **14-14** [***]　能否将上题的结果推广到 M 维空间？即设计出效率和上题一样的算法，但是 M 很大。

习题 **14-15**　证明公式（14-4）实际上定义了一个超平面，其中 $\vec{w} = \vec{\mu}(c_1) - \vec{\mu}(c_2)$，$b = 0.5 \times (|\vec{\mu}(c_1)|^2 - |\vec{\mu}(c_2)|^2)$。

习题 **14-16**　通过嵌入一个不可分数据子集（如图 14-15 所示），我们很容易在高维空间中构造一个不可分的数据集。考虑在三维空间中嵌入图 14-15，将图中的 4 个点做轻微变动（比如随机在某个方向上移动少许距离）。为什么我们还会预测最后的分布结果是线性可分的？在 M 维空间中存在 m 个点构成的不可分子集的可能性有多大（m<<M）？

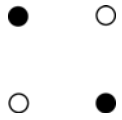


图 14-15 一个简单的非线性可分的点集

习题 **14-17** 只考虑两个类别的情况，试证明当维度$M>1$时，对超立方体顶点的任意类别分配中上出现不可分情况的百分比会下降。比如，$M=1$时，该百分比为 0；而$M=2$时，该百分比为 2/16。图 14-15 给出了两种不可分情况中的一个，另一个正好是黑白颜色互换后的结果[①]。采用解析或者模拟的方法解本习题。

习题 **14-18** 尽管我们指出 NB 分类器和线性向量空间分类器的相似性，但是将词频向量（即 NB 中的文档表示）在连续的向量空间中表示是没有意义的。然而，存在 NB 的一个类似 Rocchio 方法的形式化结果。请证明 NB 会将文档（同 13.4.1 节一样表示为词频向量，不过该向量进行了归一化使得所有分量的和为 1）分配给与其 KL 距离（参考 12.4 节）最小的类别（表示为一个参数向量）。

14.7 参考文献及补充读物

如第 9 章所述，Rocchio 相关反馈方法来自 Rocchio（1971）。Joachims（1997）对该方法进行了概率分析。在 20 世纪 90 年代的 TREC 会议当中，Rocchio 方法被广泛用于分类（Buckley 等人 1994a,b；Voorhees 和 Harman 2005）。起初，它作为信息路由（routing）的一种形式来使用。信息路由只是依照与类别的相关性将文档排序而不将它们归类。有关信息过滤（filtering）的早期工作参见 Ittner 等人（1995）及 Schapire 等人（1998）。信息过滤是真正将类别赋予文档的分类过程。这里所用的信息路由不要与它另外的一种用法相混淆。路由也可以指将文档电子分发到订阅者的过程，这种文档发布的方式也称为推模式（push model）。在拉模式（pull model）中，文档到用户的每次传递都是用户发起的，比如，通过搜索的方式或者从一个新闻网站上文档列表中选择的方式来发起请求。

有些作者仅将 Roccchio 分类器的称呼用于两类的问题，而将术语基于簇的分类（cluseter-based classification, Iwayama 和 Tokunaga 1995）和基于质心的分类（centroid-based classification，Han 和 Karypis 2000，Tan 和 Cheng 2007）用于 Rocchio 分类器在多类场合下的称呼。

关于 kNN 更细节的处理方法参见（Hastie 等人 2001），其中也包括如何调优参数 k。基于局部性的哈希方法（locality-based hashing，参见 Andoni 等人 2006）是一个快速的近似 kNN 算法。Kleinberg（1997）给出了一个时间复杂度大约是 $\Theta[(M\log^2 M)(M+\log N)]$的 kNN 算法，其中 M 是空间的维度，N 是数据点的个数，但是其存储开销也达到了 $\Theta[(N\log M)^{2M}]$。Indyk（2004）

①图 14-15 中四个点上的类别指派情况总共有 $2^4=16$ 种，其中只有 2 种情况不可分。——译者注

总结了高维空间下的近邻算法。kNN 在文本分类中的早期工作主要是受可用的大规模并行硬件架构（Creecy 等人 1992）所推动。Yang（1994）使用倒排索引来加速 kNN 的分类过程。有关 1NN 最优结果（即渐近错误率以贝叶斯错误率的两倍为上界）的工作归功于 Cover 和 Hart（1967）。

Rocchio 和 kNN 的分类效果高度依赖于参数的调优（具体地，对于 Rocchio 方法是参数 b' 而对于 kNN 是参数 k）、特征工程（feature engineering，参考 15.3 节）及特征选择（参考 13.5 节）等因素。对于这些因素，Buckley 和 Salton（1995）、Schapire 等人（1998）、Yang 和 Kisiel（2003）及 Moschitti（2003）等针对 Rocchio 方法进行了探讨，而 Yang（2001）及 Ault 和 Yang（2002）对 kNN 方法进行了探讨。Zavrel 等人（2000）比较了 kNN 分类中的多种特征选择方法。

偏差–方差折衷准则由 Geman 等人（1992）引入。虽然其在 14.6 节是由 MSE(γ)导入，但是该准则也应用到很多损失函数的推导中（参考 Friedman 1997, Domingos 2000）。Schütze 等人（1995）及 Lewis 等人（1996）讨论了文本中的线性分类器，而 Hastie 等人（2001）则讨论了通用领域的线性分类器。对本章中提及但没有详细介绍的算法感兴趣的读者，可以参考 Bishop（2006）来了解神经网络，参考 Hastie 等人（2001）来了解线性回归和 logistic 回归，参考 Minsky 和 Papert（1988）来了解感知机算法。Anagnostopoulos 等人（2006）的工作表明，倘若分类器通过特征选择后在不多的特征下也能训练出很好的效果，那么倒排索引就能够用于基于任意线性分类器的高效文档分类中去。

本章当中我们只给出了将多个二类分类器组合成单标签分类器的简单方法。另外一种重要的方法是使用校正码，即对每篇文档构造一个不同二类分类器的决策向量，然后，基于训练集决策向量的分布，来对测试文档的决策向量进行“纠正”，这其中要将所有二类分类器及其关联信息融入到最后的分类决策中（Dietterich 和 Bakiri 1995）。Ghamrawi 和 McCallum（2005）也将类别之间的依赖性引入到多标签分类器当中。Allwein 等人（2000）给出了一个二类分类器的通用组合框架。

第15章 支持向量机及文档机器学习方法

近20年来，如何提高分类器的效果一直是机器学习研究中一个集中性话题，也因此产生了一系列的高效分类器，如支持向量机、提升式（boosted）决策树、正则化logistic回归、神经网络和随机森林（random forest）[①]等。其中大部分，包括作为本章主要话题的支持向量机方法，已经成功地应用到很多IR问题特别是文本分类当中。SVM是最大间隔分类器的一种，它是基于向量空间的机器学习方法，其目标是找到两个类别之间的一个决策边界，使之尽量远离训练集上的任意一点（当然一些离群点和噪音点可能不包括在内）。

本章首先从二类线性可分问题出发，导出SVM分类器（15.1节），然后将模型推广到非线性可分数据、多类问题及非线性模型，另外也将讨论一些SVM的性能（15.2节）。之后，我们将讨论在实际开发文本分类器时遇到的问题（15.3节），包括什么时候用哪类分类器、如何在分类中融入领域文本特征等。最后，我们将介绍如何把分类中采用的机器学习技术应用到ah hoc检索中的文档排序中去（15.4节）。尽管有多个机器学习方法已经用于检索文档排序当中，但SVM的使用仍然占主导地位。虽然除了在小训练集情况下，或许SVM会占点优势之外，在其他情况下它并不一定会优于其他机器学习方法，但是它却表现出了目前最高的分类水平，并且在理论和经验方面更具有吸引力。

15.1 二类线性可分条件下的支持向量机

如图14-8所示，二类线性可分问题存在大量可能的线性分界面。直观地看，一个处于中间空白处的决策面比那些靠近某个类的决策面更好。尽管有些学习方法（如感知机）只需找到任一线性分界面即可，而另一些方法（如NB）则需按照某个准则找到最优的线性分界面。对于SVM而言，它定义的准则是寻找一个离数据点最远的决策面。从决策面到最近数据点的距离决定了分类器的间隔（margin）。这种构建方法也意味着SVM的决策函数完全由部分（通常的数量很小）的数据子集所确定，并且这些子集定义了分界面的位置。这些子集中的点称为支持向量（support vector，在向量空间中，一个点对应从原点到它的向量）。图15-1给出了分类间隔和支持向量的例子。除支持向量外的其他数据点对于最终分界面的确定并不起作用。

① 在机器学习中，随机森林是一个包含多个决策树的分类器，并且其输出的类别是由个别树输出的类别的众数而定。这个术语是1995年由贝尔实验室的Tin Kam Ho所提出的随机决策森林（random decision forests）而来的。Leo Breiman和Adele Cutler发展并推论出随机森林的算法。来自中文维基百科。——译者注

最大化分类间隔看上去很合理，这是因为在分界面附近的点代表了不确定的分类决策，分类器会以两边各 50%的概率做出决策。具有很大分类间隔的分类器不会做出确定性很低的决策，它给出了一个分类的安全间隔：度量中的微小错误和文档中的轻微变化不会导致错误分类。另一种引入 SVM 分类器的直观动机参见图 15-2。在分类器构建过程中，SVM 强调在分类决策面上下有一个大的分类间隔。与决策超平面相比，如果不得不在类别之间放置一个宽分隔器，那么分隔器可供选择的放置位置就很少。这样做会导致此时模型的记忆能力降低，因此我们可以预期它在测试数据上泛化出的分类能力就会提高（参考第 14 章有关偏差–方差折衷准则的讨论）。

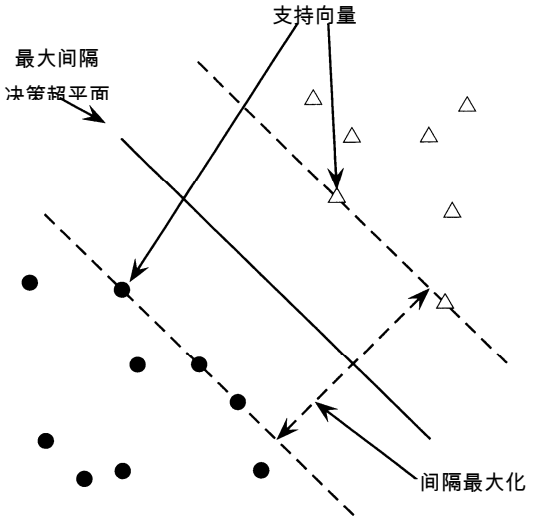


图 15-1　分类器间隔两端的 5 个点是支持向量

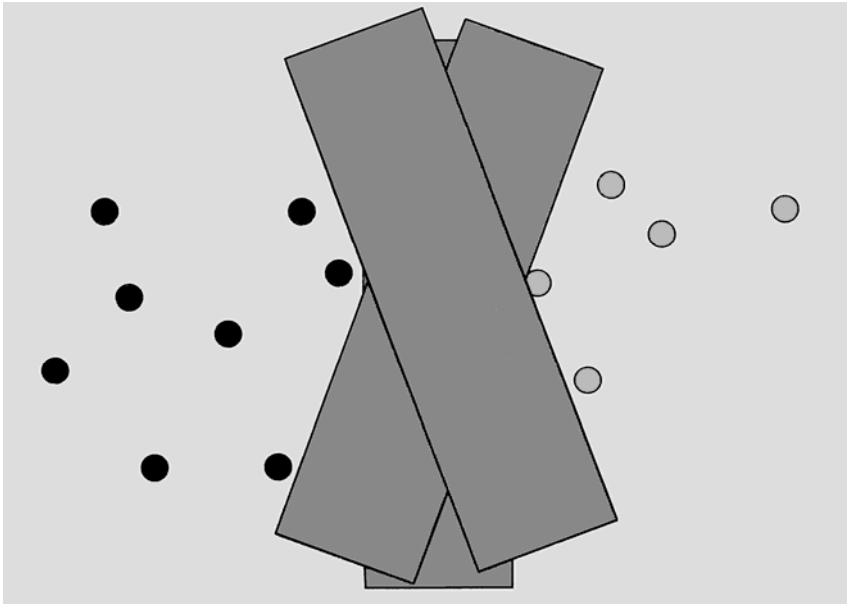


图 15-2　大间隔分类器的直观示意图。强调大间隔会降低模型的记忆能力：宽决策面放置的角度范围比会比决策超平面要小（参考图 14-8）

下面我们将对SVM进行形式化处理。决策超平面可以通过截距b和与它正交的法向量$\vec{w}$来定义。这个法向量在机器学习领域通常被称为权重向量（weight vector）。为了在所有超平面中选择与法向量正交的超平面，我们将给定截距b。由于超平面与法向量正交，因此超平面上所有的点$\vec{x}$都满足$\vec{w}^{\mathrm{T}}\vec{x}=-b$。假定训练集上的所有点为$\mathbb{D}=\{(\vec{x}_i, y_i)\}$，其中$\vec{x}_i$是点，$y_i$是对应的类别标签[①]。在SVM中，两类标签通常表示成+1 和−1（而不是 1 和 0），截距往往明确地表示为b（而不是将它作为一个特征整合到权重向量$\vec{w}$中）。如果采用上述表示，那么数学上的推导就更清晰，下面我们马上就会看到这一点。线性分类器可以表示成：

$$f(\vec{x})=sign(\vec{w}^{\mathrm{T}}\vec{x}+b)。\qquad (15\text{-}1)$$

结果为−1 时表示一个类别，为+1 时则表示另一个类别。

如果一个点离决策边界很远，那么我们就有信心对它进行分类决策。在给定数据集合决策超平面的情况下，第 i 个点$\vec{x}_i$相对于超平面$<\vec{w},b>$的函数间隔（functional margin）定义为$y_i(\vec{w}^{\mathrm{T}}\vec{x}_i+b)$。一个数据集的函数间隔是数据集点中最小函数间隔的两倍（两倍这个因子来自对整个间隔的计算，参考图 15-3）。但是，这种定义还存在一个问题，即具体的值没有大小限制，这样的话就可以通过放大$\vec{w}$和b来获得我们所需要的任意函数边界。例如，如果将$\vec{w}$和b分别替换成$5\vec{w}$和$5b$，那么函数间隔就变成$y_i(5\vec{w}^{\mathrm{T}}\vec{x}_i+5b)$，即值变大了五倍。这表明需要对向量$\vec{w}$的大小进行限制。要想理解其具体做法，首先从实际的几何图形看起。

① 正如 14.1 节所讨论的那样，我们给出的是向量空间中一般情况下的点，但是如果这些点是长度归一化的文档向量，那么所有的计算都发生在单位球体的表面，而决策面和球体表面相交。

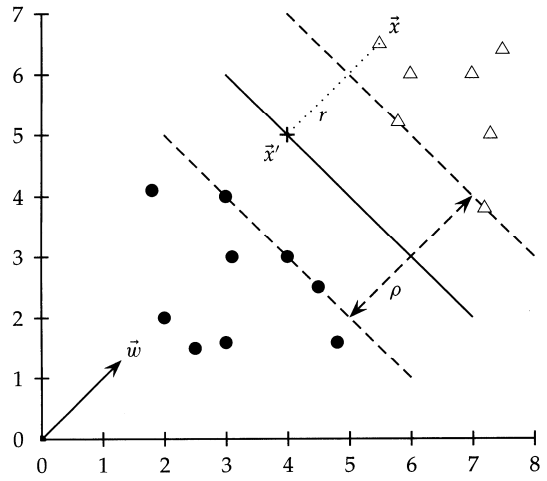


图 15-3　点到决策边界（ρ）的几何间隔（r）

一个点 $\vec{x}$ 到决策边界的欧氏距离是多少？在图 15-3 中，我们把这个距离记为 r。我们知道，点到超平面的最短距离垂直于该平面，也就是说和 $\vec{w}$ 平行。这个方向的单位向量是 $\vec{w}/|\vec{w}|$。图中的点线也就是向量 $r\vec{w}/|\vec{w}|$的平移结果。将超平面上离 $\vec{x}$ 最近的点标记为 $\vec{x}'$，于是有：

$$\vec{x}' = \vec{x} - yr\frac{\vec{w}}{|\vec{w}|'}。\qquad (15\text{-}2)$$

其中，乘以 y 的结果将改变在决策面上下的点 $\vec{x}$ 所对应的符号。另外，由于 $\vec{x}'$ 在决策边界上，因此有 $\vec{w}^{\mathrm{T}}\vec{x}'+b=0$，于是：

$$\vec{w}^{\mathrm{T}}\left(\vec{x} - yr\frac{\vec{w}}{|\vec{w}|}\right)+b=0。\qquad (15\text{-}3)$$

对r求解得[①]：

$$r = y\frac{\vec{w}^{\mathrm{T}}\vec{x}+b}{|\vec{w}|}。\qquad (15\text{-}4)$$

前面提到过，距离分界面最近的点是支持向量。分类器的几何间隔（geometric margin）就是中间空白带的最大宽度，该空白带可以用于将两类支持向量分开。也就是说，它是等式（15-4）中 r 的最小值的两倍，也等价于最宽间隔器的宽度（参见图 15-2）。很显然，不管参数如何缩放，几何间隔总是一个不变量。比如：即使将 $\vec{w}$ 和 b 分别替换成 $5\vec{w}$ 和 $5b$，其几何间隔仍然保持不变，这是因为它已经基于 $\vec{w}$ 的长度做了归一化处理。这也意味着我们可以根据需要对 $\vec{w}$ 进行缩放处理，而不用担心这样做会对几何间隔造成影响。在这里我们主要采用单位向量（参见第 6 章）的表示方法，即要求$|\vec{w}|=1$。在这种情况下，几何间隔等于函数间隔。

由于可以将函数间隔缩放到任意区间，为了方便解决大规模 SVM 问题，我们要求所有点的函数间隔至少为 1，而且至少有一个点的函数间隔为 1。这样的话，对于数据中的所有点，都有：

① $|\vec{w}|=\sqrt{\vec{w}^{\mathrm{T}}\vec{w}}$。

$$y_i(\vec{w}^{\mathrm{T}}\vec{x}_i+b)\ \mathrm{H}\ 1。 \qquad (15\text{-}5)$$

除此之外，还需存在支持向量使得上式等号成立。由于每个点到超平面的距离为 $r_i = y_i(\vec{w}^{\mathrm{T}}\vec{x}_i+b)/|\vec{w}|$，因此几何间隔的大小是 $\rho=2/|\vec{w}|$。我们的目标仍然是最大化这个间隔，也就是说，我们要寻找 $\vec{w}$ 和 b 来满足：$\rho=2/|\vec{w}|$极大化，对所有的 $(\vec{x}_i, y_i) \in$ λ，有 $y_i(\vec{w}^{\mathrm{T}}\vec{x}_i+b)$H1。

最大化 $2/|\vec{w}|$等价于最小化$|\vec{w}|/2$。于是我们最后得到SVM对应的最小化问题 ①：

寻找 $\vec{w}$ 和 b，使得

$$\rho=\frac{1}{2}\vec{w}^{\mathrm{T}}\vec{w} \text{ 时最小，对所有的 } \{(\vec{x}_i, y_i)\} \text{，有 } y_i(\vec{w}^{\mathrm{T}}\vec{x}_i+b)\mathrm{H}1。 \qquad (15\text{-}6)$$

现在我们面对的是一个在线性约束条件下的二次优化问题。二次优化（quadratic optimization）是数学上一个经典的优化问题，存在很多求解方法。原则上我们基于 QP（Quadratic Programming，二次规划）库来建立 SVM，但是近年来这一领域也有很多进展，它们可以将 SVM 中 QP 的结构信息引入到实现中。这样做的结果是，现在出现了很多虽然更复杂但是更快、扩展性也更好的程序库，它们都可以用于建立 SVM，几乎所有的人也都是用这些库来构建 SVM 模型。在这里，我们并不给出这些实现算法的细节。

然而，理解这种优化问题的解决方式很有益处。该问题在在优化过程中，建立了一个对偶问题，其中原问题中的每个约束条件 $y_i(\vec{w}^{\mathrm{T}}\vec{x}_i+b)\mathrm{H}1$ 在对偶问题中都对应一个拉格朗日因子 α_i。

寻找满足下列条件的 $\alpha_1, \ldots, \alpha_N$，使得 $\sum\alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \vec{x}_i^{\mathrm{T}}\vec{x}_j$ 极大化：

$\sum_i \alpha_i y_i = 0$；对所有的 $1\leq \alpha_i \leq N$，有 $\alpha_i\mathrm{H}0$。

最后的求解结果形式如下： （15-7）

$$\vec{w}=\sum\alpha_i y_i\vec{x}_i \ ； \qquad (15\text{-}8)$$

对于任意 $\alpha_k \neq 0$ 的点 $\vec{x}_k$ 有，$b = y_k - \vec{w}^{\mathrm{T}}\vec{x}_k$。

上述结果中，大部分 α_i 的值都为 0。每个非零的 α_i 所对应的 $\vec{x}_i$ 是一个支持向量。于是，最后的分类函数为：

$$f(\vec{x}) = sign\left(\sum_i \alpha_i y_i x_i^{\mathrm{T}}\vec{x}+b\right)。 \qquad (15\text{-}9)$$

不论是在对偶问题的最大化目标中还是最后的分类函数中，都包含着点之间的内积操作（$\vec{x}$ 和 $\vec{x}_i$ 或者 $\vec{x}_i$ 和 $\vec{x}_j$），这也是使用数据的唯一方式，我们还将在后面谈到这一点的重要性。

我们再简单重述一遍，给定训练集后，它能够确定唯一的最优超平面，超平面的确定过程则可以通过一个二次线性优化问题来实现。给定一个新的点 $\vec{x}$，公式（15-1）或公式（15-9）中的分类函数 $f(\vec{x})$ 实际上是计算点到超平面法线上的投影，计算结果的符号决定了点的类别归属。如果某个点在分类器的间隔（或者事先给出的某个用于最小化分类器错误的阈值t）之内，分类器可以返回“不知道”，而不一定非要给出两类中的一类。$f(\vec{x})$ 的值也可以转化成某个分类概率，通过Sigmoid函数拟合转化是一种常规做法（Platt 2000）。由于间隔是固定的，所以如

① 由于$|\vec{w}|\sqrt{\vec{w}^{\mathrm{T}}\vec{w}}$，所以最小化$|\vec{w}|/2$相当于最小化$\sqrt{\vec{w}^{\mathrm{T}}\vec{w}}/2$，也相当于最小化$\vec{w}^{\mathrm{T}}\vec{w}/2$。——译者注

果模型中包含了不同源的特征维的话，就需要对某些维进行细致的重缩放（rescale）处理 ①。当然，假如所有的文档点都落在单位球体表面的话，这就不成问题。

例 **15-1**　考虑为图 15-4 中的极小规模的数据建立 SVM。从几何上来求解，对于图中的例子，最大间隔的权重向量将与连接两个类的最短线段平行，即与（1，1）及（2，3）的连线平行，于是得到一个权重向量（1，2）。并且该线段与最优决策面垂直交于线段中点。因此，它会经过点（1.5，2）。于是，SVM 的决策边界是：

$$y = x_1+2x_2-5.5。$$

从算术上来求解，我们的目标是在约束 $sign(y_i(\vec{w}^{\mathrm{T}}\vec{x}_i+b))$H1 下，将 $|\vec{w}|$ 最小化。当两个支持向量正好满足上述约束条件的等值条件时，最小化成立。我们更进一步了解到，对于某个 a，$\vec{w}=(a，2a)$。因此，我们有：

$$a+2a+b=-1$$
$$2a+6a+b=1。$$

解上述方程组得，a=2/5，b= −11/5，因此最终的最优超平面为 $\vec{w}$=(2/5, 4/5)，b= −11/5。

最终的间隔为 $\rho=2/|\vec{w}|=2/\sqrt{4/25+16/25}=2/(2\sqrt{5}/5)=\sqrt{5}$。这个结果可以在图 15-4 上得到验证。

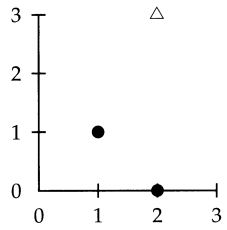


图 15-4　只包含 3 个数据点的训练集上的 SVM

习题 **15-1 [*]**　一个数据集里支持向量的最小数目是多少（此时的数据集每个类别中都包含实例）？

习题 **15-2 [**]**　SVM 中能够使用核函数（参考 15.2.3 节）的基础是，分类函数能够写成公式（15-9）的形式，其中，当数据规模较大时，大部分αi 都是 0。利用例 15-1，请说明为什么分类函数可以写成这种形式，$\vec{x}$ 是其中的唯一变量。

习题 **15-3 [**]**　安装某个 SVM 包，比如 SVMlight （http://svmlight.joachims .org/），对例子 15-1 建立一个 SVM 分类器。请确认程序会给出与文中一样的结果。对于 SVMlight 或者其他包来说，训练数据格式是一样的，它们的格式如下所示。

+1 1:2 2:3
−1 1:2 2:0

① 这里的意思是说，如果 SVM 的特征有不同来源，这些特征的原始取值范围可能相差很大，通过缩放处理可以转到具有可比性的同一取值空间。——译者注

−1 1:1 2:1

SVMlight 的训练命令为：

svm_learn -c 1 -a alphas.dat train.dat model.dat。

-c 1 选项是为了使用 15.2.1 节中将要讨论的松弛变量。核对一下归一化后的权重向量，看看是否和例 15-1 一致。检查一下最后输出的 alphas.dat 文件，看看 α_i 是否和习题 15-2 的解答一致。

15.2 支持向量机的扩展

15.2.1 软间隔分类

对于在文本分类中很普遍的高维空间问题来说，有时数据是线性可分的。但是一般情况下这都不成立，而且即使线性可分成立，我们也可能优先考虑那些能够将大部分数据分开而忽略一些奇异噪音文档的解决方案。

如果训练数据集不非线性可分，常规的做法是允许决策间隔犯一些错误（有些离群点或者噪音点在间隔内部或者在决策面的错误一方）。于是，我们要根据每个错分例子满足间隔的程度（参见公式 15-5）定义其惩罚代价。为实现这一目的，我们引入了松弛变量 ξ_i，一个非零的 ξ_i 表示允许 $\vec{x}_i$ 在未满足间隔需求下的惩罚量或代价因子。参见图 15-5。

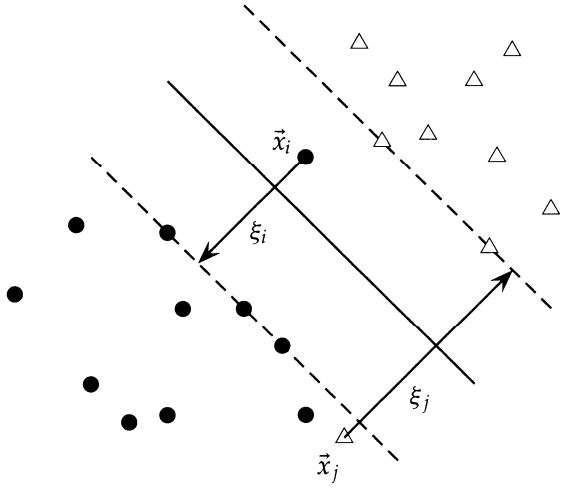


图 15-5

带松弛因子的 SVM 优化问题的形式化结果如下：

寻找 $\vec{w}$、b 及 ξ_iH0，使得

$$\rho=\frac{1}{2}\vec{w}^{\mathrm{T}}\vec{w}+C\sum_i\xi_i \qquad (15\text{-}10)$$

极小化，对所有的 $\{(\vec{x}_i, y_i)\}$，有 $y_i(\vec{w}^{\mathrm{T}}\vec{x}_i+b)H1-\xi_i$。

于是，优化问题就转化为间隔的宽度和允许进入间隔内的数据点数目之间的均衡性问题。

对于点 $\vec{x}_i$ 来说，通过设置 $\xi_i > 0$ ，间隔的宽度可以小于 1，但是在最小化过程中要付出惩罚代价 $C\xi_i$。ξ_i 的和是训练错误的上界。基于软间隔的 SVM 在训练错误和间隔大小之间权衡，并使目标函数最小化。参数 C 是正则化（regularization）因子，可以通过它来控制过拟合问题：如果 C 变大，那么会对出现在间隔内的点的惩罚较大，即更尊重数据本身，当然这时候的代价就是减少几何间隔。而当 C 很小时，则很容易通过松弛变量来考虑噪音点，此时可以得到能够对大部分数据点建模的更宽的间隔。

软间隔分类的对偶问题是

寻找 $\alpha_1,\ldots,\alpha_N$，使得 $\sum\alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \vec{x}_i^{\mathrm{T}}\vec{x}_j$ 极大化，此时要满足下列条件：$\sum_i \alpha_i y_i = 0$ ；对所有的 $1 \leq i \leq N$，有 $0 \leq \alpha_i \leq C$。（15-11）

不管是松弛变量 ξ_i 自己还是它们的拉格朗日因子都不在上述式子中出现。和前面的最大间隔分类器相比，上述最优化问题中唯一的改变就是每个支持向量的拉格朗日因子都有常数上界 C。同前面一样，与非零 α_i 对应的 $\vec{x}_i$ 向量就是支持向量。上述对偶问题的解如下：

$$\vec{w} = \sum \alpha y_i \vec{x}_i \tag{15-12}$$

$b = y_k(1-\xi_k) - \vec{w}^{\mathrm{T}}\vec{x}_k$，其中 $k = \arg\max_k \alpha_k$。

同样，在分类时不需要显式地使用向量 $\vec{w}$，而只需要计算数据点的点积即可（参考公式 15-9）。

通常情况下，支持向量只是训练数据中的很小一部分。但是，对于一个非线性问题或是间隔很小的问题，任何一个分错的或在间隔内部的点都会对应一个非零 α_i。如果这类点的数目变多的话，那么对于 15.2.3 节提到的非线性情况，SVM 在分类时的速度将会大大降低。

表 15-1　包括 SVM 在内的多个分类器的训练及测试的复杂度。训练时间是指学习方法在 λ 上学到分类器的时间，而测试时间指的是分类器对一篇文档分类的时间。对 SVM 来说，假定多类分类是通过 $|\amalg|$ 个一对多分类器完成的。L_{ave} 是文档的平均词条数目，M_{ave} 是文档的平均词汇量（即词项个数）。L_{a} 及 M_{a} 分别是测试文档的词条个数和词汇量

分类器	阶段	处理策略	时间复杂度
NB	训练		$\Theta(\|\lambda\|L_{\mathrm{ave}}+\|\amalg\|\|V\|)$
NB	测试		$\Theta(\|\amalg\|M_{\mathrm{a}})$
Rocchio	训练		$\Theta(\|\lambda\|L_{\mathrm{ave}}+\|\amalg\|\|V\|)$
Rocchio	测试		$\Theta(\|\amalg\|M_{\mathrm{a}})$
kNN	训练	预处理	$\Theta(\|\lambda\|L_{\mathrm{ave}})$
kNN	测试	预处理	$\Theta(\|\lambda\|M_{\mathrm{ave}}M_{\mathrm{a}})$
kNN	训练	不做预处理	$\Theta(1)$
kNN	测试	不做预处理	$\Theta(\|\lambda\|L_{\mathrm{ave}}M_{\mathrm{a}})$
SVM	训练	传统方法	$O(\|\amalg\|\|\lambda\|^3 M_{\mathrm{ave}})$；经验上≈ $O(\|\amalg\|\|\lambda\|^{1.7}M_{\mathrm{ave}})$
SVM	训练	切平面法（cutting plane）	$O(\|\amalg\|\|\lambda\|M_{\mathrm{ave}})$
SVM	测试		$O(\|\amalg\|M_{\mathrm{a}})$

在表 15-1 中显示了线性SVM训练和测试的复杂度。SVM的训练时间主要取决于二项规划的时间。因此，理论上的复杂度和实际上的复杂度会随着所使用的具体方法的变化而变化。常规二项规划的实现时间复杂度是数据集大小的三次方（Kozlov等人 1979）。近年来一些有关SVM

训练的工作试图降低时间复杂度，它们往往通过近似方法来求解，也取得了令人满意的结果。通常来说，经验的复杂度大概是$O(|\lambda|^{1.7})$（Joachims 2006a）。然而，传统的SVM算法的超线性时间复杂度使得它很难也不可能在大规模训练集上使用。其他的与训练集大小呈线性关系的SVM传统解决方法很难扩展到特征数目很多的情况，而特征数目很多却是文本问题的一个常规特点。然而，一个基于切平面（*cutting plane*）技术的SVM新训练算法给出了令人鼓舞的结果，它可以在训练集大小和文档中非零特征数目的线性时间以内运行（Joachims 2006a）。尽管如此，二次优化的实际速度仍然远远小于NB模型中的简单词项计数速度。将SVM算法扩展到非线性SVM（参见下一节）通常会将训练的时间复杂度提高一个因子$|\lambda|$（这是因为要计算样本点之间的内积），因此非线性SVM使用起来有点不太现实。实际操作中一个降低代价的方法可以是，通过直接实现（materializing）高阶特征来训练一个线性模型[①]。

15.2.2 多类情况下的支持向量机

SVM 天生就是一个二类分类器。我们在 14.5 节提供的方法可以作为将其用于多类问题的传统做法。具体地说，实际操作中最普遍的技术是建立$|\coprod|$个“一对多”（one-versus-rest，或者称“一对其余”）的二类分类器，这种分类器通常也被称为“OVA”（one-versus-all，“一对全部”）分类器，并选择对测试文档具有最大分类间隔的分类器的分类结果。另外一种方法是在两两类别之间建立$|\coprod|(|\coprod|-1)/2$个“一对一”（one-versus-one）分类器，测试时会根据这些分类器的投票结果选择最后的类别。和 OVA 方法相比，尽管这种方法需要训练出$|\coprod|(|\coprod|-1)/2$个分类器，但是由于每个分类器训练时的训练集要小得多，所以实际上训练所有分类器的时间可能会更少。

但是，以上两种方法都不是解决多类问题的最简便的方法。一种更好的方法是首先基于输入特征及其类别推导出特征向量$\Phi(\vec{x},y)$，然后在这个特征空间上构造一个二类分类器。这种构造的结果可以直接得到一个支持多类的 SVM。测试时，分类器选择得分最大的类别$y=\text{argmax}_{y'}\ \vec{w}^{\mathrm{T}}\Phi(\vec{x},y')$。训练时的分类间隔是该式在正确类及最近的其他类上得分的差值，因此二次规划求解时需要满足：

$$\forall i \forall y \neq y_i \quad \vec{w}^{\mathrm{T}}\Phi(\vec{x}_i, y_i) - \vec{w}^{\mathrm{T}}\Phi(\vec{x}_i, y) \geq 1 - \xi_i$$

这种通用的方法可以应用到任何一种线性分类器的多类扩展中。另外，这也是某些情况下分类泛化的一个实例，这时候类别可能并不是一些独立的类别标签，而可以是定义了相互之间关联的任意结构化对象。在 SVM 领域，这些工作归属于结构化 SVM（structural SVM）的范畴，这部分内容我们在 15.4.2 节中还会提到。

15.2.3 非线性支持向量机

到目前为止，我们讨论的数据集都是线性可分的（最多包括少数例外点或噪音点）。对这些数据集合我们都能够很好地进行处理。然而，如果数据集不允许线性分类器分类时怎么办？我们看一个一维空间中的例子。图 15-6 中上面的数据集显然可以被线性分类器直接分开，而中间的数据集却显然不可能被线性分类器直接分开。我们需要做的就是将它们间隔开。一个解决该

① 这里实现特征指的是直接计算高阶互操作特征然后将它们放入到线性模型当中。

问题的方法是将数据映射到一个高维空间并在此空间上使用线性分类器将数据分开。例如，图 15-6 中最下面的那个图就表明，如果采用二次函数将原始数据映射到二维空间（也可以采用极坐标的方法进行映射），那么在新空间中就可以很容易将数据分开。也就是说，一个面对这类问题时的一般思路就是，将原始的特征空间映射到某个更高维的线性可分的特征空间上去。当然，在映射的同时我们希望能保留与数据相关性有关的特征维。这样的话，最终的分类器仍然有很好的泛化能力。

SVM 以及很多其他的线性分类器可以通过一个非常高效的方法来映射到高维空间中去，这种方法称为核技巧（kernel trick）。当然它实际上并不是一个技巧，而是使用了我们见到过的数学方法。SVM 线性分类器依赖于数据点之间的内积操作。令 $K(\vec{x}_i,\vec{x}_j)=\vec{x}_i{}^{\mathrm{T}}\vec{x}_j$，那么支持向量机中的分类器可以写成：

$$f(\vec{x})=sign\left(\sum_i \alpha_i y_i K(\vec{x}_i,\vec{x})+b\right)。\qquad (15\text{-}13)$$

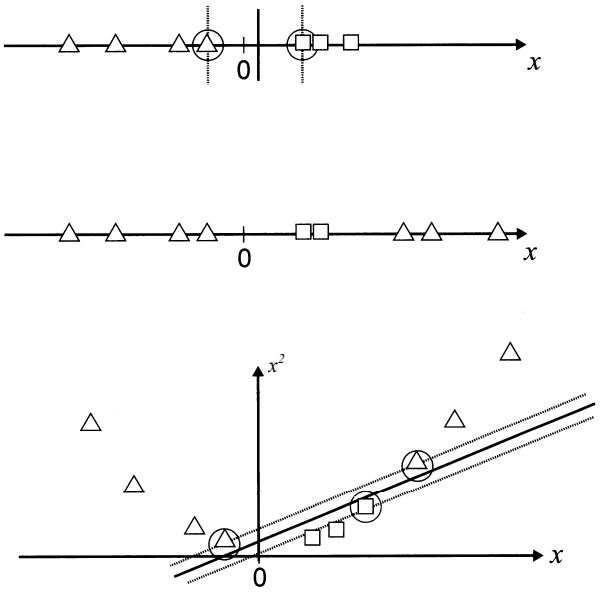
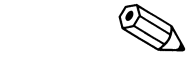


图 15-6　将非线性可分的数据映射到高维空间中使它们可分

假设我们通过某个映射函数Φ可以将原始空间的点映射到新空间，即Φ: $\vec{x}\mapsto\Phi(\vec{x})$。那么，新空间下的点积计算为$\Phi(\vec{x}_i)^{\mathrm{T}}\Phi(\vec{x}_j)$。如果能够证明该点积（也就是一个实数）能够通过原始数据点简单高效地计算出来，那么我们就不必真的要将$\vec{x}\mapsto\Phi(\vec{x})$。相反，我们可以直接通过$K(\vec{x}_i,\vec{x}_j)=\Phi(\vec{x}_i)^{\mathrm{T}}\Phi(\vec{x}_j)$来计算，然后在公式（15-13)中直接使用这个值。核函数K实际上对应于新特征空间上的内积 ①。



例 15-2　二维空间上一个二次核函数的例子。对于二维的向量$\vec{u}=(u_1\ u_2)$，$\vec{v}=(v_1$

① 核函数定义域在原始空间，其输出结果却是新空间下的内积。——译者注

$v_2)$，考虑 $K(\vec{u},\vec{v})=(1+\vec{u}^{\mathrm{T}}\vec{v})^2$。下面我们说明这是一个核函数，也就是说，对于某个函数 ϕ 有 $K(\vec{u},\vec{v})=\phi(\vec{u})^{\mathrm{T}}\phi(\vec{v})$。考虑 $\phi(\vec{u})=(1\ u_1^2\ \sqrt{2}u_1u_2\ u_2^2\ \sqrt{2}u_1\ \sqrt{2}u_2)$，我们有

$$\begin{aligned}K(\vec{u},\vec{v}) &= (1+\vec{u}^{\mathrm{T}}\vec{v})^2\\ &= 1+u_1^2v_1^2+2u_1v_1u_2v_2+u_2^2v_2^2+2u_1v_1+2u_2v_2\\ &= (1\ u_1^2\ \sqrt{2}u_1u_2\ u_2^2\ \sqrt{2}u_1\ \sqrt{2}u_2)^{\mathrm{T}}(1\ v_1^2\ \sqrt{2}v_1v_2\ v_2^2\ \sqrt{2}v_1\ \sqrt{2}v_2)\\ &= \phi(\vec{u})^{\mathrm{T}}\phi(\vec{v})\ 。\end{aligned} \qquad (15\text{-}14)$$

下面我们将利用泛函分析的语言来介绍有效核函数所满足的条件。核函数有时可以更精确地表达成 Mercer 核(Mercer kernels)，这是因为它们必须满足 Mercer 条件：对于任何 $\int g(\vec{x})^2 d\vec{x}$ 有穷的函数 $g(\vec{x})$，必须要有

$$\int K(\vec{x},\vec{z})g(\vec{x})g(\vec{z})d\vec{x}d\vec{z}\ \mathrm{H}\ 0。 \qquad (15\text{-}15)$$

核函数 K 必须连续、对称，且拥有一个正定格莱姆矩阵（positive definite gram matrix）。这种条件也意味着存在到再生核希尔伯特空间（reproducing kernel Hilbert space，即希尔伯特空间在内积操作下的闭空间）的映射，在该空间下的内积结果等于函数 K 的结果。如果核函数不满足 Mercer 条件，那么相应的二次规划问题可能无解。如果要深入理解这些内容，可以阅读 15.5 节提到的 SVM 相关书籍。否则，只需要知道有关核的 90%工作都用到了下面定义的两类基于向量的函数，并且这两类方法都能定义合法的核函数。

最常用的两类核函数是多项式核函数（polynomial kernels）和径向基核函数（radial basis functions）。多项式核函数的形式是：$K(\vec{x},\vec{z})=(1+\vec{x}^T\vec{z})^d$，当 d=1 时是个线性核函数，本节之前的 SVM 实际上就是采用了线性函数，而这里的常数 1 只是改变了阈值。当 d=2 时是个二次核函数，其用途非常普遍。例 15-2 中给出了一个二次核函数。

径向基函数的一个最普遍的形式是采用高斯分布，计算公式为：

$$K(\vec{x},\vec{z})=e^{-(\vec{x}-\vec{z})^2/(2\sigma^2)}。 \qquad (15\text{-}16)$$

径向基函数等价于将数据映射到无穷维的希尔伯特空间，因此我们无法像二项式核函数一样给出径向基函数的具体形式。除了上面两类核函数之外，还有其他建立核函数的有趣工作，其中某些工作在文本应用中也取得了令人满意的结果，其中一个具体的例子是字符串核函数（参见 15.5 节）。

SVM 领域有着自己的表述语言，它们和机器学习领域的表述语言相差很大。这些术语深深扎根于数学，但是很重要的一点是，不要对这些术语心存畏惧。实际上，我们正在讨论的问题非常简单。多项式核函数可以让我们对特征的联合（联合个数不超过多项式的阶）建模。也就是说，如果想对词的共现建模，而词项共现能够为分类提供单个词所不能提供的区分性信息，比如 operating AND system 或者 ethnic AND cleansing，那么我们就必须用到二次核函数(quadratic kernel)。如果三个词的共现能够提供区分性信息，则需要用到三次核函数(cubic kernel)。同时，

我们仍然能保留基本特征的作用。在大部分文本应用中，上述做法可能没什么用，但是它来自于数学推导，很可能也不会有害处。径向基函数允许从圆或球面上选取特征，尽管由于多个此类特征之间的交互，决策边界变得更加复杂。字符串核函数得到的特征是词项的字符子串序列。这些核函数都是非常易懂的概念，并且已经以不同的名称运用于很多其他地方。

15.2.4　实验结果

在 13.6 节当中，我们已经通过实验结果表明SVM是一个效果非常好的文本分类器。表 13-9 给出的是Dumais 等人（1998）的研究结果，它清楚地表明SVM效果最好。这也是从此为SVM树立良好声誉的一系列工作之一。另一个对SVM文本分类进行规模扩展（scaling）和评估的开拓性工作是（Joachims 1998）。我们在表 15-2[①]中给出了其结果的一部分。Dumais 等人（1998）的工作中使用MI来进行特征选择（参考 13.5.1 节），并将特征数目降到非常低的水平，而与之不同的是，Joachims则使用了大量词项特征。相对于其他线性分类方法（如NB），线性SVM重现了 14.6 节所示的成功结果。看来基于简单词项特征的方法还有很长的路可以走。值得再次提醒的是，不同论文中同一机器学习方法的结果有一定程度的差别。具体来说，和其他研究者的结果相比，（Joachims 1998）中的NB方法表现很差，而表 13-9 中的结果基本上更具有代表性。

表15-2　SVM分类算法的正确率-召回率等值点上的F_1值（Joachims 2002a, p. 114）。下面给出了Reuters-21578语料中10个最大类上的F_1结果和所有90个类别的微平均值（rbf指径向基核函数）

					Linear SVM		rbf-SVM
	NB	Rocchio	dec.Trees	kNN	C=0.5	C=1.0	σ≈7
earn	96.0	96.1	96.1	97.8	98.0	98.2	98.1
acq	90.7	92.1	85.3	91.8	95.5	95.6	94.7
money-fx	59.6	67.6	69.4	75.4	78.8	78.5	74.3
grain	69.8	79.5	89.1	82.6	91.9	93.1	93.4
crude	81.2	81.5	75.5	85.8	89.4	89.4	88.7
trade	52.2	77.4	59.2	77.9	79.2	79.2	76.6
interest	57.6	72.5	49.1	76.7	75.6	74.8	69.1
ship	80.9	83.1	80.9	79.8	87.4	86.5	85.8
wheat	63.4	79.4	85.5	72.9	86.6	86.8	82.4
corn	45.2	62.2	87.7	71.4	87.5	87.8	84.6
microavg.	72.3	79.9	79.4	82.6	86.7	87.5	86.4

15.3　有关文本文档分类的考虑

文本分类被广泛应用于商业领域，垃圾邮件过滤可能是目前最普遍的应用之一。Jackson 和 Moulinier (2002) 指出：“能够基于内容对文档进行自动分类的商业价值毋庸置疑，在公司内网、

① 这些结果用等值点的 F_1 值表示（参见 8.4 节）。很多研究人员并不愿意采用这个指标来度量文本分类的结果，它在计算中可能包含一些插值而不是系统的参数调节。我们还不清楚为什么采用这个值而不是根据任务需要采用最大的 F_1 值或者召回率-正确率曲线上的其他点来表示。在（Joachims 1998）的这个早期的工作中，利用高阶多项式或者径向基核函数能够显著地提高效果。此时 SVM 采用的也是硬间隔。对于软间隔，一个 C=1 的简单线性 SVM 就获得了最好的结果。

政府机构及互联网出版等机构或领域中存在大量的潜在应用。”

前面我们对于分类的讨论主要集中于介绍不同的机器学习方法，而对文档分类的具体特征并没有特别关注。对于教材来说，这种取舍是合理的，但是对于应用开发者而言就有点本末倒置。一个普遍的事实就是，采用领域相关的文本特征在性能上会比采用新的机器学习方法获得更大的提升。Jackson 和 Moulinier（2002）指出：“对数据的理解是分类成功的关键之一，然而这又是大部分分类工具供应商非常不擅长的领域。市场上很多所谓的通用分类工具并没有在不同类型的内容上进行广泛的测试。”本节当中，我们将回到文本分类的实际应用中来，并介绍实际问题的可能解决方案及如何使用与应用相关的启发式策略。

15.3.1 分类器类型的选择

当面对一个建立分类器的需求时，第一个要问的问题就是：训练数据有多少？一点都没有？很少？还是很多？或者数据量非常大且每天都在增长？通常来说，实际应用中建立机器学习分类器的一个最大挑战就是构建或者获取足够的训练数据。对于很多问题和算法来说，为了获得高性能的分类器，每个类都需要成百上千的训练文档。另外，在实际环境下也常常包括大量的类别集合。一开始我们假设分类器需要尽快提供；如果有足够多的时间用于系统实现的话，那么大部分时间可能要花在数据的准备上。

如果没有标记好的训练文档，但是有数据所在领域的专家的话，那么不要忘了可以采用人工编写规则的解决方法。也就是说，可以给出第 13 章一开始所讨论的固定查询来描述规则，例如：

IF (wheat OR grain) AND NOT (whole OR bread) THEN c = grain

实际中的规则要比这个例子长很多，并且可以采用远比布尔表达式复杂的查询语言来表示，其中还包括数值得分的使用。经过精心调整（也就是说，人们可以在开发集上调整规则）之后，利用这些规则分类的精度可以非常高。Jacobs 和 Rau（1990）报告说，采用基于人工规则的分类方法，有关收购(takeover)主题的文章的识别正确率可以达到 92%，召回率可以达到 88.5%。Hayes 和 Weinstein (1990) 也给出了在 675 个类的路透社新闻文档上获得 94%召回率及 84%正确率的报道。然而，要构造这样好的人工规则需要做大量的工作。一个基本合理的估计数字是每个类别需要两天的时间，由于类别中的文档内容会发生漂移，所以必须还要利用很多额外的时间去维护规则。

如果拥有的训练数据非常少，而又要训练出一个基于监督学习的分类器，那么机器学习理论指出，此时应该采用具有高偏差的分类器（参见 14.6 节的讨论）。例如，理论上和经验上的结果都表明，在这种情况下 NB 能取得较好的结果（Ng 和 Jordan 2001; Forman 和 Cohen 2004），尽管在实际中使用正则化（regularized）模型处理文本数据时不一定能观察到上述效果（Klein 和 Manning 2002）。不管怎样，此时诸如 kNN 的低偏差模型大概是不可取的。当然，无论采用何种模型，模型的质量始终会因训练数据有限而受到不利影响。

在这里，一个很有趣的理论解决方案是使用半监督的训练方法（semisupervised training

method)。这些方法包括自助法(bootstrapping)或在 16.5 节中将要讨论的 EM (the Expectation-maximization，期望最大化)算法。在这些方法中，系统有一些标注好的数据以及大量的有待学习的未经标注的数据。NB 的一个很大优点就是它能直接扩展成半监督学习算法。但对 SVM 来说，虽然也有半监督学习方面的工作，但是这些工作主要冠以直推式 SVM (*transductive* SVM)的名字。具体的内容可以参见参考文献部分。

通常，实际的解决方案是如何尽快地获得更多的已标注数据。一个最好的方法是将自己也放入到标注的过程中去，在这个过程中人们自愿为你标注数据，并且这也是他们正常工作的一部分。比如，很多情况下，人们会根据需要整理或者过滤邮件数据，这些动作能够提供类别相关的信息。由于人们很难完全理解分类器训练的目的，因此人工标注的方法在组织上往往非常困难。又由于人们并不是在一个实际的上下文场景中进行标注，因此标注结果的质量也往往不高。除了让人们标注全部或者部分随机样本数据之外，一个值得注意的研究动向是使用主动学习(active learning)的方法，即建立一个系统来确定应该标注的那些文档。通常情况下，这些文档主要指那些分类器不确定能否正确分类的文档。通过主动学习方法，能够将标注的开销降低 2 到 4 倍。不过这种方法的一个问题就是，对于训练某类分类器较好的文档往往并不适合于训练其他不同类型的分类器。

如果拥有较多的标注数据，那么我们就处于一个极其有利的位置，能够使用我们在前面所介绍的任何文本分类技术。比如，我们希望使用 SVM。但是，如果使用诸如 SVM 的线性分类器时，或许应该在这个机器学习的分类器之上再建立一个基于布尔规则的分类器。用户往往希望能够对不正确的结果直接进行调整，如果老板在电话的另一头要求马上对某个特定文档的分类结果进行修正的话，那么在不损害整体精确率的前提下，利用人工撰写规则直接调整会比 SVM 权重调整的方法容易得多。这就是像决策树这类能够产生用户可读规则的机器学习方法具有大量应用的原因。

如果具有极大规模的数据，那么分类器的选择也许对最后的结果没有什么影响，目前我们还不清楚是否有最佳的选择方法(参考 Banko 和 Brill 2001)。也许最好的方法是基于训练的规模扩展性或运行效率来选择。为达到这个目的，需要极大规模的数据。一个通用的经验法则是，训练数据规模每增加一倍，那么分类器的效果将得到线性的提高。但是对于极大规模的数据来说，效果提高的幅度会降低成亚线性。

15.3.2　分类器效果的提高

对于一个特定的应用来说，分类器的效果往往具有显著的提升空间，这可以通过使用领域或者数据集相关的特征来实现。文档常常包含对分类特别有用的域，也往往可以通过对特定的子词汇表进行特殊对待，从而对分类器的效果进行优化。

大规模高难度分类体系

如果文本分类问题仅包含少量具有区分度的类别，那么很多分类算法都可能取得很好的结果。但是实际的文本分类问题往往包含大量非常类似的类别。看到这里，读者可能会想到诸如 Web 目录（如 Yahoo!目录或 ODP（Open Directory Project）目录）、图书馆分类机制（杜威十进制分类法或美国国会图书馆分类法），或者用于法律和医学领域的分类机制。比如，Yahoo!分类目录包含了超过 200 000 个类别，并具有很深层次的分类体系。对大量相近的类别进行精确分类是一个固有的难题。

大多数大型分类体系都具有层次结构，利用该层级结构进行层次分类（hierarchical classification）是一个很有希望的做法。然而到现在为止，相对于只基于层级目录的叶节点的分类而言，这样做获得的效果提高幅度还非常有限[①]。当然，对于提高大型层次目录体系下分类的扩展性而言，层次分类的技术还是非常有用的。另外一种提高分类器在大型层次目录体系下的扩展性的简单方法是进行排他式特征选择（aggressive feature selection）[②]。在 15.5 节中将提供一些有关层次分类的参考文献。

在机器学习中，一个一般性的结果就是，仅仅假定不同分类器的错误在某种程度上是相互独立的，那么就可以通过组合这些分类器来略微提升分类精度。目前对多个分类器进行投票（voting）、装袋（bagging）及提升（boosting）的研究工作很多，这些工作可以参见参考文献部分。然而，最终可能需要一个自动和手工混合的解决方法来获得足够的分类精确率。这种情况下，一个普遍的做法是首先运行一个分类器，然后接受所有高置信度的判定结果，而将那些低置信度的结果放入某个队列供人工浏览来确定。这种过程也会自动地导出新训练文档，这些文档又可以用于机器学习分类器的下一个版本中去。然而，需要注意的是，这种做法中得到的训练数据显然并不是从文档空间随机抽样的结果。

文本中的特征

不论是 ad hoc 检索还是文本分类，其默认的特征都是词项。但是，对于文本分类来说，如果对特定的问题加入合适的额外特征，那么分类效果还可能会显著提高。同 IR 的查询语言不同，文本分类中的这些特征都会在分类器内部，不存在特征和用户直接交互的问题。上述过程通常称为特征工程（feature engineering）。目前来说，特征工程仍然依赖于人的技巧而不是机器学习的结果。好的特征工程往往可以大幅度提高分类器的效果，对于文本分类的某些重要应用来说尤其具有价值，这些应用包括垃圾邮件过滤和色情过滤等。

分类问题常常会包含大量的词项，这些词项能够很方便地聚类，并且对于分类而言具有相似的“投票”效果。典型的例子包括年份的提及、惊叹号字符串，或者像 ISBN，

① 以图 13-1 给出的小规模层次分类体系为例，叶节点类别包括 poultry 和 coffee，而高层类别包括 industries 等等。

② 简单地说，这种特征选择方法不仅要考虑每个特征自身的作用，而且不同特征之间的信息量要尽量不冗余。详细的介绍请参考 GabrilovichE. MarkovitchS. Text categonization with many redundant features: Using aggressive feature selection to make SUMs competitive with Cu.5 In Proceedings of the ?lst Internatimal Conference on Machine Learning(ICML-04)。——译者注

或是化学式一样更特定的子串等。通常来说，在分类器中直接使用这些特征会大大增加词汇量，但此时我们只想告诉分类器某种现象的发生而并不关注特殊子串本身（比如说只关注文档中是否存在化学式）。这种情况下，可以通过正则表达式对这些项进行匹配，并将它们转换为更知名的词条，来降低特征数目及特征的稀疏性。这样做往往能提高分类器的效果和效率。有时，所有的数字都转换成一个单一特征，但是不同的数字形式往往可以具有不同的值，比如 4 位数（往往对应年份）、基数和十进制小数等。同样的技术可以用于日期、ISBN 号及比赛比分等。

从另一个方向来考虑，某些词的一部分或者多词模式往往具有特殊的区分能力，对它们进行匹配往往导致特征的增加，但是这种增加往往是很有用的。词的一部分往往通过 k-gram 字符特征来匹配。这些特征可以提供分类线索，特别是当存在未定义词的时候。比如，一个以-rase 结尾的未定义词可能是一种酶的名称，即使它没有在训练集中出现。通过查找特定的普通词对（或许可以采用词之间互信息计算的方法，和 13.5.1 节的特征选择用法类似）可以发现好的多词模式，然后通过特征选择方法进行挑选。在单个词本身存在错误的类别提示信息时，上述多词模式特征的方法就很有用。比如，如果关键词 ethnic 对类别 food 和 arts 具有高度的指示作用，而 cleansing 又对类别 home 具有指示作用，但是搭配 ethnic cleansing 却表示类别 world news。有些文本分类器将命名实体识别的结果作为特征进行分类（参考第 10 章）。

诸如词干还原和小写转换之类的技术（参见 2.2 节）会对文本分类有用吗？同以往一样，最终的测试要在合适的测试集上进行经验性的评估。然而，需要注意的是，这些技术对分类结果有用的机会非常有限。对于 IR 来说，往往需要将 oxygenate 和 oxygenation 之类的词统一成一个形式，这是因为对于有关 oxygenation 的查询来说，它们当中任何一个词出现在文档中都是表明文档相关的良好线索。在给定丰富的训练集的情况下，词干还原对于文本分类的价值并不大。如果词的不同形式表达相似信号的话，那么对它们估计出来的参数权重也相差不大。像词干还原一样的技术只在数据稀疏时能起到补偿作用，此时它们也许会成为重要角色（在本节开始也提到了这一点），但是在文档分类中，一个词的不同形式往往表达了不同的指示作用。过度的词干还原很容易就会降低分类的性能。

文档中的域在文本分类中的使用

正如 6.1 节提到的那样，文档通常都有域，比如像主题和作者一样的邮件头信息，或者学术论文的标题和关键词等。在训练和分类中，文本分类器能够利用这些域信息来获益。

- 提高文档域的权重。在文本分类中，往往可以通过对不同的域设置不同的权重来提升分类的效果。其中，对标题词提高权重尤其有效（Cohen 和 Singer 1999）。一个经验法则就是在文本分类中将标题词的权重翻倍，这往往是一个很有效的做法。对于一段文本，尽管它并不属于清晰定义的文档域，但是文档的结构或内容却表明它非常重要，

也同样可以通过提高其中的词权重来获得效果的提升。Murata 等人（2000）指出，在 ad hoc 检索任务中，可以通过提高（新闻）文档中第一句话的权重来提高效果。

- 将文档域中的特征空间分开。对于文档域有两种策略可以使用。上面我们对出现在某些域中的词提高了权重。这意味着我们仍然使用同样的特征（也就是说，不同域的同一名称参数要综合在一起，这种做法称为参数集成 parameter tying），而只是对特定域中的特征出现赋予了更高的权重。另外一种方法是将不同域当中的特征及对应参数完全分开。理论上来说这种做法更加强大：当一个词出现在标题中时表示主题 Middle East，而在正文中出现时则表示 Commodities。然而，实际当中集成不同域的参数的方法往往更成功。采用独立的特征集同时也意味着 2 倍或更多的参数，而大部分参数在训练集上将会很稀疏，因此对它们的估计将会非常糟糕。而权重提升的方法却没有这类问题。另外，对于词出现在不同域而赋予不同的优先级不是太普遍的做法，主要应该调节它们的整体效果。尽管如此，最终的结果还是要依赖于训练数据的本质和数量，要因事论事。
- 与文本摘要的关联。在 8.7 节中，我们提到了文本摘要这个领域，以及大多数工作如何利用句子的位置和内容等特征来从原始的文本中提取重要的句子构成文本摘要。上述工作可以使人想到利用文档中的域来进行文本分类工作。比如 ,Kołcz 等人 (2000) 考虑了一种特征选择形式，这种形式可以仅仅基于某些域中的词汇进行文档分类。基于文本摘要的研究，他们考虑了 (i) 仅仅使用标题，(ii) 仅仅使用第一段，(iii) 仅仅使用包含最多标题词或关键词的段落，(iv) 前两段或者第一段和最后一段，或 (v) 包含最少标题词或关键词的所有句子。总的来说，这些基于位置的特征选择方法取得了和互信息（参见 13.5.1）相当的结果，也获得了极具竞争力的分类器。Ko 等人（2004）也从文本摘要出发，提出了对包含标题词或者其他文档中心词的句子提高权重的方法，得到的分类精度大概提高了 1%。由于这些句子大都在某种程度上属于文档的中心句子，所以即使是基于推测，上述做法也应该会有效果。

? 习题 **15-4 [**]** 垃圾邮件往往使用各种掩藏（*cloaking*）技术来逃避过滤器。其中一种方法就是拼凑或者替换一些字符来避开基于词的文本分类器的检查。比如，你会在垃圾邮件中看到如下的字符串。

Rep1icaRolex	bonmus	Viiiaaaagra	pi11z
PHARlbdMACY	[LEV]i[IT]l[RA]	se∧xual	ClAfLlS

讨论如何通过特征工程的方法来检测出上述掩藏方法。

习题 **15-5 [**]** 垃圾邮件散播者常常采用的另外一个策略是，在它们想发送的信息（比如 buying a cheap stock 或者其他什么内容）后面增加一段正常的文本内容（如一篇新闻）。为什么这种策略能够有效地对抗一般的分类器？如何在文本分类器中解决这个问题？

习题 **15-6[*]** 对于垃圾邮件分类器来说，还有哪些特征可能对分类有用？

15.4　ad hoc 检索中的机器学习方法

在第 6 章，我们主要通过人工建立词项和文档权重函数来进行 ad hoc 检索，这里我们可以将相关性的各种因素（余弦相似度、标题匹配度等）都看成是可用于学习的特征。将与查询集中每个查询相关的文档和不相关的文档作为例子输给分类器，便可以得到上述不同因素的相对权重。如果将问题考虑成由一系列的查询和文档对组成，并且给每对赋予了相关性判断：相关和不相关，那么也就可以将该问题看成文本分类问题。考虑成这样的分类方式并不是最好的，在 15.4.2 节中将给出另外一种方式。尽管如此，在给定相关材料的前提下，最简单的处理方法还是看成一个二类文本分类问题，并根据相关判定的置信度对文档进行排序。这种做法并不单纯是基于教学目的，确切地说该做法有时会在实际中使用。

15.4.1　基于机器学习评分的简单例子

本节当中，我们将把 6.1.2 节中的方法推广为基于机器学习的评分函数。在 6.1.2 节中，我们考虑过一个要将不同布尔相关值进行组合的场景，这里我们将考虑更一般的因子，并给出机器学习相关度的概念。具体地说，我们现在考虑的因子超出了 6.1.2 所介绍的查询项在不同文档域出现或不出现所对应的布尔量。

考虑评分函数是两个因子的线性组合：(i) 向量空间下查询和文档的余弦相似度；(ii) 包含查询项的最小窗口宽度 ω。正如我们在 7.2.2 节提到的那样，查询项邻近度对于文档的主题往往具有很强的指示作用，特别是对长文档和 Web 网页尤其有用。在提供其他信息之余，这个量实际上给出了隐式短语的某种使用方式。因此，我们给出的第一个因子依赖于查询项在文档中的统计信息，此时可以将文档看成词袋模型。而第二个因子则依赖邻近度权重计算。由于足以通过两个特征的可视化来展示方法，所以在这里我们仅仅考虑两个特征因子，显然，该技术可以扩展到多个特征因子上去。

同 6.1.2 节一样，我们给出一系列训练集，其中的每个样本包含一个查询、一篇文档以及文档是否和查询相关的相关性判定结果。对每个样本，可以计算向量空间下的余弦相似度，以及上面提到的窗口宽度 ω。计算的结果如表 15-3 所示，它与 6.1.2 节的图 6-5 有些类似。

表 15-3　机器学习评分的训练样例

例　子	文档ID	查　询	余弦得分	ω	相　关　性
Φ_1	37	linux operating system	0.032	3	*relevant*
Φ_2	37	penguin logo	0.02	4	*nonrelevant*
Φ_3	238	operating system	0.043	2	*relevant*
Φ_4	238	runtime environment	0.004	2	*nonrelevant*
Φ_5	1741	kernel layer	0.022	3	*relevant*
Φ_6	2094	device driver	0.03	2	*relevant*
Φ_7	3191	device driver	0.027	5	*nonrelevant*

…	…	…	…	…	…

这里，两个特征（余弦相似度记为 α，窗口宽度记为 ω）取值都是实数。如果我们将相关性判断结果中的相关看成 1，不相关看成 0，那么我们的目标就是寻找一个评分函数，它能够组合不同特征，产生（接近）0 或 1 的值。我们希望该函数尽可能接近训练样例的结果。不失一般性，线性分类器可以看成不同特征的线性组合形式：

$$Score(d,q) = Score(\alpha,\omega) = a\alpha + b\omega + c \text{。} \qquad (15\text{-}17)$$

其中，a、b、c 是系数，它们可以从训练集上通过学习得到。尽管也可以采用类似 6.1.2 节中错误最小化问题的思路进行求解，但是将公式（15-17）的几何性质通过图形化的方式展示出来非常有益。以余弦相似度 α 为横坐标，窗口宽度 ω 为纵坐标，可以将表 15-3 所示的例子转化成二维平面中的点，结果参见图 15-7。

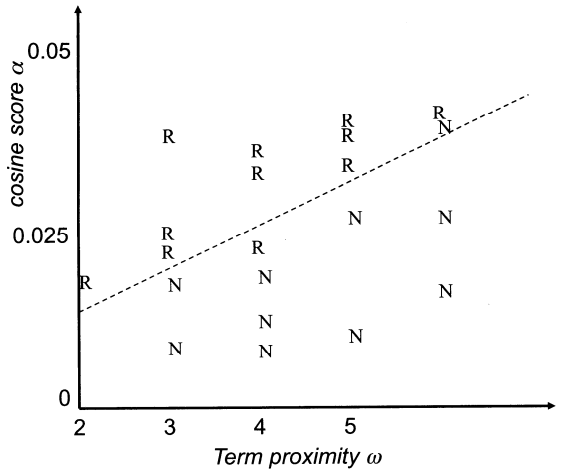


图 15-7 一个训练集的例子。每个 R 表示一篇相关文档，而 N 则表示不相关文档

这种设置下，公式（15-17）中的函数 $Score(\alpha,\omega)$可以看成悬挂在图 15-7 上空的一个平面。理想情况下，该平面（与图 15-7 所在的页面垂直）在 R 表示的点之上的函数值接近于 1，而在 N 表示的点之上的函数值接近于 0。由于该平面上的值不可能只接近 0 或者 1，我们引入了阈值方法（thresholding）：给定查询要确定某篇文档的相关性，选择某个阈值 θ，如果 $Score(\alpha,\omega) > \theta$，则认为文档和查询相关，否则不相关。从图 14-8 我们知道，所有满足 $Score(\alpha,\omega) = \theta$ 的点会构成一条直线（如图 15-7 中的虚线所示），因此，我们就得到了一个能够将相关文档和不相关文档分开的线性分类器。几何上，我们可以通过下列方法找到该分隔线：考虑通过平面 $Score(\alpha,\omega)$ 的一条直线，且它相对图 15-7 所在的平面的高度是 θ。将该直线直接投影到图 15-7 当中，就会得到图 15-7 中的虚线。然后，对于任意查询文档对，如果它们对应的点出现在该直线下面，则认为是不相关的，否则认为是相关的。

因此，在给定训练集的情况下，给出相关或不相关二值判定，就转化成在图 15-7 中寻找那

条能将相关文档和不相关文档分开的虚线的问题。在 α-ω 平面中，该直线可以写成仅包括 α 及 ω（分别代表斜率和截距）两个参数的直线方程。从第 13 章到第 15 章给出的一系列线性分类方法都适用于选择这条直线。如果训练集合足够大，就可以避免像 7.2.3 一样对评分函数进行人工调节。当然，这种做法的瓶颈在于获得合适的、有代表性的训练集的能力，在这些训练集上的相关性判定都要通过人工专家来完成。

15.4.2　基于机器学习的检索结果排序

通过将两个变量扩展到多个变量显然可以对上一节的思想进行扩展。还存在许多其他的能够表征文档相关性的得分，包括静态质量（比如将在第 21 章中讨论的 PageRank 之类的指标）、文档时间、域贡献因子、文档长度等。假定对于训练集的每篇文档来说，这些指标都能算出，再加上相关性判断，那么任意数量的指标都可以用于训练出一个机器学习分类器。比如，基于二值相关性判断，我们可以训练出一个 SVM 分类器，并按照文档的相关概率排序，其中的相关概率是文档到决策边界带符号距离（signed distance）的单调函数。

然而，通过这种方法来对文档进行排序不一定是考虑该问题的一条正确途径。统计学家通常首先将问题分成分类问题（当预测的变量是类别型变量）或回归（regression）问题（当预测的变量是实数型变量）。在它们之间是一个称为序回归（ordinal regression）的特殊领域，它预测的是一个序结果。将 ad hoc 检索中的机器学习问题考虑成一个序回归问题是最合适不过的，这是因为 ad hoc 检索是在给定训练集中文档均已排好序的情况下，对每个查询将一系列文档进行同类型排序的过程。这种形式化方法具有另外的作用，这是因为对于同一查询，只须考虑文档之间的相对排序，而不需要将文档映射成一个全局的得分。同时，这种处理方法也弱化了问题的空间，最后仅仅需要排序而不需要定义一个衡量相关性的绝对指标。排序对于 Web 搜索来说尤其重要，若干个最高结果的排名非常关键，而一篇文档和查询的相关性决策的重要度却要弱得多。这类工作可以通过 15.2.2 节中提到的结构化 SVM 来实现，其中预测的类别是结果的排序，但是，下面我们将介绍稍微简单一点的排序 SVM。

排序 *SVM*(ranking SVM)的构造过程如下：给定一系列已经判定的查询，对每个训练查询 q，都有一系列对应的返回文档，这些文档之间已经按照相关性的强弱排序。对于每个“文档–查询”对，构建一个特征向量 $\psi_j=\psi(d_j,q)$，其中的特征可以使用 15.4.1 提到的那些特征，甚至也可以引入更多的特征。对于两篇文档 d_i 和 d_j，可以计算特征向量的差：

$$\Phi(d_i,d_j,q)=\psi(d_i,q)-\psi(d_j,q)\text{。} \qquad (15\text{-}18)$$

基于假设，d_i 和 d_j 中的一个相关度更高。如果 d_i 被判定为比 d_j 更相关，记为 $d_i \prec d_j$，此时在结果列表中 d_i 应该排在 d_j 的前面，那么将向量 $\Phi(d_i,d_j,q)$ 赋给 y_{ijq}=+1 类，否则赋给–1 类。因此我们的目标是建立一个分类器，满足

$$\vec{w}^T\Phi(d_i,d_j,q)>0 \quad \text{iff } d_i \prec d_j\text{。} \qquad (15\text{-}19)$$

这里的 SVM 学习任务的形式化表示同以前看到的极其类似。

寻找 $\vec{w}$ 及 $\xi_{i,j}$ H 0 ，使得 $\frac{1}{2}\vec{w}^T\vec{w}+C\sum_{i,j}\xi_{i,j}$ 极小化，对所有的 $\{\Phi(d_i,d_j,q): d_i \prec d_j\}$ 有 $\vec{w}^T\Phi(d_i,d_j,q)$ H $1-\xi_{i,j}$ 。 (15-20)

在约束条件中可以将 y_{ijq} 去掉，由于 $\prec$ 满足反对称性，所以我们只需要考虑一个方向的排序结果即可。可以和以前一样对上述约束进行求解，于是得到一个可以对文档对排序的线性分类器。利用这个方法可以构造排序函数，目前在标准的文档集合中取得的结果已经超过了现有人工构造排序函数的结果。关于这一内容请参考相关文献的结果。

同本领域的大部分工作一样，上面刚刚介绍的两种方法都使用了表征文档相关性特征的线性组合。因此，这里面一个值得注意的有趣问题是，很多传统的 IR 权重计算机制中都包含了基本指标的非线性缩放过程（比如词项频率或 idf 的对数权重计算）。目前为止，机器学习非常擅长特征线性组合（或者其他类似的受限模型）中的权重优化，但是并不擅长基本指标的非线性缩放。这个领域仍然需要人工的特征工程方法。

排序函数学习已经提出了许多年，但是一直到最近，随着机器学习知识、训练文档集和计算能力等的发展，才足以使这种思想得以实现并取得了令人兴奋的结果。因此，虽然现在对于 IR 中基于机器学习的排序方法给出定论还为时过早，但是我们有理由期待机器学习排序方法的使用和重要性将会随时间的推移而不断增长。尽管高水平的人可以通过手工来定义一个很好的排序函数，但是手工调节是非常困难的，并且当面对新文档集或者不同类型的用户时，上述工作都需要重做。

? 习题 **15-7** 将表 15-3 的前 7 列数据画在 α–ω 平面上，得到类似图 15-7 中的图像。

习题 **15-8** 在 α–ω 平面上给出能够区分 R 和 N 所标识的文档的直线。

习题 **15-9** 给出一个训练样例（包括 α、ω 和它们的相关性判定结果），当将其加入到训练集中时，在 α–ω 平面上就不存在一条直线把所有 R 和 N 所标识的文档分开。

15.5 参考文献及补充读物

支持向量机这个有些奇怪的称呼最早起源于神经网络领域，该领域的学习算法往往被看成某种架构，因此往往被称为“机器”。SVM 的独特之处在于，决策边界完全由少数训练数据点所决定（“支持”），这些点就称为支持向量。

有关 SVM 的更详细阐述，可以参考（Burges 1998），它是一篇有关 SVM 的著名介绍材料。Chen 等人(2005)介绍了最近发展起来的 v-SVM，它对非线性可分问题提供了另外一种参数化解决方案，与指定惩罚因子 C 的做法不同，它可以通过指定参数 v 来给出出现在决策边界两边的错误样本数目的上界。目前有不少有关 SVM、大间隔学习、核函数的专著：其中（Cristianini 和

Shawe-Taylor 2000)及(Schölkopf 和 Smola 2001)更偏数学 ,而(Shawe-Taylor 和 Cristianini 2004)则更面向实际。对于 SVM 的起源可以参考（Vapnik 1998)。最近一些有关统计学习的一般性书籍也对 SVM 进行了全面的介绍，其中包括（Hastie 等人 2001)。

有关多类 SVM 构造的内容参见（Weston 和 Watkins 1999)、(Crammer 和 Singer 2001 ）及(Tsochantaridis 等人 2005)。其中，最后一篇文献中给出了结构化 SVM 一般性框架的介绍。

核技巧最早见于（Aizerman 等人 1964)。对于有关字符串核函数和其他结构化数据的核函数的介绍可以参见（Lodhi 等人 2002 ）和（Gaertner 等人 2002)。NIPS（Neural Information Processing，神经信息处理进展）会议已经成为机器学习理论工作的一个首要场所，这里面当然也包括有关 SVM 的一些工作。其他（如 SIGIR ）则更强调实验的评价结果，它们往往通过利用文本相关的特征来提高分类器的效果。

当前一个最新的有关机器学习分类器的比较参见（Caruana 和 Niculescu-Mizil 2006)，其中的问题与典型的文本问题有所不同。在 13.6 节中，我们讨论的（Li 和 Yang 2003 ）有关机器学习分类器的对比工作也是最新的比较工作。更早的文本分类器的比较实验可以参见(Yang 1999; Yang 和 Liu 1999；Dumais 等人 1998)。Joachims（2002a ）详细给出了基于 SVM 的文本分类工作。Zhang 和 Oles (2001) 对 NB、正则化 logistic 回归方法及 SVM 方法进行了深入的对比。.

Joachims（1999 ）讨论了在大规模数据集上实用的 SVM 方法。Joachims（2006a ）对该工作进行了改进。

对于层次分类问题，即类别之间存在着很自然的层级组织结构，有很多相关的研究工作，具体可参见（Koller 和 Sahami 1997；McCallum 等人 1998；Weigend 等人 1999；Dumais 和 Chen 2000)。在最近的一项大型研究中，Liu 等人（2005 ）试图将 SVM 推广到 Yahoo!分类目录上去，最后的结论是，基于层次分类的方法明显会超过扁平分类，当然目前提高的程度还很有限。对于很多类别来说，极其有限的训练集合还是会限制分类器的效果。一个能够对类别之间关系建模的更一般的方法可以参见 Tsochantaridis 等人（2005)，其中的关系不仅仅是简单的层次关系而是任意的关系。

Moschitti 和 Basili (2004) 考察了以复合名词、专有名词及词义为特征的文本分类。

Dietterich (2002) 综述了组合多个分类器的集成分类方法，而 Schapire（2003 ）主要关注其中的提升（boosting ）型集成方法，这种方法在（Schapire 和 Singer 2000 ）得到应用。

Chapelle 等人（2006 ）给出了有关半监督学习方法相关研究工作的介绍，其中有些章节特别关注 EM 在半监督文本分类中的应用(Nigam 等人 2006)以及直推式 SVM(Joachims 2006b)。Sindhwani 和 Keerthi (2006)给出了大规模数据集条件下的直推式 SVM 的快速实现。

Tong 和 Koller (2001) 将主动学习用于 SVM 文本分类。Baldridge 和 Osborne (2004) 指出，在主动学习中针对某个分类器选出的待标注样本如果用到另一分类器，那么结果不一定比随机选择方法好。

将机器学习排序的思路引入到 ad hoc 检索的先导性工作包括（Wong 等人 1988)、(Fuhr

1992）及（Gey 1994）。但是由于受当时的训练数据和机器学习方法所限，上述工作只取得了中等结果，因此在当时的影响非常有限。

Taylor 等人（2006）研究了如何利用机器学习方法来调节 BM25 系列排序函数（参考 11.4.3 节）的参数以使 NDCG（参见 8.4 节）最大化。利用机器学习的序回归方法出现在（Herbrich 等人 2000）和（Burges 等人 2005）中，并在点击流数据中得到应用（Joachims 2002b）。Cao 等人（2006）研究如何采用该方法在 IR 中取得好的效果，Qin 等人（2007）提出了包括多个超平面的扩展方法。Yue 等人（2007）研究了如何利用结构化 SVM 方法来排序，并特别考察了对 MAP（参见 8.4 节）进行直接优化的构建方法，他们的研究工作主要基于 MAP 指标而不是其他的诸如精确率或 ROC 曲线下的面积指标。Geng 等人（2007）考察了排序问题中的特征选择方法。

在 Web 搜索中，其他一些排序学习方法也表现出了很好的效果，这些工作包括（Burges 等人 2005; Richardson 等人 2006）等。

第16章 扁平聚类

聚类算法将一系列文档聚团成多个子集或簇（cluster），其目标是建立类内紧密、类间分散的多个簇。换句话说，聚类的结果要求簇内的文档之间要尽可能相似，而簇间的文档之间则要尽可能不相似。

聚类是无监督学习（unsupervised learning）的一种最普遍的形式。无监督也意味着不存在对文档进行类别标注的人类专家。聚类中，数据的分布和组成结构决定最后的类别归属。图 16-1 给出了一个简单的例子，通过肉眼就可以看出其中包含三个不同的簇。本章和下一章将介绍在无监督模式下寻找这种簇的算法。

乍看起来，聚类和分类的区别并不大，毕竟这两种任务都会将文档分到不同的组中。然而，我们将会看到，这两个问题之间存在着本质的差异。分类是监督学习的一种形式（参见第 13 章），其目标是对人类赋予数据的类别差异进行学习或复制。而在以聚类为重要代表的无监督学习当中，并没有这样的人来对类别的差异进行引导。

聚类算法的一个关键输入是距离计算方法。在图 16-1 中，计算距离时采用的是二维平面上的距离计算方法。基于这种距离计算方法在图中得出了三个不同的簇。在文档聚类当中，距离计算方法往往采用欧氏距离。不同的距离计算方法会导致不同的聚类效果。因此，距离的计算方法是影响聚类结果的一个重要因素。

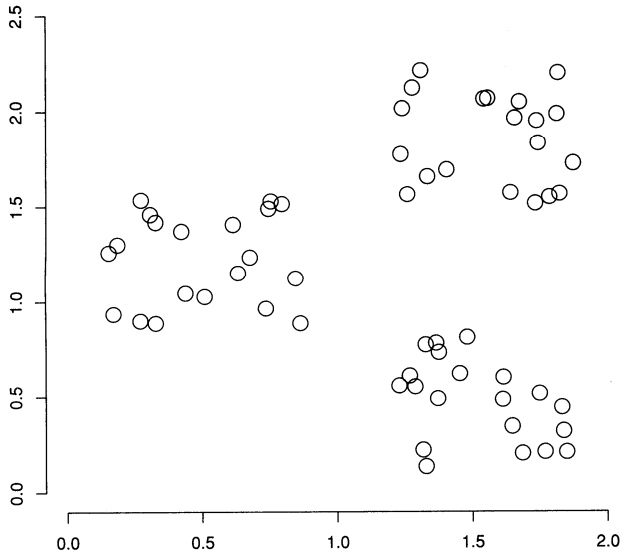


图 16-1　一个具有清晰簇结构的数据集例子

扁平聚类（flat clustering）算法会给出一系列扁平结构的簇，它们之间没有任何显式的结构来表明彼此的关联性。而层次聚类（hierarchical clustering）算法则会产生层次性的聚类结果。本章主要介绍前者，而后者将在第 17 章讨论，第 17 章还将讨论簇标签的自动生成这个难题。

了解硬聚类和软聚类之间的差别也相当重要。硬聚类（hard clustering）计算的是一个硬分配（hard assignment）过程，即每篇文档仅仅属于一个簇。而软聚类(soft clustering)算法的分配过程是软的，即一篇文档的分配结果是在所有簇上的一个分布。在软分配结果中，一篇文档可能对多个簇都具有隶属度。作为一种降维方法，隐性语义索引（参见第 18 章）就是一个软聚类算法。

本章首先通过一些应用来引入 IR 中的聚类算法（见 16.1 节），紧接着会给出聚类中需要解决的若干个问题的定义（见 16.2 节），进而讨论聚类结果的评价指标（见 16.3 节）。然后将介绍两种聚类算法：*K*-均值算法（见 16.4 节）和 EM 算法（见 16.5 节）。前者是硬聚类算法，而后者是软聚类算法。由于 *K*-均值算法简单高效，它或许是目前使用最广泛的扁平聚类算法。而 EM 是 *K*-均值算法的一般化形式，可以应用到许多文档表示和文档分布中去。

16.1　信息检索中的聚类应用

聚类假设（clustering hypothesis）给出了将聚类应用于 IR 时的基本假设。

聚类假设：在考虑文档和信息需求之间的相关性时，同一簇中的文档表现互相类似。

聚类假设所表达的是，如果簇中某篇文档和查询需求相关，那么同一簇中的其他文档也和查询需求相关。这是因为聚类算法将那些共有很多词项的文档聚在一起。聚类假设实质上就是第 14 章的邻近假设。两种情况下，我们都认为内容相似的文档在相关性上的表现也相似。

表 16-1 给出了 IR 中聚类算法的一些主要应用。一方面，这些应用在聚类的对象上有所不同，其中包括搜索结果、整个文档集或文档子集上的聚类；另一方面，它们应用在 IR 系统的不同方面，并希望对这些方面能有所改进，其中包括用户体验、用户界面及搜索系统的效率和效果等。尽管上面提到的应用之间有很多不同，但其中的聚类过程都基于上面提到的聚类假设。

表16-1　IR中一些聚类应用的例子

应　　用	聚类对象	优　　点	例　　子
搜索结果聚类	搜索结果	提供面向用户的更有效的展示	图16-2
“分散-集中”界面	文档集和文档子集	提供了另一种用户界面，即不需要人工输入关键词的搜索界面	图16-3
文档集聚类	文档集	提供了一种面向探索式浏览的有效的信息展示方法	McKeown 等人（2002），http://news.google.com
基于语言建模的IR	文档集	提高了正确率和/或召回率	Liu 和 Croft（2004）
基于聚类的检索	文档集	加快了搜索的速度	Salton（1971a）

表 16-1 中提到的第一个应用是搜索结果的聚类，这里的搜索结果指的是针对某个查询返回的一系列文档。默认情况下，IR 中搜索结果以一个简单的列表方式来展示。用户从上到下浏览文档直到发现所需的文档位置为止。与这种做法不同，搜索结果聚类可以将相似的文档放在一起呈现。通常来说，浏览几个内容连贯的文档子集会比浏览一篇篇独立的文档更容易。这种做法在查询词项存在歧义时尤其有用。图 16-2 中给出的查询例子是 jaguar。该词在 Web 中的三种最常见的意义分别是指车、动物或者 Apple 操作系统。Vivisimo 搜索引擎（http://vivisimo.com）所返回的 Clustered Results 面板提供了一个更有效的用户界面，可以给出比简单的文档列表更容易理解的搜索结果。

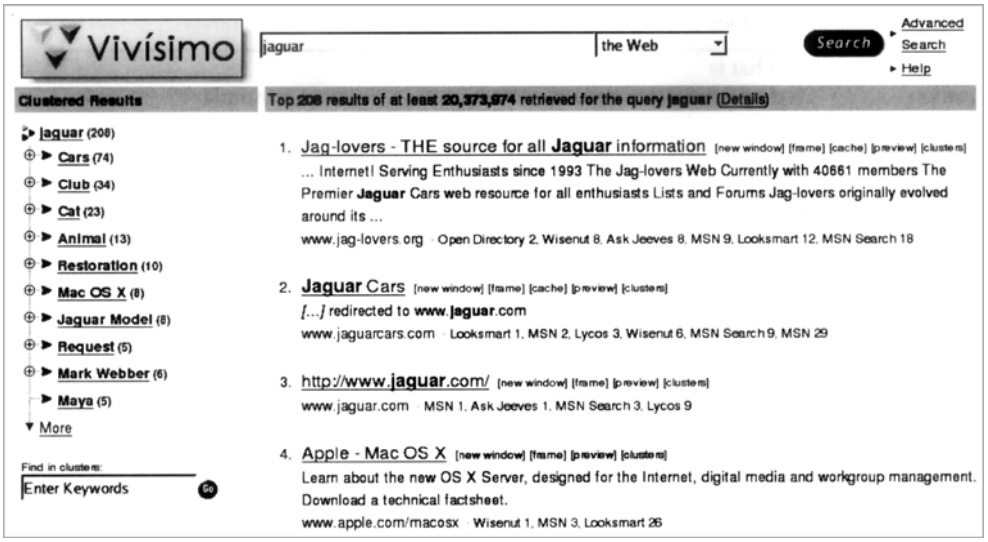


图 16-2 对搜索结果聚类以提高召回率的示例。返回结果中前面的文档并没有覆盖 jaguar 作为动物的那个词义。但是用户很容易通过在 Clustered Results 面板中点击 Cat 簇（从上往下的第三个箭头）得到该类结果

表 16-1 中的第二个应用给出了一个更好的用户界面，这也是分散-集中（scatter-gather）方法的目标。分散-集中方法首先通过聚类将文档集中所有文档分散成不同的簇供用户选择。选出的簇将集中合并在一起，然后将结果集合中的文档再次聚类。这个过程会反复进行直至找到用户满意的簇为止。图 16-3 给出了这种做法的一个例子。

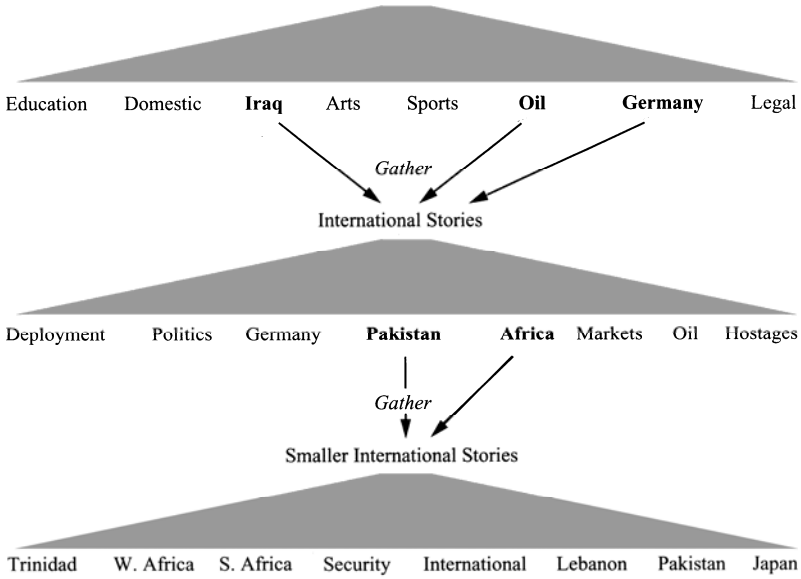


图 16-3　分散–集中方法的一个用户会话（session）的例子。一个 New York Times 的新闻报道集聚类（“分散”）成 8 个簇（顶行）。用户手工将其中三个簇“集中”成一个小的文档集 International Stories，然后再次进行分散操作。重复上述过程直至找到包含相关文档的簇（比如 Trinidad）为止

如图 16-3 所示，自动生成的簇不会像人工构造的层次树（如 ODP 目录，参见 http://dmoz.org）那样系统和精确。另外，自动寻找簇的描述性标签也是一个非常困难的问题（参见 17.7 节）。但是，相对于常规 IR 中关键词查找的模式，基于簇的导航方法是一个非常有趣的做法。当用户不确定采用什么词项进行查询时，他可能更希望采用浏览方式，这时候采用基于簇的导航方法尤其有效。

与在分散-集中模式中以用户为中介有所不同的是，我们可以采用另外一种方法，即将文档集进行静态层次聚类（表 16-1 中的文档集聚类），该聚类结果不受用户交互影响。Google News 及更早的 Columbia NewsBlaster 系统就是此类方法的两个例子。在新闻场景下，需要不断重新计算聚类结果以保证用户能够访问到最新的报道。聚类非常适合于新闻报道集的访问，这是因为新闻阅读往往不是搜索过程，而是一个对近期事件的选择过程。

聚类算法的第四个应用是，直接应用聚类假设即对文档集聚类来提高搜索的效果。首先我们基于倒排索引获得与查询匹配的初始文档集，然后将处于同一簇中、甚至与查询相似度不高的其他文档也会加入到结果文档集合。比如，查询为 car，返回结果中的若干文档来自汽车文档簇，我们将该簇中的其他文档也加入到结果当中，这些文档中使用了 car 之外的很多词项（如 automobile、vehicle 等）。这种做法可以提高结果的召回率，相互之间相似度很高的一组文档往往在整体上与查询相关。

近年来，文档集聚类的思想也被应用到语言建模中。公式（12-10）利用整个文档集的模型

来对文档 d 的模型进行插值，从而避免语言模型中的数据稀疏性问题。但是，整个文档集中包含了很多文档，它们中的很多词项并不能代表 d 的内容。通过将文档集模型替换成 d 所在的簇模型，就可以对文档 d 的模型中的词项出现概率进行更精确的估计。

聚类同样可以加快搜索的过程。正如我们在 6.3.2 节所看到的那样，向量空间模型中的搜索就是寻找与查询最近的的近邻文档的过程。在常规的 IR 配置下，倒排索引能够支持快速的近邻搜索。然而，有时倒排索引使用起来效率并不高。比如，当使用第 18 章将要提到的隐性语义索引时就会如此。这类情况下，我们要计算查询和所有文档的相似度，但是这样做很慢。聚类假设提供了另外一种做法，即首先寻找和查询最接近的簇，然后只考虑簇中文档和查询之间的相似度。在这个小得多的文档集上，我们可以遍历其中的所有文档，计算它们和查询之间的相似度并像以往一样进行排序。由于簇的个数远远小于文档的个数，因此找到最近簇的过程很快。又由于与某个查询匹配的文档之间彼此相似，因此它们很可能出现在同一簇中。尽管聚类算法不是特别精确，但是预期的搜索质量的下降程度却很小。这实际上也是 7.1.6 节所介绍的聚类应用。

? 习题 **16-1** 如果两篇文档同时包含至少两个专有名词（如 Clinton 或 Sarkozy），那么定义它们为相似文档。在这种相似度定义下，试给出某个信息需求和两篇文档的例子，此时它们并不满足聚类假设。

习题 **16-2** 在一维空间（即一条直线上的点）中构造一个包含两个簇的简单例子，此时基于簇的搜索结果并不是精确的。也就是说，在给出的例子中，返回和查询相近的簇的效果还不如直接返回近邻文档的效果。

16.2 问题描述

硬扁平聚类的目标可以定义如下：

给定 (i) 一系列文档 $D = \{d_1, \ldots, d_N\}$，(ii) 期望的簇数目 K，以及(iii)用于评估聚类质量的目标函数（objective function），计算一个分配映射 $\gamma : D \rightarrow \{1, \ldots, K\}$，该分配下的目标函数值极小化或者极大化。大部分情况下，我们要求 γ 是一个满射，也就是说，K 个簇中的每一个都不为空。

目标函数通常基于文档的相似度或者距离来定义。下面我们将看到，K-均值算法的目标是最小化文档和其所在簇的质心的平均距离，或者说最大化文档到其所在簇的质心的平均相似度，这两种说法实际上是等价的。第 14 章中有关相似度计算和距离度量方法的讨论同样适用于本章。同第 14 章一样，在谈到文档之间的相关度时，我们同时使用相似度和距离这两个概念。

对于文档而言，通常期望的相似度类型都是主题相似度，或者说此时不同文档向量当中多个相同维上的权值较高。比如，和 China 相关的文档向量可能在如 Chinese、Beijing 或 Mao 这几个词所对应维上的权值较大，而和 UK 相关的文档向量的较大值则可能出现在如 London、

Britain 或 Queen 对应的向量维下。我们可以通过向量的余弦相似度或欧氏距离（参见第 6 章）来近似计算主题相似度。如果要计算的相似度类型不属于主题相似度（比如要计算语言上的相似度），那么采用另一种不同的表示方法可能更合适。计算主题相似度时，停用词往往可以被忽略，这样做通常也比较安全。但是在将英语文档（the 出现频繁而 la 出现不频繁）和法语文档（the 出现不频繁而 la 出现频繁）区分开时，停用词却很重要。

关于术语的一个注解。关于硬聚类还存在另一种定义，在这种定义下，一篇文档可以同时完全属于多个簇。分割式聚类（partitional clustering）通常指每篇文档仅属于一个簇的聚类方法。然而，在第 17 章的分割式层次聚类当中，属于某个簇的文档同时也属于其父簇。在允许一篇文档属于多个簇的硬聚类定义中，软聚类和硬聚类的区别在于：硬聚类的簇隶属度取 0 或者 1，而软聚类的簇隶属度可以取任意的非负值。

有些学者对穷尽式（exhaustive）聚类和非穷尽式（nonexhaustive）聚类算法加以区分，在前者当中每篇文档都会归入一个簇，而在后者中某些文档可能不归入任何簇。非穷尽式聚类中，每篇文档要么属于某个簇要么不属于任何簇，这种聚类方法称为独占式（exclusive）聚类。本书将基于穷尽式思想来定义聚类算法。

聚类的势– 簇的数目

聚类算法中的一个难点是如何确定簇的数目或者说聚类的势（cardinality），这个数目记为 K。通常 K 只不过是基于经验或领域知识的一个良好的猜测结果。但是，在 K-均值算法中，我们将引入一个选择 K 值的启发式方法，并将 K 的选择融入到目标函数当中。有时候，具体的应用会对 K 的取值范围有所限制。比如，在图 16-3 所显示的“分散-集中”界面中，由于受 20 世纪 90 年代的计算机显示器大小和分辨率所限，其中每层都不能显示超过 K=10 个簇。

由于聚类的目的是对目标函数进行优化，因此聚类实际上是一个搜索问题。穷举法可以枚举出所有可能的聚类结果，然后从中选择最好的结果。但是，由于存在指数级的划分结果，穷举法实际上是不可行的 ①。基于这个原因，大部分扁平聚类算法会在初始划分结果的基础上不断迭代更新。如果搜索的初始点选择不当，那么最终可能不会达到全局最优。因此，扁平聚类的另一个重要问题是选择好的初始点。

16.3　聚类算法的评价

聚类算法中，典型的目标函数是将簇内高相似度（簇内文档相似）及簇间低相似度（不同簇之间的文档不相似）的目标进行形式化后得到的一个函数。这是聚类质量的一个内部准则（internal criterion）。但是，内部准则的高分并不意味着应用中的效果就一定好。另一种方法是

① 可能的聚类结果数目的上界是 $K^N/K!$。将 N 篇文档分到 K 个簇中的不同分割的精确数目是一个第二类 Stirling 数。参见 http://mathworld.wolfram.com/StirlingNumberoftheSecondKind.html 或 Comtet (1974)。

根据应用的需求来直接评价。对于搜索结果的聚类，我们需要度量在不同聚类算法下用户找到所需答案的时间。这是最直接的度量方法，但是这种做法很昂贵，尤其在需要进行大量用户调查时更是如此。

代替用户判断的一种做法是，利用已有的分类文档集合作为评价基准或黄金标准（参考 8.5 节和 13.6 节）。理想情况下，黄金标准来自人工的判定，并且不同人之间的判定一致性具有很高的水平（参考第 8 章）。于是，我们可以计算一个所谓外部准则（external criterion），即计算聚类结果和已有的标准分类结果的吻合程度。比如，对于图 16-2 中针对 jaguar 的搜索结果聚类而言，最优的结果是聚出 car、animal 及 operating system 所代表的三个类来。当然，在这种类型的评价中，我们只使用黄金标准的分割结果，而不必考虑具体的类别标签。

本节中将介绍四种衡量聚类质量的外部准则。其中，纯度（purity）是一个简单、明晰的评价指标。NMI（Normalized Mutual Information，归一化互信息）可以基于信息论进行解释。RI（Rand Index，兰德指数）将惩罚聚类中的 FP（False-positive，假阳性）和 FN（False-negative，假阴性）的错误判定。另外，F 值（F measure）可以对上述两种类型的错误进行加权组合。

计算纯度时，每个簇被分配给该簇当中出现数目最多的文档所在的类别，然后可以通过正确分配的文档数除以文档集中的文档总数 N 来得到该分配的精度。形式地有

$$\text{purity}(\Omega, \mathrm{C}) = \frac{1}{N}\sum_{k}\max_{j}|\omega_k \cap c_j| \qquad (16\text{-}1)$$

其中，$\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$ 是聚类的结果，即文档簇的集合，而 $\coprod = \{c_1, c_2, \ldots, c_J\}$ 是类别集合。每个 ω_k 和 c_j 都是指一些文档组成的集合。

对于图 16-4[①]，我们给出了如何计算纯度的例子。很糟糕的聚类结果会得到一个接近 0 的纯度，而完美的聚类结果的纯度为 1。表 16-2 给出了纯度和其他三种度量指标的比较结果。

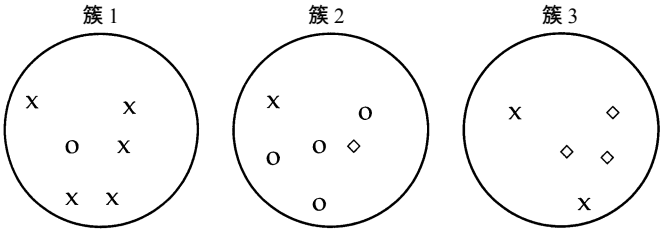


图 16-4 作为度量簇质量的外部准则的纯度示例。每个簇所对应的主类及属于其的文档数目分别是：X, 5（簇 1）；O, 4（簇 2）；及◇, 3（簇 3）。纯度为 $(1/17) \times (5 + 4 + 3) \approx 0.71$

表16-2 图16-4中聚类算法中的四种外部评价指标

	purity	NMI	RI	F_5

① 在浏览本章和后续章节的这幅或者其他的二维图像时，不妨回忆一下我们基于图 14-2 所做的提示：这些图有可能会对我们的理解产生误导，这是因为长度归一化向量的二维映射可能会使点之间的相似度和距离产生失真。

下界	0.0	0.0	0.0	0.0
最大值	1.0	1.0	1.0	1.0
图16-4中的值	0.71	0.36	0.68	0.46

当簇的数目很大时，很容易得到高纯度的结果。特别是，当每篇文档都独自作为一个簇时，结果的纯度为 1。因此，纯度并不能在聚类质量和簇数目之间保持均衡。

一种能在聚类质量和簇数目之间维持均衡的指标是 NMI，它的定义如下：

$$NMI(\Omega,\mathrm{C})=\frac{I(\Omega,\mathrm{C})}{[H(\Omega)+H(\mathrm{C})]/2}。\qquad(16\text{-}2)$$

其中，I 是互信息（参见第 13 章）：

$$I(\Omega,\mathrm{C})=\sum_k\sum_j P(\omega_k\cap c_j)\log\frac{P(\omega_k\cap c_j)}{P(\omega_k)P(c_j)}\qquad(16\text{-}3)$$

$$=\sum_k\sum_j\frac{|\omega_k\cap c_j|}{N}\log\frac{N|\omega_k\cap c_j|}{|\omega_k||c_j|}。\qquad(16\text{-}4)$$

其中，$P(\omega_k)$、$P(c_j)$及 $P(\omega_k\cap c_j)$分别是一篇文档属于 ω_k、c_j 及 $\omega_k\cap c_j$ 的概率。公式（16-4）是对公式（16-3）的概率进行 MLE 估计后的结果，其中每个概率都估计为相应的相对频率值。

如第 5 章一样，H 代表是信息的熵：

$$H(\Omega)=-\sum_k P(\omega_k)\log P(\omega_k)\qquad(16\text{-}5)$$

$$=-\sum_k\frac{|\omega_k|}{N}\log\frac{|\omega_k|}{N}。\qquad(16\text{-}6)$$

同样，（16-6）式是（16-5）式进行 MLE 估计的结果。

公式（16-3）中的 $I(\Omega,\mathrm{C})$ 度量的是在知道簇的情况下关于类的知识所增加的信息量。如果聚类算法产生的簇与真正的类归属之间只是随机关系的话，那么 $I(\Omega,\mathrm{C})$ 取最小值 0。任何情况下，知道某篇文档属于某个特定的簇并不会为它到底属于哪个类带来任何新信息。一个重构原始类别的完美聚类算法 Ω_{exact} 会取得最大的互信息值，但是如果 Ω_{exact} 的结果簇可以进一步分成更小的簇，那么最后的结果仍然同样可以取得最大值（参考习题 16-7）。特别是，$K=N$，即每篇文档都属于一个簇的聚类算法也能取得最大的 MI 值。所以，MI 也存在和纯度类似的问题，即它并不对数目较大的聚类结果进行惩罚，因此也不能在其他条件一样的情况下，对簇数目越小越好的这种期望进行形式化。

由于熵会随着簇的数目的增长而增大，所以公式（16-2）中基于分母 $[H(\Omega)+H(\mathrm{C})]/2$ 的归一化能够解决上述问题。比如，当 $K=N$ 时，$H(\Omega)$ 会达到其最大值 $\log N$，此时就能保证 NMI 的值较低。由于 NMI 做了归一化，所以可以用它来比较具有不同簇的数目的聚类结果。公式（16-2）中之所以采用 $[H(\Omega)+H(\mathrm{C})]/2$ 作为分母，是因为它是 $I(\Omega,\mathrm{C})$ 的紧上界（参考习题 16-8）。因此，NMI 的值在 0 到 1 之间。

与上述基于信息论来解释聚类结果不同的是，我们也可以将聚类看成是一系列的决策过程，

即对文档集上所有 $N(N-1)/2$ 个文档对进行决策。当且仅当两篇文档相似时，我们将它们归入同一簇中。TP（True-positive，真阳性）决策将两篇相似文档归入一个簇，而 TN（True-negative，真阴性）决策将两篇不相似的文档归入不同的簇。在此过程中会犯两类错误：FP 决策会将两篇不相似的文档归入同一簇，而 FN 决策将两篇相似的文档归入不同簇。RI 计算的是正确决策的比率，它实际上就是在 8.3 节中提到的精确率（accuracy）：

$$\mathrm{RI}=\frac{\mathrm{TP+TN}}{\mathrm{TP+FP+FN+TN}}。$$

我们以图 16-4 为例来计算其 RI 值。首先计算 TP+FP。三个簇中包含的点数分别为 6 个、6 个、5 个，因此，所有正例即文档对出现在同一簇的数目为：

$$\mathrm{TP}+\mathrm{FP}=\binom{6}{2}+\binom{6}{2}+\binom{5}{2}=40。$$

在这些正例当中，簇 1 的 x 点对、簇 2 的 o 点对、簇 3 的◇点对及簇 3 中的 x 点对都是真正例，因此

$$\mathrm{TP}=\binom{5}{2}+\binom{4}{2}+\binom{3}{2}+\binom{2}{2}=20。$$

因此，FP = 40 – 20 = 20 。

我们同样可以计算出 FN 和 TN，最后得到如下列联表：

	同一簇	不同簇
同一类	TP=20	FN=24
不同类	FP=20	TN=72

于是，RI = (20+72)/(20+20+24+72) ≈ 0.68。

上述 RI 的计算当中 FP 和 FN 采用了相等的权重，有时候，将相似文档分开的后果比将不相似文档归成一类更严重。所以，我们可以使用 8.3 节讨论的 F 值来度量聚类结果，并通过设置 $\beta > 1$ 以加大对 FN 的惩罚，此时实际上也相当于赋予召回率更大的权重。

$$P=\frac{\mathrm{TP}}{\mathrm{TP+FP}}\qquad R=\frac{\mathrm{TP}}{\mathrm{TP+FN}}\qquad F_\beta=\frac{(\beta^2+1)PR}{\beta^2P+R}。$$

基于上面列联表中的数字，可以计算出 P=20/40=0.5，R=20/44 ≈ 0.455，因此当 $\beta = 1$，$F_1 \approx 0.48$，当 $\beta = 5$，$F_5 \approx 0.456$。在 IR 中，通过 F 值来评价聚类方法的一个优点在于，这个领域的研究人员对该指标已经非常熟悉。

? 习题 **16-3** 对于图 16-4，同一类中的每个点 d 都用两个同样的 d 的副本来替换。(i) 那么，对于新的包含 34 个点的集合进行聚类，会比图 16-4 中 17 个点的聚类更容易、一样难还是更难？(ii) 计算对 34 个点聚类的纯度、NMI、RI 及 F_5 值。在点数增加一倍之后，哪些指标增大？哪些指标

保持不变？(iii) 在得到(i)中的判断和(ii)中的指标之后，哪些指标更适合于上述两种聚类结果的质量比较？

16.4　*K*-均值算法

K-均值算法是最重要的扁平聚类算法，它的目标是最小化文档到其簇中心的欧氏距离平方的平均值，其中，簇中心的定义为簇 ω 中文档向量的平均值或者说质心 $\vec{\mu}$ ：

$$\vec{\mu}(\omega)=\frac{1}{|\omega|}\sum_{\vec{x}\in\omega}\vec{x}。$$

在上述定义中，我们假设每个文档向量都表示为实数空间下的长度归一化向量。这里我们用到了第 14 章 Rocchio 分类算法中的质心向量，它们在这里的作用和以前类似。*K*-均值算法中理想的簇是以质心为中心的一个球体。理想情况下，簇之间不会重叠。我们在 Rocchio 中的理想目标也是如此，所不同的是，聚类中并没有已标注文档来告诉我们哪些文档应该归于同一簇。

一个衡量质心对簇中文档的代表程度的指标是 RSS(*Residual Sum of Squares* ,残差平方和)，即所有向量到其质心距离的平方和：

$$\mathrm{RSS}_k=\sum_{x\in\omega_k}|\vec{x}-\vec{\mu}(\omega_k)|^2。$$

$$\mathrm{RSS}=\sum_{k=1}^{K}\mathrm{RSS}_k。\qquad(16\text{-}7)$$

RSS 是 *K*-均值算法的目标函数，我们的目的就是要是让这个函数取最小值。由于 *N* 是固定的，最小化 RSS 也就等价于最小化平方距离，而平方距离度量的正是质心对文档的代表能力。

K-均值算法的第一步是随机选择 *K* 篇文档构成初始的簇中心，这些文档也被称为种子 (seed)。然后，算法在空间中不断移动这些簇中心使得 RSS 极小化。图 16-5 给出了 k-均值算法的伪代码，它可以通过反复迭代执行下列两步直至满足停止条件：先将文档重新分配到距它最近的簇质心 (即中心) 所在的簇，然后基于簇中目前的文档重新计算簇质心。图 16-6 给出了 *K*-均值算法在一个点集上进行 9 次迭代的快照图。下一章表 17-2 中标识“ centroid” 的那列给出的是质心的示例。

```
K-MEANS({x⃗_1, ..., x⃗_N}, K)
 1  (s⃗_1, s⃗_2, ..., s⃗_K) ← SELECTRANDOMSEEDS({x⃗_1, ..., x⃗_N}, K)
 2  for k ← 1 to K
 3  do μ⃗_k ← s⃗_k
 4  while stopping criterion has not been met
 5  do for k ← 1 to K
 6     do ω_k ← {}
 7     for n ← 1 to N
 8     do j ← arg min_{j'} |μ⃗_{j'} − x⃗_n|
 9        ω_j ← ω_j ∪ {x⃗_n}  (reassignment of vectors)
10     for k ← 1 to K
11     do μ⃗_k ← (1/|ω_k|) Σ_{x⃗∈ω_k} x⃗  (recomputation of centroids)
12  return {μ⃗_1, ..., μ⃗_K}
```

图 16-5 *K*-均值算法。对于大部分 IR 应用，向量 $\vec{x}_n \in R^M$ 应该基于长度进行归一化处理。其他的种子选择和初始化方法将在后面讨论

K-均值算法中可以采用如下终止条件。

- 当迭代一个固定次数 *I* 后停止。该条件能够限制聚类算法的运行时间，但是有些情况下，由于迭代次数不足，聚类结果的质量并不高。
- 当文档到簇的分配结果（即划分函数 γ）不再改变后停止。除了某些情况下会使算法陷入局部最优外，该停止条件通常会产生较好的聚类结果，但是运行时间不宜太久。
- 当质心向量 $\vec{\mu}_k$ 不再改变后停止。这等价于函数 γ 不再改变（参考习题 16-5）。

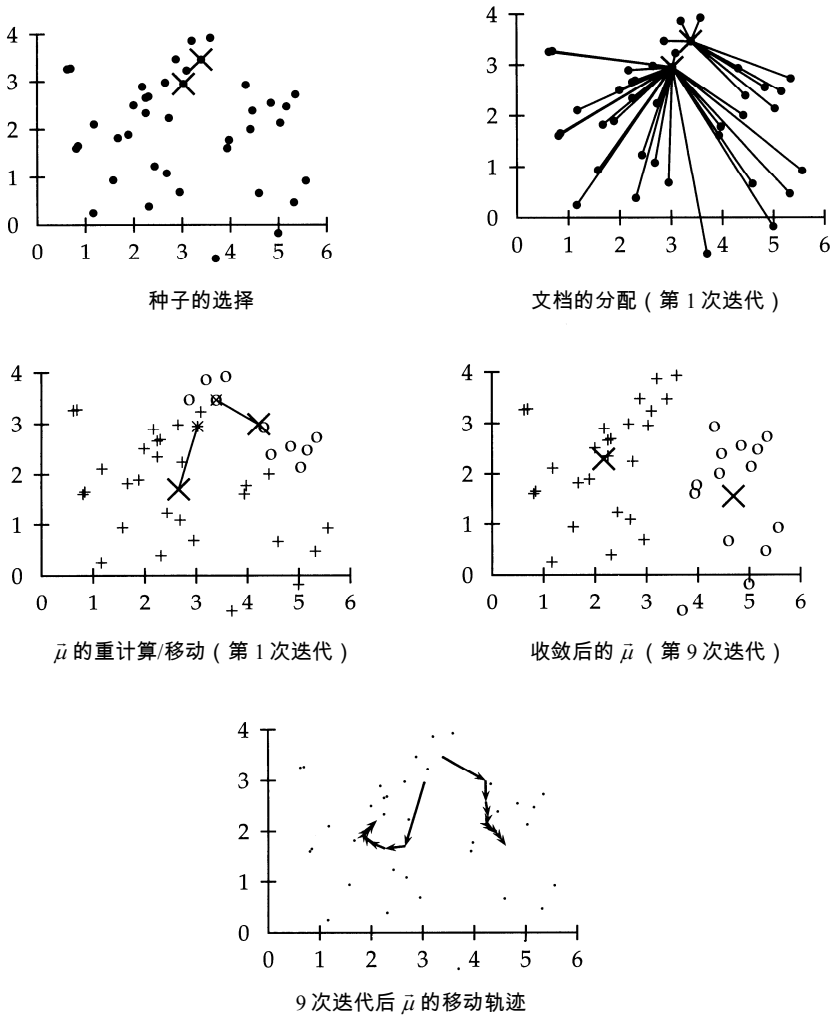


图 16-6　$\Re^2$ 空间下 K=2 的一个 K-均值算法运行示例。在 9 次迭代后两个质心向量的位置（质心向量 $\vec{\mu}$ 在最上面的 4 个图当中标记为 X）收敛

- 当 RSS 低于某个阈值时停止。该条件能够保证停止后的聚类结果具有一定的质量。在实际当中，我们必须要将它和迭代次数一起使用以保证迭代能够停止。
- 当 RSS 的减小值低于某个阈值 θ 时迭代停止。对于较小的 θ 而言，这意味着迭代接近收敛。同样，这个停止条件也往往和迭代次数限制条件一起使用以避免运行时间太长。

下面我们将证明，如果能够证明在每次迭代后 RSS 的值单调递减，那么 K-均值算法就会收敛。需要说明的是，这里的单调递减包括值减少或值不变这两种含义。第一，在每次文档的重分配过程中，每个向量都分配给最近的质心，此时它对 RSS 计算的距离贡献值也会减小，因此 RSS 会单调减少。第二，在簇中心重计算过程中，由于新的质量向量使得每个 RSS_k 达到最小值，因此，此时 RSS 也会单调递减。

$$\mathrm{RSS}_k(\vec{v}) = \sum_{\vec{x}\in\omega_k} |\vec{v}-\vec{x}|^2 = \sum_{\vec{x}\in\omega_k}\sum_{m=1}^{M}(v_m - x_m)^2 \text{。} \tag{16-8}$$

$$\frac{\partial \mathrm{RSS}_k(\vec{v})}{\partial v_m} = \sum_{\vec{x}\in\omega_k} 2(v_m - x_m) \text{。} \tag{16-9}$$

其中，x_m 和 v_m 分别是相应向量的第 m 个分量。令其偏导数为 0，则有

$$v_m = \frac{1}{|\omega_k|}\sum_{\vec{x}\in\omega_k} x_m \text{。} \tag{16-10}$$

这正好是基于每个向量分量来计算的质心的定义。因此，当将旧质心替换为新质心时，我们让 RSS_k 极小化。重新计算之后，作为 RSS_k 之和的 RSS 一定也会下降。

由于所有可能的聚类结果数目是有限的，单调下降的算法最终会达到（局部）最小值。然而，算法进行过程中要始终注意打破可能出现的平局局面。比如，如果一篇文档和多个的簇中心的距离相等时，则将该文档分配给具有最小索引号的那个簇。否则的话，聚类算法可能会陷入具有相同消耗的死循环中。

尽管上面证明了 K-均值算法的收敛性，但是并不能保证目标函数一定会达到全局最小值（global minimum）。特别是当文档集包含很多离群点（outliers）时，这个问题尤其明显。这些离群点远离其他文档，因此不适合归入任何一个簇。通常来说，如果离群点被选为初始种子，那么在后续迭代中不会有任何其他的向量被分配到该簇中。所以，即使存在具有更低 RSS 值的聚类算法，我们最后也会得到只包含一篇文档的所谓单点簇（singleton cluster）。图 16-7 给出了一个在初始种子较差的情况下得到非理想聚类结果的示例。

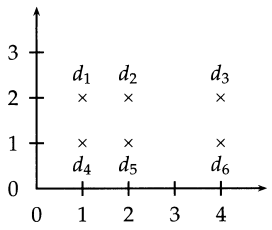


图 16-7 K-均值算法的结果依赖于初始种子的选取。对于种子 d_2 和 d_5，K-均值算法最后收敛为 $\{\{d_1, d_2, d_3\}, \{d_4, d_5, d_6\}\}$。而对种子 d_2 和 d_3，收敛结果为 $\{\{d_1, d_2, d_4, d_5\}, \{d_3, d_6\}\}$，这也是 K=2 时的全局最优值

空簇则是另一个频繁出现的非理想聚类结果（参考习题 16-11）。

种子文档选取的高效启发式策略包括：

(i) 要从种子集合中排除离群点；

(ii) 尝试多种可能初始点，并选择代价最小的聚类结果；

(ii) 借助其他方法（如层次聚类）来获得种子。

由于层次聚类是确定性的方法，其结果也比 K-均值方法更容易预测。在一个容量为 iK（例

如 i=5 或 i=10）的随机样本基础上进行层次聚类往往能提供很好的种子结果（参考第 17 章有关 Buckshot 算法的描述）。

还有其他一些初始化种子的方法，它们并不从待聚类的文档向量中直接选择种子。一个较好的方法是，为每个簇都选出 i 个（比如，i=10）随机向量，然后将它们的质心向量作为该簇的种子向量，该方法对很多文档分布都具有鲁棒性。更复杂的种子初始化方法请参见 16.6 节。

下面来讨论 K-均值算法的时间复杂度。该算法将绝大部分时间都用于计算向量的距离，其中每次计算的代价是 $\Theta(M)$。而每次重分配过程需要计算 KN 个距离，因此总的时间复杂度是 $\Theta(KNM)$。在簇中心的每次重新计算过程中，每个向量都要被累加一次，因此其复杂度为 $\Theta(NM)$。所以，对于某个固定的迭代次数 I，K-均值算法的时间复杂度与上面所有的相关因子之间都是线性关系，这些因子包括：迭代次数、簇的数目、向量的数目及向量的维数。这也意味着 K-均值算法的时间效率要高于第 17 章将要介绍的层次聚类算法。k-均值聚类算法在实现过程中必须要确定迭代次数 I，这在实际操作当中可能会存在各种处理技巧。但是，大部分情况下，K-均值算法会很快达到完全或近似收敛。后一种情况下，会有少数文档在进一步迭代中不断更换簇类别，但是这对聚类结果的整体质量影响并不大。

上面关于 K-均值算法线性时间复杂度的结论仍然有微妙之处。对于一个线性算法 $\Theta(\ldots)$而言，如果其中的某个因子非常大的话，那么该算法也有可能会非常慢，而上面提到的向量维数 M 往往就非常大。当然，计算文档之间的距离时，向量高维并不是个问题，这是因为这些向量都是稀疏向量，对于理论上可能的 M 个分量的相减运算来说，实际上只需要计算极少一部分分量。然而，由于质心向量将簇中所有文档中的词项综合在一起，所以它却是一个密集向量。因此，在 K-均值算法的原始实现中，距离计算的开销很大。但是，还有一些简单有效的启发式方法，能使得质心和文档的相似度计算与文档之间的相似度计算一样快。在几乎不降低聚类质量的情况下，可以通过仅保留质心向量中最重要的 k 个词项（比如，k=1 000）来显著加快重分配的速度（参见 16-6 的参考文献）。

通过另一种被称为 K-中心点（K-medoid）的方法可以解决 K-均值算法中的距离计算开销问题。这种方法是选取簇中某个中心点而不是质心向量来作为簇中心。我们可以将簇的中心点定义为离质心向量最近的文档向量。由于中心点都是稀疏向量，所以距离的计算速度非常快。

✂ K-均值算法中簇的个数

我们在 16.2 节中指出，对于大多数扁平聚类算法来说，簇的个数 K 往往是算法的输入参数。当我们不能对 K 的合理取值进行猜测时，应该怎么处理？

一个很自然的想法显然是选择使目标函数最优的 K 值，也就是使 RSS 极小化的 K 值。将所有可能输出 K 个聚类结果的 RSS 中的最小值记为 $\mathrm{RSS}_{\min}(K)$，则 $\mathrm{RSS}_{\min}(K)$ 会随着 K 的增大而单调递减（参考习题 16-13），当 K=N 时，$\mathrm{RSS}_{\min}(K)$ 会取最小值 0，其中 N 是所有文档的

数目。也就是说，我们应该在每篇文档都成为一个簇时结束。很显然，这并不是最优的聚类结果。

一个处理上述问题的启发式方法是采用下列方法来估计 $\mathrm{RSS}_{\min}(K)$。首先我们进行 i 次（例如 i=10）聚类过程，每次聚类的结果都包含 K 个簇（每次聚类的初始化是不同的），接着计算每次聚类结果的 RSS 值。然后我们取 i 个 RSS 中的最小值，记为 $\overline{\mathrm{RSS}}_{\min}(K)$。现在将 K 增大，根据 $\overline{\mathrm{RSS}}_{\min}(K)$ 的值的变化来寻找曲线中的拐点，过了该点之后，$\overline{\mathrm{RSS}}_{\min}$ 的减小量会明显降低。图 16-8 中给出了两个拐点的例子，一个在 K=4 处，此时的梯度变得稍微平坦一些。而另一个点在 K=9 处，变平的趋势相当明显。这种两个拐点的结果具有一定的代表性，也就是说只存在一个最佳的簇数目这种情况并不常见。所以我们必须要借助外部的约束条件从多个可能的 K 值中进行选择（比如在图 16-8 的情况中，就需要从 K=4 或 K=9 中选择）。

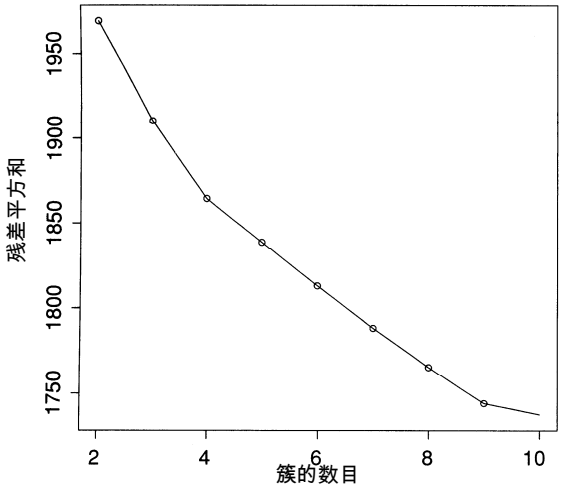


图 16-8 K-均值算法中，估计出的最小残差平方和（$\overline{\mathrm{RSS}}_{\min}$）是簇数目的函数。该聚类中，总共对 1203 篇 Reuters-RCV1 的文档进行了聚类。$\overline{\mathrm{RSS}}_{\min}$ 曲线分别在簇数目为 4 和 9 的两个拐点处变平坦。聚类中的文档从 China、Germany、Russia 及 Sports 等 4 个类中选取，因此，$K=4$ 的聚类结果接近 Reuters-RCV1 中的分类

第二种确定簇数目的准则是对每个新簇给予一定的惩罚：首先一开始将所有文档归入单个簇中，然后不断地连续增大 K 值并从中寻找出最优的 K 值。为了确定簇的数目，这种做法引入了一个包含两个要素的一般化目标函数：一个是失真率（distortion），它衡量的是文档和它们的簇原型之间的偏离程度（比如对 K-均值算法可以是 RSS）；另一个要素是衡量模型复杂度（model complexity）的一个指标。这里我们将聚类结果解释为数据的一个模型。聚类中的模型复杂度往往是指簇的数目或是它的一个函数。对于 K-均值算法而言，K 的选择准则如下：

$$K = \arg\min_{K}[\mathrm{RSS}_{\min}(K) + \lambda K]。 \qquad (16\text{-}11)$$

其中 λ 是权重因子。λ 取较大值时则意味着簇的数目倾向于取较小值。λ=0 表明并不对簇的数目取较大值时进行惩罚，此时最好的结果是 $K=N$。

显然，公式（16-11）中的一个难点是确定 λ 的值。如果确定 λ 值比直接确定 K 值还难，那就不如直接确定 K 值。某些情况下，我们可以选择一个在以前的相似数据集上效果很好的 λ 值。比如，如果我们定期地对某个新闻报社的新闻报道聚类，那么就可能取得基于某个固定的 λ 值得到后续聚类中的正确 K 值。在这个应用中，我们不可能根据过去的经验来确定 K 值，因为 K 是不断变化的。

对于公式（16-11）的一个理论性的佐证是 AIC（Akaike Information Criterion，赤池信息量准则）。AIC 是一个基于信息论的指标，它可以权衡失真率与模型复杂度。AIC 通常的形式是：

$$\textbf{AIC: } K = \arg\min_{K}[-2L(K) + 2q(K)] \text{。} \qquad (16\text{-}12)$$

其中，$-L(K)$是 K 个簇上数据的最大对数似然函数（maximum log-likelihood）的相反数，实际上它是失真率的一个衡量指标。而 $q(K)$是具有 K 个簇的模型的参数数目，它是模型复杂度的衡量指标。这里我们并不打算对 AIC 准则进行推导，实际上它在直观上很容易理解。一个好的数据模型的第一个特性是每个数据点都被模型很好地建模，这是低失真率的目标。但是模型本身也应该要小（即模型复杂度低），一个仅仅只能描述所有数据的模型（此时失真率为 0）是毫无价值的。在选择模型时，AIC 准则为基于失真率和模型复杂度两个因素进行加权的具体方法提供了理论依据。

对于 K-均值聚类算法，AIC 可以采用如下形式

$$\textbf{AIC: } K = \arg\min_{K}[\mathrm{RSS}_{\min}(K) + 2MK] \text{。} \qquad (16\text{-}13)$$

很显然，公式（16-13）是（16-11）的一个特例，其中 $\lambda = 2M$。

下面我们将介绍如何从公式（16-12）推导出公式（16-13）。在 K-均值算法中，由于 K 个质心向量的每个分量都是独立变化的，所以我们有 $q(K) = KM$。如果将 K-均值算法的总体看成是一个具有硬分配、簇先验满足均匀分布、具有相等球协方差矩阵的高斯混合分布的话，那么 $L(K) = -(1/2)\mathrm{RSS}_{\min}(K)$（以常数为模）。

在 AIC 准则的推导中引入了很多假设，比如，数据满足独立且同分布。这些假设在 IR 数据集上只能说是近似为真。因此，不能将没有做任何修改的 AIC 准则直接应用到文本聚类当中。图 16-8 中，向量空间的维数 $M \approx 50\,000$。因此，$2MK > 50\,000$，它与较小的 RSS 值相比占了主导地位（这是因为 RSS min(1)<5000，而图中并没有显示），因此（16-13）式在 K=1 时取最小值。但是我们知道，K=4 是一个比 K=1 更好的选择，此时 4 个簇分别对应四个类 China、Germany、Russia 及 Sports。实际操作中，公式（16-11）往往比公式（16-13）更有用，当然此时需要给出 λ 的估计。

? 习题 **16-4** 在 *K*-均值算法中，为什么对同一概念 car 使用不同词项来表示的文档最后可能会被归入同一簇中？

习题 **16-5** *K*-均值算法的两个停止条件为：(i) 文档的分配不再改变；(ii) 簇质心不再改变。请问这两个条件是否等价？

16.5 基于模型的聚类

本节当中将介绍 *K*-均值算法的一个一般化形式——EM 算法（Expectation Maximization Algorithm，期望最大化方法），可以应用它的文档表示和分布比 *K*-均值算法更多。

K-均值算法中，我们试图找到能够具有良好代表性的质心，*K* 个质心的集合可以被看成是数据的生成模型。在这个模型中，一篇文档的生成过程包括两步：首先随机选择一个质心，然后加入一些噪音便可生成文档。如果噪音满足正态分布且协方差为球形，那么上述文档生成的结果就是一系列球形的簇。基于模型的聚类（Model-based clustering）方法假定数据产生自某个模型并试图从数据中恢复出该模型，这个恢复出的模型即定义了簇及文档到簇的归属。

最大似然法是一种被广泛使用的模型参数估计方法。在 *K*-均值算法中，−RSS 的幂 exp(−RSS) 与某个具体模型（即质心集合）生成数据的似然成正比。*K*-均值算法中的最大似然准则和最小 RSS 准则是等价的。我们将模型的参数标记为 Θ，在 *K*-均值算法中，$\Theta=\{\vec{\mu}_1,\ldots,\vec{\mu}_K\}$。

一般来说，最大似然准则是选择能够使得数据 *D* 的生成似然极大化的参数值 Θ，即

$$\Theta = \arg\max_{\Theta} L(D \mid \Theta) = \arg\max_{\Theta} \log \prod_{n=1}^{N} P(d_n \mid \Theta) = \arg\max_{\Theta} \sum_{n=1}^{N} \log P(d_n \mid \Theta)\ 。$$

其中，$L(D|\Theta)$是度量聚类结果质量好坏的目标函数。给定具有同样簇数目的两个聚类结果，我们会选择具有更大 $L(D|\Theta)$值的那个结果。

在第 12 章的语言模型和 13.1 节的文本分类当中，也采用了和上述方式相同的做法。在文本分类中，我们选择某篇具体文档的生成似然最大的那个类。在这里，我们选择生成给定文档集的似然最大的聚类方法 Θ。一旦有了 Θ，就可以对每个“文档-簇”对计算一个分配概率 $P(d|\omega_k;\Theta)$。这一系列的分配概率实际上定义了一个软聚类结果。

一个软分配的示例是，一篇有关 Chinese cars 的文档可能对 China 和 automobiles 这两个簇各有 0.5 的隶属概率，这也反映出该文档与两个主题都相关的事实。而诸如 *K*-均值算法的硬聚类方法则不能反映同时与两个主题相关的情况。

基于模型的聚类方法能够提供一个融入领域知识的框架。*K*-均值算法和下一章将要介绍的层次聚类方法对数据的假设相当严格。例如，*K*-均值算法中假定簇满足球形结构，而基于模型的方法则能够提供更大的灵活性。聚类模型可以适用于我们所知的任何数据分布，如表 16-3 例子中对应的贝努利分布、非球形方差的高斯分布（文档聚类的另外一个重要模型）或者其他类型的一些分布。

一个在基于模型的聚类中被广泛使用的算法是 EM 算法。EM 是一个最大化 $L(D|\Theta)$的迭代算法，它可以应用于不同类型的概率建模中。这里我们使用的概率分布模型是多元贝努利混合分布，关于它的介绍可以参看 11.3 节和 13.3 节：

$$P(d \mid \omega_k;\Theta)=\left(\prod_{t_m\in d} q_{mk}\right)\left(\prod_{t_m\notin d}(1-q_{mk})\right)。\qquad (16\text{-}14)$$

其中，$\Theta=\{\Theta_1,\ldots,\Theta_K\}$，$\Theta_k=(\alpha_k, q_{1k}, \ldots, q_{Mk})$，$q_{mk}=P(U_m=1|\omega_k)$是模型的参数[①]。$P(U_m=1|\omega_k)$是簇$k$中的文档包含词项$t_m$的概率。$\alpha_k$是簇$\omega_k$的先验概率，即在对$d$没有任何先验知识的情况下$d$属于$\omega_k$的概率。

于是，混合模型如下：

$$P(d \mid \Theta)=\sum_{k=1}^{K}\alpha_k\left(\prod_{t_m\in d} q_{mk}\right)\left(\prod_{t_m\notin d}(1-q_{mk})\right)。\qquad (16\text{-}15)$$

该模型中，生成一篇文档的过程如下：首先以概率 α_k 选择一个簇 ω_k，然后按照参数 q_{mk} 生成文档的词项。需要提醒的是，基于多元贝努利模型的文档表示是一个 M 维的布尔向量而不是实数向量。

那么，如何利用 EM 算法从数据中推导出这些参数呢？也就是说，如何选择参数 Θ 使得 $L(D|\Theta)$极大化？EM 在步骤上和 K-均值算法类似，它在 E 步（*Expectation Step*，期望）和从步（*Maximization Step*，最大化)这两步之间交替迭代，而 K-均值算法中分别对应的是文档重分配和模型参数重计算这两步。所不同的是，K-均值算法当中的参数是质心，而本节中的 EM 算法实例中的参数是 α_k 和 q_{mk}。

在 M 步中，可以按照如下方式对参数 q_{mk} 和 α_k 进行计算

$$\text{M 步：}\ q_{mk}=\frac{\sum_{n=1}^{N} r_{nk} I(t_m\in d_n)}{\sum_{n=1}^{N} r_{nk}}\quad \alpha_k=\frac{\sum_{n=1}^{N} r_{nk}}{N}。\qquad (16\text{-}16)$$

其中，如果 $t_m\in d_n$，那么 $I(t_m\in d_n)=1$，否则为 0。r_{nk} 表示 d_n 到簇 k 的软分配概率，这将在前一次迭代中计算得到（后面我们马上会讲到初始化问题）。这就是表 13-3 中多元贝努利分布参数的最大似然估计，与之不同的是，这里的文档按概率会分配给多个簇。在给定模型下，这些最大似然估计值能够使得数据的似然取最大值。

$$\text{E 步：}\ r_{nk}=\frac{\alpha_k\left(\prod_{t_m\in d_n} q_{mk}\right)\left(\prod_{t_m\notin d_n}(1-q_{mk})\right)}{\sum_{k=1}^{K}\alpha_k\left(\prod_{t_m\in d_n} q_{mk}\right)\left(\prod_{t_m\notin d_n}(1-q_{mk})\right)}。\qquad (16\text{-}17)$$

E 步应用了公式（16-14）和（16-15）来计算 ω_k 生成文档 d_n 的似然。这对于表 13-3 的多元贝努利模型来说就是分类过程。因此，E 步实际上就是贝努利朴素贝叶斯的分类过程，这其中还包括在不同簇上的概率的归一化过程。

① U_m 是 13.3 节为贝努利朴素贝叶斯模型定义的随机变量。当 t_m 在文档中出现时其值为 1，否则为 0。

对于表 16-3 中的 11 篇文档，我们通过使用 EM 算法将它们聚成 2 个簇。经过 25 次迭代之后，前面的 5 篇文档被分配给簇 1($r_{i,1}$ = 1.00)，而后面的 6 篇文档被分配给簇 2 ($r_{i,1}$ = 0.00)。这里的结果不是特别具有代表性，因为这里的最后聚类结果是个硬分配结果，而 EM 却往往收敛于软分配结果。第 25 次迭代中，由于 11 篇文档中有 5 篇属于簇 1，所以簇 1 的先验概率 α_1=5/11≈0.45。由于初始的分配会很快无歧义地传播给某些词项，所以这些词项会很快与某个簇关联上。比如，在第 1 次迭代中，由于文档 7 和文档 8 都包含 sugar，所以文档 7 对簇 2 的归属度会很快地传播到文档 8（第一次迭代中 $r_{8,1}$=0），对于出现在歧义语境中的词项的参数来说，收敛时间则会更长。种子文档 6 和文档 7 都包含 sweet。但是，需要 25 次迭代之后才能将该词项无歧义地与簇 2 关联上（在第 25 次迭代中，$q_{sweet,1}$ =0）。

表16-3 EM聚类算法的一个示例。

(a)

文档ID	文本内容	文档ID	文本内容
1	hot chocolate cocoa beans	7	sweet sugar
2	cocoa ghana africa	8	sugar cane brazil
3	beans harvest ghana	9	sweet sugar beet
4	cocoa butter	10	sweet cake icing
5	butter truffles	11	cake blank forest
6	sweet chocolate		

(b)

	聚类的迭代次数							
参数	0	1	2	3	4	5	15	25
α_1		0.50	0.45	0.53	0.57	0.58	0.54	0.45
$r_{1,1}$		1.00	1.00	1.00	1.00	1.00	1.00	1.00
$r_{2,1}$		0.50	0.79	0.99	1.00	1.00	1.00	1.00
$r_{3,1}$		0.50	0.84	1.00	1.00	1.00	1.00	1.00
$r_{4,1}$		0.50	0.75	0.94	1.00	1.00	1.00	1.00
$r_{5,1}$		0.50	0.52	0.66	0.91	1.00	1.00	1.00
$r_{6,1}$	1.00	1.00	1.00	1.00	1.00	1.00	0.83	0.00
$r_{7,1}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$r_{8,1}$		0.00	0.00	0.00	0.00	0.00	0.00	0.00
$r_{9,1}$		0.00	0.00	0.00	0.00	0.00	0.00	0.00
$r_{10,1}$		0.50	0.40	0.14	0.01	0.00	0.00	0.00
$r_{11,1}$		0.50	0.57	0.58	0.41	0.07	0.00	0.00
$q_{\text{africa},1}$		0.000	0.100	0.134	0.158	0.158	0.169	0.200
$q_{\text{africa},2}$		0.000	0.083	0.042	0.001	0.000	0.000	0.000
$q_{\text{brazil},1}$		0.000	0.000	0.000	0.000	0.000	0.000	0.000
$q_{\text{brazil},2}$		0.000	0.167	0.195	0.213	0.214	0.196	0.167
$q_{\text{cocoa},1}$		0.000	0.400	0.432	0.465	0.474	0.508	0.600
$q_{\text{cocoa},2}$		0.000	0.167	0.090	0.014	0.001	0.000	0.000
$q_{\text{sugar},1}$		0.000	0.000	0.000	0.000	0.000	0.000	0.000
$q_{\text{sugar},2}$		1.000	0.500	0.585	0.640	0.642	0.589	0.500
$q_{\text{sweet},1}$		1.000	0.300	0.238	0.180	0.159	0.153	0.000
$q_{\text{sweet},2}$		1.000	0.417	0.507	0.610	0.640	0.608	0.667

（续）

注：表格中的（a）部分给出了一系列文档，（b）部分给出了 EM 聚类中若干次循环中的参数值。这些参数包括簇 2 的先验分布 α_1、软分配得分 $r_{n,1}$ 以及部分词项对应的词汇参数 $q_{m,k}$。作者一开始在第 0 次迭代中将文档 6 分配给簇 1、文档 7 分配给簇 2。EM 算法在 25 次迭代以后收敛。为了平滑，公式（16-16）中的 r_{nk} 替换为 $r_{nk}+\varepsilon$，其中 $\varepsilon = 0.0001$。

与k–均值算法相比，寻找好的种子文档对EM算法成为关键。如果种子选择不当，那么EM算法很容易陷入局部最优。这也是EM在其他应用中面对的一个通用问题[①]。因此，同*K*-均值算法一样，EM中初始文档的分配往往通过其他算法来实现。例如，采用*K*-均值硬聚类算法可以提供一个初始的分配结果，然后利用EM进行“软处理”。

? 习题 **16-6**　在上面我们看到 *K*-均值算法的时间复杂度是 Θ(*IKNM*)，那么 EM 算法的时间复杂度是多少？

习题 **16-7**　令 Ω 为一个重现类别结构∐的聚类方法（即 Ω 的聚类结果为∐），而 Ω'在 Ω 的基础上将一些簇进一步分解。请证明：$I(\Omega;\coprod) = I(\Omega';\coprod)$。

习题 **16-8**　请证明 $I(\Omega;\coprod) \le [H(\Omega) + H(\coprod)]/2$。

习题 **16-9**　如果将簇和类别的角色交换，那么互信息的结果不会改变，即 $I(\Omega;\coprod) = I(\coprod;\Omega)$。在这种意义上说，我们称互信息是对称的。同样基于这种意义，其他 3 种评价指标中哪些是对称的？

习题 **16-10**　对于图 16-7，计算两种聚类结果的 RSS 值。

习题 **16-11**　(i) 请给出这样一个例子，由多个点构成的点集合和 3 个初始质心组成（初始质心不一定是点集合中的点），采用 3-均值聚类方法收敛得到的聚类结果中包含空簇 (ii) 这种包含空簇的聚类结果在 RSS 的意义上说有没有可能是全局最优？

习题 **16-12**　下载 Reuters-21578 语料。只考虑以下 10 个类中的文档：acquisitions、corn、crude、earn、grain、interest、money-fx、ship、trade 及 wheat，并且不考虑同时出现在这 10 个类中的两类中的文档。(i) 利用 *K*-均值算法将上述文档子集聚成 10 个类。有很多可以实现 *K*-均值算法的软件包，如 WEKA（Witten 和 Frank 2005）和 R（R Development Core Team 2005）。(ii) 根据上述 10 个类别中的文档情况计算聚类结果的纯度、归一化的互信息、F_1 及 RI 值。(iii) 为 10 个类和 10 个簇建立一个混淆矩阵（参见表 14-5），识别发生假阳性和假阴性的类别。

习题 **16-13**　请证明 $RSS_{min}(K)$ 会随 *K* 的增长而单调递减。

习题 **16-14**　存在一个 *K*-均值算法的软聚类版本，它将文档属于簇的隶属度定义为其到质心的距离 Δ 的单调递减函数，比如 $e^{-\Delta}$，对硬聚类 *K*-均值算法进行修改使之能够进行上述软聚类。

习题 **16-15**　对于表 16-3 中的最后一次迭代，尽管文档 6 一开始作为簇 1 的种子，但是最后却归入簇 2。为什么该文档的归属会改变？

习题 **16-16**　表 16-3 中的第 25 次迭代的参数 q_{mk} 做了四舍五入，那么 EM 将要收敛到的精确值是多少？

习题 **16-17**　对表 16-3 给出的文档进行 *K*-均值算法聚类。经过多少次迭代之后，*K*-均值算法会收敛？将 *K*-均值算法得到的聚类结果和表 16-3 中 EM 算法的结果进行比较并讨论其不同之处。

习题 **16-18 [***]**　针对一个高斯混合模型，对 EM 算法的 E 步和 M 步进行修改。M 步计算每个簇的 α_k、$\vec{\mu}_k$ 及 Σ_k 的最大似然估计值，E 步基于当前参数计算每个向量到簇的软分配概率。写出在高斯混合分布下与公式（16-16）及公式（16-17）对应的公式。

习题 **16-19 [***]**　请证明高斯混合分布下的 EM 算法在方差很小、所有协方差都是 0 的时候会退化成 *K*-均值算法，即此时 *K*-均值算法是 EM 算法的一个受限形式。

① 例如，该问题也发生在 NLP 领域的 HMM 模型、概率文法及机器翻译的参数估计中 (参见 Manning 和 Schäutze 1999)。

习题 **16-20 [***]** 聚类结果的点内分散度（within-point scatter）定义为：$\sum_k \frac{1}{2}\sum_{\vec{x}_i \in \omega_k}\sum_{\vec{x}_j \in \omega_k} |\vec{x}_i - \vec{x}_j|^2$。请证明最小化 RSS 和最小化点内分散度是等价的。

习题 **16-21 [***]** 为公式（16-12）中的多元贝努利混合模型推导出一个 AIC 准则。

16.6 参考文献及补充读物

Berkhin（2006b）给出了一个当前聚类方法的综合介绍，其中聚类方法的扩展托备受关注。模式识别领域有关聚类的经典参考文献是（Duda 等人 2000），其中包括了 *K*-均值算法和 EM 算法。Rasmussen（1992）从 IR 的角度介绍了聚类方法。Anderberg（1973）则对应用中的聚类算法给出了一个一般性介绍。除了欧氏距离和余弦相似度之外，KL 距离也往往用于度量文档和簇之间的相似或不相似程度（Xu 和 Croft 1999; Muresan 和 Harper 2004; Kurland 和 Lee 2004）。

聚类假设归功于Jardine 和 van Rijsbergen（1971），当时的说法如下[①]："文档之间的关联能够为文档和需求的相关性传递信息。" Salton（1971a，1975）、Croft（1978）、Voorhees（1985a）、Can and Ozkarahan（1990）、Cacheda 等人（2003）、Can 等人（2004）、Singitham 等人（2004）以及Altingövde 等人（2008）考察了基于聚类的检索的效率和效果。尽管其中有些研究表明效果或效率或两者同时有所提高，但是对于基于聚类的检索在各种场景下是否都能很好地工作并没有一致结论。基于聚类的语言建模方法由Liu 和 Croft（2004）首创。

有可靠的证据可以表明，尽管搜索结果的聚类并不如人工精心编辑的层次类别那样具有良好的结构（Hearst 2006），但是它确实能够提高用户的体验和搜索结果的质量（Hearst 和 Pedersen 1996; Zamir 和 Etzioni 1999; Tombros 等人 2002; Käki 2005; Toda 和 Kataoka 2005）。采用"分散-集中"策略的文档集浏览界面来自 Cutting 等人（1992）。一个分析分散-集中以及其他信息探求用户界面的理论框架参见 Pirolli（2007）。Schütze 和 Silverstein（1997）评价了 LSI（参见第 18 章），并为提高 *K*-均值算法的效率对质心的表示进行了约简。

哥伦比亚大学的 NewsBlaster 系统（McKeown 等人 2002），是一个比现在最著名的 Google News（http://news.google.com）早得多的先驱系统，它使用了层次聚类（第 17 章）给出了新闻主题的两级粒度。具体的细节可以参考 Hatzivassiloglou 等人（2000），其他的相关系统可以参见 Chen 和 Lin（2000）以及 Radev 等人（2001）等。聚类在 IR 中的其他应用还包括查重（Yang 和 Callan（2006），参见 19-6 节）、新信息检测（novelty detection，参考 17-9 的参考文献）及语义 Web 上的元数据发现（Alonso 等人 2006）。

有关外部评价指标的讨论部分基于 Strehl（2002）的工作。Dom（2002）提出了一个在理论上比 NMI 动机更优的评价指标 Q_0，它是在假定簇归属已知的情况下传递归属信息所需要的

① 原文是"*Associations between documents convey information about the relevance of documents to requests*"。——译者注

比特位的数目。兰德指数 RI 归功于 Rand（1971）的工作。Hubert 和 Arabie（1985）提出一个调整的兰德指数（*adjusted Rand index*）方法，它的值在−1 到 1 之间，当它等于 0 时簇和类别之间只存在偶然的一致性（与第 8 章的 κ 指标类似）。Basu 等人（2004）认为三种评价指标 NMI、兰德指数 RI 和 *F* 值的结果非常类似。Stein 等人（2003）提出一种称为预期边缘密度（*expected edge density*）的内部指标，并给出证据来表明它是预测聚类质量的一个很好的指标。Kleinberg（2002）和 Meilă（2005）给出了比较聚类方法的公理化框架。

以 *K*-均值算法而闻名于世的作者包括 Lloyd（1982）（最早发布在 1957 年）、Ball（1965）、MacQueen（1967）和 Hartigan 和 Wong（1979）。Arthur 和 Vassilvitskii（2006）考察了 *K*-均值算法在最坏情况下的复杂度。Bradley 和 Fayyad（1998）、Pelleg 和 Moore（1999）及 Davidson 和 Satyanarayana（2003）凭借经验考察了 *K*-均值算法的收敛性质及其对初始种子的选择的依赖情况。Dhillon 和 Modha（2001）比较了 *K*-均值算法中的簇和基于 SVD 分解的簇（参见第 18 章）。*K*-中心点（*K*-medoid）算法由 Kaufman 和 Rousseeuw（1990）提出。EM 算法最早由 Dempster 等人（1977）引入。有关 EM 的深入处理请参见（McLachlan 和 Krishnan 1996）。隐性分析（latent analysis）可以看成一种软聚类方法，有关它的介绍请参考 18-5 节给出的有关文献。

有关 AIC 的工作归功于 Akaike（1974）（也请参考 Burnham 和 Anderson（2002））。一个可替代 AIC 的准则是 BIC，它可以被看成是一个贝叶斯模型选择过程（Schwarz 1978）的推动结果。Fraley 和 Raftery（1998）给出了基于 BIC 来选择最优簇数目的方法。BIC 在 *K*-均值算法中的一个应用可以参见（Pelleg 和 Moore 2000）。Hamerly 和 Elkan（2003）又给出了一个替代 BIC 的方法，并通过实验说明了它的有效性。另一个具有影响力的决定簇数目（同时也给出了文档到簇的分配方法）的贝叶斯方法的介绍参见 Cheeseman 和 Stutz（1996）。Tibshirani 等人（2001）提出了两个不依赖外部准则来确定簇数目的方法。

由于本书篇幅所限，这里我们只介绍了完全无监督的聚类方法。实际上，一个非常重要的研究课题是如何融入用户的先验知识来指导聚类（比如 Ji 和 Xu（2006）的工作），以及如何在聚类中融入用户的交互反馈信息（比如 Huang 和 Mitchell（2006）的工作）。Fayyad 等人（1998）提出了一个 EM 聚类的初始化方法。通过单遍数据扫描就能对大规模数据集聚类的算法参见 Bradley 等人（1998）。

表 16-1 中的例子都是基于文档的聚类。其他的 IR 应用中可以对词聚类（比如 Crouch 1988 的工作）、词的上下文聚类（比如 Schütze 和 Pedersen 1995 的工作）或者同时对词和文档聚类（比如 Tishby 和 Slonim 2000、Dhillon 2001 及 Zha 等人 2001 的工作）。词和文档同时聚类可以被认为是协同聚类（co-clustering）或双聚类（biclustering）的一个例子。

第17章 层次聚类

在第16章中介绍的扁平聚类具有概念简单、速度快的优点，但是同时也有很多缺点。上一章中介绍的算法返回的是一个无结构的扁平簇集合，它们需要预先定义簇的数目，并且聚类结果具有不确定性。与之不同的是，层次聚类（hierarchical clustering或hierarchic clustering）则会输出一个具有层次结构[①]的簇集合，因此能够比扁平聚类输出的无结构簇集合提供更丰富的信息。层次聚类不需要事先指定簇的数目，并且大部分用于IR中的层次聚类算法都是确定性算法。当然，层次聚类在获得这些好处的同时，其代价就是效率降低。最普遍的层次聚类算法的时间复杂度至少是文档数目的平方级，而前面提到的K-均值算法和EM算法（参考 16.4 节）的时间复杂度都是线性的。

本章将首先在 17.1 节中介绍HAC（Hierarchical Agglomerative Clustering，凝聚式层次聚类），然后从 17.2 节到 17.4 节将分别给出四种不同的凝聚式算法，它们分别采用了不同的相似度计算方法[②]：单连接（single-link）、全连接（complete-link）、组平均（group-average）及质心（centroid）相似度方法。17.5 节中讨论了层次聚类的最优性条件。17.6 节介绍了自顶向下（也称为分裂式divisive）的层次式聚类方法。17.7 节介绍了簇标签自动生成的问题，只要存在用户和聚类结果的交互，那么我们就必须解决这个问题。17.8 节讨论了聚类方法的实现问题。17.9 节给出了一些补充读物，其中给出了本书没有覆盖的软层次聚类的相关参考文献。

在IR中，扁平聚类和层次聚类的应用领域几乎没有差别。特别是，对于表16-1（参见16.6节）所示的任何应用，层次聚类都能适用。实际上，表中给出的文档集聚类就是一个层次聚类的例子。通常来说，当效率因素非常重要时，我们选择扁平聚类算法。而当扁平算法的问题（如结构信息不足、簇数目需要预先定义、聚类结果非确定性）需要加以考虑时，我们则采用层次算法。此外，有很多研究人员相信，层次方法产生的聚类结果会比扁平方法更好。然而，关于这一点目前还没有形成一致的结论（参考17-9给出的参考文献）。

① 本章只考虑图17-1所示的那种二叉树层次结构，但是层次聚类很容易扩展到其他类型的树结构。

② 这四种方法有时也被称为单链(单链接、单连通)、全链(全链接、全连通)、平均连通及中心相似度方法。——译者注

17.1 凝聚式层次聚类

层次聚类可以是自顶向下或自底向上的一个过程。自底向上的算法一开始将每篇文档都看成是一个簇，然后不断地对簇进行两两合并（或称凝聚（agglomerate）），直到所有文档都聚成一类为止。自底向上的聚类方法也因此被称为 HAC。而自顶向下的方法则首先将所有文档看成一个簇，然后不断利用某种方法对簇进行分裂直到每篇文档都成为一个簇为止（参见 17.6 节）。在 IR 领域，HAC 方法的使用比自顶向下方法更普遍，本章主要介绍的也是这种方法。

在介绍 HAC 算法中的相似度计算方法之前，首先我们先介绍层次聚类的一个图形化表示方法，然后介绍 HAC 的一些特点，最后给出一个简单的 HAC 实现算法。

HAC 聚类结果往往可以采用图 17-1 中所示的树状图（dendrogram）来表示。其中，每篇文档一开始都被看成一个单独的簇，聚类过程中的每次合并都表示成一条水平线，而 y 轴对应的是合并时两个簇的相似度，这个相似度也被称为合并后的簇的结合相似度（combination similarity）。例如，图 17-1 中标题为 Lloyd's CEO questioned 及 Lloyd's chief /U.S. grilling 的两篇文档合并成的簇的结合相似度约等于 0.56。对于单篇文档构成的簇，我们将它的结合相似度定义成文档的自相似度，当采用余弦相似度计算时，这个值为 1.0。

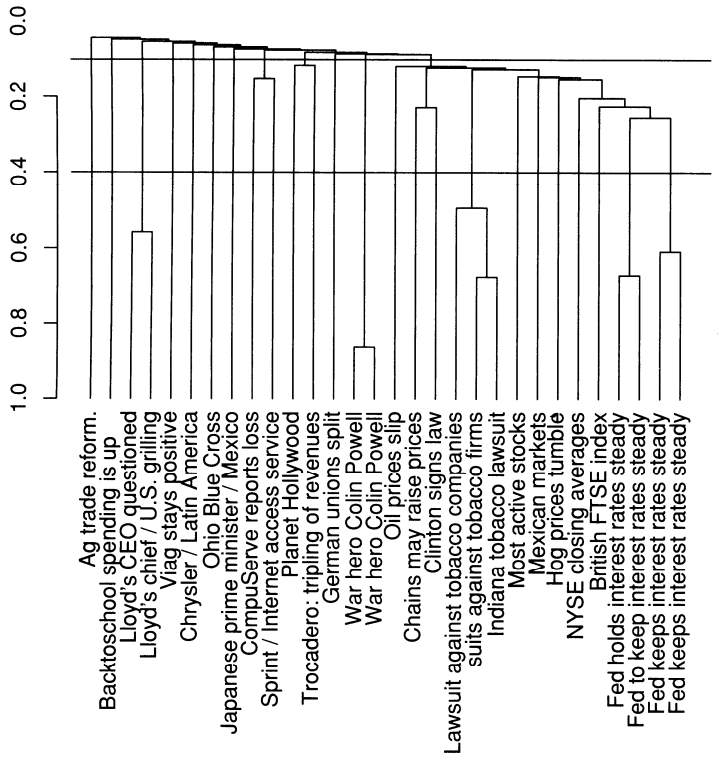


图 17-1 Reuters-RCV1 语料中 30 篇文档的单连接聚类树状图。图中用水平线给出了

其中两步的聚类结果：在 0.4 处聚成 24 个簇，在 0.1 处聚成 12 个簇

通过从底层节点一直上升到顶层节点，树状图能够记录整个合并历史并能重构整个合并过程。例如，在图 17-1 中我们看到，两篇标题为 War hero Colin Powell 的文档被首先合并，而最后一次合并则是将标题为 Ag trade reform 的文档加入到其它 29 篇文档组成的簇中。

HAC 的一个基本假设是，HAC 聚类中的合并操作是单调的（monotonic）。这里的单调性意味着，如果 HAC 中的合并结果的结合相似度依次是 $s_1, s_2, \ldots, s_{K-1}$，那么有 $s_1 \geq s_2 \geq \ldots \geq s_{K-1}$。而非单调的层次聚类中则至少包含一个颠倒现象（inversion），即存在某个 $s_i < s_{i+1}$，这会和每次选择最好的合并操作所依赖的假设相矛盾。在图 17-12 中我们将会看到这样一个颠倒现象。

层次聚类不需要事先给定簇的数目。但是，在一些应用中，我们希望像扁平聚类一样得到不相交的多个簇。这些情况下，需要在层次结构中的某一点上进行截断（即停止聚类），截断点的确定方法则有如下多种做法。

- 在某个事先给定的相似度水平上进行截断。比如，如果我们希望结果簇的结合相似度不低于 0.4 的话，那么我们就在 0.4 处截断树状图。图 17-1 中，在 y=0.4 处截断后会产生 24 个簇，此时产生的簇的结合相似度较高，而在 y=0.1 处截断后会产生 12 个簇，此时产生的簇的结合相似度较低，聚类结果由一个大的金融新闻簇和其他 11 个较小的簇组成。
- 当两个连续的聚类结果的结合相似度之差最大时进行截断。这种较大的差值也意味着很“自然”的聚类结果，多增加一个簇之后的聚类结果质量会显著下降。这种策略与图 16-8 中寻找 K-均值聚类拐点的方法类似。
- 应用公式（16-11）进行截断：

$$K = \arg\min_{K'}[\mathrm{RSS}(K') + \lambda K']。$$

其中，K' 指的是对层次结构进行截断后产生的结果簇的数目，RSS 是残差平方和，而 λ 是每额外增加一个簇时的惩罚量。这里也可以使用 RSS 之外的失真率计算方法。

- 和扁平聚类一样，也可以事先指定结果簇的数目 K，在产生 K 个簇时进行截断。

图 17-2 中给出了一个简单的朴素HAC算法。该算法中，首先计算一个 $N \times N$ 的相似度矩阵 C，然后算法执行 $N-1$ 步，每一步都将最近的两个簇进行合并。每次迭代中，合并最相近的两个簇 i 和 m，合并后得到的簇仍然用 i 来标识，此时簇 i 在 C 中对应的行和列要进行更新[①]。每次聚类的结果不断添加到 A 中。数组 I 表示仍然可以被合并的簇（此时对应的值为 1）。$\mathrm{SIM}(i, m, j)$ 计算的是簇 i 及 m 的合并簇与簇 j 的相似度。对于某些HAC算法而言，$\mathrm{SIM}(i, m, j)$ 可以是 $C[j][i]$ 和 $C[j][m]$ 的一个简单函数，比如，在单连接算法中取这两个值中的较大值。

① 在这里我们假定，在遇到多个相似度得分相等的情况下，我们会使用一个确定性的算法来选择聚类方式。比如，如果将文档集 D 的所有子集进行编号的话，那么在上述情况下，我们始终从多个可能选项中选择基于此编号的第一个簇进行合并。

```
SIMPLEHAC(d_1, ..., d_N)
 1  for n ← 1 to N
 2  do for i ← 1 to N
 3     do C[n][i] ← SIM(d_n, d_i)
 4     I[n] ← 1  (keeps track of active clusters)
 5  A ← []  (collects clustering as a sequence of merges)
 6  for k ← 1 to N − 1
 7  do ⟨i, m⟩ ← arg max_{{⟨i,m⟩:i≠m∧I[i]=1∧I[m]=1}} C[i][m]
 8     A.APPEND(⟨i, m⟩)  (store merge)
 9     for j ← 1 to N
10     do C[i][j] ← SIM(i, m, j)
11        C[j][i] ← SIM(i, m, j)
12     I[m] ← 0  (deactivate cluster)
13  return A
```

图 17-2 一个简单但是低效的 HAC 算法

在接下来的各节当中，我们将采用不同的相似度计算方法对图 17-2 所示的算法进行修改：17.2 节将介绍单连接和全连接聚类算法，17.3 节和 17.4 节将介绍组平均和质心聚类算法。图 17-3 中给出了这四种 HAC 聚类算法的合并准则。

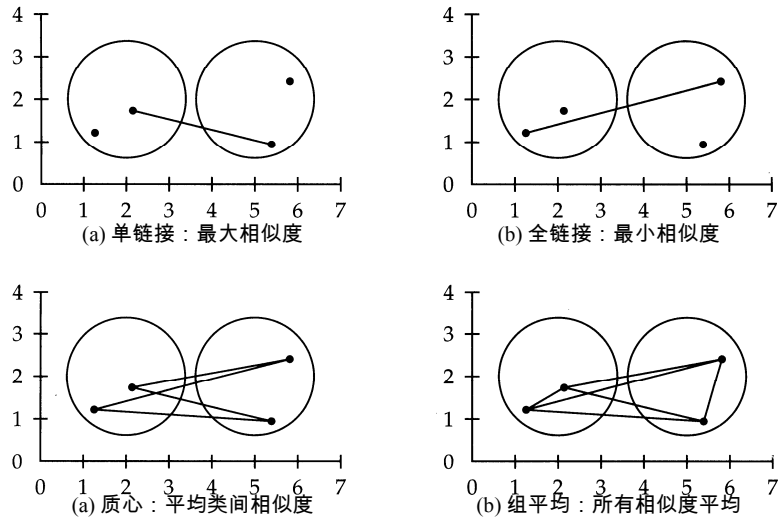


图 17-3 四种 HAC 算法中所使用的不同的簇相似度概念。类间相似度（inter-similarity）指的是来自不同簇的文档的相似度

17.2 单连接及全连接聚类算法

在单连接聚类（single-link clustering或single-linkage clustering）中，两个簇之间的相似度定

义为两个最相似的成员之间的相似度（参考图 17-3（a））[①]。这种单连接的合并准则是局部的（local），即它仅仅关注两个簇互相邻近的区域，而不考虑簇中更远的区域和簇的总体结构。

在全连接聚类（complete-link clustering 或 complete-linkage clustering）中，两个簇之间的相似度定义为两个最不相似的成员之间的相似度（参考图 17-3（b）），这也相当于选择两个簇进行聚类，使得合并结果具有最短直径。全连接聚类准则是非局部的，聚类结果中的整体结构信息会影响合并的结果。这种聚类实际上相当于优先考虑具有较短直径的紧凑簇，而不是具有长直径的松散簇，当然这种做法可能会对离群点较为敏感，比如某个远离中心的文档会显著增加候选簇的直径从而完全改变最后的聚类结果。

图 17-4 给出了 8 篇文档在单连接及全连接算法下的聚类结果。前四步中，两个算法每次均产生一个由两篇文档组成的簇，它们的结果完全一样。然后，由于按照单连接算法中簇最大相似度的定义，上面的两个簇之间的相似度大。所以，单连接算法先合并上面的两个簇，然后根据同样的准则再合并下面的两个簇。而按照全连接算法中簇相似度的定义，左边的两个簇相似度最大。所以，全连接算法先合并左边的两个簇，然后根据同样的准则再合并右边的两个簇。

图 17-1 给出的是在一个文档集合上进行单连接聚类的例子，而图 17-5 给出的是在同样的文档集合上进行全连接聚类的例子，在最后一次合并之前，有两个大小相当的簇（文档 1-16 组成一个簇，标题从 NYSE closing averages 到 Lloyd's chief / U.S. grilling。文档 17 到 30 组成另外一个簇，标题从 Ohio Blue Cross 到 Clinton signs law）。而在图 17-1 所示的树状图中，不存在一个可以得到非常均衡的聚类结果的截断点。

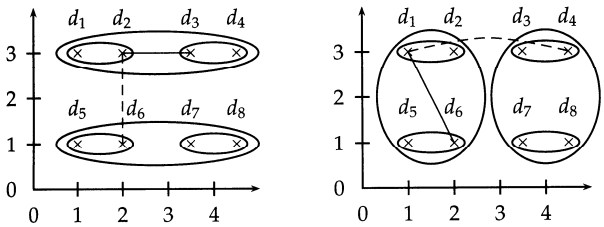


图 17-4　8 篇文档的单连接聚类（左图）和全连接聚类（右图）算法。每个椭圆对应聚类的一步。左图：在单连接聚类中，上面两个簇的相似度为d_2和d_3的相似度（用实线表示），左边两个簇的相似度为d_2和d_6的相似度（用虚线表示），前者显然大于后者；右图：在全连接聚类中，上面两个簇的相似度为d_1和d_4的相似度（用实线表示），左边两个簇的相似度为d_1和d_6的相似度（用虚线表示），前者显然小于后者[②]

① 本章中，我们将相似度和二维空间描述的聚类中的邻近度这两个概念等价。

② 如果读者对这里面的各种距离组合理解不清的话，那么就可以通过坐标系上的距离来理解，比如可以设想 d_1 的坐标是（$1+\varepsilon, 3-\varepsilon$），其中ε 是一个很小的数，而其他各点的坐标都是整数。

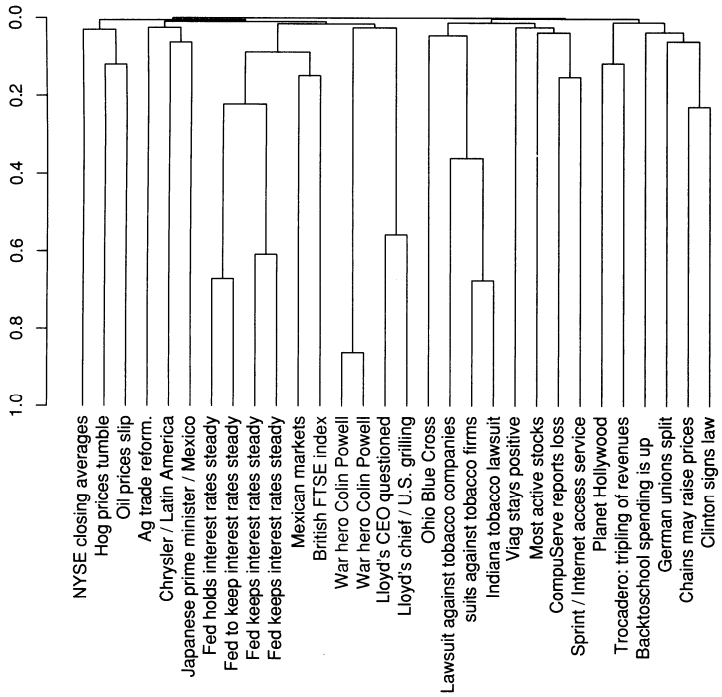


图 17-5 全连接聚类树状图。参加聚类的 30 篇文档同图 17-1 中的一样。

不论是单连接还是全连接匹配算法都可以用图论来解释。令 s_k 为第 k 步中两个簇合并后的结合相似度，$G(s_k)$为任意两点间相似度不低于 s_k 的点的连接图。因此，单连接聚类第 k 步后得到的簇其实是 $G(s_k)$的连通分支，而采用全连接聚类在第 k 步后得到的簇则构成 $G(s_k)$的最大团。连通分支（connected component）指的是在任意两点之间都存在可达路径的最大点集。而团（clique）指的是所有点之间都存在边的全连通图。

上述基于图论的解释方法也是单连接和全连接聚类方法得名的由来。单连接方法中第 k 步的所谓单连接簇指的是这样一些点的集合：它们当中至少存在两个点之间的相似度满足 $s \geq s_k$；而全连接方法中第 k 步的簇中所有点之间的相似度均满足 $s \geq s_k$。

单连接和全连接聚类方法将簇质量的计算过程简化成两个文档的单一相似度计算，其中在单连接方法中计算的是两篇最相似的文档之间的相似度，而在全连接方法中计算的是两篇最不相似的文档之间的相似度。仅仅根据两篇文档来计算显然不能完全反映出簇中的文档分布情况，因此，这两种聚类方法产生的结果簇往往不是非常理想。比如，在图 17-6 中的单连接聚类算法产生了非常分散的簇。由于上述方法中的合并准则的局部性要求非常严格，一连串的多个点会因此合并成一个长链，而不会考虑合并后的簇的整体形状。这种效果称为链化（chaining）。

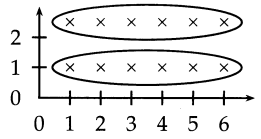


图 17-6　单连接聚类中的链化现象。单连接聚类算法中的局部准则会导致非期望的狭长簇

图 17-1 也给出了具有链化效果的一个示例。单连接聚类当中的最后 11 次合并（0.1 对应的直线之上）中，每次要么加入单篇文档、要么加入一对文档构成的簇，这样最后就形成一条链。而图 17-5 中的全连接聚类就可以避免这种问题。在最后一次合并前，所有文档被分成两个大小近似相等的组。通常来说，这样的数据组织结构会比链式结构要更有用。

然而，全连接聚类会遇到另外一个问题，它对那些与总体结构不太一致的离群点给予了过多的关注。在图 17-7 的示例中，由于存在左边离群点 d_1（参考习题 17-1），d_2、d_3、d_4、d_5 这四篇文档被分成两部分。也就是说，该例子中，全连接聚类并不能找出最直观的簇结构。

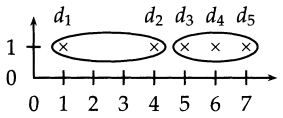


图 17-7　全连接聚类中的离群点。五篇文档的 x 坐标分别是 $1+2\varepsilon$、4、$5+2\varepsilon$、6 及 $7-\varepsilon$。全连接聚类算法会产生图示的两个椭圆形聚类结果。直观上看，上述文档应该分成两个簇 $\{\{d_1\}, \{d_2, d_3, d_4, d_5\}\}$，但是采用全连接聚类，离群点 d_1 会将 $\{d_2, d_3, d_4, d_5\}$ 分开

时间复杂度

在图 17-2 所示的朴素 HAC 算法中，需要进行 N-1 次迭代，而每次迭代需要在 $N \times N$ 的矩阵 C 中进行彻底搜索直至找到最大的相似度，因此该算法的复杂度为 $\Theta(N^3)$。

对于本章提到的四种HAC方法，图 17-8 给出了一个基于优先队列[①]的更高效的实现算法，它的时间复杂度是$\Theta(N^2\log N)$。$N \times N$矩阵C的第k行$C[k]$按照降序存入优先队列P中。然后，$P[k]$.MAX()返回当前的P[k]中与ω_k相似度最大的簇，其中ω_k表示第 16 章中所说的第k个簇。将两个簇ω_{k1}及ω_{k2}合并后，可以用ω_{k1}作为整个簇的代表。SIM是簇之间的相似度计算函数，在单连接、全连接、组平均（参见 17.3 节）及质心方法（参见 17.4 节）分别被定义为最大相似度、最小相似度、平均相似度及质心相似度。以下将给出C中处理每一行的例子（参见图 17-8 的底部）。由于支持删除、插入的优先队列的实现复杂度是$\Theta(\log N)$，所以两个高层的循环（第 1-7 行及第 9-21 行）的复杂度都是$\Theta(N^2\log N)$。因此，整个算法的总复杂度是$\Theta(N^2\log N)$。在SIM函数的定义中，$\vec{v}_m$ 和 $\vec{v}_i$ 分别是簇$\omega_{k1} \cup \omega_{k2}$及簇$\omega_i$的向量和，而$N_m$和$N_i$分别是这两个簇中的文档数目。

① 在数据结构中，优先队列（priority queue）是不同于先进先出队列的另一种队列。每次从优先队列中取出的是具有最高优先权的元素。通常，优先队列可以通过堆（heap）来实现。——译者注

```
EFFICIENTHAC(d⃗_1, ..., d⃗_N)
 1  for n ← 1 to N
 2  do for i ← 1 to N
 3     do C[n][i].sim ← d⃗_n · d⃗_i
 4        C[n][i].index ← i
 5     I[n] ← 1
 6     P[n] ← priority queue for C[n] sorted on sim
 7     P[n].DELETE(C[n][n]) (don't want self-similarities)
 8  A ← []
 9  for k ← 1 to N − 1
10  do k_1 ← arg max_{k:I[k]=1} P[k].MAX().sim
11     k_2 ← P[k_1].MAX().index
12     A.APPEND(⟨k_1, k_2⟩)
13     I[k_2] ← 0
14     P[k_1] ← [ ]
15     for each i with I[i] = 1 ∧ i ≠ k_1
16     do P[i].DELETE(C[i][k_1])
17        P[i].DELETE(C[i][k_2])
18        C[i][k_1].sim ← SIM(i, k_1, k_2)
19        P[i].INSERT(C[i][k_1])
20        C[k_1][i].sim ← SIM(i, k_1, k_2)
21        P[k_1].INSERT(C[k_1][i])
22  return A
```

	SIM(i, k_1, k_2)
聚类算法	max(SIM(i, k_1), SIM(i, k_2))
单连接方法	min(SIM(i, k_1), SIM(i, k_2))
全连接方法	$(\frac{1}{N_m}\vec{v}_m) \cdot (\frac{1}{N_i}\vec{v}_i)$
组平均方法	$\frac{1}{(N_m+N_i)(N_m+N_i-1)}[(\vec{v}_m + \vec{v}_i)^2 - (N_m + N_i)]$

质心方法

计算 C[5]

1	2	3	4	5
0.2	0.8	0.6	0.4	1.0

通过排序建立 P[5]

2	3	4	1
0.8	0.6	0.4	0.2

合并 2 和 3，更新 2 的

2	4	1
0.3	0.4	0.2

删除并插新插入 2

4	2	1
0.4	0.3	0.2

图 17-8 HAC 中的优先队列算法。上部：算法。中部：四种不同的相似度计算方法。底部：第 6 行及 16-19 的执行例子，这里用的是一个 5×5 的矩阵 C 中的元素 P[5]

由于组平均凝聚式聚类和质心聚类方法都需要向量作为输入，所以图 17-8 中 EFFICIENTHAC 函数的输入不是原始的文档集，而是一系列向量的集合。当然，EFFICIENTHAC 算法的全连接版本可以处理非向量表示的文档。

对于单连接聚类方法来说，可以引入 NBM（*next-best-merge*，下次最佳合并）数组来做进一步优化（参见图 17-9）。NBM 保留了对每个簇的最佳合并轨迹。图 17-9 中的两个高层循环的复杂度为 $\Theta(N^2)$，因此单连接聚类的总复杂度为 $\Theta(N^2)$。

```
SINGLELINKCLUSTERING(d_1, ..., d_N)
 1  for n ← 1 to N
 2  do for i ← 1 to N
 3     do C[n][i].sim ← SIM(d_n, d_i)
 4        C[n][i].index ← i
 5     I[n] ← n
 6     NBM[n] ← arg max_{X∈{C[n][i]:n≠i}} X.sim
 7  A ← []
 8  for n ← 1 to N − 1
 9  do i_1 ← arg max_{i:I[i]=i} NBM[i].sim
10     i_2 ← I[NBM[i_1].index]
11     A.APPEND(⟨i_1, i_2⟩)
12     for i ← 1 to N
13     do if I[i] = i ∧ i ≠ i_1 ∧ i ≠ i_2
14           then C[i_1][i].sim ← C[i][i_1].sim ← max(C[i_1][i].sim, C[i_2][i].sim)
15        if I[i] = i_2
16           then I[i] ← i_1
17     NBM[i_1] ← arg max_{X∈{C[i_1][i]:I[i]=i∧i≠i_1}} X.sim
18  return A
```

图 17-9　使用 NBM 数组的单连接聚类算法。两个簇 i_1 和 i_2 合并之后，我们用 i_1 来代表合并后的簇结果。如果 I[i]=i，那么 i 就是当前簇的代表。如果 I[i]≠i，那么 i 就已经合并到 I[i]所表示的簇中，因此当对 NBM[i_1]进行更新时不再考虑

那么，能否利用 NBM 数组来对其他三种 HAC 算法进行加速呢？答案是否定的这是由于只有单连接方法才具有最佳合并持续性（best-merge persistent）。也就是说，如果假定在单连接聚类中 ω_k 的最佳合并簇是 ω_j，那么在将 ω_j 与第三个簇 $\omega_i \neq \omega_k$ 合并之后，那么 ω_i、ω_j 的合并结果将会是 ω_k 的最佳合并簇（参见习题 17-6）。换句话说，在单连接聚类中，两个簇合并后得到新簇，其最佳合并候选簇是其合并前的两个簇的最佳合并簇之一。这也就意味着每次迭代中，C 可以在 $\Theta(N)$时间内更新，这时只需要在图 17-9 中的第十四行对剩余的多个簇（数目$\leqslant N$）中的每个簇进行一个简单的最大值计算。

图 17-10 表明，全连接聚类并不满足最佳合并持续性，这也意味着不能利用一个 NBM 数组来加速其聚类过程。在将 d_3 的最佳合并簇 d_2 与簇 d_1 合并之后，一篇不相关的簇 d_4 变成了 d_3 的最佳合并簇。这是因为全连接聚类算法中的合并准则是非局部的，离两个合并簇区域很远的点能够对聚类产生很大影响。

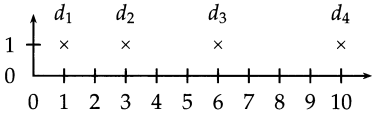


图 17-10　关于全连接聚类不具最佳合并持续性的一个示例。一开始，d_2 是 d_3 的最佳合并簇，但是将 d_1、d_2 合并之后，d_4 变成了 d_3 的最佳合并簇。在一个具有合并持续性的算法（如单连接算法）当中，d_3 的最佳合并簇应该是 $\{d_1, d_2\}$。

实际当中，由于文档的相似度计算（如计算向量内积）比两个值的大小比较要慢一个数量级，因此与复杂度为 $\Theta(N^2)$的单连接算法相比，复杂度为 $\Theta(N^2\log N)$的算法在效率上的损失并没有那么大。本章中提到的所有四种 HAC 算法相对于相似度计算的时间复杂度为 $\Theta(N^2)$。因此，在实际操作当中，选择这几种算法时时间复杂度的差异并不是主要考虑因素。

? 习题 **17-1** 给出利用全连接聚类方法建立图 17-7 所示的两个簇的过程。

17.3 组平均凝聚式聚类

GAAC（Group-average Agglomerative Clustering，组平均凝式聚类）通过计算所有文档之间的相似度来对簇的质量进行计算，因此可以避免在单连接和全连接准则中只计算一对文档相似度的缺陷。GAAC 也被称为组平均聚类（group-average clustering）或平均连接聚类（average-link clustering）。GAAC 可以计算所有文档之间相似度的平均值 SIM-GA，其中也包括来自同一簇的文档。当然，这种自相似度在这里并没有使用。计算公式如下：

$$\text{SIM-GA}(\omega_i,\omega_j)=\frac{1}{(N_i+N_j)(N_i+N_j-1)}\sum_{d_m\in\omega_i\cup\omega_j}\sum_{d_n\in\omega_i\cup\omega_j,d_n\neq d_m}\vec{d}_m\cdot\vec{d}_n \quad 。 \qquad (17\text{-}1)$$

其中，$\vec{d}$ 是文档 d 的长度归一化向量，$\cdot$ 是内积运算符，N_i 和 N_j 分别是 ω_i 和 ω_j 中的文档数目。

GAAC 的动机在于，如果在 HAC 算法中选择 ω_i 和 ω_j 进行下一次合并，那么就要保证合并之后的结果 $\omega_k=\omega_i\cup\omega_j$ 具有一致性（coherence）。为了判断 ω_k 的一致性，我们就要计算 ω_k 中两两文档之间的相似度，其中也包括 ω_i 中内部文档之间以及 ω_j 中内部文档之间的相似度。

由于向量相似度之和等于向量和的相似度，所以可以采用下列方法对 SIM-GA 值进行高效计算：

$$\sum_{d_m\in\omega_i}\sum_{d_n\in\omega_j}(\vec{d}_m\cdot\vec{d}_n)=\left(\sum_{d_m\in\omega_i}\vec{d}_m\right)\cdot\left(\sum_{d_n\in\omega_j}\vec{d}_n\right)。 \qquad (17\text{-}2)$$

于是，有

$$\text{SIM-GA}(\omega_i,\omega_j)=\frac{1}{(N_i+N_j)(N_i+N_j-1)}\left[\left(\sum_{d_m\in\omega_i\cup\omega_j}\vec{d}_m\right)^2-(N_i+N_j)\right]。 \qquad (17\text{-}3)$$

其中，式子右部的（N_i+N_j）是 N_i+N_j 个值为 1 的自相似度之和。假定两个向量和 $\sum_{d_m\in\omega_i}\vec{d}_m$ 和 $\sum_{d_m\in\omega_j}\vec{d}_m$ 已知，那么利用上述技巧就可以在常数时间而不是 $\Theta(N_iN_j)$内计算两个簇的相似度。因为在图 17-8 给出 EFFICIENTHAC 算法中，必须要能够对第 18 行和第 20 行的 SIM 函数在常数时间内进行计算，从而保证 GAAC 的高效性。需要指出的是，如果对两个单文档簇进行计算时，公式（17-3）实际上就等价于内积计算。

公式（17-2）成立的原因在于内积计算满足加法分配律。由于对于 GAAC 的高效实现来说，

这一点至关重要，所以不容易将该方法推广到非实数向量的其他文档表示上。同样，公式（17-2）仅在使用内积计算时成立。尽管本书介绍的很多算法不论是采用内积、余弦相似度还是欧氏距离（参考 14.1 节）都有近似于等价的描述，但是公式（16-2）仅仅只能采用内积来表示。这是单连接或全连接聚类和 GAAC 最根本的区别，前面两种聚类方法仅仅需要相似度平方矩阵作为输入，而并不关心到底如何计算这些相似度。

概括地说，GAAC 要求 (i) 文档要表示成向量；(ii) 向量要基于长度归一化，以保证自相似度为 1.0；(iii) 采用内积计算向量与向量和之间的相似度。

除了使用图 17-8 中公式（17-3）进行相似度计算外，GAAC 中的合并算法和全连接聚类算法完全一样。因此，GAAC 的总时间复杂度和全连接聚类一样都是 $\Theta(N^2\log N)$。同全连接聚类算法一样，GAAC 不具有最佳合并持续性（参考习题 17-6）。这也意味着，不存在时间复杂度为 $\Theta(N^2)$的类似于图 17-9 中单连接算法的 GAAC 实现方法。

我们也可以将自相似度考虑在内，定义这种情况下的组平均相似度：

$$\text{SIM-GA}'(\omega_i,\omega_j)=\frac{1}{(N_i+N_j)^2}\left(\sum_{d_m\in\omega_i\cup\omega_j}\vec{d}_m\right)^2=\frac{1}{N_i+N_j}\sum_{d_m\in\omega_i\cup\omega_j}[\vec{d}_m\cdot\vec{\mu}(\omega_i\cup\omega_j)]。\quad(17\text{-}4)$$

其中，采用公式（14-1）定义质心向量 $\vec{\mu}(\omega)$。公式(17-4)中给出的组平均相似度定义与将所有文档 $\vec{d}_m$ 和簇质心向量 $\vec{\mu}$ 的平均相似度作为簇质量的直观定义相等价。

自相似度永远等于 1.0，这也是采用长度归一化向量进行相似度计算所能得到的最大值。对于一个大小为 i 的簇来说，公式（17-4）中的自相似度比例是 $i/i^2=1/i$。这样，小簇所得到的自相似比率更大，于是在计算当中也就给了小簇更多好处，这显然是不公平的。对于两篇文档相似度为 s 的两篇文档 d_1、d_2，我们有 $\text{SIM-GA}'(d_1,d_2)=(1+s)/2$。而此时，$\text{SIM-GA}(d_1,d_2)=s\leq(1+s)/2$。$\text{SIM-GA}(d_1,d_2)$和单连接、全连接及质心聚类方法中的相似度值都是一样的。由于我们不希望因大簇自相似度比率较低而对其进行更大的惩罚，另外，我们也希望在所有四种 HAC 算法中，文档相似度的计算始终保持一致，所以，我们优先采用公式（17-3）中的组平均相似度定义。

? 习题 **17-2**　对于图 17-6 和 17-7 中的点使用组平均聚类方法，将这些点映射到三维空间下单位球体的表面来获得长度归一化向量。采用组平均方法的聚类结果是否和单连接及多连接方法的聚类结果不同？

17.4　质心聚类

在质心聚类中，将通过两个簇的质心相似度来定义这两个簇的相似度：

$$\begin{aligned}\text{SIM-CENT}(\omega_i,\omega_j)&=\vec{\mu}(\omega_i)\cdot\vec{\mu}(\omega_j)\qquad(17\text{-}5)\\&=\left(\frac{1}{N_i}\sum_{d_m\in\omega_i}\vec{d}_m\right)\cdot\left(\frac{1}{N_j}\sum_{d_n\in\omega_j}\vec{d}_n\right)\end{aligned}$$

$$= \frac{1}{N_i N_j} \sum_{\vec{d}_m \in \omega_i} \sum_{\vec{d}_n \in \omega_j} \vec{d}_m \cdot \vec{d}_n \text{。} \qquad (17\text{-}6)$$

公式（17-5）就是质心相似度。公式（17-6）则表明质心相似度等价于不同簇文档之间的平均相似度。因此，GAAC 和质心聚类的区别在于，GAAC 在计算平均相似度时考虑了所有文档之间的相似度（参见图 17-3（d）），而质心聚类中仅仅考虑来自同簇的文档之间的相似度（参考图 17-3（c））。

图 17-11 个给出了质心聚类的前三步。前两次迭代中，由于$<d_5,d_6>$和$<d_1,d_2>$具有最高的质心相似度，所以迭代后形成质心为 μ_1 的簇$\{d_5,d_6\}$及质心为 μ_2 的簇$\{d_1,d_2\}$。在第三次迭代中，最高的质心相似度在 μ_1 和 d_4 之间，因此产生以 μ_3 为质心的簇$\{d_4,d_5,d_6\}$。同 GAAC 一样，质心聚类方法也不具有最佳合并持续性，因此其时间复杂度为 $\Theta(N^2 \log N)$（参考习题 17-6）。

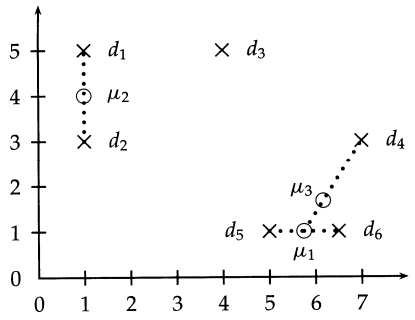


图 17-11 质心聚类中的三次迭代过程。每次迭代合并质心距离最近的两个簇

与其他三种 HAC 算法相比，质心聚类方法不是单调的，可能会发生相似度的颠倒现象。也就是说聚类过程中相似度值有可能会下降。图 17-12 给出了一个相似度颠倒的例子，其中相似度的定义为距离的倒数。第一次合并中，d_1 和 d_2 的相似度为$-(4-\varepsilon)$。第二次合并中，d_1 和 d_2 的质心（用小圆圈表示）和 d_3 的相似度约为$-\cos(\pi/6)\times 4 = -\sqrt{3}/2\times 4 \approx -3.46 > -(4-\varepsilon)$。这样我们就说发生了相似度颠倒现象，也就是说在聚类的前后两步中相似度有所增加。在单调的 HAC 算法中，相似度会随着迭代次数不断下降。

HAC 步骤中相似度的增加与小簇比大簇一致性更强的假设相矛盾。在聚类树状图中，一次相似度颠倒也就意味着某次合并对应的水平直线反而低于前一步的合并直线。而在图 17-1 和图 17-5 中，由于单连接和全连接是单调聚类算法，所以所有的合并直线都会高于先前的合并直线。

尽管质心聚类具有非单调性，但是在概念上来说，由于其相似度计算比 GAAC 采用的所有文档平均相似度的方法更简单，所以人们往往选择使用质心聚类。图 17-11 已经足够使人理解质心聚类。但是，对于 GAAC 而言，没有这样简单的示意图可以解释其原理。

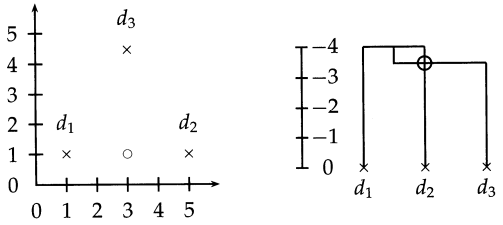


图 17-12　质心聚类的非单调性。文档 d_1（1+ε，1）、文档 d_2（5，1）和文档 d_3（3, 1+2$\sqrt{3}$）之间几乎等距。其中 d_1 和 d_2 的距离略小于其他文档之间的距离，这三篇文档进行层次聚类时会发生相似度反转现象，在聚类树状图中表现为两条直线相交（交点用圆圈表示）

? 习题 **17-3**　对于固定的 N 篇文档，在单连接和全链接聚类中最多有 N^2 个不同的相似度。那么对于 GAAC 和质心聚类算法，不同的簇相似度数目是多少？

17.5　层次凝聚式聚类的最优性

为了精确阐述层次聚类的最优条件，首先将聚类结果 $\Omega=\{\omega_1,\ldots,\ \omega_K\}$ 的结合相似度（combination similarity）COMB-SIM 定义为这 K 个簇中的最小结合相似度：

$$\text{COMB-SIM}(\{\omega_1,...,\omega_K\}) = \min_k \text{COMB-SIM}(\omega_k)。$$

我们在前面提到过，由 ω_1 和 ω_2 合并成的簇 ω，其结合相似度为 ω_1 和 ω_2 的相似度。

然后，如果对于任意具有 k 个簇（$k\leq K$）的聚类结果 Ω’，其结合相似度都低于 Ω，那么我们就称 Ω 是最优的，即：

$$|\Omega'|\ \mathbb{G}\ |\Omega|\ \Rightarrow \text{COMB-SIM}(\Omega')\ \mathbb{G}\ \text{COMB-SIM}(\Omega)。$$

图 17-12 表明质心聚类的结果不是最优的。聚类结果$\{\{d_1, d_2\}, \{d_3\}\}$（$K=2$）的结合相似度为$-(4-\varepsilon)$，而聚类结果$\{\{d_1, d_2, d_3\}\}$（$K=1$）的结合相似度为$-3.46$。由于后者簇数目更小，并且它的结合相似度低于前者的结合相似度，所以$\{\{d_1, d_2\}, \{d_3\}\}$不是最优的。也可以这么说，由于会发生相似度颠倒现象，所以质心聚类的结果并不是最优的。

如果上面的最优化定义仅仅只能应用于具有合并历史的聚类结果，那么它的使用就很有限。然而，我们能够证明，在无历史情况下，可以直接基于簇本身来得到三种非反转聚类算法的结合相似度。这些情况下结合相似度的直接定义如下。

单连接算法：簇 ω 的结合相似度为所有可能的簇二分结果中的最小相似度，其中簇二分的相似度为二分后两部分中文档相似度的最大值，即

$$\text{COMB-SIM}(\omega) = \min_{\{\omega':\omega'\subset\omega\}} \max_{d_i\in\omega'} \max_{d_j\in\omega-\omega'} \text{SIM}(d_i,d_j)。$$

其中，每个〈ω', ω-ω'〉都是 ω 的一个可能的二分结果。

全连接算法：簇 ω 的结合相似度是 ω 中任意两点之间相似度的最小值，即 $\min_{d_i\in\omega} \min_{d_j\in\omega}$

$\mathrm{SIM}(d_i,d_j)$。

GAAC：簇 ω 的结合相似度是 ω 中所有两点间相似度的平均值（不包含自相似度），即公式（17-3）。

如果使用上面的三种结合相似度定义，那么最优性就是一系列簇集合本身的性质，而不是簇集合产生过程的性质。

以下我们将基于簇数目 K 采用数学归纳法来证明单连接聚类的最优性。我们将给出任意两篇文档之间相似度都不等时的证明，当然很容易将该证明推广到具有等相似度文档的情况。

数学归纳的第一步，当 $K=N$ 时的聚类结果的结合相似度为 1.0，即取到最大值。

归纳假设为，对包含 K 个簇的单连接聚类结果 Ω_K，如果对任意其他包含 K 个簇的聚类结果 Ω_K'，都有 COMB-SIM (Ω_K) > COMB-SIM(Ω_K')，那么说 Ω_K 是最优的。

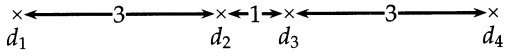
通过将 Ω_K 中的两个最相似簇合并我们得到 Ω_{K-1}，假定 Ω_{K-1} 不是最优的，那么也就存在另一个聚类序列 Ω_K'、Ω'_{K-1}，使得 Ω'_{K-1} 最优。也就是说，此时 COMB-SIM (Ω'_{K-1}) > COMB-SIM (Ω_{K-1})。以下分为两种情况。

情况 1：通过 s = COMB-SIM (Ω'_{K-1})连接的两篇文档在 Ω_K 中属于同一簇。这只有在 Ω_K 的产生序列当中发生一个比 s 小的合并才有可能。也就是说 s > COMB-SIM (Ω_K)，因此，COMB-SIM (Ω'_{K-1}) = s > COMB-SIM (Ω_K) > COMB-SIM (Ω'_K) > COMB-SIM (Ω'_{K-1})，这出现了矛盾！

情况 2：通过 s= COMB-SIM (Ω'_{K-1})连接的两篇文档在 Ω_K 中属于不同的簇。但是由于 s= COMB-SIM (Ω'_{K-1}) > COMB-SIM(Ω_{K-1})，因此，根据单连接算法的合并规则，在处理 Ω_K 时又应该合并这两个簇。

因此，综上所述，Ω_{K-1} 是最优的。

相比于上述单连接算法，全连接算法和 GAAC 都不是最优的，具体示例为：



全连接算法和 GAAC 首先将距离为 1 的 d_2 和 d_3 这两个点合并，因此不可能得到两个簇的聚类结果{{d_1, d_2}, {d_3, d_4}}，而按照全连接算法和 GAAC 的最优准则，这个结果却是最优的。

然而，如果期望聚类结果簇近似球状时，全连接聚类和 GAAC 的合并准则要优于单连接算法。而在很多应用中，我们都希望得到球形的簇。因此，尽管乍看上去基于最优性原则而言，单连接算法更可取，但是在很多聚类应用中，这种最优性可能会建立在错误的准则上。

表 17-1 概括了本章介绍的四种 HAC 算法的性质。由于在应用中，GAAC 通常产生的聚类结果性质最优，所以在文档聚类时我们推荐 GAAC 算法。它不会受链化、异常点敏感及相似度颠倒的影响。

当然，上述推荐也有两个例外情况：第一，非向量表示情况下，GAAC 不太合适，此时通常采用全连接方法来处理；第二，很多应用中，聚类的目的并不是为文档集建立一个层次化、穷尽式的划分体系。比如，首报道检测（first story detection）或被称为新信息检测（novelty

detection）指的是在新闻报道流当中检测出某个事件的第一次出现。该任务的一个做法是从某个短时间内发布的新闻报道中找到一个紧密的簇，这个簇与所有以前的文档不相似。比如，美国 2001 年 9 月 11 日发生“9-11”事件后的有关新闻报道就会构成这样一个簇。由于该任务中涉及的不是文档集的全局结构，而是向量空间中的一个局部的结构，在上例中这一点相当重要，所以单连接算法的多个版本都能很好地处理这种任务。

表17-1　HAC算法的对比

方　法	结合相似度	时间复杂度	是否最优	注　释
单连接法	簇间文档的最大相似度	$\Theta(N^2)$	是	链效应
全连接法	簇间文档的最小相似度	$\Theta(N^2\log N)$	否	对异常点敏感
组平均法	所有文档相似度的平均值	$\Theta(N^2\log N)$	否	大部分应用的最佳选择
质心法	所有簇间相似度的平均值	$\Theta(N^2\log N)$	否	可能存在相似度反转

类似地，我们将介绍用于 Web 重复检测的一种方法，其中并查算法（union-find algorithm）使用了单连接聚类方法。同样，判断一组文档是否互相重复并不受那些离它们较远的文档所影响，因此，对于重复检测来说单连接聚类算法是一个很好的选择。

? 习题 **17-4**　证明结合相似度的两种定义方法是等价的，一种是基于过程的定义（参见 17-1 节），而另一种是静态的定义（参见 17.5 节）。

17.6　分裂式聚类

到现在为止我们仅仅介绍了凝聚式聚类，然而，簇层次结构也可以自顶向下生成。这种层次聚类方法称为自顶向下的聚类（top-down clustering）或分裂式聚类（divisive clustering）。一开始在顶层将所有文档组成一个簇，然后利用某个扁平聚类算法分割该簇簇。上述过程反复迭代直至每篇文档都变成一个单独的簇为止。

在概念上，自顶向下的聚类上比自底向上的聚类要复杂，我们还需要另外一个扁平聚类算法作为“子过程”来调用。当不需要产生包含到文档叶节点各种路径的完整层次结构时，自顶向下的方法效率更高。对于固定数目的某个顶层结构而言，可以利用诸如 K-均值的高效扁平算法，

自顶向下的聚类方法时间复杂度与文档和簇的数目成线性关系。因此，它们远比时间复杂度至少为平方级的 HAC 算法快得多。

有证据表明，在某些场景下，分裂式聚类算法能够生成比自底向上的方法更精确的层次结构。关于这一点请参考 17.9 节中有关二分 K-均值算法（bisecting K-means）的文献。自底向上的方法并不在一开始就考虑全局分布的信息，而是基于局部模式来做聚类决策。在聚类过程中，

早期的决策也不可能被重新修正。而自顶向下的方法在进行高层分割决策时能够利用完整的全局分布信息。

17.7 簇标签生成

在扁平聚类和层次聚类的很多应用特别是分析任务和用户界面(参考表 16-1 中的应用)中，人们需要和簇进行交互。这种情况下，必须要对簇生成标签，以便让用户对簇的相关内容有所了解。

差别式簇标签生成（differential cluster labeling）方法通过比较某个簇和其他簇中的词项分布情况来对簇进行标识。13.5 节所介绍的特征选择方法均可以用于差别式簇标签生成[①]。尤其是， MI（参见 13.5.1 节，另一种等价的称呼为IG（Information Gain，信息增益））以及χ^2等方法都可以找出能够刻画本簇与其他簇差异的簇标签。

采用差别式方法同时又对罕见词项进行惩罚，就可以得到最好的标签结果。这是因为罕见词项在代表某个簇的整体性时并不是必要的。

下面我们将针对 *K*-均值算法介绍 3 种标签生成方法（参见表 17-2）。该例子中，MI 和 χ^2 得到的结果几乎没有差异。因此，我们只列出了前者的结果。

表17-2 自动生成的簇标签。这是对Reuters-RCV1语料的前10 000篇文档进行*K*-均值聚类得到的10个簇中的3个簇（第4、9和第10个簇）的标签生成情况。最后三列分别是采用不同方法得到的簇摘要文本：质心中权重最高的词项（质心法）、互信息最高的若干词项及最接近簇质心的文档标题。仅仅通过前两种方法之一得到的词项用黑体表示

		标签生成方法		
	文档数目	质　心	互信息	标　题
4	622	oil plant mexico production crude **power 000 refinery gas** bpd	plant oil production **barrels** crude bpd mexico **dolly capacity petroleum**	MEXICO: Hurricane Dolly heads for Mexico coast
9	1017	police security **russian** people military peace killed told **grozny court**	police killed military security peace told **troops forces rebels** people	RUSSIA: Russia's Lebed meets rebel chief in Chechnya
10	1259	00 000 tonnes traders futures wheat prices **cents september** tonne	**delivery** traders futures tonne tonnes **desk** wheat prices 000 00	USA: Export Business - Grain/oilseeds complex

基于簇内信息的标签生成（cluster-internal labeling）方法只根据簇本身计算出标签。一种做法是用最接近簇质心的文档标题作为簇标签。标题比一系列词项的可读性要强，并且完整的标题也可以包含上下文信息 ,而这是利用 MI 选择得到的前 10 个词项所不能包含的。在 Web 下，由于指向某个网页的锚文本上往往是该网页的一个简介，所以它也可以起到类似标题的作用。

表 17-2 中，簇 9 的标题表明其中很多文档与车臣冲突（Chechnya conflict）有关，但是基

① 13.5 节讨论的高频词选择并不是一种差别式特征选择技术，但是它也可以用于簇标签生成。

于 MI 选出的词项却难以表达这个事实。然而，单篇文档不可能代表簇中的所有文档。一个明显的例子就是簇 4，它选出的标题就具有误导性。这个簇的主题应该与石油相关，但是有关飓风多利（hurricane Dolly）的文章只是因为其影响了石油价格才被选入簇中。

我们也可以选择簇质心中一些具有较高权重的词项作为簇标签。即使在差别式方法中它们不会被选出用作区分词项，但是这些高权重词项（或者短语，特别是名词短语更佳）往往比一些标题更有代表意义。当然，用户理解一系列短语所花费的时间会比理解标题更多。

基于簇内信息的标签生成效率很高，但是它们不能将全局高频词项和簇内高频词项区分开来。诸如 year 或者 Tuesday 之类的词项可能会在簇中高频出现，但是它们对于理解某个诸如石油主题的簇却毫无帮助。

在表 17-2 中，质心方法和 MI 方法都选出了一些无信息量的词项（比如质心方法中选出的 000、court、cents 和 september，MI 方法选出的 forces 和 desk），前者选出的数目还要更多一些。但是这两种方法选出的大部分词项都是很好的描述特征，我们通过浏览这些词项可以很好地了解文档的内容。

对于层次聚类来说，簇标签生成还会遇到一个额外的复杂问题。我们不仅要将树中内部节点和它的兄弟节点区分开，而且要将它和其父子节点区分开。从定义上说，子节点的文档也属于其父节点，因此我们不能采用原始的差别式方法来区分父子节点。当然，利用将全局频率和当前簇中频率相结合的复杂准则，就可以确定某个词项对子节点还是父节点更具代表性（参见 17.9 节）。

17.8　实施中的注意事项

很多需要大量内积计算的问题可以通过倒排索引来提高效率，对于 HAC 聚类来说也是如此。如果很多文档之间公共词项很少或者采用了较大规模的停用词表，那么相似度计算中就存在很多零值，这时采用倒排索引就会节省很多计算方面的开销。

在低维情况下，可以采用更积极的优化方法来减少大部分不必要的相似度计算（参考习题 17-10）。但是，在高维空间下这种方法并不存在。在 kNN 分类中我们也遇到过这个问题（参见 14.7 节）。

当在高维空间下利用 GAAC 对大型文档集进行聚类时，那么要注意避免密集（即非稀疏的）的质心向量。对于密集型质心向量来说，聚类的时间复杂度可能会达到 $\Theta(MN^2\log N)$，其中 M 是整个词汇表的大小。而对于全连接方法来说，时间复杂度为 $\Theta(M_{\text{ave}}N^2\log N)$，其中 M_{ave} 是一篇文档的平均词汇量。因此，对于大型词汇表来说，全连接算法比未优化的 GAAC 的实现效率要高得多。关于这个问题，我们在第 16 章 K-均值聚类环境中讨论过，当时也给出了两个解决建议：对中心向量进行约简（仅保留高权重词项）和采用稀疏向量而不是稠密向量来表示簇。这些优化策略同样可以用于 GAAC 和质心聚类方法。

即使采用这些优化策略，HAC 算法的复杂度仍然是 $\Theta(N^2)$或 $\Theta(N^2\log N)$，因此对于 1 000 000 篇或者更多文档组成的大规模文档集而言，它们的实现也是不可能的。对于这么大的文档集，HAC 只能同某个扁平聚类方法（如 *K*-均值算法）组合在一起使用。前面我们提到 *K*-均值算法需要一系列种子进行初始化（参见 P332 图 16-5），如果种子选择不好，那么聚类结果质量也会很差。我们可以利用 HAC 算法来计算得到高质量的种子，如果 HAC 算法应用在一个规模为 $\sqrt{N}$ 的文档子集上，那么 *K*-均值连同 HAC 种子生成的总时间就是 $\Theta(N)$。这是因为一个平方级算法应用在规模为 $\sqrt{N}$ 的样本上的总时间复杂度为 $\Theta(N)$。对于时间复杂度为 $\Theta(N^2\log N)$的算法也可以通过适当调整来保证实现上的线性时间复杂度。这种算法称为 Buckshot 算法（Buckshot algorithm），它结合了 HAC 的确定性、高可靠性以及 *K*-均值算法的高效性。

? 习题 **17-5** 单连接聚类算法也可以通过图的最小生成树（minimum spanning tree）算法来计算。最小生成树以最小的代价连接图中的顶点，其中代价被定义为图中各边上的权重之和。在聚类中，边上的代价权重就是两篇文档的距离。若$\Delta_{k-1}>\Delta_k>\ldots>\Delta_1$是最小生成树中各边上的权重，请证明这些边分别对应单连接聚类当中的 *k*–1 次合并。

习题 **17-6** 证明单连接聚类具有最佳合并持续性，而 GAAC 和质心聚类则不具最佳合并持续性。

习题 **17-7**

a. 考虑在一个两种语言组成的文档集上进行 2-均值聚类，你预期的结果是什么？

b. 当使用 HAC 算法时，预期的结果是否仍然一样？

习题 **17-8** 下载 Reuters-21578 语料。仅保留类别 crude、interest 及 grain 中的文档，去掉那些同时属于这三个类别中的多个类别的文档。计算采用 (i) 单连接方法；(ii) 全连接方法；(iii) GAAC；(iv) 及质心方法的文档聚类结果；(v) 对于聚类树状图，在从顶层往下的第二枝上截断来得到 K = 3 个簇。计算四种聚类方法所得结果的兰德指数并指出最好的聚类结果。

习题 **17-9** 假定运行 HAC 算法找到 *K*=7 的某个聚类结果在某个预先给定的质量指标上表现最好，那么能否说我们找到了 *K*=7 的所有聚类结果中的最高值？

习题 **17-10** 考虑对如下的直线上的 *N* 个点进行单连接聚类。

试证明我们总共只需要计算 *N* 次相似度。另外，请给出对直线上的点进行单连接聚类的总时间复杂度。

习题 **17-11** 证明单连接、全连接、组平均聚类方法满足 17.1 节所定义的单调性。

习题 **17-12** 对于 *N* 个点，存在≤ N^K 种不同的最后聚成 *K* 个簇的聚类结果（参考 16.2 节）。那么，*N* 篇文档的不同层次聚类结果有多少种？对于给定的 *K* 和 *N* 来说，是否存在更多的扁平聚类或层次聚类结果？

17.9 参考文献及补充读物

一个出色地介绍了聚类的一般性综述请参见（Jain 等人 1999）。早期介绍具体 HAC 算法的

文献包括(King 1967)（单连接方法）、（Sneath 和 Sokal 1973)（全连接方法和 GAAC）和（Lance 和 Williams 1967)（讨论了层次聚类算法的很多变形）等。图 17-9 的单连接算法与 Kruskal 的最小生成树算法类似。Kruskal 最小生成树算法的一个基于图论的正确性证明(类似于 17.5 节证明）由 Cormen 等人（1990, 定理 23.1）给出。有关最小生成树和单连接聚类算法的关联可以参考习题 17-5。

尽管近来有实验得到了相反的结论（Zhao 和 Karypis 2002），但是很多人仍然认为层次聚类算法能够比扁平聚类算法产生更好的聚类结果(Jain 和 Dubes(1988, p.140）;Cutting 等人(1992）; Larsen 和 Aone（1999））。尽管在平均表现上还没有一致公认的结论，但是毫无疑问，由于 EM 及 K-均值往往收敛于低质量的局部最优结果，所以其聚类结果的变数很大。而这里我们介绍的 HAC 算法是确定性的、结果可以预期的算法。

由于文档相似度计算的复杂度比简单的比较高出一个数量级，所以在计算出 $N \times N$ 的相似度矩阵之后，也就基本完成了合并步骤中的主要操作，因此有些文献(Day 和 Edelsbrunner 1984; Voorhees 1985b; Murtagh 1983）给出的全连接、组平均和质心聚类方法的时间复杂度是 $\Theta(N^2)$。

这里介绍的质心算法来自 Voorhees（1985b）的工作，Voorhees 建议在检索应用中优先采用全连接和质心聚类方法而不是单连接方法。Buckshot 算法最早发表在 Cutting 等人(1993）。Allan 等人（1998）将单连接聚类方法应用于首报道检测。

这里没有介绍的一个重要的 HAC 技术是 Ward 方法(Ward method Jr. 1963; El-Hamdouchi 和 Willett 1986），也被称为最小方差聚类（minimum variance clustering）。该算法每一步都选择具有最小 RSS 值的合并(参见第 16 章)进行操作。Ward 方法的合并准则（一个到质心的多个距离的函数）与 GAAC（一个到质心的多个相似度的函数）密切相关。

尽管对聚类结果的展示非常有用，簇标签自动生成的工作却相对较少。Popescul 和 Ungar（2000）通过词项的 χ^2 及文档集频率的组合获得了较好的结果。Glover 等人 (2002b) 使用信息增益来标识 Web 网页所生成的簇。Stein 和 zu Eissen 提出了基于本体的方法（Stein 和 zu Eissen 2004）。Glover 等人（2002a）和 Treeratpituk 和 Callan（2006）对层次结构下更复杂的簇标识问题（需要区分更通用的父节点和更具体的子节点）进行了处理。有些聚类算法试图首先找到标签集合然后在标签周围建立簇（簇之间往往存在重叠），这样就可以避免标签独立生成的问题（Zamir 和 Etzioni 1999; Käki 2005; Osiński 和 Weiss 2005）。据我们所知，目前还没有一项完整的工作可以对这种基于标签的聚类算法和本章及上一章的聚类算法的结果质量进行比较。理论上说，多文档摘要（McKeown 和 Radev 1995）也可以用于簇标签生成，但是多文档摘要往往比所需要的簇标签要长得多（参考 8.7 节）。将簇以用户能理解的方式进行展示属于人机交互问题，我们推荐通过阅读（Baeza-Yates 和 Ribeiro-Neto 1999，第 10 章）来了解 IR 中用户界面的相关问题。

一个高效的分裂式聚类算法的例子是二分 K-均值方法（Steinbach 等人 2000）。谱聚类（*spectral clustering*）算法（Kannan 等人 2000; Dhillon 2001; Zha 等人 2001; Ng 等人 2001a），包括 PDDP（Principal Direction Divisive Partitioning，主方向分裂式分割，其二分决策基于第 18

章介绍的 SVD 分解）方法（Boley 1998; Savaresi 和 Boley 2004）的计算开销高于二分 *K*-均值方法，但是其优点是具有确定性。

与 EM 及 *K*-均值算法不同的是，大部分层次聚类算法并没有概率上的解释。当然，基于模型的层次聚类方法是个例外（Vaithyanathan 和 Dom 2000; Kamvar 等人 2002; Castro 等人 2004）。

16.3 节中所介绍的评价方法同样也适用于层次聚类。一些用于层次聚类的特殊评价指标在 Fowlkes 和 Mallows（1983）、Larsen 和 Aone（1999）及 Sahoo 等人（2006）中有所讨论。

R 开发环境（R Development Core Team 2005）为层次聚类提供了良好的支撑。R 函数 hclust 实现单连接、全连接、组平均、质心及 Ward 方法。另一个可供选择的是 median 聚类函数，其中每个簇用它的中心点（medoid）来代表（参见第 16 章中提到的 *K*-中心点聚类算法）。软件包 CLUTO（http://glaros.dtc.umn.edu/gkhome/views/cluto）能够为高维空间下的向量聚类提供支持。

第 18 章

矩阵分解及隐性语义索引

在第 6 章中我们介绍了词项-文档矩阵的概念，即由 M 个词项和 N 篇文档组成的一个 $M \times N$ 的权重矩阵 C，矩阵的每行代表一个词项，每列代表一篇文档。即使对于一个中等规模的文档集来说，词项–文档矩阵 C 可能都会有上万的行和列。在 18.1.1 节中，我们首先给出了线性代数中的一类所谓矩阵分解（matrix decomposition）的运算。18.2 节中，我们将使用矩阵分解的某个具体形式来建立词项–文档矩阵的低秩（low-rank）逼近矩阵。18.3 节考察了该低秩逼近矩阵在索引和检索文档时的应用，这也就是人们常常提到的 LSI（*Latent Semantic Indexing*，隐性语义索引）技术。尽管 LSI 在 IR 目前的评分和排名当中并不是一个非常重要的技术，但是它在某些领域的文档聚类当中仍然充满生命力（参见 16.6 节）。如何理解并发挥 LSI 的潜力仍然是一个活跃的研究领域。

不需要复习基本线性代数知识的读者可以直接跳过 18.1 节，当然我们还是建议阅读一下例子 18-1，因为它强调了后面要用到的特征值的一些重要性质。

18.1 线性代数基础

本节当中我们简单地回顾一些必要的线性代数背景知识。令 C 为一个 $M \times N$ 的词项-文档矩阵，其中的每个元素都是非负实数。矩阵的秩（rank）是线性无关的行（或列）的数目，因此有 $rank(C) \leq \min\{M,N\}$。一个非对角线上元素均为零的 $r \times r$ 方阵被称为对角阵（diagonal matrix），它的秩等于其对角线上非零元素的个数。如果上述对角阵上的 r 个元素都是 1，则称之为 r 维单位矩阵（identity matrix），记为 I_r。

对于 $M \times M$ 的方阵 C 及非零向量 $\vec{x}$，有

$$C\vec{x} = \lambda\vec{x}。 \qquad (18\text{-}1)$$

满足上式的 λ 被称为矩阵 C 的特征值（eigenvalues）。对于特征值 λ，满足等式（18-1）的 M 维非零向量 $\vec{x}$ 称为其右特征向量（*right eigenvector*）。对应最大特征值的特征向量被称为主特征向量（principal eigenvector）。同样，矩阵 C 的左特征向量（left eigenvectors）是满足下列等式的 M 维向量 y：

$$\vec{y}^T C = \lambda\vec{y}^T。 \qquad (18\text{-}2)$$

C 的非零特征值的个数最多是 rank(C)。

等式（18-1）可以改写成 $(C - \lambda I_M)\vec{x} = 0$，这个等式称为特征方程（characteristic equation），

可以通过求解这个方程来得到矩阵的特征值。因此，C 的特征值也就是方程 $|(C-\lambda I_M)|=0$ 的解，其中 $|S|$ 表示的是方阵 S 的行列式（determinant）。$|(C-\lambda I_M)|=0$ 是一个以 λ 为变量的 M 阶多项式方程，因此它最多有 M 个根，这些根也就是矩阵 C 的特征值。即使 C 中所有元素都是实数，那么这些特征值通常也有可能是复数。

下面我们将进一步考察特征值和特征向量的更多性质，从而为 18.2 节的 SVD（Singular Value Decomposition，奇异值分解）建立起核心思想。首先，我们考察矩阵-向量乘法和特征值之间的关系。

例 **18-1**　考虑矩阵

$$S=\begin{pmatrix}30 & 0 & 0\\ 0 & 20 & 0\\ 0 & 0 & 1\end{pmatrix}。$$

很明显，矩阵的秩是 3，并且具有 3 个非零特征值 λ_1=30、λ_2=20 及 λ_3=1，它们对应的特征向量分别是：

$$\vec{x}_1=\begin{pmatrix}1\\0\\0\end{pmatrix},\ \vec{x}_2=\begin{pmatrix}0\\1\\0\end{pmatrix},\ \vec{x}_3=\begin{pmatrix}0\\0\\1\end{pmatrix}。$$

对每个特征向量而言，它与 s 相乘相当于用单位矩阵的某个倍数去乘以该特征向量，对于不同的特征向量，具体的倍数也有所不同。现在我们考虑任意一个向量，比如 $\vec{v}=\begin{pmatrix}2\\4\\6\end{pmatrix}$，我们总是可以将 $\vec{v}$ 表示成 S 的三个特征向量的线性组合，对于本例有：

$$\vec{v}=\begin{pmatrix}2\\4\\6\end{pmatrix}=2\vec{x}_1+4\vec{x}_2+6\vec{x}_3。$$

假定用 S 乘以 $\vec{v}$，则有：

$$\begin{aligned}S\vec{v}&=S(2\vec{x}_1+4\vec{x}_2+6\vec{x}_3)\\&=2S\vec{x}_1+4S\vec{x}_2+6S\vec{x}_3\\&=2\lambda_1\vec{x}_1+4\lambda_2\vec{x}_2+6\lambda_3\vec{x}_3\\&=60\vec{x}_1+80\vec{x}_2+6\vec{x}_3\end{aligned}\qquad(18\text{-}3)$$

上例表明，即使 $\vec{v}$ 是一个任意的向量，用 S 去乘以它的效果都取决于 S 的特征值及特征向量。另外，从公式（18-3）看一个非常直观的结论就是，相对而言，$S\vec{v}$ 的大小更不受 S 的小特征值影响。上例中，由于 λ_3=1，所以（18-3）式中最右边的加数的影响较小。实际上，如果完

全忽略（18-3）式的最右边对应于 λ_3=1 的特征向量，那么 $S\vec{v}$ 的结果就是 $\begin{pmatrix}60\\80\\0\end{pmatrix}$ 而不是正确结果 $\begin{pmatrix}60\\80\\6\end{pmatrix}$，但是不论采用哪一种指标（比如差向量的长度）来计算，这两个向量都相对比较接近。

这也意味着，对于矩阵–向量的乘积来说，较小的特征值及其特征向量的影响也较小。我们将带着这种直观理解来研究 18.2 节中的矩阵分解和低秩逼近问题时我们也带着这种直观理解。在这之前，我们来考察某些特殊形式的矩阵的特征向量和特征值，这些才是我们特别感兴趣的内容。

对于对称（*symmetric*）矩阵 S，不同特征值所对应的特征向量之间是正交的（*orthogonal*）。另外，如果 S 是实对称矩阵，那么所有特征值也都是实数。

例 **18-2** 考虑如下实对称矩阵：

$$S=\begin{pmatrix}2 & 1\\1 & 2\end{pmatrix}。\qquad (18\text{-}4)$$

将上述矩阵代入特征方程 $|S-\lambda I|=0$ 可以得到二次方程 $(2-\lambda)^2-1=0$，对这个方程求解可以得到两个特征值 3 和 1，它们对应的特征向量 $\begin{pmatrix}1\\1\end{pmatrix}$ 和 $\begin{pmatrix}1\\-1\end{pmatrix}$ 是正交的。

18.1.1 矩阵分解

本节将介绍将方阵分解成多个矩阵因子乘积的方法，并且这几个矩阵因子都可以从方阵的特征向量导出。这个过程我们称之为矩阵分解（matrix decomposition）。18.3 节中将要讨论的文本分析技术依赖于矩阵分解技术，那里的矩阵分解技术与本节所介绍的技术类似。所不同的是，在 18.3 节中我们主要考虑的是非方阵形式的矩阵分解。本节当中的方阵分解非常简单，数学上也充分严谨，它们能够帮助读者理解矩阵分解的过程。18.2 节介绍的更复杂的矩阵分解技术的数学推导过程不在本书范围之内。

下面我们先给出将方阵分解成特殊矩阵乘积的两个定理，第一个是定理 18-1，它给出了实方阵的基本因子分解方法。而第二个，也就是定理 18-2 给出了实对称方阵的分解方法，这是后面奇异值分解定理（定理 18-3）的基础。

定理 **18-1**（矩阵对角化定理）令 S 为 $M\times M$ 的实方阵，并且它有 M 个线性无关的特征向量，那么存在一个特征分解：

$$S=U\Lambda U^{-1}。\qquad (18\text{-}5)$$

其中，U 的每一列都是 S 的特征向量，Λ 是按照特征值从大到小排列的对角阵，即：

$$\begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \cdots & \\ & & & \lambda_M \end{pmatrix}, \quad \lambda_i \text{ H } \lambda_{i+1}。 \tag{18-6}$$

如果特征值都不相同，那么该分解是唯一的。

为了理解定理 18-1，我们将 U 记为 S 的特征向量的一个表示：

$$U = (\vec{u}_1\ \vec{u}_2 ... \vec{u}_M)。 \tag{18-7}$$

于是，我们有：

$$\begin{aligned} SU &= S(\vec{u}_1\ \vec{u}_2 \cdots \vec{u}_M) \\ &= (\lambda_1\vec{u}_1\ \lambda_2\vec{u}_2 \cdots \lambda_M\vec{u}_M) \\ &= (\vec{u}_1\ \vec{u}_2 \cdots \vec{u}_M)\begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \cdots & \\ & & & \lambda_M \end{pmatrix}。 \end{aligned}$$

因此，我们得到 $SU = U\Lambda$ 或者 $S = U\Lambda U^{-1}$。

我们接下来将介绍另一个紧密相关的分解方法，它将对称方阵分解成其特征向量导出的矩阵的乘积。这将为我们后面介绍的文本分析的主要工具—SVD 技术奠定基础。

定理 **18-2**（对称对角化定理）　假定 S 是一个 $M \times M$ 的实对称方阵，并且它有 M 个线性无关的特征向量，那么存在如下一个对称对角化分解：

$$S = Q\Lambda Q^T。 \tag{18-8}$$

其中，Q 的每一列都是 S 的互相正交且归一化（单位长度）的特征向量，Λ 是对角矩阵，其每个对角线上的值都对应 S 的一个特征值。另外，由于 Q 是实矩阵，所以有：$Q^{-1} = Q^T$。

我们将基于该定理来建立词项-文档矩阵的低秩逼近矩阵。

? 习题 **18-1**　如下 3×3 对角矩阵的秩是多少？

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 2 & 1 \end{pmatrix}$$

习题 **18-2**　证明 λ=2 是如下矩阵的特征值，并求出相应的特征向量：

$$C = \begin{pmatrix} 6 & -2 \\ 4 & 0 \end{pmatrix}。$$

习题 **18-3**　计算式（18-4）中 2×2 矩阵的唯一的特征分解结果。

18.2 词项–文档矩阵及 SVD

迄今为止我们介绍的分解都是基于方阵，然而，我们感兴趣的是 $M\times N$ 的词项–文档矩阵 C，如果排除极端罕见的情况，那么我们有 $M\neq N$。另外，C 基本上也不可能是对称矩阵。为此，我们先给出对称对角化分解的一个被称为 SVD 的扩展形式，然后在 18.3 节中说明如何将它用于构建 C 的近似矩阵。SVD 的完整数学推导超出了本书的范围，这里我们按照定理 18-3 的陈述将 SVD 与 18.1.1 节的对称对角化分解关联起来。给定矩阵 C，U 是一个 $M\times M$ 的矩阵，其每一列是矩阵 CC^T 的正交特征向量，而 $N\times N$ 矩阵 V 的每一列都是矩阵 C^TC 的正交特征向量。这里 C^T 是 C 的转置矩阵。

定理 18-3 令 r 是 $M\times N$ 矩阵 C 的秩，那么 C 存在如下形式的 SVD（参见图 18-1）：

$$C=U\Sigma V^T。 \tag{18-9}$$

其中

1．CC^T 的特征值 $\lambda_1, \lambda_2, \cdots, \lambda_r$ 等于 C^TC 的特征值；

2．对于 $1\leq i\leq r$，令 $\sigma_i=\sqrt{\lambda_i}$，并且 $\lambda_i\geq\lambda_{i+1}$。$M\times N$ 的矩阵 Σ 满足 $\Sigma_{ii}=\sigma_i$，其中 $1\leq i\leq r$，而 Σ 中其他元素均为 0。

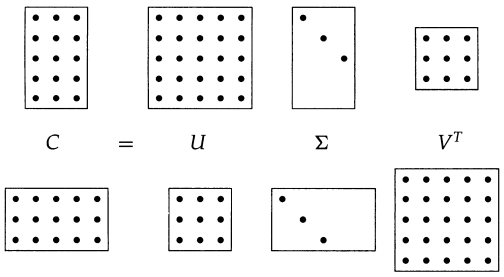


图 18-1 SVD 的示意图。该示意图给出了两种情况：上图中，$M\times N$ 的矩阵 C 满足 $M>N$。而下图中 $M<N$

其中，σ_i 就是矩阵 C 的奇异值（*singular value*）。了解定理 18-3 和 18-2 之间的关系是非常有益的，下面我们将介绍这一点。而对于定理 18-3 我们不加以证明，正如前面提到的那样，这并不在本书范围之内。

将（18-9）式与其转置相乘，有：

$$CC^T=U\Sigma V^TV\Sigma^TU^T=U\Sigma\Sigma^TU^T。 \tag{18-10}$$

在（18-10）式中，左边是一个实对称方阵，而最右边正好是定理 18-2 给出的对称对角化分解形式。那么左边的 CC^T 代表什么呢？它实际上是一个方阵，其每行和每列都对应 M 个词项中的一个。矩阵中的第 i 行、第 j 列的元素实际上是第 i 个词项与第 j 个词项基于文档共现次数的一个重合度计算指标。其精确的数学含义依赖于构建 C 所使用的词项权重方法。假定 C 是第

1 章图 1-1 所示的词项-文档出现矩阵，那么 CC^T 的第 i 行、第 j 列的元素是词项 i 和词项 j 共现的文档数目。

当记录 SVD 分解的数值结果时，由于其他部分都是零，常规做法是将 Σ 表示成一个 $r \times r$ 的对角方阵，所有奇异值排列在对角线上。同样，对应于 Σ 中被去掉的行，U 中的最右 $M-r$ 列也被去掉。对应于 Σ 中被去掉的列，V 中的最右 $N-r$ 列也被去掉。这种 SVD 的书写形式有时被称为简化的 *SVD*（*reduced SVD*）或截断的 *SVD*（*truncated SVD*），这个概念我们还会在习题 18-9 中遇到。下面，所有的例子和习题都将使用这种形式。

例 **18-3**　这里给出一个秩为 2 的 4×2 矩阵的 SVD 例子，奇异值 $\Sigma_{11}=2.236$，$\Sigma_{22}=1$。

$$C=\begin{pmatrix}1&-1\\0&1\\1&0\\-1&1\end{pmatrix}=\begin{pmatrix}-0.632&0.000\\0.316&-0.707\\-0.316&-0.707\\0.632&0.000\end{pmatrix}\begin{pmatrix}2.236&0.000\\0.000&1.000\end{pmatrix}\begin{pmatrix}-0.707&0.707\\-0.707&-0.707\end{pmatrix}。\qquad (18\text{-}11)$$

同 18.1.1 节的矩阵分解定义一样，矩阵的 SVD 分解可以通过很多算法来实现，其中大部分都有可公开使用的软件包，在 18.5 节给出了一些参考文献材料。

习题 **18-4**　令

$$C=\begin{pmatrix}1&1\\0&1\\1&0\end{pmatrix}。\qquad (18\text{-}12)$$

为某个文档集上的词项–文档出现矩阵，计算词项的共现矩阵 CC^T。当 C 是一个词项–文档出现矩阵时，CC^T 对角线上的元素是什么？

习题 **18-5**　验证公式（18-12）所示矩阵的 SVD 结果是：

$$U=\begin{pmatrix}-0.816&0.000\\-0.408&-0.707\\-0.408&0.707\end{pmatrix},\ \Sigma=\begin{pmatrix}1.732&0.000\\0.000&1.000\end{pmatrix},V^T=\begin{pmatrix}-0.707&-0.707\\0.707&-0.707\end{pmatrix}。\qquad (18\text{-}13)$$

并验证定理 18-3 给出的所有性质。

习题 **18-6**　假定 C 是词项–文档出现矩阵，那么 C^TC 的元素的含义是什么？

习题 **18-7**　令

$$C=\begin{pmatrix}0&2&1\\0&3&0\\2&1&0\end{pmatrix}。\qquad (18\text{-}14)$$

上式为一个词项–文档矩阵，其中每个元素都是词项频率，因此词项 1 在文档 2 中出现 2 次，而在文档 3 中出现 1 次。计算 CC^T，并找出两个词项的最高词频都出现在同一文档时所对应的元素。

18.3 低秩逼近

接下来我们介绍一个看似与 IR 无关的矩阵逼近问题，我们可以利用 SVD 对该问题求解，然后将之应用到 IR 领域。

给定 $M \times N$ 的矩阵 C 及正整数 k，我们想寻找一个秩不高于 k 的 $M \times N$ 的矩阵 C_k，使得两个矩阵的差 $X = C - C_k$ 的 F–范数（Frobenius Norm，弗罗宾尼其范数）最小，即下式最小：

$$\| X \|_F = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N} X_{ij}^{\ 2}}。 \tag{18-15}$$

因此，X 的 F–范数度量了 C_k 和 C 之间的差异程度。我们的目标是找到一个矩阵 C_k，会使得这种差异极小化，同时又要限制 C_k 的秩不高于 k。如果 r 是 C 的秩，那么很显然 C_r=C，此时矩阵差值的干一范数为 0。当 k 比 r 小得多时，我们称 C_k 为低秩逼近（low-rank approximation）矩阵。

SVD 可以用于解决矩阵低秩逼近问题，接着我们将其应用到词项-文档矩阵的逼近问题上来。为此，我们要进行如下三步操作：

(1) 给定 C，按照公式（18-9）构造 SVD 分解，因此 $C = U\Sigma V^T$；

(2) 把 Σ 中对角线上 r-k 个最小奇异值置为 0，从而得到 Σ_k；

(3) 计算 $C_k = U\Sigma_k V^T$ 作为 C 的逼近。

由于 Σ_k 最多包含 k 个非零元素，所以 C_k 的秩不高于 k。然后，我们回顾一下例 18-1 的直观性结果，即小特征值对于矩阵乘法的影响也小。因此，将这些小特征值替换成 0 将不会对最后的乘积有实质性影响，也就是说该乘积接近 C。接下来 Eckart 及 Young 给出的定理将会告诉我们，实际上，上述过程产生了一个秩为 k 的矩阵，它的 F–范数误差最小。

定理 18-4

$$\min_{Z|rank(Z)=k} \| C - Z \|_F = \| C - C_k \|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^{\ 2}}。 \tag{18-16}$$

由于奇异值按照降序排列，即$\sigma_1 \geq \sigma_2 \geq \cdots$，因此，我们从上述定理可以知道$C_k$是$C$的秩为$k$的最佳逼近，其中误差的大小（采用$C-C_k$的F–范数来度量）等于 $\sqrt{\sum_{i=k+1}^{r} \sigma_i^{\ 2}}$ ①。所以，k越大，该误差也越小。当k=r时，由于 Σ_r=Σ，所以此时的误差为 0。给定 $r < M, N$的话，有σ_{r+1}=0，因此，C_r=C。

为了深入理解为什么从 Σ 去掉最小的 r–k 个奇异值的过程能够产生低误差的 k–秩逼近，下面我们考察一下 C_k 的形式：

$$C_k = U\Sigma_k V^T \tag{18-17}$$

① 原文为σ_{k+1}，有误。——译者注

$$= U \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \sigma_k & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots \end{pmatrix} V^T \qquad (18\text{-}18)$$

$$= \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^{\,T} \text{ 。} \qquad (18\text{-}19)$$

其中，$\vec{u}_i$ 和 $\vec{v}_i$ 分别是 U 和 V 的第 i 列。因此，$\vec{u}_i \vec{v}_i^{\,T}$ 是一个 1-秩矩阵，于是我们将 C_k 表示成 k 个 1-秩矩阵的加权和，每个矩阵的权重是一个奇异值。由于 $\sigma_1 \geq \sigma_2 \geq \cdots$，所以当 i 增加时，1-秩矩阵 $\vec{u}_i \vec{v}_i^{\,T}$ 的权重也随之减小。

习题 **18-8**　利用（18-13）式所示的 SVD 分解，计算公式（18-12）中矩阵 C 的 1-秩逼近 C_1，并给出该逼近下的 F–范数误差值。

习题 **18-9**　考虑习题 18-8 中计算结果。在图 18-2 中，我们注意到 1-秩逼近情况下 σ_1 是个标量。令 U_1 表示 U 的第一列，V_1 表示 V 的第一列。试证明 C 的 1-秩逼近可以写成 $U_1 \sigma_1 V_1^{\,T} = \sigma_1 U_1 V_1^{\,T}$。

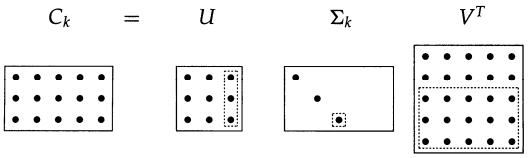


图 18-2　利用 SVD 的低秩逼近示意图。虚线框内给出的是受小奇异值影响的可以去除的矩阵元素

习题 **18-10**　习题 18-9 可以推广到 k-秩逼近情况。令 U'_k 和 V'_k 分别表示保留 U 和 V 前 k 列后得到的矩阵，那么 U'_k 是一个 $M \times k$ 矩阵，而 $V'^{\,T}_k$ 是个 $k \times N$ 矩阵。因此，我们有

$$C_k = U'_k \Sigma'_k V'^{\,T}_{k'} \text{。} \qquad (18\text{-}20)$$

其中 Σ'_k 是 Σ_k 的一个 $k \times k$ 的子方阵，其对角线上的奇异值是 $\sigma_1, \sigma_2, \ldots \sigma_k$。利用（18-20）的最大好处是可以去掉 U 和 V 中很多冗余的零列向量，从而可以明显地减少那些不影响低秩逼近结果的乘法计算。SVD 的这个版本有时也被称为简化的 SVD 或者截断的 SVD，它在计算低秩逼近时更简单。试对例 18-3 中的矩阵 C，求出 Σ_2 和 Σ'_2。

18.4　LSI

下面我们来介绍如何利用SVD分解来找到词项-文档矩阵C的某个低秩逼近，在这个低秩逼近下能够为文档集中的每篇文档产生一个新的表示。同样，查询也可以映射到这个低秩表示空间，从而可以基于新的表示来进行查询和文档的相似度计算。这个过程被称为LSI[①]。

① 这个术语也有人译成潜语义索引/标引、潜在语义索引/标引、潜性语义索引/标引、隐含语义索引/标引等。LSI

首先，我们来介绍引入 LSI 的目的所在。回顾一下在第 6 章中介绍的将查询和文档均表示成向量的向量空间表示方法。这种表示的优点包括：可以将查询和文档转换成同一空间下的向量，可以基于余弦相似度进行评分计算，能够对不同的词项赋予不同的权重，除了文档检索之外还可以推广到诸如聚类和分类等其他领域，等等。但是，向量空间表示方法没有能力处理自然语言中的两个经典问题：一义多词（*synonymy*）和一词多义（*polysemy*）问题。一义多词指的是不同的词（比如 car 和 automobile）具有相同的含义。向量空间表示不能捕捉诸如 car 和 automobile 这类同义词之间的关系，而是将它们分别表示成独立的一维。因此，计算查询 $\vec{q}$（如 car）和文档 $\vec{d}$（同时包含 car 和 automobile）的相似度 $\vec{q}\cdot\vec{d}$ 时，就会低估了用户所期望的相似度。而一词多义指的是某个词项（如 charge）具有多个含义，因此在计算相似度 $\vec{q}\cdot\vec{d}$ 时，就会高估了用户所期望的相似度。一个很自然的问题就是，能否利用词项的共现情况（比如，charge 是和 steed 还是 electron 在某篇文档中共现），来获得词项的隐性语义关联从而减轻这些问题的影响？

即使对一个中等规模的文档集来说，词项-文档矩阵 C 也可能有成千上万个行和列，它的秩数目大概也是这个数量级。在 LSI 中，我们使用 SVD 分解来构造 C 的一个低秩逼近 C_k，其中 k 远小于矩阵 C 原始的秩。在本节后面所引用的一些研究工作当中，实验时 k 的取值往往在几百以内。这样，我们就可以将词项–文档矩阵中每行和每列（分别对应每个词项和每篇文档）映射到一个 k 维空间，为 CC^T 和 C^TC 的 k 个主特征向量（对应 k 个最大的特征值）可以定义该空间。需要注意的是，不管 k 取值如何，矩阵 C_k 仍然是一个 $M\times N$ 的矩阵。

接下来，和原始空间一样，我们利用新的 k-维空间的 LSI 表示来计算向量的相似度。向量 $\vec{q}$ 可以通过下式变换到 LSI 空间：

$$\vec{q}_k = \sum\nolimits_k^{-1} U_k^T \vec{q}\text{。} \qquad (18\text{-}21)$$

现在我们利用第 6 章介绍的余弦相似度来计算查询和文档、文档之间或者查询之间的相似度。我们注意到，（18-21）式中的 $\vec{q}$ 并不一定要求是查询，它可以是词项空间中的一个任意向量。这也意味着，如果我们有某个文档集的 LSI 表示，那么一篇不属于该文档集的新文档就可以通过（18-21）式转换成 LSI 表示。这样就允许将文档增量式地加入到 LSI 表示当中。当然，这种增量式的加入过程并不能体现新加入文档中的词项的共现关系（有时甚至忽略这些文档所包含的新词项）。因此，当文档被不断加入时，LSI 表示的质量会有所降低，最终必须要对 LSI 表示进行重新计算。

C_k 到 C 的逼近性使得我们希望仍然可以保留原有的余弦相似度的相对大小：如果在原始空间中查询和文档相近，那么在新的 k 维空间中它们仍然比较接近。但是这本身并不是十分有趣，特别是当原始的稀疏向量 $\vec{q}$ 转换成低维空间中的密集向量 $\vec{q}_k$ 时，新空间下的计算开销会高于原始空间。

的另外一个名称是（latent semantic analysis，简称 LSA。）——译者注

例 **18-4**　考虑如下词项–文档矩阵 C =

	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
voyage	1	0	0	1	1	0
trip	0	0	0	1	0	1

利用 SVD 分解，可以将其分解生成三个矩阵的乘积。首先，矩阵 U 为

	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
voyage	−0.70	0.35	0.15	−0.58	0.16
trip	−0.26	0.65	−0.41	0.58	−0.09

当在词项–文档矩阵上使用 SVD 分解时，得到的矩阵 U 被称为 *SVD* 词项矩阵（SVD term matrix）。奇异值矩阵Σ =

2.16	0.00	0.00	0.00	0.00
0.00	1.59	0.00	0.00	0.00
0.00	0.00	1.28	0.00	0.00
0.00	0.00	0.00	1.00	0.00
0.00	0.00	0.00	0.00	0.39

最后，我们得到 V^T，在词项–文档矩阵进行 SVD 分解的情况下，它被称为 *SVD* 文档矩阵（SVD document matrix）。

	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

去掉Σ 中除两个最大值之外的奇异值，我们有 Σ_2=

2.16	0.00	0.00	0.00	0.00
0.00	1.59	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
0.00	0.00	0.00	0.00	0.00

基于该矩阵，可以计算 C_2=

	d_1	d_2	d_3	d_4	d_5	d_6
1	−1.62	−0.60	−0.44	−0.97	−0.70	−0.26
2	−0.46	−0.84	−0.30	1.00	0.35	0.65
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00

需要注意的是，该低秩矩阵可能会包含负值，这一点和原始矩阵 C 是不同的。

考察上例中的矩阵 Σ_2 及 C_2 就会发现，每个矩阵的最后三行都是零。这表示，公式（18-18）中的 SVD 分解结果 $U\Sigma V^T$ 只需要保留 Σ_2 和 V^T 的两行即可。于是，将这些矩阵表示成其截断矩阵 Σ'_2 和 $(V')^T$。此时，截断 SVD 分解中的文档矩阵 $(V')^T$ 为

	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41

图 18-3 给出了 $(V')^T$ 中文档向量在二维空间下的示意图，需要指出的是，相对于矩阵 C 而言，C_2 更密集。

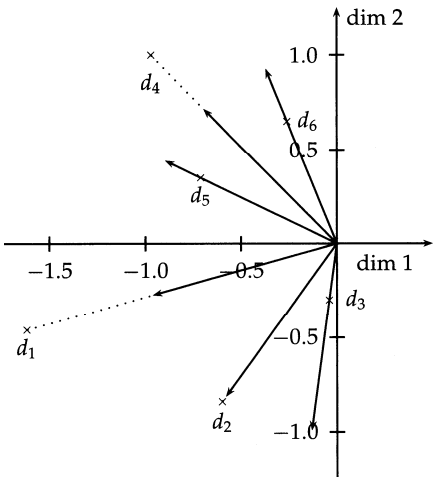


图 18-3　18-4 示例中文档简化成 $(V')^T$ 中 2 维向量后的示意图

一般来说，可以将求 C 的低秩逼近看成是一个约束优化问题（constrained optimization problem）：在 C_k 的秩最多为 k 的条件下，从 C 出发寻找词项和文档的一个表示 C_k，使得 $C–C_k$ 的 F–范数误差值最小。当将词项/文档表示到 k 维空间时，SVD 应该将共现上相似的词项合在一起。这个直觉也意味着，检索的质量不仅不太会受降维的影响，而且实际上有可能会提高。

文献 Dumais（1993）和 Dumais（1995）基于普遍所使用的 Lanczos 算法来计算 SVD 分解，并在 TREC 语料和任务上对 LSI 进行了一系列实验。在实验当时（20 世纪 90 年代早期），数万篇文档上的 LSI 计算在单机上大约需要一整天。这些实验也达到或超过了当时 TREC 参加者的中游水平。在 20%左右的 TREC 主题中，他们的系统得分最高，在平均水平上使用大约 350 维

的 LSI 也比常规的向量空间方法稍高。下面列出了最早从他们工作中得到的结论，而这些结论在后续的其他实验中也得到了验证。

- SVD 的计算开销很大，在撰写本书之际，还没听说过有成功分解超过一百万篇文档的实验。这也是一个阻碍 LSI 推广的主要障碍。一个解决这个障碍的方法是对文档集随机抽样然后基于抽取出的样本子集建立 LSI 表示，剩余的其他文档可以基于公式（18-21）进行转换。
- 如果减低 k 值，那么如预期一样，召回率将会提高。
- 令人奇怪的是，当 k 取几百之内的数目时，某些查询的正确率实际上也会得到提高。这也意味着，对于合适的 k 值，LSI 能部分解决一义多词的问题。
- 当查询和文档的重合度很低时，LSI 的效果最好。

上述实验中也记录了一些 LSI 的失效模式，在这些模式下一些 LSI 不如更传统的索引和相似度计算方法。最值得一提（或许是很明显）的是，LSI 仍然延续了向量空间检索的两个缺点：无法表示否定（比如寻找包含 german 但同时不包含 shepherd 的文档）和无法完成布尔查询条件。

如果将简化后的空间中的每一维都解释成一个簇，而每篇文档在该维上的（相似度得分）值看成是到这个簇的隶属度，那么就可以把 LSI 看成软聚类（soft clustering）的一种。

? 习题 **18-11**　假定有一个文档集合，其中每篇文档可以是英文或者是西班牙文。整个文档集如图 18-4 所示。图 18-5 给出了与图 18-4 相关的英语和西班牙语的术语表。当然，该术语表只用于帮助理解，对检索系统来说是不可见的。

DocID	document text
1	hello
2	open house
3	mi casa
4	hola Profesor
5	hola y bienvenido
6	hello and welcome

图 18-4　习题 18-11 中的文档

Spanish	English
mi	my
casa	house
hola	hello
profesor	professor
y	and
bienvenido	welcome

图 18-5　习题 18-11 中的术语表

(1) 构造该文档集的词项-文档矩阵 C。为简单起见，只使用原始词频而不是归一化的 tf-idf 权重。请务必清晰标出矩阵的每一维。

(2) 写出矩阵 U_2、Σ'_2 及 V_2，并推出 2-秩逼近矩阵 C_2。

(3) 简述矩阵 C^TC 的元素（i,j）代表的意义。

(4) 简述矩阵 $C_2{}^TC_2$ 的元素（i,j）所代表的意义，解释它为什么与 C^TC 中的元素(i,j)不同。

18.5 参考文献及补充读物

Strang（1986）给出了一篇相当出色的综述，该综述包含了SVD在内的矩阵分解。定理 18-4 的工作归功于Eckart 和 Young（1936）。将词项-文档矩阵的低秩逼近与IR进行关联的工作来自 Deerwester 等人（1990），后来的相关结果的综述参见Berry 等人（1995）。Dumais（1993）及 Dumais（1995）介绍了他们在TREC上的实验，其结果表明，至少在某些基准测试上LSI所产生的结果的正确率和召回率会高于常规的向量空间检索方法。www.cs.utk.edu/~berry/lsi++/ 及 http://lsi.argreenhouse.com/lsi/LSIpapers.html 网站给出了有关LSI文献及软件的详细链接。Schütze 和 Silverstein（1997）评估了在高效K-均值聚类（参考 16.4 节）中LSI及质心截断表示的性能。Bast 和 Majumdar（2005）详细讨论了LSI中简化空间的维度k所承担的角色，并考察了k值取不同水平时不同词项对之间结合的情况。Berry 和 Young（1995）及Littman 等人（1998）将LSI 应用到跨语言检索[①]（cross-language information retrieval）中。LSI（更通常的情况下被称为LSA）已经应用到计算机科学的很多其他领域，并涵盖了从内存建模（memory modeling）到计算机视觉的诸多内容。

Hofmann（1999a, 1999b）给出了 LSI 基本技术的一个初始概率扩展方法。一个更令人满意的用于降维的概率隐性变量模型是 LDA（Latent Dirichlet Allocation）模型（Blei 等人 2003），它是个生成模型并能够给训练集外的文档赋予概率。该模型被 Rosen-Zvi 等人（2004）扩展为一个层次聚类算法。Wei 和 Croft（2006）第一次给出了在大规模情况下的 LDA 评价，发现它的性能要明显高于 12.2 节中提到的查询似然模型，但是不如 12.4 节中介绍的相关性模型，然而相关性模型需要对每个查询进行额外的处理，这一点上 LDA 却不需要。Teh 等人（2006）对 LDA 进行了进一步推广，提出了 HDP（Hierarchical Dirichlet processes）过程，它不仅允许在隐性主题的一个无穷混合上抽出一个组（对我们来说，就是一篇文档），而且还允许这些主题在文档之间共享。

①这种 IR 模式下，查询用某种语言描述，而目标文档则由另外一种或多种语言来描述。

第 19 章

Web 搜索基础

在本章之前，我们主要介绍的是在传统的文档集上进行的 IR 工作。而在本章和后续的两章中，我们将主要关注 Web 搜索引擎。19.1 到 19.4 节给出了相关背景和历史介绍，以帮助读者理解为什么纷繁杂乱、变化迅速的 Web（从 IR 的角度上看）和传统的文档集完全不同。19.5 节到 19.6 节分别讨论了 Web 搜索引擎的索引规模估计及 Web 索引中的文档查重问题。这两节的内容也为后两章的学习提供了背景资料。

19.1 背景和历史

Web 在很多方面都是空前的：不仅在规模上史无前例，而且其创建过程中协调机制的缺乏也是空前的；另外，Web 参与者的背景和动机的多样性同样也是空前的。以上的每一个因素都使得 Web 搜索有别于传统的文档搜索，并且，一般来说 Web 搜索要困难得多。

Vannevar Bush 于 20 世纪 40 年代提出了超文本的构想，并于 70 年代首次实现了超文本的技术，这大大早于 Web（Word Wide Web，万维网）形成的时间（20 世纪 90 年代）。Web 的使用已经显示出巨大的增长趋势，可以说它目前已经成为网络用户生活的一部分。而 Web 的使用往往基于一个简单开放的客户端-服务器（client-server）机制就可以实现：(i) 服务器通过某个轻量级的协议（HTTP（Hypertext Transfer Protocol，超文本传输协议））和客户端通讯，该协议是异步的，并且可携带各种内容（开始是文本、图像，后来随着时间的推移出现了更丰富的包含音频、视频在内的媒体格式），这些媒体通过一个简单的 HTML（Hypertext Markup Language，超文本标记语言）标记语言来编码；(ii) 客户端通常是浏览器，即一个具有图形用户环境的应用，并且会忽略掉它不能理解的部分。上述这些看上去并不起眼的特征对 Web 的增长居功至伟，所以有必要对它们做进一步的介绍。

基本的操作流程如下：客户端（如浏览器）发送一个 *http* 请求（http request）给 Web 服务器（*web server*）。浏览器指定某个 URL(Universal Resource Locator，统一资源定位符)，例如 http://www.stanford.edu/home/atoz/contact.html。该 URL 中的字符串 http 指定了用于传输数据的协议。字符串 www.stanford.edu 被称为域（domain），它给出了网页层次结构（往往是 Web 服务器上某个文件系统的层次结构的映像）的根目录位置。而/home/atoz/contact.html 给出的是层次结构的路径，其中 contact.html 包含了位于 www.stanford.edu 的 Web 服务器应答请求而返回的信息。这个文件采用 HTML 语言编码，其中包含链接和内容信息（本例中的内容信息是 Stanford

University 的联系信息）以及在浏览器中显示的格式化规则。因此，这种 http 请求允许我们获得网页的内容，而这些内容在采集和索引文档（参考第 20 章）时非常有用。

第一代浏览器的设计者确保了人们很容易就能阅读某个 URL 对应的原始 HTML 标记文档，这样新用户就无需太多的学习或者经验便可以创建自己的 HTML 内容，甚至可以选择喜欢的网页作为样例直接学习。与此同时，浏览器的第二个特点是会忽略其不能解析的内容，这个特点使得 Web 内容的创建和使用能够被迅速扩散开来。有人可能会担心这样做将导致大量不兼容的 HTML 语言出现，但是事实并非如此。实际上这大大促进了业余网页制作者的热情。他们可以自由地实验并从新建网页中学习，而不必担心一个简单的语法错误会导致整个系统崩溃。在 Web 上发布网页已经不是少数训练有素的编程人员的特权，而是上亿普通网民参与的活动。对于大部分的用户和需求来说，Web 已经迅速成为提供和消费各种信息的重要场所，这些信息包括从疑难杂症到地铁时刻表的任何内容。

如果其他用户不能发现并使用 Web 上发布的大量信息，那么这些信息实际上就毫无价值。有关 Web 信息发现的早期尝试可以归成两类：(i)像 Altavista、Excite 和 Infoseek 一样的基于全文索引的搜索引擎；(ii) 诸如 Yahoo!的 Web 网页分类体系。前者在前台给用户提供了关键词搜索界面，而在后台则采用前面介绍的倒排索引和排序机制。后者可以允许用户沿树形结构的类别体系进行浏览。尽管乍一看这种方法可以非常方便直观地发现 Web 网页，但是它也具有很多缺点：首先，将 Web 网页精确地分类到层次树节点上的大部分工作都是人工编辑完成，那么随着 Web 规模的扩大这种做法很难扩展。也许有人会说，在分类体系中我们只需要那些高质量的 Web 网页，对每一类只需要其中最好的网页即可。然而，仅仅是发现并将这些网页精确、一致地分到分类体系中也需要大量的手工工作。另外，要准确发现 Web 网页并把它们分到类别节点上去，用户必须要了解应该要到哪些子树下去搜索特定的主题，并且他的理解必须要和分类体系的编辑人员一致。当类别体系的规模急剧增长时，这一点也变得相当具有挑战性。Yahoo!的分类体系很早就超过了 1 000 个的节点。由于存在上述挑战，随着时间的推移，尽管涌现了一些利用主题专家收集和标注每类 Web 网页的新做法（如 About.com 和 ODP（Open Directory Project，开放目录项目））, 分类体系的受欢迎程度还是有所下降。

第一代 Web 搜索引擎将前面介绍的传统搜索技术应用到 Web 领域，它们主要关注规模上的挑战性。最早的 Web 搜索引擎必须要处理包含上千万文档的索引，而这个规模比以前所有的公共域 IR 系统的数据规模要高很多个数量级。要完成这种规模下的索引、查询服务和排序任务，需要利用数十台计算机来建立一个具有高可靠性的系统。同样，这种规模在之前的面向消费者的搜索服务中也从未见到过。第一代 Web 搜索引擎针对上述挑战取得了巨大成功，它们能够持续索引很大一部分 Web 页面，并且能在亚秒级时间内完成对查询的应答。然而，由于 Web 内容创建的特殊性（参见 19.2 节）, Web 搜索结果的质量和相关性离期望还有很大的距离。这就亟需研发出新的排序机制和反作弊技术来提高搜索质量。尽管 Web 搜索中仍然需要传统的 IR 技术（比如本书前面介绍的各种技术）, 但是仅仅使用这些技术已经远远不够了。很重要的一个方

面（将在第 21 章讨论）是，传统技术只度量了文档和查询的相关性，但是在进行 Web 搜索时，必须要度量文档的权威度（authoritativeness），即计算权重度时可利用诸如其所在网站之类的信息。

19.2 Web 的特性

导致 Web 爆炸式增长的最根本原因在于无法集中控制的无中心的网页内容发布机制，这也为 Web 搜索引擎索引和检索这些发布的内容带来了巨大挑战。网页作者可能会采用数十种自然语言和数千种专业语言来撰写网页，因此需要很多不同的词干还原工具及其他的语言处理方法。由于 Web 发布现在已经对数千万用户开放，网页在其规模惊人增长的同时也在很多重要方面表现出不同的特点。首先，内容创作不再是少数受过编辑培训的作家的特权，尽管这一点意味着内容创作的极大民主化，但同时也造成网页中存在大量语法和风格上的巨大差异（很多情况下，语法或风格甚至无法辨认）。确实，在某种意义上说，Web 出版完全释放出了在全球范围内进行桌面出版的最好和最糟的一面，通过不同的颜色、字体和结构来装饰的页面迅速涌现。某些网页，包括一些大公司的经过专业制作的主页，几乎全是图像（点击之后可以看到文字内容），这些网页并不包含可索引的文本。

那么网页中的文本内容所表达的主旨又怎样呢？由于网络内容的创作具有极大的民主化，所以这就意味着几乎在任何一个话题上都会出现一些粒度更细的不同观点。这也表示 Web 中包含真理、谎言、矛盾和大量猜测。这就引发了一个问题：我们应该相信哪些 Web 网页？假定我们绕过搜索引擎如何衡量网页的可信度这个问题，那么对于上述疑问的一种简单的解决方法是，直接通过用户来认定哪些发布者是可信的，而哪些发布者不可信。在第 21 章中，我们将考察理解上述问题的方法。需要指出的一点是，可能并不存在统一的、与用户无关的可信度标准，对某个用户可信的网页内容不一定对其他用户可信。在传统的非 Web 出版方式下，这并不是个问题，用户可以自己选择他们认为可信的来源。因此，可能某个读者会觉得《纽约时报》的报道更可靠，而另一个读者却更喜欢《华尔街时报》。但是，当搜索引擎成为用户了解（更不用说选择）大部分内容的唯一可行途径时，上述问题就带来了巨大的挑战性。

Web 到底有多大？这个问题不易回答，换一种问法：“某个搜索引擎中索引的网页数目是多少？”这样问要求答案比较精确，当然，回答这个问题也不是那么容易。到 1995 年底，Altavista 声称它采集并索引了大概三千万个静态网页。所谓静态网页（static web page），指的是那些内容不会因请求不同而不同的网页。按照这个定义，一个教授每周都会手工修改的主页是静态网页，而机场航班状态页面一般是动态的。动态页面（dynamic page）通常是由应用服务器应答数据库的查询需求时产生的，图 19-1 给出了一个动态网页生成的例子。这种页面的一个标志是 URL 中通常包含字符“?”。在 1995 年时，由于大家相信每过几个月静态页面的数目就会翻番，

所以早期的包括 Altavista 在内的 Web 搜索引擎必须要经常增加硬件和带宽来采集和索引网页。

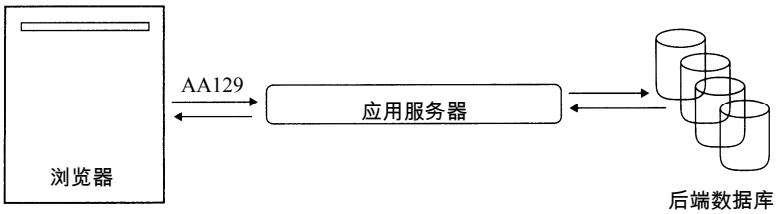


图 19-1　一个动态生成的网页。浏览器发送有关 AA129 次航班的请求给 Web 应用服务器，服务器从后端数据库中获取信息并且生成一个动态网页返回给浏览器

19.2.1　Web 图

我们可以将整个静态 Web 看成是静态 HTML 网页通过超链接互相连接而成的有向图，其中每个网页是图的顶点，而每个超链接则代表一个有向边。

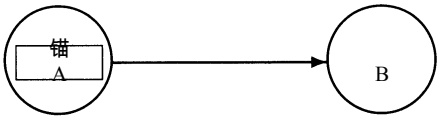


图 19-2　两个顶点通过链接构成的 Web 图

图 19-2 给出了包含两个顶点 A、B 的 Web 图，每个顶点代表一个网页，A 网页上有一个超链接指向 B。我们将所有这样的顶点和有向边集合称为 Web 图。图 19-2 还表明，在 A 网页上的超链接周围还有一些文本，大部分网页链接的实际情况也是如此。这些文本通常被嵌在<a>标签（称为锚）的 href 属性中。 正如有人所怀疑的那样，该有向图可能不是一个强连通（strongly connected）图，也就说，从一个网页出发，沿着超链接前进，有可能永远不会到达另外某个网页。我们将指向某个网页的链接称为入链接（in-link），而从某个网页指出去的链接称为出链接（out-link）。一个网页的入链接数目被称为这个网页的入度（in-degree），在一系列研究中得到的网页的平均入度大概从 8 到 15 左右不等。同样，我们可以定义某个网页的出链接数目为其出度（out-degree）。图 19-3 给出了展示这些概念的一个例子。

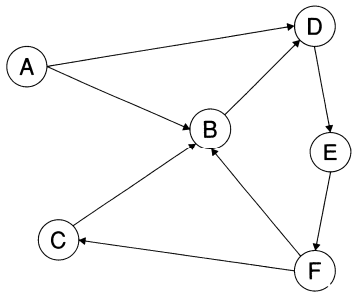


图 19-3　一个小型 Web 图的例子。该例子中共有 6 个网页（分别以 A 到 F 标识），网

页 B 的入度为 3、出度为 1。该图不是强连通图，因为 B 不可能到 A

有充分的证据表明，这些链接并不满足随机分布。一方面，如果每个网页都是随机均匀地选择链接目标时，那么链接到一个网页的链接数目应该满足泊松分布，但是实际中的数目并不满足预想的泊松分布。实际上，有大量研究表明这个分布满足幂分布定律（power law），具有入度为i的网页总数目正比于 $1/i^{\alpha}$，研究中一个有代表性的α值是 2.1[①]。另外，一些研究表明，整个Web有向图结构是个蝴蝶结(*bowtie*)形，其中主要包含三大类网页，分别是IN、OUT和SCC。Web冲浪者能够从IN中的任一网页出发通过超链接到达SCC的任一网页。同样，冲浪者可以从SCC中的网页达到OUT中的任一网页。最后，从SCC中的任一网页可以到达SCC中的其他网页。然而，不可能从SCC中的网页到达IN的任一网页，也不能从OUT中的网页到达SCC中的任一网页（当然此时也不能到达IN中的任一网页）。

值得注意的是，在一些研究中，IN 和 OUT 的规模大致相当，而 SCC 的规模则稍大，大部分网页都落入到这三大类中。剩余的网页构成了所谓管道（tube），它由少部分 SCC 之外的网页组成，可以直接将 IN 和 OUT 中的网页相连。另外，还有一些不能从 IN 到达或者不能到达 OUT 的网页构成的所谓卷须（tendril）。图 19-4 给出了 Web 图的这种结构。

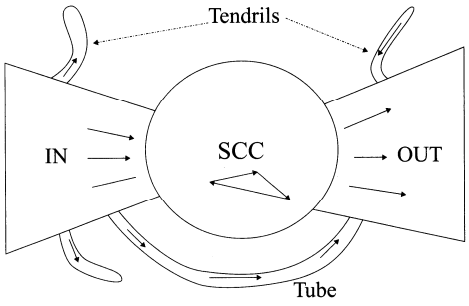


图 19-4 Web 图的蝴蝶结形结构。这里给出了一个管道和三个卷须

19.2.2 作弊网页

在早期的 Web 搜索历史上，Web 搜索引擎显然是连接广告商和顾客的一种重要途径。用户在搜索“maui golf real estate”时，他想做的不仅仅是搜索有关 Maui 岛上高尔夫球场地产的新闻或者娱乐信息，而且很可能要寻找并购买这样的地产。因此，地产销售商及其代理机构就有强烈的动机来建立针对该查询的高排名网页。在一个基于词项频率排名的搜索引擎中，一个反复出现“maui golf real estate”的网页排名将会很高。这导致了第一代作弊网页（spam）的产生，即通过操作网页内容来达到在某些关键词的搜索结果中排名较高的目的（这里指的是在 Web 搜索的场景下）。为了避免用户对这些冗余和重复信息的极度反感，一些老练的作弊者还会采用一些手段和技巧，比如将这些重复的词设置成和背景一样的颜色。尽管这些词对用户不可见，但

① 参考第 5 章有关词分布的 Zipf 定律，它实际上也是一种幂分布，只不过那里的 $\alpha=1$。

是搜索引擎却会从网页的 HTML 表示中分析出这些词并会对它们建立索引。

从根本上说，网页作弊起源于网页内容建设动机的多样性。实际上，很多网页内容的建设者都有商业动机，因此他们希望通过操作搜索引擎的结果来获益。你可能认为：这样做与公司采用大字体在黄页上列出其电话号码并没有什么不同，但是公司通常花费了很大代价，而且这可能是一种更公平的机制。一个更恰当的比喻也许是公司名的使用，它们可以在公司名前面加上一长串字母 A，这样该公司名在黄页列表中就会排名靠前。实际上，这种公司为黄页中大黑字体付费的模型已经被复制到 Web 搜索中：在很多 Web 搜索引擎中，有可能可以通过付费来将自己的网页放入到搜索引擎的索引中，这个模型称为付费收录（paid inclusion）。对于是否允许付费收录、付费是否会影响搜索引擎的排名结果，不同的搜索引擎会有不同的政策。

针对上述作弊技术，搜索引擎也很快变得更加成熟，已经可以筛选出通过大量复制某些特定关键词的作弊网页。为此，作弊者也发展出了更多的作弊技术，下面将介绍它们当中比较著名的几种。第一种技术被称为伪装（cloaking），图 19-5 给出了一个示意图。这里，根据 http 请求是来自搜索引擎的采集器（搜索引擎中获取网页的部件，将在第 20 章介绍）还是用户所使用的浏览器，作弊 Web 服务器会返回不同的网页结果。如果是前者，那么会返回一个包含欺骗性关键词的作弊网页供搜索引擎索引。这样，当用户输入这些关键词并选择该网页进行浏览时，他看到的却是与搜索引擎索引的内容不同的另一个网页。这种对搜索引擎索引器的欺骗在传统的 IR 当中是没有的，它主要是由于网页发布者和搜索引擎之间的不完全协作而造成的。

桥页（doorway page）包含了精心挑选的文字和元信息，通过这些信息能够针对某些选定的搜索关键词来提高排名。当某个浏览器请求访问桥页时，它会重定向到一个更具商业性的网页。更复杂的作弊技术还包括操纵与网页相关的元数据及指向网页的链接等。由于作弊的根源

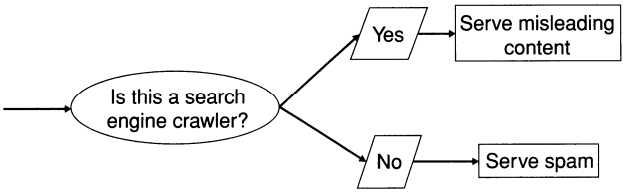


图 19-5　作弊者使用的“伪装”技术

来自经济利益的驱动，因此涌现了一个被称为 SEO（Search Engine Optimizers，搜索引擎优化）的产业，SEO 为那些希望在搜索结果中提高关键词排名的客户提供顾问服务。搜索引擎对这些试图破解和适应其排名技术的做法十分头疼，并颁布了一系列政策来规定一些不能容忍的 SEO 行为。并且，据我们所知，一旦某些 SEO 违反了上述政策，搜索引擎可能会切断他们的搜索请求。这些 SEO 能逐渐推断出每个 Web 搜索引擎排名算法的特性，而搜索引擎公司则会不断做出应对，他们之间的斗争将永不停止。实际上，这场斗争的一个结果是，研究领域里也出现了一个被称为对抗式信息检索（adversarial information retrieval）的子领域。为了对抗作弊者通过操

作网页内容进行作弊的做法，人们开发出了一种利用 Web 中链接结构的被称为链接分析（link analysis）的方法。尽管几乎所有的现代搜索引擎都使用了链接分析技术（随之而来的是，现在作弊者下了很大力气在开发利用链接分析进行作弊的所谓链接作弊（link spam）技术），但目前所知的最早大规模使用链接分析方法（细节参见第 21 章）的搜索引擎仍然是 Google。

? 习题 **19-1** 如果入度为 i 的网页的数目正比于 $1/i^{2.1}$，那么随机选择一个网页的入度是 1 的概率是多少？

习题 **19-2** 如果入度为 i 的网页的数目正比于 $1/i^{2.1}$，那么一个网页的平均入度是多少？

习题 **19-3** 如果入度为 i 的网页的数目正比于 $1/i^{2.1}$，那么当最大入度变成无穷大时，入度为 i 的网页比例是升高还是降低？如果指数不是 2.1，那么上述问题的结论如何？

习题 **19-4** 在某个 Web 快照图中的所有节点的平均入度是 9，那么该快照图中的平均出度如何？

19.3 广告经济模型

在 Web 发展的早期，公司会将图形化的广告横幅放在流行的网站（新闻或者娱乐网站，比如 MSN、America Online、Yahoo!或 CNN 等）上。这些广告的主要目的是品牌推广（branding），即让浏览用户对公司的品牌产生好感。通常来说，这些广告是按照 CPM（*Cost per Mil*，每千次显示会费）机制付费，即横幅广告显示 1 000 次为一次付费单位。按照某些网站和广告客户之间的合同规定，广告并不是按照显示的次数（也被称为印象数 impression）来付费，而是按照用户的点击次数来付费。这种付费模型被称为按 CPC（*Cost per Click*，每次点击付费）模型。这种情况下，对广告的点击会将用户引导到广告商所建立的页面上来，从而能促进用户对商品的购买。在这里，广告的目的就不再是品牌推广而是促进交易了。面向品牌及面向交易的广告之间的区别已经在一些传统的媒体(如广播和印刷品)上得到广泛公认。Web 的交互性使得 CPC 付费模型成为可能，用户的点击可以被网站记录和监控，然后根据记录的情况就可以给广告商寄去账单。

采用 CPC 机制的先驱是一个名叫 Goto 的公司，它在最终被 Yahoo!收购之前改名为 Overture。Goto 并不是一个传统意义上的搜索引擎公司，实际上，对于每个查询项 q，它会接受那些期望将网页与 q 进行关联的公司的竞标。为应答查询 q，Goto 会返回所有投标广告商的网页结果，并按照他们的投标价格进行排序。此外，当用户点击某个返回结果时，相应的广告商会付费给 Goto（在 Goto 最初的实现当中，广告商是按照投标的价格来付费）。

Goto 模型的某些方面值得我们重点强调一下。首先，用户在 Goto 搜索界面上输入查询 q 时，实际上积极地表达了与 q 相关的用户兴趣和意图。比如，输入“golf clubs”的用户可能比那些浏览高尔夫新闻的人拥有更迫切的购买愿望。其次，Goto 仅仅在用户对广告表现出真实兴趣时才向广告商收费，即用户此时真正点击了广告。综上所述，以上这些方面创造了一个连接广告商和消费者的强有力的机制，这很快也为 Goto/Overture 带来了每年上亿美元的丰厚利润。

这种模式的搜索引擎在后来被称为赞助搜索（sponsored search）或者搜索广告（search advertising）。

给定两种搜索引擎，其中一种是类似 Google、Altavista 的“纯”搜索引擎，而另一种是上面提到的赞助搜索引擎。一个很自然的步骤就是将它们合并到统一的用户体验当中。现有的搜索引擎基本都是按照如下模式：对于用户的查询，搜索引擎会将“纯”搜索结果（通常也被称为基于算法的搜索结果）作为主要结果返回给用户，同时赞助搜索结果在算法结果的右侧独立并有区别性地显示出来。图 19-6 给出了一个例子。现在，针对某个查询返回赞助搜索结果并对它们排序已经变成一个远比 Goto 机制复杂得多的过程，其中包含了来自 IR 和微观经济学的各种思想。这些内容已经超过了本书的讨论范围。对于广告商来说，了解搜索引擎如何对广告排序、如何给不同关键词及不同搜索引擎分配不同的营销预算已经发展为一门被称为 SEM（*Search Engine Marketing*，搜索引擎营销）的职业。

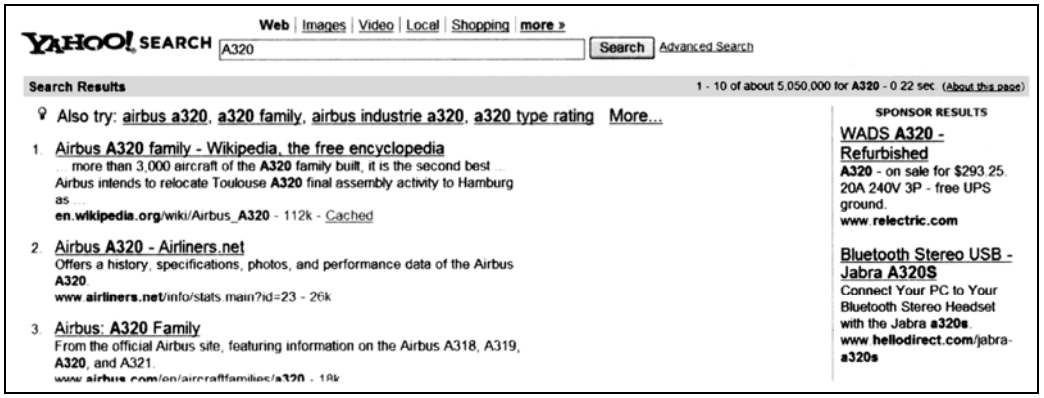


图 19-6　查询关键词触发的搜索广告。这里输入查询 A320 会返回一系列有关空中客车的算法性结果，同时也返回了一些货物号码为 A320 的、与飞机无关的物品，广告商希望通过这些查询来寻找市场。飞机广告的缺乏也反映了几乎没有人会在 Web 上出售 A320 飞机这个事实

赞助搜索背后所固有的经济利益也驱使一些人为达到自己的目的来暗中破坏系统。有很多种破坏形式，其中一种被称为垃圾点击（*click spam*）[①]。目前垃圾点击在国际上还没有公认的定义。正如其名字所体现的，它指的是搜索用户对赞助搜索结果的非善意点击。比如，一个心术不正的广告商可能会试图通过重复点击（使用机器点击生成器）其竞争者的赞助搜索广告来耗尽其广告预算。搜索引擎面对的一个挑战是，如何从一系列点击中发现垃圾点击的模式，从而避免广告客户为这些垃圾点击付费。

① 这里主要指欺诈点击或者恶意点击，但是译成垃圾点击还可以包括那些无意图的无用点击，因为后者也常常用 click spam 这个说法。——译者注

? 习题 **19-5** Goto 方式根据报价的高低来对广告进行排序，出价最高的广告商获得最高的位置，出价第二高的次之，其余以此类推。如果出价最高的广告商给出的广告与查询无关时会出现什么问题？为什么会出现这样的情况？

习题 **19-6** 假定除了出价之外，我们还有每个广告商的点击率（click-through rate）数据，即在历史上用户点击该广告商广告的次数与广告显示次数的比值。请对 Goto 机制进行修改，使之能够利用点击率信息来避免习题 19-5 中的问题。

19.4 搜索用户体验

理解 Web 搜索的用户也极其重要。传统 IR 中的用户通常至少是受过查询措辞训练的专业人士，并且他们查询的文档库通常也是良好的创作结果，其风格和结构都很容易被用户所理解。而 Web 则体现出明显的不同，Web 搜索用户往往不知道或者根本不关心 Web 内容的多样性、查询语言的语法以及查询的措辞方法。实际上，主流的工具（包括 Web 搜索）都不应该对数十亿用户施加这种烦琐的要求。一系列的研究结果都表明，Web 搜索中的平均查询关键词个数大概是 2 到 3 个，并且用户很少使用语法操作符（布尔连接符、通配符等）。之所以会出现上述结果，当然还是因为搜索用户的组成主要是“普通”人而不是信息专家的缘故。

很显然，搜索引擎吸引的用户流量越大，那么从赞助搜索中获得的利润也就更多。搜索引擎如何使自己与众不同从而提高他们的流量？这里，Google 确定了两个帮助自己增长同时使竞争者下降的原则：(i) 关注相关性，特别是排名靠前的一些结果的正确率而不是召回率；(ii) 用户体验要轻量级，也就是说查询页面和返回结果页面应该简洁整齐，并且这些页面上基本没有图像成分，而应该几乎全是文本内容。第一个原则可以帮助用户减少信息定位的时间。第二条原则能够保证用户得到飞快的用户体验，或者说无论如何，载入查询页面或者结果页面都不会成为瓶颈。

用户查询需求

普通的 Web 搜索查询似乎可以分成三大类：(i) 信息类（informational）；(ii) 导航类（navigational）；(iii) 事务类（transactional）。下面我们将分别解释这几个类别的含义，很显然有些查询可能会同时属于多个类，而有些查询则根本不在上述类别中。

信息类查询（informational query）主要查找的是与某个宽泛主题相关的一般信息，比如有关白血病（leukemia）或者普罗旺斯（Provence）的查询。通常来说，这时候不存在能够包含所有要查询信息的单个网页。实际上，用户输入信息类查询的目的往往是想从多个不同的网页中抽取信息。

导航类查询（navigational query）查找的是用户心目中某个实体的网站或者主页，比如汉莎航空（Lufthansa airlines）。这种情况下，用户最期望的结果就是汉莎航空公司的主页出现在搜索结果的第一个位置。用户对包含词项 Lufthansa 的其他很多文档都不感兴趣，对这种用户来说，

最好的用户满意度评价指标就是正确率为 1。

事务类查询（transactional query）是用户在 Web 上进行事务处理的一个先导型查询，这些事务处理包括产品购买、文件下载或进行预订等。这种情况下，搜索引擎应该在结果中列举出一些服务，它们能够提供上述事务处理的交互接口。

将查询分到上述几类中也是一个挑战性问题。类别信息不仅可以用于控制基于算法的搜索结果，也可以用于查询和赞助搜索结果的匹配当中（由于查询可能表示出购买意图）。对于导航类查询来说，有些人认为搜索引擎应该仅仅返回一条结果记录甚至是直接返回目标页面。然而，历史上很多 Web 搜索引擎都参与了索引量大小的竞争，大家都鼓吹自己的索引量比其他搜索引擎大。那么，用户是否会真正关注索引量大小呢？也许并不这样，但是媒体往往突出不同搜索引擎大小的估计结果（这些结果往往在统计上站不住脚）。用户很容易受到这些报道的影响，因此，搜索引擎必须要重视自己及其竞争者的索引量大小。对于信息类查询（以及在一定程度上也包括事务类查询）而言，用户确实关心搜索引擎的覆盖面。

图 19-7 给出了一个搜索引擎的组成示意图，包括采集器、网页索引及广告索引。虚线下面的部分属于搜索引擎的内部结构。

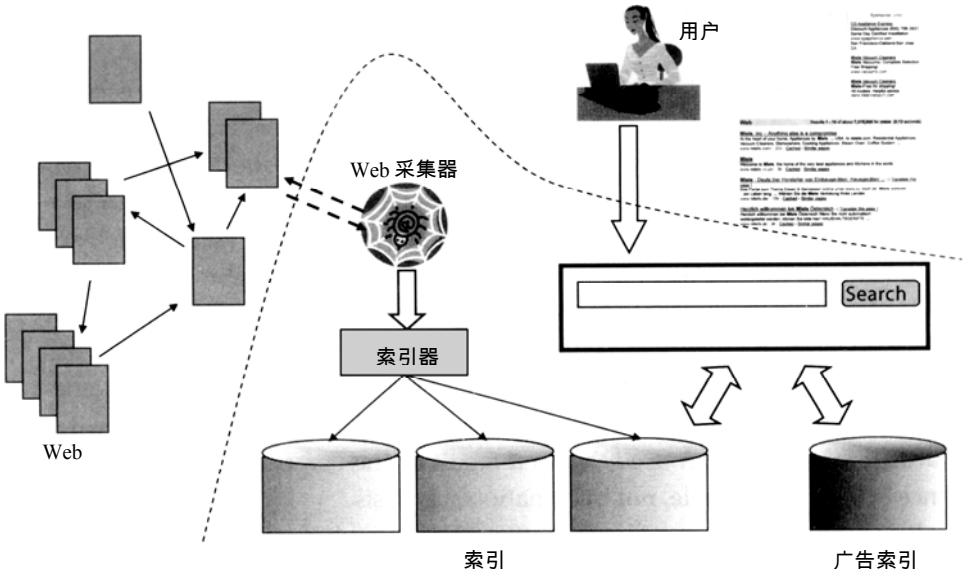


图 19-7　一个 Web 搜索引擎的组成示意图

19.5　索引规模及其估计

尽管由于有些网页包含的信息量更大，搜索引擎具体索引了哪些网页至关重要，然而初步估计的话，可以说索引规模越大，搜索引擎的覆盖面也越大。估计出某个搜索引擎的索引规模占整个Web的比例仍然是非常困难的，这是因为Web中存在无数的动态网页。比如，地

址 http://www.yahoo.com/any_string 会返回一个有效的HTML页面而不是错误页面，它很友好地告诉用户Yahoo!中不存在这个页面。这种“软 404 错误”页面只是Web服务器产生的无数动态有效页面的一个例子。实际上，其中有些页面可能是恶意的采集器（搜索引擎抓取网页信息的部件）的“陷阱”页面，它们能将搜索引擎的采集器陷入到作弊者的网站中去，从而能够索引这个网站上的大量网页。

一个具有更好定义的问题如下：给定两个搜索引擎，它们索引的相对规模如何？当然，即使是这个问题也是不精确的，具体原因有以下几个方面。

(1) 在查询应答时，搜索引擎可以返回(全部或者部分)内容未被索引的网页。一方面，搜索引擎通常只索引每个网页的前几千个词。另一方面，有些情况下，搜索引擎知道某个页面 p 链接到它索引的某些网页，但是它并没有直接对 p 建立索引。正如我们将要在第 21 章中看到的那样，在检索结果中返回 p 仍然是很有意义的。

(2) 搜索引擎通常会按照不同层次或者分区来组织索引，每次搜索过程当中不一定会调用所有的索引（参考 7.2.1 节的层次索引）。例如，一个网站的深层页面可能会被索引，而在一般的 Web 搜索中并不会被返回。但是，如果用户限定在该网站内搜索（大多数搜索引擎都提供了限定在某个网站进行搜索的功能）时，则该页面会被返回。

综上所述，搜索引擎的索引中包含了多类页面，因此无法采用单一指标来衡量索引规模。尽管如此，研究人员仍然设计出不少方法来粗略估计两个搜索引擎 E_1 和 E_2 的索引规模的相对比值。这些技术背后的基本假设是每个搜索引擎都从 Web 中随机、独立、均匀地选择了部分网页进行索引。此外，这里面还包括一些可能存在疑问的假设：第一，Web 的规模是有限的，每个搜索引擎都从中选择了一个子集；第二，每个搜索引擎都从 Web 中独立、均匀地选择了部分网页子集。正如我们将要从第 20 章有关采集的讨论中看到的那样，上述假设与实际情况相差太远。但是，如果我们基于这些假设，那么就可以采用一种被称为捕获再捕获（capture–recapture method）的传统技术来进行估计。

假定我们从E_1的索引中随机选取一个网页，并检验它是否属于E_2。同样也可以检验某个从E_2中随机选取的网页是否属于E_1。根据上述实验，可以得到两个分数x和y，其中x表示从E_1中抽出的网页属于E_2的比例，而y则表示从E_2中抽出的网页属于E_1的比例。因此，假定搜索引擎E_i的规模用$|E_i|$来表示的话，那么有[①]

$$x|E_1| \approx y|E_2|。$$

上式可以进一步写成如下形式：

$$\frac{|E_1|}{|E_2|} \approx \frac{y}{x}。 \tag{19-1}$$

如果上述关于E_1和E_2是Web中独立均匀随机抽样子集的假设是正确的话，另外再假定我们

① 实际上有，$x|E_1| \approx y|E_2| \approx |E_1 \cap E_2|$。——译者注

的抽样过程无偏，那么公式（19-1）就给出了$|E_1|/|E_2|$的一个无偏估计。这里我们区分两种情况，第一种情况是这是能够访问搜索引擎的索引的人（比如E_1的雇员）做出的估计[①]。第二种情况是不能进入搜索引擎内部的第三方机构做出的估计。前一种情况下，只需要从某个搜索引擎选出随机页面即可。而后一种情况却更具挑战性。因为，我们要从搜索引擎外部为一个搜索引擎选择随机页面，然后验证它是否在另一个搜索引擎中存在。为了实现这种抽样过程，我们可能需要从全体Web（这里的Web是规模有限的理想化网页集合）上产生一个随机网页，然后测试它是否存在于每个搜索引擎。然而，均匀随机选出一个网页是非常困难的。下面，我们首先简单介绍几种构造此类样本的方法，并指出每种方法内在的偏向性，然后对一种在多项研究上都采用的技术给出详细介绍。

(1) 随机搜索法(random search)：利用 Web 搜索的搜索日志，从日志中随机选择一个查询给E_1，从返回结果中再随机选出一个网页。由于这些日志不可在搜索引擎之外广泛使用，因此，这种方法的一种实现途径是，可以在经人同意的情况下跟踪某些用户（比如一个研究中心的所有科学家）提交的所有查询。这种做法有一些问题，比如这些人提交的查询类型的倾向性。另外，在 E_1 中随机搜索并从结果中随机选出的文档并不等于直接从 E_1 中随机选出的文档。

(2) 随机 IP 地址法（random IP address）：第二种做法是随机产生一系列 IP 地址，然后发送请求给以随机 IP 为地址的 Web 服务器，最后收集该服务器的所有网页。这里的偏向性包括：有很多主机可能会共享一个 IP（很多是因为采用了虚拟主机技术），或者选出的主机根本不接受 http 请求。另外，该技术选出的网站很可能是具有较少网页的网站中的一个，从而造成文档概率的偏向性，当然如果我们知道网站的网页数目的分布，那么就可以对上述偏向性进行修正。

(3) 随机游走法（random walk）：如果 Web 图是一个强连通有向图，那么就可以从任一个 Web 页面出发进行随机游走。这种游走会收敛到一个稳定的状态分布（参见 21.2.1 以获得更多的背景知识），这样在理论上我们就能以某个固定概率选择一个网页。当然，这种方法也有很多偏向性。第一，Web 不是强连通的，即使采用多个矫正规则，也很难说就能从任一网页出发到达一个稳态分布。第二，从随机游走到达稳态分布的时间是未知的，可能会超过实验本身的时间。

显然，上面的每种方法都远远不够完美。下面我们将介绍第四种抽样方法——随机查询法（random query）。该方法之所以值得介绍主要有两个原因：第一，它已经应用到一系列越来越精确的估计上；第二，它已被证明是最有可能被误解和错误实施的方法，那样会导致具有误导性的结果。随机查询的思路是，通过发送一个随机查询给搜索引擎来（几乎）均匀随机地从搜索引擎的索引中选出一个页面。很显然，从韦氏词典（Webster's dictionary）中随机选择词项并不是实施上述思想的一个好办法。一方面，词汇表中词项的出现频度是不等的，因此这样做并不

① 有关的估计方法可以参见 Pierre Baldi, Paolo Frasconi and Padhraic Smyth, Modeling the Internet and the Web: Probabilistic Methods and Algorithms, 2003 的第 45-46 页。——译者注

能从搜索引擎中均匀、随机地选出文档。另一方面，Web 文档中的很多词项并不出现在如韦氏词典一样的常规词典中。为了解决词项不在常规词典中出现这个问题，我们首先需要收集一个 Web 词典样本。这可以通过采集 Web 的一部分内容来实现，或者可以采集那些人工编辑的有代表性的 Web 子集，如 Yahoo!（早期的实验中就是这种做法）。考虑从这部词典中随机抽取的两个或者多个词组成的与查询。我们的处理如下：给 E_1 提交一个随机的与查询，从返回页面的前 100 条结果中随机选择一个页面 p。然后，我们再从 p 中选出 6 到 8 个低频词项组成与查询提交给 E_2 进行检测。上述过程可以重复多次以便提高估计的精度。不论是抽样过程还是检测过程都需要考虑下列问题。

(1) 我们的样本偏向于更长的文档。

(2) 从 E_1 的前 100 个结果中选择会导致结果偏向于 E_1 的排序算法，而选择 E_1 的所有结果又会造成实验很慢。由于大部分搜索引擎都会防止过多的机器人自动查询，所以上述说法在实际中更为合理。

(3) 检测过程也引入了很多额外的偏向性。比如，E_2 就可能不能正常地处理 8 个词的与查询。

(4) 如果 E_1 或 E_2 都把检测查询看成是机器的恶意查询，那么他们可能拒绝对这些查询进行应答。

(5) 处理过程中可能会有类似连接超时的问题。

基于上面提到的基本模式，人们进行了一系列索引大小评价的研究，并试图解决评价中的一些问题。当然，目前并没有完美的解决方案，但是理解这些偏向性的统计方法的成熟度也在不断提高。对每篇文档，解决这些估计中的偏向性的主要思路是估计偏向性的数量级。基于此，常规的统计抽样技术可以用于产生无偏样本。在检测阶段，一些更新的工作已经从与查询转向了短语，以及其他看上去具有更好表现的查询。最后，一些新实验使用了除随机查询之外的其他抽样方法。这当中最著名的是文档随机游走抽样（document random walk sampling）方法，它首先根据文档之间的关系形成文档虚拟图，然后通过随机游走方式从这个图中选出文档。该图当中的每个节点是一篇文档，如果两篇文档包含两个或更多的公共词，则它们之间存在一条边。虽然我们无法得到上述图的整体情形，但是我们至少可以在这个图上进行随机游走。从该图的一个文档 d 到另一个文档的随机游走过程可以这样来完成：首先从文档 d 中选择一对关键词，然后将关键词输入到某个搜索引擎，并从返回结果中随机选择一篇文档。相关的细节可以参见 19.7 所提供的参考文献。

? 习题 **19-7** 两个 Web 搜索引擎 A 和 B，每个搜索引擎都从其索引中随机均匀地产生大量网页。A 中页面有 30%属于 B，而 B 中页面有 50%属于 A，那么 A 和 B 两个引擎的索引规模的相对大小值是多少？

19.6　近似重复及 shingling

在 19.5 节中讨论索引规模时我们忽略了 Web 网页的重复（duplication）问题，也就是说 Web 上包含了大量具有相同内容的重复网页。有估计认为，Web 中大概有 40%的网页和其他网页重复。当然，这些重复现象当中，有很多是合法的。比如，对某个信息库做多份映像只是为了提供冗余备份和提高访问的可靠性。搜索引擎试图尽量避免索引重复的网页，这样就可以降低存储和处理开销。

检测重复最简单的方法就是为每个网页计算出一个指纹（fingerprint），它是整个网页文本的一个很精炼（比如说 64 位）的摘要。然后，一旦发现两篇文档的指纹一样，我们就会检查这两篇文档是否真的相同，如果相同，那么我们就认为其中一篇文档是另一篇文档的副本。但是，在面对 Web 上一个更广泛的被称为近似重复（*near duplication*）的现象时，上述的简单方法却并不成功。在很多情况下，一个网页的内容并不会完全等同于另一个网页的内容，而是在某些字符上有点差异。比如，两个网页相差的就只是最后的更新日期。即使在这种情况下，由于两个网页的重复度已经足够高，所以我们希望只索引一份网页。当规模上升到数亿时，要对这些页面进行全部比较就是一个不可能完成的任务。因此，一个问题就是，我们应该如何检查并过滤掉这些近似重复文档？

下面我们将介绍一个近似重复的解决方案，该方案主要基于一个被称为搭叠（*shingling*）[①]的技术（参见图 19-8）。给定正整数k及文档d的一个词项序列，可以定义文档d的k-shingle为d中所有k个连续词项构成的序列。例如，考虑文档 a rose is a rose is a rose，它的 4-shingle（在近似重复检测中，$k=4$是一个常用值）为“a rose is a”、“rose is a rose”及“is a rose is”，前两个 4-shingle 在文本中都出现两次。直观上看，如果两个文档的shingle集合几乎一样，那么它们就满足近似重复。下面我们将这种直观性精确化，并开发一种计算和比较所有Web网页之间shingle的高效算法。

① shingling 技术实际上是一种类似瓦片搭叠的技术，每个 shingle 相当于一块瓦片。在这个意义上说，每篇文档是由多个瓦片搭叠而成的。因此，shingling 可以译成搭叠技术，而 shingle 则可以译成搭叠块。但是从目前所看到的文章看，这个词都没有正式翻译，为了和其他文献保持一致，我们最后采用原文而不对它们进行翻译。——译者注

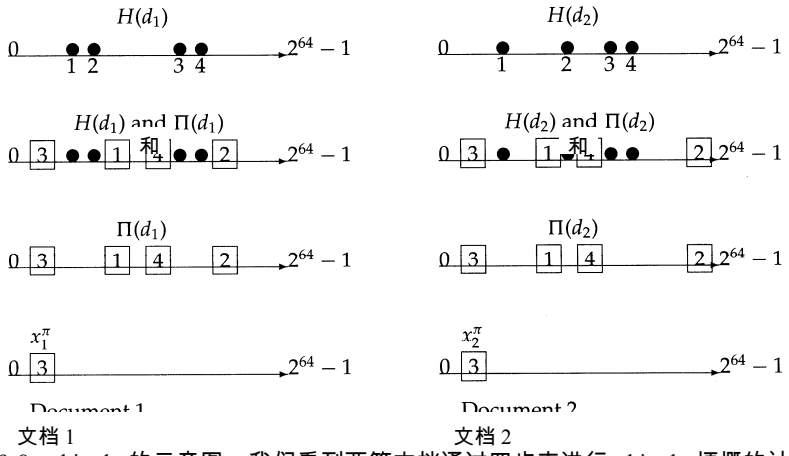


图 19-8 shingle 的示意图。我们看到两篇文档通过四步来进行 shingle 梗概的计算。第一步（最上面一行），对每篇文档的每个 shingle 应用一个 64 比特位哈希函数得到两篇文档的 $H(d_1)$和 $H(d_2)$（以圆圈表示）。然后，对 $H(d_1)$和 $H(d_2)$采用随机置换，获得 $\Pi(d_1)$及 $\Pi(d_2)$（以正方形表示）。第三行给出了 $\Pi(d_1)$及 $\Pi(d_2)$，最后一行给出了每篇文档的最小值 x_1^π 和 x_2^π

令 $S(d_j)$ 表示文档 d_j 中的 shingle 集合，我们在前面 3.3.4 节中曾经提到过一种计算两个集合 $S(d_1)$和 $S(d_2)$之间重叠度的 Jaccard 系数，它的计算公式为$|S(d_1)\cap S(d_2)|/|S(d_1)\cup S(d_2)|$，记为 $J(S(d_1), S(d_2))$。通过计算这种 Jaccard 系数就可以判断 d_1 和 d_2 是否近似重复，如果该值超过某个预先给定的阈值(比如 0.9)，那么我们就可以认为它们是近似重复文档，在索引时就会去掉其中一篇文档。然而，这似乎并没有对问题进行简化，因为我们仍然需要计算两两文档之间的 Jaccard 系数。

为了避免上述问题，我们使用某种哈希的形式。首先，我们将所有 shingle 都映射到一个大空间（比如 64 比特位）下的哈希值。令 $H(d_j)$为所有 $S(d_j)$中的 shingle 映射出的 64 比特位的哈希值集合，其中 $j=1$ 或 2。下面我们将采用一个技巧来检测出哪些文档对的 $H(\ )$集合之间具有较大的 Jaccard 重叠度。

令π 为从 64 比特位整数到 64 比特位整数的一个随机置换[①]。$H(d_j)$中所有哈希值的置换结果集合记为$\Pi(d_j)$，因此对每个$h\in H(d_j)$，都存在一个相应值$\pi(h)\in\Pi(d_j)$。

令 x_j^π 为 $\Pi(d_j)$ 中最小的整数。于是有：

定理 19-1

$$J(S(d_1),S(d_2))=P(x_1^\pi=x_2^\pi)。$$

证明：下面我们给出一个更一般情况下的证明。考虑一系列集合，它们的元素均来自同一个论域。我们将这些集合看成是一个矩阵 A 的列，这个矩阵的每一行都对应论域的一个元素。

① 随机置换（random permutation）指的是将集合对象随机排序的过程。比如 5 个整数集合{1,2,3,4,5}的一个随机置换结果可能是{3,2,1,5,4}。这个置换也相当于定义了一个函数π：π(1)=3，π(2)=2，π(3)=1，π(4)=5，π(5)=4。本节中的π的定义域为 0~264−1，值域也是 0~264−1，相当于把 264 个整数随机排序。——译者注

矩阵元素 $a_{ij}=1$ 表示元素 i 在第 j 列所代表的集合 S_j 中出现。

令 Π 为 A 中行与行元素之间的随机置换，将 Π 应用到 A 中第 j 列的结果记为 Π(S_j)。最后令 x_j^{π} 为第一个出现 Π(S_j)值为 1 的行下标。那么我们需要证明对任意两列 j_1、j_2，有

$$J(S_{j_1},S_{j_2})=P(x_{j_1}^{\pi}=x_{j_2}^{\pi})\ 。$$

如果我们能证明这个等式，那么就能证明上述定理。

考虑图 19-9 所示的两列 j_1 和 j_2，S_{j_1} 及 S_{j_2} 的有序组合可以将行划分成四种类型：两列中的值均为 0、S_{j_1} 中的值为 0 而 S_{j_2} 中的值为 1，S_{j_1} 中的值为 1 而 S_{j_2} 中的值为 0 以及两列均为 1。实际上，图 19-9 中的前 4 行就代表了所有 4 种可能的行类型。将两列均为 0 的行数目记为 C_{00}，同样可以分别将其他三种类型的行数目记为 C_{01}、C_{10} 及 C_{11}。于是有

$$J(S_{j_1},S_{j_2})=\frac{C_{11}}{C_{01}+C_{10}+C_{11}}。\qquad (19\text{-}2)$$

为了完成证明，我们需要证明公式（19-2）的右部等于 $P(x_{j_1}^{\pi}=x_{j_2}^{\pi})$。考虑对列 j_1，j_2 进行扫描并不断增加行下标直到任一列出现非零值为止。由于 Π 是一个随机置换，所以该最小行下标对应的两个列都是 1 的概率正好等于公式（19-2)的右部。证毕。

S_{j1}	S_{j2}
0	1
1	0
1	1
0	0
1	1
0	1

图 19-9　两个集合 S_{j_1} 及 S_{j_2} 的 Jaccard 计算（结果为 2/5）

因此，对 shingle 集合计算 Jaccard 系数的检测过程是基于概率的。我们计算不同文档的 x_i^{π} 值并进行比较。如果两个文档的值一致，那么就得到候选的近似重复文档。使用 200 次随机置换 π（该领域文献的建议值）来独立地重复上述过程。我们称这 200 次 x_i^{π} 的结果集合为文档 d_i 的梗概（sketch）$\psi(d_i)$，于是我们可以通过计算$|\psi_i\cap\psi_j|/200$ 来估计任意两篇文档 d_i、d_j 的 Jaccard 系数，如果它超过某个既定阈值，我们就认为 d_i 和 d_j 相似。

如何对任意的 i、j 来快速计算$|\psi_i\cap\psi_j|/200$？实际上，如果不进行优化，那么需要比较的次数是文档数目的平方级别，为了避免这种比较次数的爆炸式增长，应该如何表示那些相似的文档对？首先，我们可以通过指纹的方法找到所有那些完全相等的文档并只保留一份副本。在 shingle 计算中，也可以去掉那些普通的 HTML 标签和整数，并去掉那些在文档中出现得非常普遍但是对重复检测毫无价值的 shingle。下一步使用一个并查（union-find）算法来建立包含相似文档的簇。为了达到这一目的，必须要完成最关键的一步，即从梗概集合出发得到 i、j 对的集合，其中文档 d_i 与文档 d_j 相似。

为此，我们计算任意具有公共梗概的文档对之间的公共 shingle 数目。首先有一张按 x_i^{π} 排序的表$<x_i^{\pi},d_i>$，对每个 x_i^{π}，生成同时在梗概中包含 x_i^{π} 的 i、j 对。基于这些可以对每个梗概之

间满足非零重合度 i、j 对计算它们公有的 x_i^{π} 值的数目。应用一个预定阈值，我们可以知道哪些 i、j 对的梗概之间具有很高的重合度。比如，如果阈值为 0.8，那么对任意 i、j 来说，需要的数目至少是 160。当我们找到这些文档对时，就可以运行并查算法将文档聚成近似重复的多个“句法簇”（syntactic clusters）。本质上这是 17.2 节中所介绍的单连接聚类算法的一个变形。

最后一个技巧是缩减对所有 i、j 对计算$|\psi_i \cap \psi_j|/200$ 所需要的空间，虽然在理论上可能仍然需要文档数目的平方级空间。为了去掉那些之间几乎没有公共 shingle 的文档对，我们对每个文档的梗概进行如下预处理：将梗概中的 x_i^{π} 排序，然后将这些排好序的序列搭接起来构成每篇文档的超 shingle。如果两篇文档具有公共的超 shingle，我们才会去计算$|\psi_i \cap \psi_j|/200$ 的精确值。这仍然是一个启发式方法，但是能够大大缩减需要累加梗概重叠数目的 i、j 对数目。

习题 **19-8** 两个 Web 搜索引擎 A、B，每个引擎都采集了 Web 的一个随机子集，并且这两个子集规模一样。采集到的部分网页是重复的，即不同的 URL 包含完全一样的文本。假定这些重复在所有 A 和 B 采集到的网页中满足均匀分布。另外，我们假定这里说的重复，指的是一个网页只有两个副本，也就是说，这里假设网页的重复次数不会超过 2。A 引擎索引时没有去重，而 B 引擎对每个重复文档只索引了一个副本。在去重之前，两个网页子集的规模一样，如果 A 索引的 URL 中有 45%被 B 索引，而 B 索引的 URL 中的 50%又被 A 索引，那么 Web 中没有重复的网页的比例是多少？

习题 **19-9** 和图 19-8 不一样，我们采用另外一种估计两个集合 S_1 及 S_2 的 Jaccard 系数的方法，过程如下：从 S_1 和 S_2 的论域中随机选择一个子集，这相当于在图 19-8 的证明过程中从矩阵 A 中随机选择某些行组成子集。我们穷尽一切可能来计算这些随机子集之间的 Jaccard 系数，为什么说上述计算的结果是 S_1 和 S_2 的 Jaccard 系数的一个无偏估计？

习题 **19-10** 解释上述计算方法在实际中很难应用的原因。

19.7 参考文献及补充读物

Bush（1945）在介绍一个被他称之为 memex 的信息管理系统时便预示了 Web 的产生。Berners-Lee 等人 (1992) 描述了一种最早的 Web 具体形式。Kumar 等人 (2000)及 Broder 等人 (2000) 给出了有关 Web 图研究的全面介绍。最早使用锚文本的工作参见 McBryan (1994)。19.4 节中提到的查询分类体系归功于 Broder (2002)的工作。19.2.1 节中提到的指数为 2.1 的幂定律出现在 Kumar 等人 (1999)的文献中。而在 Web 搜索和分析方面，Chakrabarti (2002)是一篇非常好的参考资料。

有关 Web 搜索的索引规模估计的研究由来已久，有关工作参见 Bharat 和 Broder（1998）、Lawrence 和 Giles（1998）、Rusmevichientong 等人（2001）、Lawrence 和 Giles（1999）、Henzinger 等人（2000）及 Bar-Yossef 和 Gurevich（2006）。目前最先进的工作是 Bar-Yossef 和 Gurevich（2006），其中包括了 19.5 节末尾提到的多种偏向性去除技术。shingling 技术由 Broder 等人（1997）

提出，Bharat 等人（2000）则将之应用于网站（不是简单的网页）的查重。

第20章 Web采集及索引

20.1 概述

Web采集是从Web中收集网页的过程，这些网页用于索引从而为搜索引擎提供支持。采集的目标是尽可能高效地采集更多数目的有用页面，并同时获得连接这些页面的链接结构。第19章中，我们谈到Web复杂性的根源在于其创建的无协调性。本章当中，我们将研究此根源给Web采集所带来的困难。本章我们将主要关注图19-7中一个被称为Web采集器（web crawler）的组成部分，有时它也被称为网络蜘蛛（spider）[①]。

本章的目标不是介绍如何建立一个面向大规模Web搜索引擎的采集器，而是关注采集中的一系列一般性问题，这些问题不论是对学生项目还是实际研究项目的中的采集器来说都有普遍意义。本章一开始在20.1.1节中列出了Web采集器的所要支持的功能，然后在20.2节讨论如何实现这些功能。本章的剩余部分介绍了满足这些功能的分布式Web采集器的构架及实施细节。20.3节讨论了在Web级规模下如何在多台机器上实现分布式索引。

20.1.1 采集器必须提供的功能特点

下面我们将Web采集器的功能特点分成两类：一类是采集器所必须提供的功能特点，另一类是采集器应该提供的功能特点。

采集器必须要提供的功能特点包括两点。

- 鲁棒性：Web中有些服务器会制造采集器陷阱（spider traps），这些陷阱服务器实际上是Web页面的生成器，它能在某个域下生成无数网页，从而使采集器陷入到一个无限的采集循环中去。采集器必须要能从这类陷阱中跳出来。当然，这些陷阱倒不一定都是恶意的，有时可能是网站设计疏忽所导致的结果。
- 礼貌性：Web服务器具有一些隐式或显式的政策来控制采集器访问它们的频率。设计采集器时必须要遵守这些代表礼貌性的访问策略。

20.1.2 采集器应该提供的功能特点

① 其他一些名称还包括网络机器人(robot)、网络爬虫等等。——译者注

分布式：采集器应该可以在多机上分布式运行。

（规模）可扩展性：在增加额外的机器和带宽的情况下，采集器的架构应该允许实现采集率的提高。

- 性能和效率：采集器应该能够充分利用不同的系统资源，包括处理器、存储器和网络带宽等。
- 质量：在应答用户查询需求时，大部分 Web 网页的质量都很差，因此采集器应该优先考虑抓取“有用”的网页。
- 新鲜度：在很多应用中，采集器都处于连续工作状态，也就是说它应该要对原来抓取的网页进行更新。例如，只有这样做，一个搜索引擎才能保证其索引中包含索引网页的较新版本。对于这种连续式采集来说，采集器应该能够以接近网页更新的频率来采集网页。
- （功能）可扩展性：采集器的设计要能支持其在很多方面方便地进行功能扩展，比如可以处理新的数据格式、新的抓取协议等。这就要求采集器的架构要高度模块化。

20.2 采集

任何超文本采集器（不论是面向 Web、内网还是其他的超文本文档集）的基本处理如下：首先，设定一个或者多个 URL 为采集的种子集合（seed set）。接着，从种子集合中选择一个 URL 进行采集，然后对采集到的页面进行分析，并抽取出页面中的文本和链接（每个链接都链向其他的 URL）。抽取出的文本输给文本索引器（参见第 4 章和第 5 章的介绍），而抽取出的 URL 则加入到待采集 URL 池（URL frontier，以下简称 URL 池）中，任何时候 URL 池中放的都是所有待采集网页的 URL。一开始，种子集合会放入 URL 池中，一旦某个 URL 被采集，那么就从池中删除这个地址。整个采集过程可以看成是 Web 图的遍历过程。当然，在连续式采集中，一个已采集的网页的 URL 还会被重新放到 URL 池中以等待下一次重新采集。

由于实际系统的需求，这种看上去简单的 Web 图递归遍历过程变得非常复杂：在抓取高质量网页的同时，采集器要满足分布式、规模可扩展、高效、礼貌性、鲁棒性及功能可扩展等一系列要求。后面我们将考察上述每种问题的效果。以下的介绍主要遵循 Mercator 采集器的设计思路，它也是很多研究型和商业型采集器的基础。为给读者对采集速度有一个基本的印象，我们给出的一个有关采集器的参考数据是，要在一个月内抓取 10 亿网页（这么多网页只是目前静态 Web 的很小一部分）的话，大概需要在每秒之内抓取几百个网页。后面我们将会看到，如何利用多线程机制来解决整个采集系统的中的多个瓶颈，从而达到上述的采集率。

在讨论细节之前，对那些试图建立采集器的读者，我们要重申一点，即使是随便一个非专业的采集器也应该满足以下基本特性：

(1) 对任意主机每次仅仅应该开放一个连接；

(2) 在连续发送请求给某一个主机时，应该要在每两次请求之间等待几秒钟；

(3) 一定要遵守 20.2.1 节所介绍的礼貌性原则。

20.2.1　采集器架构

一个简单的采集器由多个模块构成，图 20-1 给出了一个示意图，其中包括五种模块。

(1) 待采集 URL 池：它包含了当前待采集的 URL（在连续式采集中，某个已经采集过的 URL 可能还会放回到该采集池中以便进行重新采集）。关于这一点，我们将在 20.2.3 节进一步讨论。

(2) DNS 解析模块：它在 URL 抓取网页时用于确定其对应的 Web 服务器的 IP 地址。

(3) 抓取（fetch）模块：利用 http 协议返回某个 URL 对应的网页。

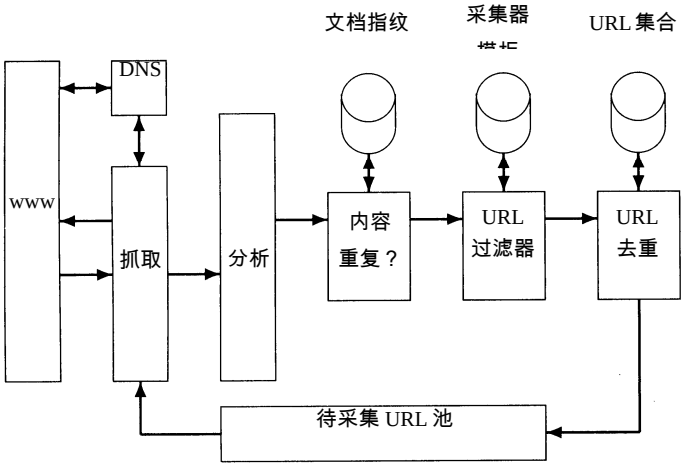


图 20-1　采集器的基本架构

(4) 分析（parse）模块：从采集到的网页中抽取文本及链接。

(5) URL 去重模块：确定某个抽取出的链接是否已在 URL 池中或者最近是否已抓取。

通过一个到数百个来自任何地方的线程来共同完成采集过程，其中每个线程会按照图 20-1 所示的流程反复循环。这些线程可以运行在单个进程中，或者分开到分布式系统的不同节点的多个进程当中。我们这里先假设 URL 池已经就位且非空，这样就可以暂时不对它进行介绍（其介绍参见 20.2.3 节）。下面，我们将详细介绍单个 URL 的采集流程，包括 URL 抓取、结果的各种检测和过滤等，最后将 URL 重新放入 URL 池中（对于连续式采集）。

一个采集线程首先从 URL 池中选择一个 URL，然后抓取该 URL 对应的网页，抓取过程通常是通过 http 协议来完成。抓取到的页面会被写入一个临时存储器中，等待它的是一系列处理。接着，该网页被分析，文本和链接都被抽取出来。文本（包含标签信息，如黑体词项）信息会传给索引器。链接信息以及锚文本也会传给索引器以用于排序过程，我们将在第 21 章中介绍该排序过程。另外，每个抽出的链接信息要经过一系列的检测来判断该链接是否要加入到 URL 池中。首先，采集线程会检查具有相同内容的另一个 URL 是否已经被采集。最简单的实施策略是

使用简单的指纹检测方法，比如校验和（在图 20-1 中放在“文档指纹”所标识的存储器中）。一种更复杂的方法是可以使用 shingle 而不是指纹，而关于 shingle 的内容已经在第 19 章做过介绍。

下一步，URL过滤器采用多个测试来确定某个抽取出的URL是否应该被URL池收录或排除。比如，采集过程可能要排除某些域（如所有.com域的URL），这种情况下，只需要将所有.com域的URL过滤掉即可。类似的检测过程也可以通过包含而不是排除的方式来实现。Web上的很多主机会在某些地方放置对本主机进行采集的限制条件，一种常规的实现措施就是采用拒绝蜘蛛协议（robots exclusion protocol）。它通过在网站服务器根目录下放置一个名为robots.txt的文件来实现。下面给出了一个robots.txt文件的例子，其中规定：除名称为“searchengine”之外的任何采集器都不能访问/yoursite/temp/ 这个目录下的文件[①]。

```
User-agent: *
Disallow: /yoursite/temp/

User-agent: searchengine
Disallow:
```

采集器必须从网站抓取其robots.txt文件，从而确定要采集的URL是否能够通过限制，这之后才能确定能否将其放入URL池中。而在对每个URL进行检测以确定它是否可以加入URL池的时候，并不需要重复抓取robots.txt文件。通常的做法是，将采集器最近访问过的主机的robots.txt文件放到一个缓存中。由于从某个网页中抽取的链接往往会和当前网页处于同一主机，从而对它们会在同一robots.txt文件上进行测试，所以采用缓存的做法特别重要。因此，在链接抽取过程中进行过滤操作时，在主机流中我们就会得到一个具有高度局部性的主机集合，它们都需要基于同一robots.txt文件进行过滤操作，这时缓存的利用率会很高。然而，这种做法可能与网站管理员所预期的礼貌性原则有所冲突[②]。一个URL，特别是低质量文档或者很少变化的文档的URL，可能会在URL池中放很多天甚至多个星期。如果在将这类URL放入URL池之前就已经对它进行过采集限制检测的话，那么在该URL出列被真正采集之时，相应的robots.txt文件可能已经改变。也就是说，原来满足采集限制的URL可能在进行真正采集时已经不满足新的采集限制。因此，我们必须在真正抓取网页之前进行采集限制检测。当然，事实证明，维持一个robots.txt文件的缓存仍然十分有效，即使是从URL队列中出列的URL流也存在着充分的局部性。

下一步，URL 必须进行如下意义下的规范化（normalization）处理。通常 HTML 页面 *p* 中的 URL 链接给出基于 *p* 的相对地址。因此，在下列 en.wikipedia.org/wiki/Main_Page 中的相对链接地址：

```
<a href="/wiki/Wikipedia:General_disclaimer"title="Wikipedia:General_disclaimer">
Disclaimers</a>
```

① 这个文本也表明，searchengine 采集器可以访问网站的任何目录。——译者注

② 也就是说，同一网站不希望在一段时间内被频繁访问。——译者注

实际上指向的是 URL http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer。

最后，需要对 URL 进行查重处理：如果某个 URL 已经在 URL 池中或者已经被采集（在非连续式采集的情况下），那么就不需要将它再放到 URL 池中。当某个 URL 加入到 URL 池时，它会被分配一个优先级，基于这个优先级来决定其最终出列真正进行采集的时机。有关这种优先级队列的细节将在 20.2.3 节讨论。

一般一些日常事务处理的工作会通过一个专门的线程来实现。除了每隔数秒被唤醒来记录采集过程的状态（包括已采集的 URL、URL 池的大小等）、确定是否需要终止采集或者对采集进行检查点检查等，一般情况下该线程都处于休眠状态。在检查点进行检查时，采集器状态（如 URL 池）会提交给磁盘保存。一旦碰到灾难性故障，采集会从最近的检查点进行恢复。

分布式采集器

前面我们提到过，采集器中的线程可以运行在分布式系统环境下不同节点的不同进程中。这种分布式架构对于规模的扩展非常重要，它也可以用于一个地理位置分布的采集系统中，其中每个节点对“就近”的主机进行采集。将要采集的主机分配到每个节点可以通过哈希函数或者一些更具体的针对性策略来完成。比如，地理位置在欧洲的采集器主要关注欧洲域名下网站的采集。当然，这种做法并不完全可靠，原因包括：Internet 上数据包的传输路线并不一定反映地理位置的邻近性。并且，在任何情况下，主机域名并不总是反映其实际的物理位置。

那么，这些不同节点之间如何互相通讯并共享 URL？其思路是在每个采集节点上复制一份图 20-1 所示的流程，但是与图 20-1 中的做法相比，这种做法有一点本质的不同，即在 URL 过滤之后，我们要使用一个主机划分器（host splitter）将通过过滤检测的 URL 分配到不同的采集节点上去。也就说，要采集的主机对象会被分配到不同节点进行采集。修改后的采集流程如图 20-2 所示。主机划分器的输出结果会输入到分布式系统每个采集节点的重复 URL 检测模块中去。

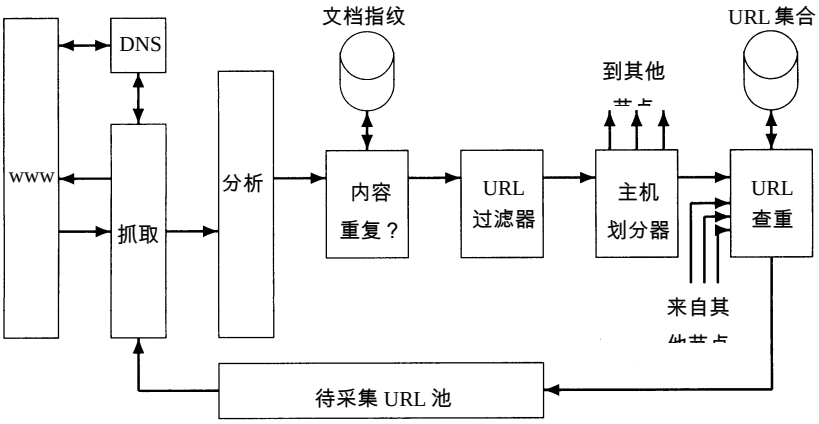


图 20-2　基本采集架构的分布式结果示意图

然而，图 20-2 所示的分布式采集架构中的内容重复检测模块会很复杂，其主要原因如下。

(1) 和 URL 池及 URL 查重模块不同的是，很难基于主机名划分文档的指纹或 shingle，也就是说无法防止相同或者高度相似的内容会出现在不同的 Web 服务器上。因此，必须要基于指纹或 shingle 集合的某些性质对它们进行节点划分，比如用指纹对节点的数目取模。这种本地失配造成的结果就是，大部分内容重复检测模块都会进行一个远程过程调用（尽管有可能会进行批量的查询请求）

(2) 文档的指纹流或 shingle 流当中几乎不存在局部性。因此，对常用的指纹进行缓存没什么作用（实际上也没有常用的指纹）。

(3) 文档会随时间不断变化，因此在连续式采集中，必须要从已下载内容集合中去掉过时的指纹或 shingle。为此，就必须要将文档的指纹或 shingle 连同 URL 本身一起放入 URL 池中。

20.2.2 DNS 解析

每个Web服务器（实际上每个连入Internet的主机）都有一个唯一的IP地址。每个IP地址是 4 个字节组成的序列，通常表示为通过点连接起来的 4 个整数。比如：207.142.131.248 就是主机www.wikipedia.org对应的IP地址。给定一个文本表示的URL（如www.wikipedia.org），将它转换成IP地址（这里就是 207.142.131.248）的过程被称为DNS解析（DNS resolution）或DNS查询（DNS lookup），这里DNS指的是域名服务（domain name service）。在DNS解析过程中，需要进行IP地址转换的程序（这里是采集器）会联系一个DNS服务器来返回IP地址。实际中，整个转换过程很可能不是单个DNS服务器完成的，初始调用的DNS服务器会不断递归调用其他的服务器来完成地址转换。对于一个更复杂的URL地址，比如 en.wikipedia.org/wiki/Domain_Name_System，采集器中负责DNS解析功能的模块就会从中抽取出主机名，这里就是 en.wikipedia.org，然后查找其对应的IP地址。

众所周知，DNS 解析在 Web 采集中是一个“ 瓶颈”。由于域名服务本身就是分布式的，所以 DNS 解析可能包括多个请求在 Internet 上的往返过程，这通常需要数秒甚至更多的时间。这样的话，就会给每秒获取数百网页的采集目标造成极大困难。一个常规的措施就是引入缓存机制。最近进行过 DNS 解析的 URL 可以放入一个 DNS 缓存中，这样新来的 URL 如果在缓存中命中的话，就不需要到 Internet 上去做域名解析。然而，遵循采集中的礼貌（参见 20.2.3 节）的要求往往又会限制缓存的命中率。

DNS 解析还存在另外一个严重的困难，采集器的开发者往往使用标准库（这个库可能被开发采集器的任何一个人使用）来实现 DNS 解析功能，但是这个库中的域名查找过程在实现上往往是同步的。这就意味着，一旦某个请求被提交给域名服务器，处于同一节点的其他采集线程就会在第一个请求完成之前被阻塞。为了避免这个问题，大部分 Web 采集器都会实现自己的 DNS 解析器。执行解析器代码的线程 *i* 会发送一条消息给 DNS 服务器，然后进入一个定时的等待过程，当收到其他线程的通知信号或者规定时间用完时该线程才恢复执行。我们使用一个单

独的 DNS 线程监听标准的 DNS 端口（端口号 53），以等待域名服务器的应答到来。一旦收到应答，它就会通知相应的采集线程（本例中为线程 i），并且如果此时线程 i 因等待时间用完而没有恢复执行其他操作时，便将相应数据包发送给它。由于等待时间用完而恢复执行其他操作，则需要重试固定的次数，每次将一个新消息发送给 DNS 服务器并执行定时等待过程。Mercator 的设计者建议尝试的次数为 5 的倍数，而每次等待的时间则随着尝试次数的增加而指数级增加。Mercator 一开始的等待时间是 1 秒，考虑有些主机名需要数十秒完成解析的事实，它结束时的等待时间大概需要 90 秒左右。

20.2.3　待采集 URL 池

在每个节点上，采集进程或其他采集进程的主机分割器会将 URL 放入本节点的 URL 池中。该采集池会维护一系列 URL，并在采集线程需要寻找 URL 时，以某种次序将 URL 输出。采集池中 URL 的输出次序必须要考虑到两个重要的方面：第一，频繁改变的高质量网页应该优先考虑频繁采集。因此，网页的优先级应该是其变化率和质量（可以采用某些合理的质量估计方法）的函数。由于大量作弊网页在每次抓取时几乎完全改变，所以同时考虑变化率和质量这两者是十分必要的。

第二个要考虑的是礼貌问题。我们必须避免在很短的时间间隔内反复访问同一主机。由于很多URL会链向同一主机的其他URL，因此会产生互相引用的局部效应，因此，如果不进行控制，那么在很短时间内访问同一主机的可能性很大。所以，如果URL池的实现中只使用简单的优先级队列，就会造成对某个主机的突发性高频访问。甚至即使在我们限制在任何时刻最多只有一个采集线程访问某个主机的情况下，上述突发高频访问仍然有可能发生[①]。一个普遍使用的启发式策略是，在对某个主机发送连续的两次抓取请求之间插入一个时间间隔，它要比最近一次从该主机抓取网页所需的时间高一个数量级。

图 20-3 给出了 URL 池的一个实现的示意图，它支持优先级处理并遵循礼貌性访问原则。其目标是为了保证：(i) 在任一时刻只有一个连接对主机开放；(ii) 在连续两次主机请求之间，需要等待数秒；(iii) 高优先级网页优先采集。

① 每次只有一个访问，但是两次访问之间的间隔太短。——译者注

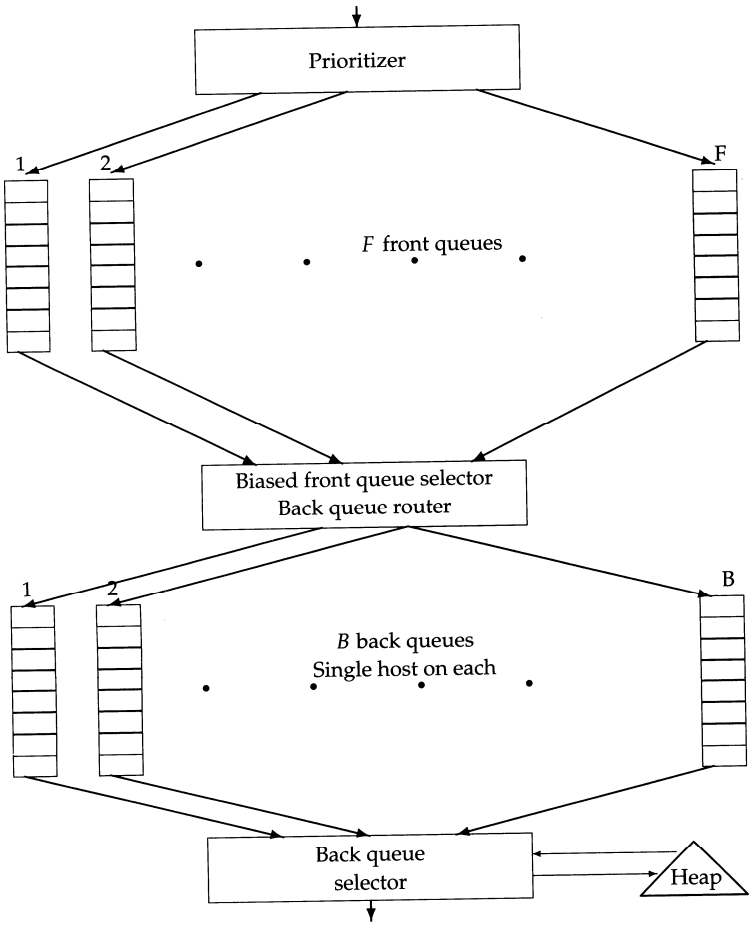


图 20-3 URL 池示意图。从已采集网页中抽取的 URL 从图上部流入。一个请求 URL 的采集线程从图底部抽取一个 URL。途中，URL 依次流过管理采集优先级的某个前端队列及管理采集器礼貌性的一个后端队列

图中的两个主要子模块分别是：上图中的 F 个前端队列（front queues）集合，以及下图中的 B 个后端队列（back queue）集合。并且，这两个队列都满足先进先出的原则。 前端队列主要是实现优先级访问功能，而后端队列实现礼貌性访问功能。在 URL 加入到采集池的流程中，会在前端队列和后端队列中走出一条通路。首先，优先级分配器（prioritizer）会基于 URL 的抓取历史（考虑在以往的采集中本 URL 对应的 Web 网页的变化率）赋给该 URL 一个整数 i 表示其优先级， 其中 i 的取值在 1 到 F 之间。比如，给一篇变化更频繁的文档分配更高的优先级。其他一些定义优先级的启发式策略依赖于具体且明确的应用。比如，来自新闻社的 URL 可能总是被赋予最高优先级。现在，一个 URL 已经被赋予优先级 i，这个 URL 就会添加到第 i 个前端队列中。

B个后端队列中的每个队列维持下列固定情况：(i) 当采集正在进行时，队列不会为空；(ii)

队列只包含来自单个主机的URL[①]。使用一个辅助表T（如图 20-4 所示）来维护从主机到后端队列的映射。当某个后端队列为空并从前端队列重新填充时，T必须进行相应的更新。

主机	后端队列
stanford.edu	23
microsoft.com	47
acm.org	12

图 20-4　一个主机到后端队列的辅助映射表

此外，我们还维护一个堆结构，其中的每个元素对应一个后端队列，元素值为该队列对应的主机重新访问的最早时间 t_e。

某个采集线程在请求 URL 池的一个 URL 时，会从上述堆中取出其根节点，并且等待相应的时间 t_e。然后，从根节点对应的后端队列 j 中取出队列首部的 URL u，并执行 u 的抓取操作。采集 u 之后，调用线程会检查 j 是否为空。如果为空，则选择一个前端队列并取出该队列的首部 URL v。在选择前端队列时会倾向于高优先级队列（通常有一个随机过程来实现），即保证高优先级 URL 能够更快地流入到后端队列中。对于 URL v，我们会检查在某个后端队列中是否已经包含了来自同一主机的 URL。如果存在，那么 v 就会加入到该队列中，这样我们就需要重新回到前端队列来寻找另外一个候选 URL 插入到现在为空的队列 j 中。该过程不断继续直到 j 不再为空。任何情况下，对队列 j，线程都会基于其中上次采集的 URL 的属性在堆中插入一个新的最早访问时间 t_e（比如上次访问主机的时间及上次抓取所花的时间），然后继续进行处理。例如，新的时间 t_e 可以是上次抓取时间的十倍加上当前时间。

前端队列的数目、优先级分配及队列选择策略决定了我们要在系统中建立的优先级的性质。后端队列的数目控制了在保持礼貌性的同时所有采集线程的忙碌程度。Mercator 的设计者给出了一个大致的建议，即后端队列的数目大概是采集线程个数的三倍。

在一个 Web 规模的采集器中，URL 池需求的内存可能会超过节点本身的可用内存大小。其解决方法是将大部分待采集 URL 驻留在磁盘上。每个队列的一部分保留在内存中，当其用完时再从磁盘将其他数据调入内存。

习题 **20-1**　在分布式采集系统中，为什么按照主机进行划分会比按照每个 URL 进行划分要好？

习题 **20-2**　为什么要在 URL 查重之前完成主机划分？

习题 **20-3 [***]**　在前面的讨论中，我们遇到两个推荐使用的“硬常数”（hard constant），一个是 t_e，它的增加量是上次抓取时间的 10 倍，另一个常数要求后端队列的数目是采集线程的 3 倍，这两个常数有什么关联？

① 假定主机的数目远远超过 B。

20.3 分布式索引

在 4.4 节中，我们介绍了分布式的索引构建过程。这里，我们考虑将索引分布到一个支持查询处理的大规模计算机集群[①]中去。显然存在两种索引实施方法：一种被称为基于词项的划分（partitioning by terms）方法，也被称为全局的索引组织方法。另一种被称为基于文档的划分（partitioning by documents）方法，也被称为局部的索引组织方法。前者将词项词典划分成多个子集，每个子集驻留在一个节点上。对该节点上的这些词项，我们保留其倒排记录表。处理某个查询时，会将查询导向那些包括含查询词项的节点。理论上说，由于包含不同查询词项的查询流会命中不同的机器集合，所以这种做法能够允许更高的并发度。

实际上，按照词项来划分索引并非易事。多词查询处理时需要在节点之间传输长倒排记录表以用于合并。这种处理开销可能会超过高并发度所带来的好处。划分结果的负载均衡不受控于相对词频的先验分析，而受控于查询词项及其共现的分布情况，而后者甚至可能会随时间变化而出现突发情况。好的划分是查询词项共现的一个函数，需要对词项进行聚类来优化一个不易定量的目标。最后，这种划分策略会使动态索引的实现更加困难。

一个更为普遍使用的方法是按照文档划分：每个节点包含某个文档子集的索引。每个查询都会被分发到所有节点上，来自不同节点的结果在呈现给用户之前会进行合并。该策略在减少节点之间通讯量的同时需要更多的本地磁盘访问次数。这种实施策略的一大难点是，尽管每个节点上只有部分文档子集，但是评分中的全局统计信息（如 idf）必须要基于全体文档集合进行计算。一些分布式“后台”进程会计算出这些统计信息，并利用它们定期刷新每个节点上的索引信息。

如何将文档划分到节点上去？基于 20.2.1 节所开发的采集器架构，一种简单的方法是将某个主机上的网页分配到一个节点上。这种划分和网页采集时的划分是一致的。但是，这种划分方法带来的危险就是，对很多查询来说，大部分结果可能只集中来自某一部分主机，也就是说只来自于一部分索引节点，这样会使得访问的均衡性很差。

另一种做法是将每个 URL 哈希到索引节点空间，这样会产生一个在不同节点间的更均匀的查询处理时间分布。查询时，查询会广播到所有节点，每个节点上会返回排名最高的 k 个结果，然后将这些结果合并并找出最终的前 k 个结果。一种普遍实现的启发式策略是：将整个文档集划分成对大多数查询都可能会得高分的文档子集（比如，可以采用第 21 章的技术）以及其他的低分文档子集。只有在高分文档子集中匹配过少时，才会搜索低分文档子集。关于这一点在 7.2.1 节中曾有所讨论。

① 请注意本书中 cluster 的不同用法，参考第 16 章和第 17 章。

20.4　连接服务器

由于某些原因（这些原因会在下一章中详细介绍），Web 搜索引擎需要一个连接服务器（connectivity server）来支持 Web 图连接查询（connectivity query）的快速处理。典型的连接查询包括“给定的 URL 被哪些 URL 所指向？”及“给定 URL 指向了哪些 URL？”等。为此，我们在内存中存储了从 URL 到出链及 URL 到入链的映射表。这个表可以具体应用在采集控制、Web 图分析、高精度的采集优化及下一章将要介绍的链接分析（link analysis）当中。

假定整个 Web 包含 40 亿网页，每个网页有 10 个链接指向其他网页。在最简单的形式下，对每个链接的首尾两端（链接源和链接目标），我们分别采用 32 比特位或者说 4 个字节来描述，于是总共需要

$$4 \times 10^9 \times 10 \times 8 = 3.2 \times 10^{11}$$

字节的内存。我们可以利用 Web 图的一些特性将上述内存的需求压缩到 10%以下。乍一看，我们面对的似乎是一个可以采用很多标准解决方案的数据压缩问题。但是，我们的目标不仅仅是将 Web 图压缩到内存中，而且要支持连接查询的高效处理。这种挑战会使我们回想起第 5 章中讨论的索引压缩。

假定每个网页都用唯一的整数来表示，具体的整数生成机制将会在下面介绍。我们建立一个类似于倒排索引的邻接表（adjacency table），其每行都对应一个网页，并按照其对应的整数大小来排序。任一网页 p 对应的行中包含的也是一系列整数的排序结果，每个整数对应的是链向 p 的网页编号。这张邻接表允许我们应答类似于“哪些网页指向 p？”（“which pages link to p?”）的查询。以同样的方法，可以建立所有 p 所指向的网页的邻接表。

原始的表示方法中，均采用 32 比特位整数来表示每个链接的源页面和目标页面。而上述这种邻接表的表示方式能够将原始表示的空间降低 50%。下面的介绍主要关注从网页中链出的链接组成的邻接表，很显然这些技术很容易应用到链入网页的邻接表上。为了进一步减少上表的存储空间，可以采用如下几种思路。

(1) 表中的相似度：表格中的很多行有很多公共元素。因此，如果我们将多个相似行表示成一个行原型（prototype），那么其他相似行就可以采用这个原型来简洁表示。

(2) 局部性：某个网页会链接到其相邻的网页，比如链接到同一主机的网页。这意味着，如果对链接目标进行编码时，往往可以通过使用小整数来达到节省空间的目的。

(3) 在排序表中使用间隔编码：我们不直接存储链接目标的编号，而是存储其与前一个元素的偏移。

下面，我们将详细讨论每一种技术。

在按照词典对所有URL排序时，我们是将每个URL看成一个包含字母数字的字符串并对它们进行排序。图 20-5 给出了这样一种排序的片段。在Web网页的这种实际排序当中，URL的主机名部分应该反过来，因此 www.stanford.edu 变成 edu.stanford.www，但是这样做也不是必要的，

因为我们只关注单个主机的局部链接。

```
1: www.stanford.edu/alchemy
2: www.stanford.edu/biology
3: www.stanford.edu/biology/plant
4: www.stanford.edu/biology/plant/copyright
5: www.stanford.edu/biology/plant/people
6: www.stanford.edu/chemistry
```

图 20-5 URL 的一个词典排序片段

对每个 URL，我们将其在上述排序中的位置设为其唯一编码。图 20-6 给出了这样编码之后最终表格的示例。在本例中，由于 www.stanford.edu/biology 在词典序中排第二，所以其最终的编码为 2。

下面我们将利用大部分网站结构化的一些特点来获得相似性和局部性。大部分网站都有这样一个特点，即网站中每个网页都有一系列链接指向网站中的一些固定页面（比如版权声明、使用条件等等）。这种情况下，该网站在表格中的网页所对应的行之间就有很多公共元素。此外，在 URL 基于词典排序时，很可能来自同一网站的网页在表格内也会处于连续的行中。

于是，我们就可以采用如下策略：从上到下遍历表格，对每一行基于前面的 7 行来编码。在图 20-6 所示的例子中，可以将第 4 行编码为“除了将 9 替换成 8 之外，同偏移为 2 的行（即表格中往前 2 行的那行）完全一样”。这里需要指定偏移、给出删除的整数（本例中为 9）和增加的整数（本例中为 8)。这里只使用前 7 行进行编码有如下两个优点：(i) 偏移可以通过 3 个比特位来表示，这种选择在经验上最优（为什么用前 7 行而不是前 8 行是习题 20-4 关注的问题）；(ii) 将最大的偏移固定在一个较小的值（如上面的 7）能够减少搜索原型花费的时间。

随之而来的问题就是，如果在前 7 行中都找不到本行的原型怎么办？这种现象有可能发生，比如在遍历到不同网站的边界时。这种情况下，我们就简单地将本行表示成从一个空集开始并不断加入本行所有整数的过程。我们可以使用间隔编码，即不使用原始整数而是整数之间的间隔来编码，由于文档之间的间隔可能比较紧，所以采用间隔编码可以进一步减少存储空间。在 20.5 节提到的实验中，运用上述技术看上去可以将每个链接的表示减少到平均 3 个比特位，这和原始的 64 个比特位相比，无疑是一个巨大的进步。

尽管上述技术可以将大规模的 Web 图在一个较小的可以放在内存的空间内进行表示，我们仍然必须要支持连接查询。在上述表示下，需要做哪些工作才能返回某个页面指出的链接呢？首先，我们需要一个将 URL（哈希）映射到其行号的索引。其次，我们需要对所有元素重构，即按照其他行来对每行进行编码。这需要按照偏移来对这些行进行重构，这个过程理论上可能需要多级间接处理来完成。然而，这实际上并不常见。在表格的创建中可以引一种控制这种现象发生的启发式策略：当考察前 7 行来确定哪行是当前行的原型时，我们需要引入一个当前行和候选行的相似度阈值。必须精心选择这个阈值。如果阈值设置太高，那么就很少会使用原型

来表示本行，此时，每行就需要重新表示。如果阈值设置太低，那么大部分行都通过原型来表示，因此，在查询处理时，这种行构造方法就会导致基于原型的多级间接处理。

? 习题 **20-4**　我们注意到，如果用前 7 行来表示当前行，就能够在不超过 3 个比特位的空间来指定原型所在的行。这里为什么是前 7 行而不是前 8 行？（提示：考虑前 7 行都不是好的原型的情况）

习题 **20-5**　我们注意到，采用 20.4 节的机制，对某个 URL 相关的链接进行解码需要多级间接处理。请构造一个例子，在该例子中，间接处理的级数会随 URL 的数目而线性增长。

20.5　参考文献及补充读物

最早的Web采集器似乎是Matthew Gray写于 1993 年春天的Wanderer。Mercator采集器的工作归功于Najork 和 Heydon（Najork 和 Heydon 2001, 2002），本章中主要参考了Mercator的相关工作。其他有关早期Web采集的经典工作介绍包括Burner (1997)、Brin 和 Page (1998)、Cho 等人 (1998)及Stanford建立的Webbase系统（Hirai 等人 2000）。Cho 和 Garcia-Molina (2002) 给出了分布式采集器的分类体系，并对分布式环境下不同的通讯方式进行了比较研究。拒绝蜘蛛协议标准在www.robotstxt.org/wc/exclusion.html中有所描述。Boldi 等人 (2002) 和 Shkapenyuk 和 Suel（2002）给出了大规模分布式Web采集器实施的最新细节。

本章中有关DNS解析的讨论主要基于当前的因特网地址规范，即IPv4（因特网协议版本 4 的简称）规范，其中每个IP用 4 个字节来表示。未来地址的规范（统称为互联网地址空间）可能会使用一种新的被称为IPv6 的标准（www.ipv6.org/）。

Tomasic 和 Garcia-Molina（1993）以及 Jeong 和 Omiecinski（1995）是早期的评估分布式索引中词项划分和文档划分两种方式的重要论文。他们的工作表明，文档划分方式更优越，至少在词项分布不均衡的情况下会是如此，然而这种不均衡情况在实际中很常见。这个实验结果也在近来的一些工作中得到了验证（MacFarlane 等人 2000）。当然，实验结果依赖于分布式系统的细节，至少有一些工作得到了与上述结果相反的结论（Ribeiro-Neto 和 Barbosa 1998; Badue 等人 2001）。Sornil（2001）主张将两种划分方法混合起来使用。Barroso 等人（2003）介绍了 Google 中所使用的分布式方法。连接服务器的首次实施方法出现在 Bharat 等人（1998）。本章所讨论的链接编码机制，也被认为是目前所发表的最好机制（对每个链接只需要 3 个比特位进行编码），关于这些机制的介绍可以参考 Boldi 和 Vigna 的一系列文章：Boldi 和 Vigna（2004a, 2004b）。

第 21 章

链接分析

超链接分析（也称链接分析或超链分析）和 Web 图结构信息已经在 Web 搜索的开发中发挥了重要作用。本章主要关注链接结构信息在 Web 搜索结果排序中的使用。在给定查询下，链接分析结果已经成为 Web 搜索引擎在计算某个网页的组合得分中的一个因子。我们首先在 21.1 节先回顾一下 Web 图的基本属性，然后介绍用于排名的链接分析方法的各个要素。

Web 搜索中链接分析思想的最早起源于引文分析领域，后者在很多方面与一个被称为文献计量学（bibliometrics）的领域有交叉。这些学科试图通过分析文献之间的引用模式来量化学术论文的影响力。文献引用代表某篇学术论文对所引用论文的权威度的认可，类似地，Web 上的链接分析方法也把超链接看成是一个网页对另一个网页的权威度的认可。很显然，并不是所有的引用或超链接都代表这种对权威度的认可，因此，仅仅简单地通过入链接的数目来衡量网页的质量是不够鲁棒的。比如，某个人可以建立多个 Web 网页来指向同一个目标网页，这样就可以通过人工手段有意地提高该目标页面的入链数目。这种现象通常被称为垃圾链接或者链接作弊（*link spam*）。尽管如此，引用现象的普遍性和可靠性已经足以使搜索引擎通过精妙的链接分析方法推导出有用的排序因子。链接分析也被证明是在 Web 采集中选择下一个采集网页的非常好的方法之一，具体实现时可以使用链接分析来指导第 20 章中前端队列中的优先级分配过程。

21.1 节给出了链接分析当中 Web 图使用的一些基本思路。而 21.2 节和 21.3 节则给出了两种具体的链接分析方法：PageRank 和 HITS。

21.1 Web 图

我们回顾一下 19.2.1 节中提到的 Web 图的概念，特别是图 19-2 所示的内容。链接分析的研究主要基于两个基本直觉。

(1) 指向页面 B 的锚文本是对 B 的一个很好的描述。

(2) A 到 B 的超链接表示 A 的作者对 B 的认可。当然，并非所有情况都会如此，比如，某个网站的网页中的很多链接源于通用模板的使用。例如，大部分公司网站的每个网页都有一个链接指向版权声明页面。这种链接显然不代表认可的意义。因此，链接分析算法在实施过程中通常会去掉这些“内部”的链接。

21.1.1 锚文本和 Web 图

下面来自某个网页的HTML代码片段给出了一个指向期刊Journal of the ACM的链接：

```
<a href="http://www.acm.org/jacm/">Journal of the ACM.</a>
```

这个例子中，链接指向页面www.acm.org/jacm/，其锚文本为Journal of the ACM。显然，在这个例子中锚文本是对目标页面的文字描述，但是目标网页（B=http://www.acm.org/jacm/）本身除了其他有关期刊的信息外也包含了这段文字描述。那么，锚文本到底起什么作用呢？

Web上随处可见的一个现象是，很多网页（如上面的目标网页B）的内容并不包含对自身的精确描述。很多情况下，问题主要是出在网页的设计者对网页内容的选择。这个问题对于公司网页来说更加普遍，因为它们往往是用做商业宣传而不是介绍公司内容。比如，在撰写这本书之际，尽管IBM被普遍认为是世界上最大的计算机制造商，但是其公司的主页(www.ibm.com) HTML代码的任何地方都不包含词项computer。类似地，Yahoo! (www.yahoo.com)主页的HTML代码中也不包含单词portal。

因此，Web网页本身携带的词项和用户用于描述同一网页的词项之间往往存在着一定的差异。因此，Web搜索者不一定要使用网页中的词项来对网页进行查询。另外，很多Web网页中的图形和图像十分丰富，并且（或者）在图像中嵌入了文字。这种情况下，采集时进行的HTML分析就无法抽出文本来构建网页索引。这个问题的标准IR解决方法会用到在第9章和12.4节所介绍的技术。那么在这里上述解决方法可以为锚文本所取代，通过它就可以聚集多个Web网页作者的集体力量。

很多指向www.ibm.com的链接上的锚文本都包含单词computer，这个事实就可以为Web搜索引擎所使用。比如，锚文本中的词项就可以作为索引目标网页的词项。因此，词项computer的倒排记录表中就会包含文档www.ibm.com，而词项portal的倒排记录表也同样会包含文档www.yahoo.com。这时通过一个特别的指示器来表示这些词项出现在锚文本中而不是页内文本中。同页内词项一样，通常也会基于词频来计算锚文本词项的权重。那些在多个锚文本中高频出现的词项（如Web锚文本中最普遍的词项是Click和here）会受到惩罚，这与idf的思想非常类似。可以通过15.4.1节中基于机器学习的评分方法来给出词项的实际权重，当前的Web搜索引擎看来给锚文本词项分配了一个较大的权重。

锚文本的使用会产生一些有趣的副作用。在大部分Web搜索引擎中来搜索big blue时，IBM公司的主页都会出现在排名靠前的结果中，这和很多人提到IBM时所常常采用的绰号是一致的。另外，网上已有并会持续存在这样的实例：当用类似evil empire的词项在Web搜索引擎中搜索时，这些贬义的锚文本往往会导致意料之外的结果。这种现象能够在针对某些特定网站进行的精心策划活动中看到。这种刻意策划的锚文本可能是一种作弊形式，某个网站可以通过构造具有误导性的锚文本来指向自己，从而提高在某些查询词项上的排名。检测并和这种对锚文本的滥用进行对抗是Web搜索引擎所从事的另外一种作弊检测工作。

锚文本周围窗口中的文本（有时被称为扩充的锚文本（extended anchor text））常常也可以当成锚文本一样来使用。这样的一个Web文本片段的例子是there is good discussion of vedic

scripture <a>here。很多场合下都考虑了这种扩充的锚文本，并且有很多学者还对有效窗口的大小进行了研究。有关这一点，请参考 21.4 节提到的参考文献。

? 习题 **21-1**　在 Web 图中，能否从任一节点出发总有可能沿着有向边（超链）到达任一其他节点？为什么？

习题 **21-2**　在 Web 上寻找一个具有误导性的锚文本例子。

习题 **21-3**　对某个 Web 网页 x，假设给定了指向它的所有锚文本短语组成的集合，请给出某种启发式方法来从这个集合中选出针对 x 的最具描述性的词项或短语。

习题 **21-4**　上述习题中的启发式方法中是否考虑到同一域名 D 下的多个网页都指向 x 并反复使用同一锚文本进行描述的现象？

21.2　PageRank

现在我们集中关注仅仅基于链接结构的评分和排序方法。链接分析的第一种技术是对 Web 图中的每个节点赋一个 0 到 1 之间的分值，这个分值被称为 PageRank。节点的 PageRank 值依赖于 Web 图的链接结构。给定查询，Web 搜索引擎会融合数以百计的特征来得到最后的综合得分，这些特征包括余弦相似度（参考 6.3 节）、词项邻近度（7.2.2 节）及 PageRank 等。我们往往利用 15.4.1 节介绍的方法来实现这种综合得分，根据它可以提供最后的结果排名。

考虑一个网上的随机冲浪者，它从某个网页（即 Web 图中的一个节点）出发，在 Web 中进行如下随机游走过程：在每一步中，冲浪者都会从当前网页 A 的链出网页中随机选出一个网页作为下一步的访问目标。图 21-1 给出了某个冲浪者处于节点 A 的例子，其中 A 有三个链接分别指向 B、C、D，那么下一步他将以 1/3 的等概率分别访问这三个节点。

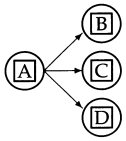


图 21-1　位于节点 A 的随机冲浪者将以 1/3 的等概率分别访问 B、C 和 D

当冲浪者在 Web 上进行节点间的随机游走时，他对某些节点的访问次数会比其他节点更多。直观地看，这些访问频繁的节点具有很多从其他频繁访问节点中指向的入链接。PageRank 的思路就是，在随机游走过程中访问越频繁的网页也越重要。

有时候冲浪者所在的当前网页A可能不存在出链，这时应该怎么做？为了解决这个问题，我们为冲浪者引入一个被称为随机跳转（teleport）[①]的额外操作。基于这个操作，冲浪者可以从一个节点跳到Web图的任一其他节点。由于他有可能在浏览器中输入一个URL，所以上述操作是有可能发生的。随机跳转操作的目标节点可以从满足均匀分布的所有Web网页中随机选择

① teleport 原意是指远距离传送，这里的意思是冲浪者不沿着链接跳转到其邻近网页，而是直接跳到整个网页空间中的网页之一（也可能包括它自身）。从这个意义上说，我们将它译成“不基于链接的随机跳转”，简称“随机跳转”。——译者注

来实现。换句话说，假如Web图中所有的节点数目是N，那么随机跳转操作使得冲浪者以$1/N$的概率跳到每个节点。当然，冲浪者也以$1/N$的概率跳到其当前位置。

给 Web 图中的每个节点分配 PageRank 值时，有两种使用随机跳转操作的方法：(i) 当节点没有出链接时，冲浪者调用随机跳转操作；(ii) 当节点包含出链接时，冲浪者将以 $0 < \alpha < 1$ 的概率调用随机跳转操作，而以 $1-\alpha$ 的概率继续进行随机游走(从满足均匀分布的出链接中随机选择一个出链前行)。其中 α 是一个事先选定的固定参数，一个典型的 α 取值为 0.1。

在 21.2.1 节中，我们将采用马尔科夫链理论来说明，当冲浪者采用这种混合过程（随机游走加上随机跳转操作）时，他就会以一个固定的时间比例 $\pi(v)$访问每个节点 v，其中 $\pi(v)$依赖于 (i) Web 图的结构；(ii) α 的值。 我们称 $\pi(v)$为 v 的 PageRank，并将在 21.2.2 节中给出 PageRank 的计算方法。

21.2.1 马尔科夫链

马尔科夫链是一个离散时间随机过程（discrete-time stochastic process），这个过程中的每一步都需要做一个随机选择。一个马尔科夫链包括 N 个状态（state）。在下面的介绍中，每个 Web 网页都对应我们所构造的马尔科夫链中的一个状态。

马尔科夫链通过一个 $N \times N$ 的转移概率矩阵（transition probability matrix）P 来刻画，其中每个元素的值在[0,1]之间，并且 P 中每一行的元素之和为 1。在任一步，马尔科夫链都可能处于 N 个状态之一，那么，元素 P_{ij} 给出的就是从当前状态 i 到下一个状态 j 的条件转移概率。P_{ij} 被称为转移概率，它仅仅依赖于当前的状态 i，这种性质被称为马尔科夫性。因此，基于马尔科夫性，我们有

$$\forall i, j, P_{ij} \in [0,1]$$

和

$$\forall i, \sum_{j=1}^{N} P_{ij} = 1 。 \qquad (21\text{-}1)$$

满足上述性质的非负矩阵被称为随机矩阵(stochastic matrix)。随机矩阵的一个重要性质是，它的最大特征值是 1，与该特征值对应的有一个主左特征向量（principal left eigenvector）。

马尔科夫链中，下一个状态的分布仅仅依赖于当前的状态，而和如何到达当前状态无关。图 21-2 给出了包含 3 个状态的简单马尔科夫链。从中间的状态 A 出发，可以分别以等概率 0.5 到达 B 或 C。而从 B 或 C 出发，都会以概率 1 到达 A。该马尔科夫链的转移概率矩阵为：

$$\begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

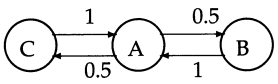


图 21-2 一个包含 3 个状态的简单马尔科夫链，链接上的数字代表的是转移概率

马尔科夫链的状态概率分布可以看成一个概率向量（probability vector），其中的每个元素都在[0,1]之间，并且所有的元素之和为 1。如果一个 N 维的概率向量的每个分量对应马尔科夫链中的一个状态的话，那么该向量就可以被看成是在状态上的一个概率分布。在 21-2 给出的简单例子中，概率向量包含 3 个总和为 1 的元素。

我们可以将Web图上的一个随机冲浪过程看成是马尔科夫链，其中马尔科夫链中的每个状态对应一个网页，而每个转移概率代表从一个网页跳转到另外一个网页的概率。teleport操作对这些转移概率的计算具有贡献。Web图的邻接矩阵A可以如下定义：如果存在网页i到网页j的一条链接，那么A_{ij}=1，否则 A_{ij}=0。这样，我们很容易就可以从N×N的矩阵A推导出马尔科夫链的转移概率矩阵P。如果A的某一行没有 1，则用 1/N代替每个元素[①]。对于其他行的处理如下[②]：

(1) 用每行中的 1 的个数去除每个 1，因此如果某行有 3 个 1，则每个 1 用 1/3 代替；

(2) 上面处理后的结果矩阵乘以 $1-\alpha$；

(3) 对上面得到的矩阵中的每个元素都加上 α/N。

这样，我们就可以通过概率向量 $\vec{x}$ 给出冲浪者在任一时间所处位置的概率分布。当 t=0 时，冲浪者可能处于某个初始状态，这时 $\vec{x}$ 中相应的元素为 1，其他元素均为 0。根据上述定义，在 t=1 时，冲浪者的状态分布可以用概率向量 $\vec{x}P$ 来表示。同样，t=2 时概率向量为 $(\vec{x}P)P = \vec{x}P^2$，可以依此一直类推下去。在 21.1.2 节中我们将详细讨论这个过程。这样我们只需要知道初始状态分布和转移概率矩阵 P，就能计算冲浪者在任一时刻所处状态的概率分布。

如果马尔科夫链被允许运行很多次，那么每个状态将会以不同的频率被访问，这个频率依赖于马尔科夫链的结构。在我们的运行模拟中，冲浪者访问某些网页（比如流行的新闻主页）会比其他网页更频繁。下面，我们将这种直观精确化，并建立访问频率收敛于固定的、稳态量的条件。然后，我们将 PageRank 设置为每个节点 v 在稳态下的访问频率，并介绍它的计算过程。

定义　一个马尔科夫链，如果存在一个正整数 T_0 使得对其中所有的状态对 i、j 都满足：若 i 是初始状态，那么对所有的 $t>T_0$，在时刻 t 处于状态 j 的概率都大于 0。此时，称该马尔科夫链是遍历马尔科夫链（ergodic Markov chain）。

一个马尔科夫链要满足遍历性，它的状态及其非零转移概率必须要满足两个专门条件，它们分别被称为不可约性（irreducibility）及非周期性（aperiodicity）。非正式地说，第一个条件保证从任意两个状态之间都存在非零概率转移序列，而第二个条件保证不存在这样的状态划分：所有的状态转移只发生在两个划分后的状态子集之间并循环往复。

定理 21-1　对任一遍历马尔科夫链，都存在一个唯一的稳态概率向量 $\vec{\pi}$，它是矩阵 P 的主

① 此时 A 没有出链接，给每个元素赋值 $1/N$ 实际上相当于给 A 到所有节点（包括 A 自身）之间增加了一条虚拟链接。——译者注

② 译者注：这样处理以后得到的矩阵 P 中的每个元素(i,j)即代表从 i 到 j 的转移概率。如果 i 没有出链接，则第 i 行的每个概率为$(1-\alpha)/N + \alpha/N=1/N$。其他情况下该概率由两部分线性求和得到。——译者注

左特征向量，并且如果 $\eta(i,t)$是在 t 步之内状态 i 的访问次数，那么有

$$\lim_{t\to\infty}\frac{\eta(i,t)}{t}=\pi(i)\text{。}$$

其中，$\pi(i)>0$ 是状态 i 的稳态概率。

按照定理 21-1，带随机跳转操作的随机游走过程最后会得到其对应的马尔科夫链中状态的一个唯一的稳态概率分布。某个状态的稳态概率就是相应网页的 PageRank。

21.1.2 PageRank 的计算

如何计算 PageRank 值？回顾公式（18-2）的左特征向量的定义，转移概率矩阵 P 的 N 维左特征向量 $\vec{\pi}$ 满足：

$$\vec{\pi}P=\lambda\vec{\pi}\text{。}\qquad(21\text{-}2)$$

主特征向量 $\vec{\pi}$ 是带随机跳转操作的随机游走过程的稳态概率，因此也就是所有 Web 网页的 PageRank 值。对于公式（21-2）可以采用如下解释：如果 $\vec{\pi}$ 是冲浪者在网页间的概率分布，那么他会保持在这个稳态概率分布 $\vec{\pi}$ 中。给定稳态概率分布为 $\vec{\pi}$ 的情况下，有 $\pi P=1\pi$，于是 1 是 P 的一个特征值。所以，如果我们计算出对应于矩阵 P 的特征值 1 的主左特征向量的话，那么就计算出了 PageRank 的值。

有很多计算左特征向量的方法，第 18 章末尾的参考文献及本章都可以作为这个问题求解的指南。这里我们只给出一个非常基本的方法，有时这个方法被称为幂迭代法（power iteration)。如果 $\vec{x}$ 是初始状态分布向量，那么时刻t的状态分布为 $\vec{x}P^t$。当t不断增大时，我们期望 $\vec{x}P^t$ 和 $\vec{x}P^{t+1}$ 非常相近[①]。当t很大时，我们期望马尔科夫链达到了它的稳态。根据定理 21-1，最后的稳态分布于初始状态分布 $\vec{x}$ 无关。幂迭代方法模拟了冲浪者的游走过程：从某个状态出发，游走一个很大的步数t，并跟踪对每个状态的访问频率。在经过一个很大步数t之后，这些频率会稳定下来，此时计算得到的访问频率的变化值会低于某个预先定义的阈值。这些访问频率[②]就是我们要计算的PageRank值。

考虑习题 21-6 中的 Web 图，其中 α=0.5。于是，冲浪者带随机跳转操作的随机游走过程的转移概率矩阵为：

$$P=\begin{pmatrix}1/6 & 2/3 & 1/6\\ 5/12 & 1/6 & 5/12\\ 1/6 & 2/3 & 1/6\end{pmatrix}\text{。}\qquad(21\text{-}3)$$

设想冲浪者的初始状态为 1，对应的初始状态概率分布向量为 $\vec{x}_0=(1\ 0\ 0)$。于是，一步之后的概率分布为：

① 需要注意的是，这里的 P^t 指的是 P 的 t 次幂而不是 P 的转置 P^{T}。

② 这个值一般要归一化到[0,1]之间。——译者注

$$\vec{x}_0 P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} = \vec{x}_1 \text{。} \tag{21-4}$$

两步之后，概率分布为：

$$\vec{x}_1 P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \end{pmatrix} = \vec{x}_2 \text{。} \tag{21-5}$$

按同样方式反复迭代下去，就可以得到图 21-3 所示的概率分布向量序列。

$\vec{x}_0$	1	0	0
$\vec{x}_1$	1/6	2/3	1/6
$\vec{x}_2$	1/3	1/3	1/3
$\vec{x}_3$	1/4	1/2	1/4
$\vec{x}_4$	7/24	5/12	7/24
...	...	...	...
$\vec{x}$	5/18	4/9	5/18

图 21-3　一个概率分布向量序列

反复迭代一定次数之后，我们会看到分布收敛于一个稳定状态 $\vec{x} = \begin{pmatrix} 5/18 & 4/9 & 5/18 \end{pmatrix}$。在这个简单的例子中，我们注意到（21-3）式中的转移概率矩阵的第一行和第三行是相等的，也就是说该马尔科夫链中状态 1 和状态 3 具有对称性，于是可以直接计算稳态的概率分布。假定状态 1 和状态 3 具有相同的稳态概率，记为 p，则最后的稳态概率分布的形式为 $\vec{\pi} = \begin{pmatrix} p & 1-2p & p \end{pmatrix}$。现在，利用恒等式 $\vec{\pi} = \vec{\pi} P$，我们就可以求解一个简单的线性方程，从而得到 $p = 5/18$，于是 $\vec{\pi} = \begin{pmatrix} 5/18 & 4/9 & 5/18 \end{pmatrix}$。

网页的 PageRank 与用户输入的查询无关，这也意味着网页按照 PageRank 的排序结果与查询无关。因此，PageRank 是与查询无关的网页静态质量衡量指标（可以回顾一下 7.1.4 节提到的这种静态质量衡量指标）。另一方面，网页的排名直观上应该与查询有关。因此，使用网页静态质量指标（如 PageRank）的搜索引擎往往将这个指标作为排名因子之一。实际上，PageRank 对总得分的相对贡献可以通过 15.4.1 节的机器学习方法来确定。

例 21-1　考虑图 21-4 中所示的图，假定随机跳转操作的概率是 0.14，其（随机）转移概率矩阵为：

0.02	0.02	0.88	0.02	0.02	0.02	0.02
0.02	0.45	0.45	0.02	0.02	0.02	0.02
0.31	0.02	0.31	0.31	0.02	0.02	0.02
0.02	0.02	0.02	0.45	0.45	0.02	0.02
0.02	0.02	0.02	0.02	0.02	0.02	0.88
0.02	0.02	0.02	0.02	0.02	0.45	0.45
0.02	0.02	0.02	0.31	0.31	0.02	0.31

该矩阵的 PageRank 向量为：

$$\vec{x} = \begin{pmatrix} 0.05 & 0.04 & 0.11 & 0.25 & 0.21 & 0.04 & 0.31 \end{pmatrix} \text{。} \tag{21-6}$$

从图 21-4 中我们观察到，节点 q_2、q_3、q_4 和 q_6 至少包含两个入链接。它们当中，q_2 的 PageRank 最小，这是因为随机游走过程倾向于从图的上部跳出，冲浪者仅仅通过随机跳转操作才能回到这部分图中。

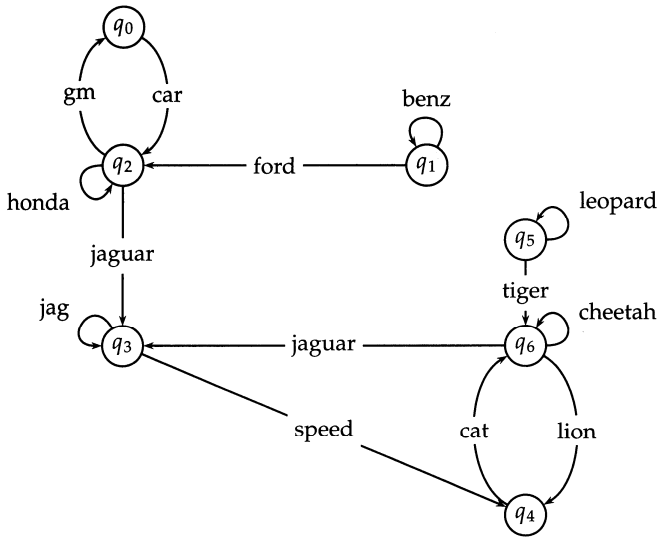


图 21-4 一个小型 Web 图。每条边上都用链接上的锚文本单词来标记

21.2.3 面向主题的 PageRank

迄今为止，我们已经讨论了带随机跳转操作的 PageRank，冲浪者可以等概率地跳到一个随机 Web 网页。现在我们来考虑非等概率跳到一个随机网页的情况。这样做的话，就可以推出基于特定的兴趣的 PageRank。比如，一个体育迷可能希望有关体育主题的网页的排名要高于非体育主题的网页。假定有关体育的网页在 Web 图中彼此“相近”。那么，在随机游走过程中，一个喜欢体育类网页的冲浪者可能会在这类网页上停留大量的时间，因此，体育类网页的稳态分布概率被提升。

假定一个随机冲浪者，仍然像往常一样能够进行随机跳转操作，它会跳到与体育主题有关的一个随机网页，而不是等概率跳到一个随机的网页。这里我们并不关注如何搜集所有的与体育主题有关的网页，实际上，我们只需要一个与体育主题相关的非空网页集合S，以保证随机跳转操作的可行性[①]，而这是可以做到的。比如，可以从一个人工编辑的体育网页的目录中得到该子集，比如从ODP（open directory project，www.dmoz.org/）或Yahoo的分类目录中获取。

假定与体育相关的网页集合S非空，因此存在一个非空网页集合 $Y \supseteq S$，其中Y中的随机游走过程存在稳态分布。我们将这个面向体育主题的PageRank的分布记为 $\vec{\pi}_s$，而对于不在Y中出

① 可以通过随机游走的方式从某些体育类网页跳到另外一些体育类的网页上，因此没有必要收集所有体育类的网页。——译者注

现的网页，令其PageRank为 0。我们称 $\vec{\pi}_s$ 为面向体育主题的PageRank[①]。

这里我们并不要求随机跳转操作下随机冲浪者会等概率地选择目标体育网页，随机跳转到 S 中目标网页的分布实际上可以是任意的。

同样，我们可以设想有面向科学、宗教或政治等其他主题的PageRank分布。每个分布会给每个Web页面一个[0,1）之间的PageRank值[②]。当搜索用户仅仅对某个主题感兴趣时，那么在对检索结果打分和排名过程只须调用相应主题的PageRank向量值进行计算。这样当搜索引擎知道用户关注哪个主题时，上述做法就使我们有调用相应主题配置的能力。由于用户有可能显式地注册了其兴趣，或者系统能够从每个用户的历史行为中学到其兴趣，因此，搜索引擎可以实现上述思路。

但是，有时候用户可能会对多个主题感兴趣，也就是说用户的兴趣由多个主题混合而成，那么此时应该如何处理？比如，某个用户的兴趣（或称为 profile）中由 60%的体育主题和 40%的政治主题混合而成，那么此时我们能否为这个用户计算出其个性化的 PageRank 值呢？乍一看，这个问题似乎令人生畏。因此我们要为每个用户 profile 构造一个不同的 PageRank 分布，而光是每个用户的 profile 都可能会有无穷多个选择。实际上，我们可以假设每个人的兴趣可通过多个主题网页分布的线性组合来很好地近似，这样就可以解决上述问题。一个具有混合兴趣的用户可以这样进行随机跳转操作：首先确定是跳到已知的体育类网页集合 S，还是跳到已知的政治类网页集合。由于对每个网页的选择是随机的，于是在上例中，用户有 60%的时间选择体育类，而其余 40%的时间则选择政治类（参见图 21-5）。一旦随机跳转操作随机选择的是某个体育类网页，那么就从 S 中等概率地随机选择一个网页作为目标。这又导致了一个具有稳态分布的遍历马尔科夫链，且该马尔科夫链是对用户的多主题兴趣的个性化表示结果（参考习题 21-16）。

尽管上述思路直观上很具吸引力，但实施起来似乎非常麻烦。因为看上去似乎要针对每个用户计算一个转移概率矩阵和稳态分布。不过，由于马尔科夫链的状态概率分布的演化过程事实上可以被看成一个线性系统，所以上述问题将迎刃而解。在习题 21-16 中将会看到，我们并不需要对每个用户的不同兴趣组合都计算一个 PageRank 向量，实际上任一用户的个性化 PageRank 都可以被表示成多个面向主题的 PageRank 的线性组合。比如，拥有 60%体育类兴趣和 40%政治类兴趣的用户的个性化 PageRank 就可以表示成：

$$0.6\vec{\pi}_s + 0.4\vec{\pi}_p \text{ 。} \qquad (21\text{-}7)$$

其中，$\vec{\pi}_s$ 和 $\vec{\pi}_p$ 分别是面向体育和政治主题的 PageRank 向量。

---

① 需要指出的是，$\vec{\pi}_s$ 中的下标 s 指的是体育主题（sports），而不是集合 s。实际上，$\vec{\pi}_s$ 向量给出的是所有 Web 网页的 PageRank 值，只不过在这个向量 ϕ，Y 中的每个网页都有非零 PageRank 值，而 Y 之外网页的 PageRank 值为 0。——译者注

② 也就是说，每个主题都会对应一个 PageRank 向量。——译者注

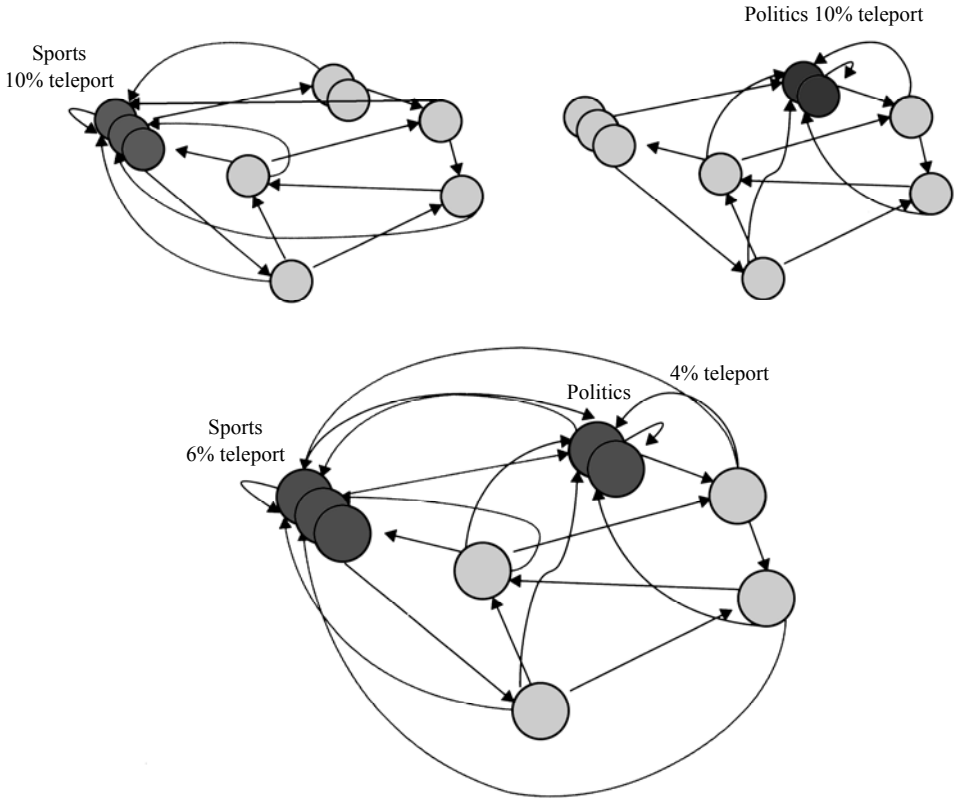


图 21-5 面向主题的 PageRank。上例中我们考虑一个体育兴趣为 60%、政治兴趣为 40%的用户。如果随机跳转操作的概率是 10%的话，那么其中有 6%是到体育类，而 4%到政治类网页

习题 **21-5** 写出图 21-2 中所示例子的转移概率矩阵。

习题 **21-6** 考虑一个具有 3 个节点 1、2、3 的 Web 图。其中的链接为：1 → 2，3 → 2，2 → 1，2 → 3。写出以下不同随机跳转操作概率情况下的带随机跳转操作的冲浪者的转移概率矩阵：

$\alpha = 0$；

$\alpha = 0.5$；

$\alpha = 1$。

习题 **21-7** 除了点击当前浏览页面 x 上的链接之外，浏览器用户还可能通过点击“后退”按钮回到上一页。用户的这种回退行为能否用马尔科夫链来建模？如何对反复的回退调用建模？

习题 **21-8** 考虑一个具有 3 个状态 A、B、C 的马尔科夫链及如下的转移概率：从 A 到 B 的概率为 1。从 B 出发，到 A 的概率为 p_A，到 C 的概率为 $1-p_A$。从 C 出发，到 A 的概率为 1。那么，p_A 在[0,1]上取何值时，该马尔科夫链是可遍历的吗？

习题 **21-9** 试证明，对于任一有向图，带随机跳转操作的随机游走过程所导出的马尔科夫链都是可遍历的。

习题 **21-10** 试证明每个网页的 PageRank 值都不小于 α/N。当 α 逐渐接近 1 时，这对所有不同网页的 PageRank 值之间的差异来说意味着什么？

习题 **21-11** 对于例子 21-1，写一段小程序或者利用一个计算器来计算公式（21-6）中的 PageRank 值。

习题 **21-12**　假定 Web 图以邻接表的形式存储在磁盘上，这种情况下假定用户仅仅查询网页的排好序的链出网页邻居。用户不可能将所有 Web 图装到内存，但是可以采用多次读入的办法。写出在这种情况下计算 PageRank 的算法。

习题 **21-13**　在 21.2.3 节开始我们提到集合 S 和 Y，集合 Y 如何与 S 发生关联？

习题 **21-14**　集合 Y 是不是总是所有的页面集合？为什么？

习题 **21-15 [***]**　S 中每个网页的面向体育主题的 PageRank 是否不低于其在整个 Web 图中的 PageRank？

习题 **21-16 [***]**　考虑对每个 Web 网页具有两个主题相关的 PageRank 的情况：一个是体育主题的 PageRank $\vec{\pi}_s$，另一个是政治主题的 PageRank $\vec{\pi}_p$。令 α 为计算两个主题相关的 PageRank 时随机跳转操作的公共概率。对于 $q \in [0, 1]$，考虑一个兴趣可被分成概率 q 到体育类、$1-q$ 到政治类的用户，试证明带随机跳转操作（到体育类的概率是 q，到政治类的是 $1-q$）的随机游走过程得到的稳态概率分布就是这个用户的个性化 PageRank。

习题 **21-17**　试证明对应于习题 21-16 的马尔科夫链是可遍历的，因此用户的个性化 PageRank 可以通过求解该马尔科夫链的稳态分布来计算。

习题 **21-18**　试证明习题 21-17 的稳态分布中，每个网页 i 的稳态概率等于 $q\pi_s(i) + (1-q)\pi_p(i)$。

21.3　Hub 网页及 Authority 网页

本节中，给定某个查询，我们对每个网页给出两个得分：一个得分被称为hub值，另外一个被称为authority值[①]。因此对于任一查询，我们都可以得到两个排序结果列表，其中一个基于hub值，而另一个基于authority值。

上述方法源于对 Web 网页制作的深入理解，也就说，在泛主题搜索（broad-topic search）时，主要有两类网页结果非常有用。这里说的泛主题搜索指的是信息类的查询，比如“我想了解白血病相关的知识”（I wish to learn about leukemia）。对于这个主题而言，存在一些权威性的网页，比如美国国家癌症研究所(National Cancer Institute)的有关白血病的网页就是其中之一。这种网页我们将之称为权威型网页（authority），在下面的计算中我们就会知道，它们是那些具有很高 authority 值的网页。

另一方面，Web 中存在很多网页，它包含很多人工编辑的指向某些特定主题权威网页的链接。这些所谓的导航型网页（hub）本身并不是某个主题的权威性网页，而是对某个主题感兴趣的人花时间编辑整理出的权威型网页列表。那么，我们的方法能够利用这些导航型网页来帮助我们找到权威型网页。在下面的计算中我们就会知道，这些导航型是那些具有很高 hub 值的网页。

一个好的 hub 网页会同时指向多个好的 authority 网页，而一个好的 authority 网页同时会被多个好的 hub 网页所指向。因此，我们似乎可以给出一个 hub 值和 authority 值的循环定义，然后通过迭代计算来求解。假定我们有一个包含好的 hub 网页和 authority 网页的 Web 子集及它们之间的链接。下面我们将介绍如何基于这个子集迭代计算每个网页的 hub 值和 authority 值，而

① hub 值指的是该网页的导航能力，也可以称为导航度。而 authority 值指的是网页的权威度。为和其他文献保持一致，这里我们直接用 hub 值和 authority 值来表达。——译者注

有关子集的选择问题我们将在 21.3.1 节介绍。

在上述 Web 子集中，某个网页 v 的 hub 值记为 $h(v)$，authority 值记为 $a(v)$。对于任一节点 v，初始化赋值为 $h(v)=a(v)=1$。如果从 v 到 y 存在一条超链接，则记为 $v \mapsto y$。迭代算法的核心环节就是 hub 值和 authority 值的双重更新过程，这个过程如公式（21-8）所示。而整个迭代过程体现了“好 hub 网页指向好 authoriy 网页、好 authority 网页被好 hub 网页所指向”的直觉思想。

$$\begin{aligned} h(v) &\leftarrow \sum_{v \mapsto y} a(y) \\ a(v) &\leftarrow \sum_{y \mapsto v} h(y)。 \end{aligned} \tag{21-8}$$

上式第一行中，每个网页的 hub 值设为它指向的所有网页的 authority 值之和。换句话说，如果 v 指向的页面的 authority 值越高的话，那么它的 hub 值也越高。同样，上式第二行中两者的角色互换，如果页面 v 被 hub 值更高的网页所指向，那么其 authority 值也越高。

在循环迭代过程中，会重新计算所有网页的 hub 值，接着根据更新后的 hub 值又来计算所有网页的 authority 值，接着又根据更新后的 authority 值重新计算所有网页的 hub 值，如此可以反复迭代下去，那么最终结果会如何？我们将（21-8）式转写成矩阵-向量形式。令 $\vec{h}$ 和 $\vec{a}$ 分别表示所有网页的 hub 值和 authority 值向量，A 表示我们所处理的 Web 子集的邻接矩阵。很显然 A 是一个方阵，其每一行和每一列都对应 Web 子图的一个网页。如果存在页面 i 到页面 j 的链接，则有 $A_{ij}=1$，否则 $A_{ij}=0$。于是，可以将（21-8）式写成：

$$\begin{aligned} \vec{h} &\leftarrow A\vec{a} \\ \vec{a} &\leftarrow A^T\vec{h}。 \end{aligned} \tag{21-9}$$

其中，A^T 表示 A 的转置矩阵。于是，(21-9) 中的每个式子的右部都是另一个式子左部的一个向量。于是，将它们互相代入，就会得到

$$\begin{aligned} \vec{h} &\leftarrow AA^T\vec{h} \\ \vec{a} &\leftarrow A^TA\vec{a}。 \end{aligned} \tag{21-10}$$

现在，我们得到了两个特征方程，它们之间具有不可思议的相似性（参见 18-1 节）。实际上，如果将上式中的 $\leftarrow$ 替换为 = 号并引入未知特征值的话，那么公式（21-10）的上面那个式子就变成矩阵 AA^T 的特征方程，而下面那个式子则会变成矩阵 A^TA 的特征方程。

$$\begin{aligned} \vec{h} &= (1/\lambda_h)\ AA^T\vec{h} \\ \vec{a} &= (1/\lambda_a)\ A^TA\vec{a}。 \end{aligned} \tag{21-11}$$

其中，λ_h、λ_a 分别是矩阵 AA^T 及 A^TA 的特征值。

于是，可以得到以下重要推论。

(1) 如果配以合适的特征值，那么（21-8）式也即（21-9）式的迭代更新过程等价于采用幂迭代法求解 AA^T 及 A^TA 的特征向量的过程。假定 AA^T 的主特征向量是唯一的，那么 $\vec{h}$ 和 $\vec{a}$ 最后会收敛于某个唯一的稳态向量，而具体稳态向量的取值取决于矩阵 A，也就是图的结构。

(2) 在计算这些特征向量时，我们的方法并不仅限于幂迭代方法。实际上，我们可以选择

任何随机矩阵主特征向量的快速求解方法。

于是，最后的计算形式如下：

(1) 收集 Web 网页构成网页子集，利用网页的链接形成图结构，计算 AA^T 及 A^TA；

(2) 计算 AA^T 及 A^TA 的主特征向量，得到最后的 *hub* 值向量 $\vec{h}$ 和 authority 值向量 $\vec{a}$；

(3) 输入排名靠前的 hub 网页和 authority 网页。

上述链接分析的方法被称为 HITS（Hyperlink-Induced Topic Search，超链导向的主题搜索）。

例 21-2　假定查询为 jaguar，我们将锚文本包含该查询词的链接的权重加倍，则图 21-4 对应矩阵 A 如下：

$$\begin{matrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 2 & 1 & 0 & 1 \end{matrix}$$

于是，可以计算得到最后的 hub 值及 authority 值向量为：

$$\vec{h} = \begin{pmatrix} 0.03 & 0.04 & 0.33 & 0.18 & 0.04 & 0.04 & 0.35 \end{pmatrix}$$

$$\vec{a} = \begin{pmatrix} 0.10 & 0.01 & 0.12 & 0.47 & 0.16 & 0.01 & 0.13 \end{pmatrix}$$

这里，q_3 是主要的 authority 网页，两个 hub 网站通过包含 jaguar 的高权重链接指向它。

由于上述迭代过程刻画了好的 hub 和好的 authority 的直觉思想，上述输出就会从目标 Web 子集中得到靠前的结果。在下一节中，我们将介绍剩余细节：对于给定的主题（如 leukemia），如何得到其目标网页子集？

21.3.1　Web 子集的选择

为构造与给定主题（如 leukemia）相关的 Web 子集，我们必须要能处理这样一个事实：好的权威型网页也许并不包含查询项 leukemia。正如我们在 21.1.1 节所说的那样，当某个权威型网页是利用用于突出其宣传图片的时候，尤其会发生这种情况。比如，尽管 IBM 网站上的很多网页本身都不包含词项 computer 或者 hardware，但是它们都是有关计算机硬件（computer hardware）的权威型网页。然而，一个编辑计算机硬件相关信息的 hub 网页很可能会使用这些词项并会链接到 IBM 的这些网页。

基于上述观察结果，下列过程被推荐为 hub 值和 authority 值计算的 Web 子集的选取。

(1) 给定某个查询（比如 leukemia），利用某个文本索引获得包含 leukemia 的所有网页。这些网页被称为根集（root set）。

(2) 将根集及指向根集中的网页和根集所指向的网页加入到基本集（base set）中。

我们利用上述过程产生的基本集进行 hub 值和 authority 值的计算。之所以如此构造基本集的原因在于：

(1) 一个好的 authority 网页可能不包含查询文本（如刚才提到的 computer hardware 的例子）；

(2) 如果文本查询要设法从根集中获得一个好的 hub 网页 v_h，那么将所有根集指向的网页全部包括进来的话，就能在基本集获得 v_h 所指向的所有好的 authority 网页；

(3) 反之，如果文本查询要设法从根集中获得一个好的 authority 网页 v_a，那么将所有指向根集的网页全部包括进来的话，就将其他的好的 hub 网页加入到基本集中。换句话说，从根集到基本集的扩展能够增加更多好的 hub 网页及 authority 网页。

对于很多不同的查询还运行 HITS 算法，会发现一些有关链接分析的有趣的结论。排名靠前的 hub 网页和 authority 网页往往包括不止查询所在的一种语言。推测起来，这些不同语言的网页可能在根集扩展到基本集的过程中被加入。因此，一些跨语言检索元素的出现在这里看起来很明显。有趣的是，这里的跨语言的效果完全来自于链接分析，而不需要任何语言翻译的工作。

最后，我们给出本节的结论，并指出一些算法实施上的注意事项。根集包含所有的与文本查询匹配的网页，我们在实际实施（参考 21.4 给出的参考文献）时，建议取 200 左右的网页就足够了。任意计算特征向量的算法都可以用于计算 hub 值和 authority 值向量。实际上，我们并不需要计算这些值的精确值，而只需要知道这些值的相对大小以便能够进行排序即可。为此，利用幂迭代算法有可能只需要少量迭代次数就可以获得高 hub 值和高 authority 值网页的相对次序。实验结果表明，实际上公式（21-8）只需要大概 5 次迭代就可以产生相当好的结果。另外，由于 Web 图结构非常稀疏（平均每个网页指向 10 个其他网页），我们并不采用矩阵-向量积的方式进行计算，而是采用公式（21-8）的加法进行更新。

图 21-6 给出了查询“ japan elementary schools”（日本小学）的 HITS 算法运行的结果。图中给出排名靠前的 hub 网页和 authority 网页，每一行给出了从相应网页中抽取的 title 标签。由于结果字符串不一定是拉丁字母，所以输出的结果有很多看上去是一毫无意义的乱码串。每个串都对应于一个使用非拉丁字母的网页，本例中很可能是日语网页。另外似乎还出现了其他的非英语的网页，这些网页对应于给定的英文查询来说有些奇怪。实际上，这正是使用 HITS 的标志性结果特征。因为在根集建立之后，（英文）查询字符串被忽略。基本集可能会包含其他语言的网页，比如如果有一个英文的 hub 网页指向一个日文描述的日本小学的主页的话，那么后者会被引入到基本集中。由于后续的 hub 值和 authority 值的计算全部都是基于链接的，那么部分非英文的页面就有可能在最后的 hub 值或 authority 值结果中排名靠前。

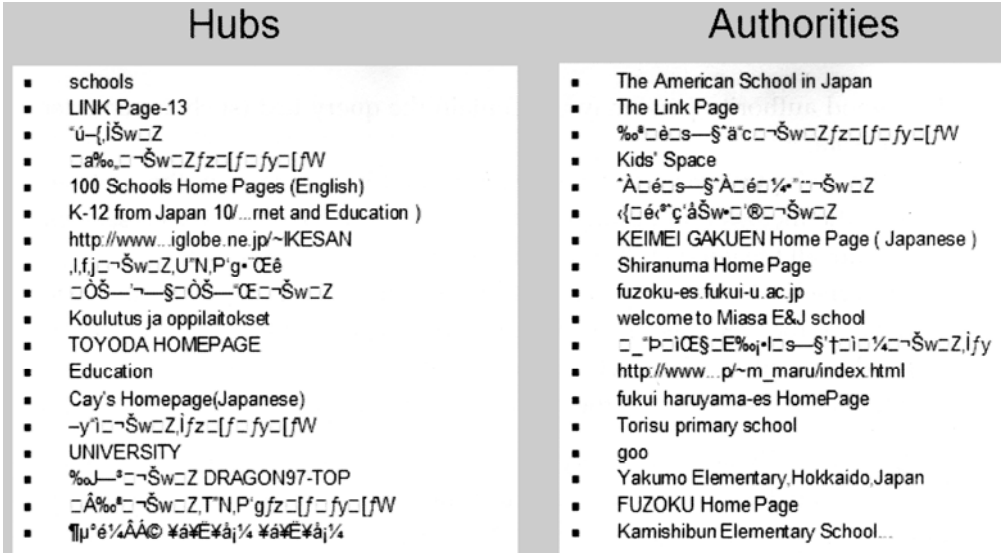


图 21-6　查询 japan elementary schools 的 HITS 算法的运行结果

习题 **21-19**　如果所有网页的 hub 值和 authority 值的初始值都设为 1 的话，那么一次迭代之后的 hub 值和 authority 值是多少？

习题 **21-20**　如何解释矩阵 AA^T 和 A^TA 中每个元素的含义？它与第 18 章的共现矩阵 CC^T 有什么关系？

习题 **21-21**　什么是 AA^T 和 A^TA 的主特征向量？

习题 **21-22**　对图 21-7 所示的 Web 图，计算每个网页计算其 PageRank、hub 值及 authority 值。依照这三个值分别给出三个网页的排序结果，如果出现相等值，则请标明。

PageRank：假定 PageRank 的每一步，我们都有 0.1 的概率进行随机跳转操作，其中会等概率地选择随机跳转的目标网页。

Hub/authority：将 hub 值或 authority 值进行归一化，以保证最大的 hub 或 authority 值为 1。

提示 1：利用对称性来简化线性方程并求解，这种做法可能会比迭代方法容易得多；

提示 2：对每个衡量指标，给出三个节点的相对次序。

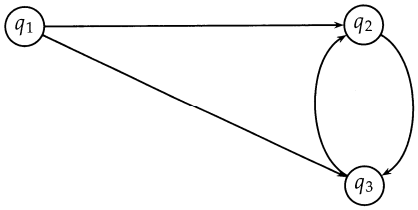


图 21-7　习题 21-22 对应的 Web 图

21.4 参考文献及补充读物

Garfield（1955）是引文分析科学中的开创性工作。基于该项工作，Pinski 和 Narin（1976）提出了期刊影响权重（*journal influence weight*）的概念，其定义与 PageRank 非常类似。利用锚文本进行搜索和排名源自 McBryan（1994）的工作。扩充的锚文本在他的工作中有所暗示，更系统的结果可以参考 Chakrabarti 等人（1998）。

Kemeny 和 Snell（1976）是有关马尔科夫链的经典教材。PageRank 在 Brin 和 Page（1998）及 Page 等人（1998）中提出并得以发展。PageRank 值的一些快速计算方法总结在 Berkhin（2005）和 Langville 和 Meyer（2006）中，前者详细介绍了如何将 PageRank 特征求解看成线性系统进行求解的过程，这也给出了习题 21-16 的一种解答方法。有关随机跳转操作概率 α 的研究可以参考 Baeza-Yates 等人（2005）及 Boldi 等人（2005）。面向主题的 PageRank 及其变形可以参考 Haveliwala（2002）、Haveliwala（2003）及 Jeh 和 Widom（2003）。Berkhin（2006a）给出了面向主题的 PageRank 的另外一个视角。

Ng 等人（2001b）认为 PageRank 的结果比 HITS 算更鲁棒，也就说，相对于图拓扑结构的微小变化而言，PageRank 的结果比 HITS 更不敏感。然而，他也同时指出，随机跳转操作对 PageRank 的这种鲁棒性具有非常重要的贡献。不论是 PageRank 还是 HITS 算法，都可以通过在 Web 图中插入构造的链接来对它们进行作弊。实际上，我们已经知道，在 Web 中存在所谓链接农场（link farm）现象，即多个网页联合来提高它们在各种链接分析方法下的得分结果。

HITS 算法归功于 Kleinberg（1999）的工作。Chakrabarti 等人（1998）提出了多种变形方法，主要思路是在迭代计算中基于查询项在被链接网页中的存在信息给链接赋予不同权重。他将这些方法与一些搜索引擎的结果进行了比较。Bharat 和 Henzinger（1998）进一步发展了这些方法并给出了其他的一些启发式策略，结果表明对这些方法进行一定的组合能够超过基本 HITS 算法的结果。Borodin 等人（2001）对多种不同的 HITS 的变形算法进行了系统的研究。Ng 等人（2001b）提出了链接分析的稳定性（stability）的概念，文章认为链接拓扑结构的微小变化不会导致最后排名结果的显著变化。其他研究人员也提出了许多其他的 HITS 变形算法，其中最出名的可能是 SALSA（Lempel 和 Moran 2000）。

参 考 文 献

以下参考文献中，我们使用如下期刊和会议名称的缩写形式。

CACM Communications of the Association for Computing Machinery.
IP&M Information Processing and Management.
IR Information Retrieval.
JACM Journal of the Association for Computing Machinery.
JASIS Journal of the American Society for Information Science.
JASIST Journal of the American Society for Information Science and Technology.
JMLR Journal of Machine Learning Research.
TOIS ACM Transactions on Information Systems.
Proc. ACL Proceedings of the Annual Meeting of the Association for Computational Linguistics. Available from: http://www.aclweb.org/anthology-index/
Proc. CIKM Proceedings of the Conference on Information and Knowledge Management.
Proc. ECIR Proceedings of the European Conference on Information Retrieval.
Proc. ECML Proceedings of the European Conference on Machine Learning.
Proc. ICML Proceedings of the International Conference on Machine Learning.
Proc. IJCAI Proceedings of the International Joint Conference on Artificial Intelligence.
Proc. INEX Proceedings of the Initiative for the Evaluation of XML Retrieval.
Proc. KDD Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
Proc. NIPS Proceedings of the Neural Information Processing Systems Conference.
Proc. PODS Proceedings of the ACM Conference on Principles of Database Systems.
Proc. SDAIR Proceedings of the Annual Symposium on Document Analysis and Information Retrieval.
Proc. SIGIR Proceedings of the Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval. Available from: http://www.sigir.org/proceedings/Proc-Browse.html
Proc. SPIRE Proceedings of the Symposium on String Processing and Information Retrieval.
Proc. TREC Proceedings of the Text Retrieval Conference.
Proc. UAI Proceedings of the Conference on Uncertainty in Artificial Intelligence.
Proc. VLDB Proceedings of the Very Large Data Bases Conference.
Proc. WWW Proceedings of the International World Wide Web Conference.

Aberer, Karl. 2001. P-grid: A self-organizing access structure for P2P information systems. In *Proc. International Conference on Cooperative Information Systems*, pp. 179-194. Springer.

Aizerman, Mark A., Emmanuel M. Braverman, and Lev I. Rozonoér. 1964. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control* 25:821-837. [319]

Akaike, Hirotugu. 1974. A new look at the statistical model identification. *IEEE Transactions on automatic control* 19(6):716-723. [345]

Allan, James. 2005. HARD track overview in TREC 2005: High accuracy retrieval from documents. In *Proc. TREC*. [160]

Allan, James, Ron Papka, and Victor Lavrenko. 1998. On-line new event detection and tracking. In *Proc. SIGIR*, pp. 37-45. ACM Press. DOI: http://doi.acm.org/10.1145/ 290941.290954. [367]

Allwein, Erin L., Robert E. Schapire, and Yoram Singer. 2000. Reducing multiclass to binary: A unifying approach for margin classifiers. *JMLR* 1:113-141. URL: www.*jmlr* .org/papers/volume1/allwein00a/allwein00a.pdf. [292]

Alonso, Omar, Sandeepan Banerjee, and Mark Drake. 2006. GIO: A semantic web application using the information grid framework. In *Proc. WWW,* pp. 857-858. ACM Press. DOI: http://doi.acm.org/10.1145/1135777.1135913. [344]

Altingövde, Ismail Sengör, Engin Demir, Fazli Can, and Özgür Ulusoy. 2008. Incremental cluster-based retrieval using compressed cluster-skipping inverted files. *TOIS*. To appear. 372

Amer-Yahia, Sihem, Chavdar Botev, Jochen Dörre, and Jayavel Shanmugasundaram. 2006. XQuery full-text extensions explained. *IBM Systems Journal* 45(2):335-352. [200]

Amer-Yahia, Sihem, Pat Case, Thomas Rölleke, Jayavel Shanmugasundaram, and Gerhard Weikum. 2005. Report on the DB/IR panel at SIGMOD 2005. *SIGMOD Record* 34(4):71-74. DOI: http://doi.acm.org/10.1145/1107499.1107514. [200]

Amer-Yahia, Sihem, and Mounia Lalmas. 2006. XML search: Languages, INEX and scoring. *SIGMOD Record* 35(4):16-23. doi: http://doi.acm.org/10.1145/1228268. 1228271. [200]

Anagnostopoulos, Aris, Andrei Z. Broder, and Kunal Punera. 2006. Effective and efficient classification on a search-engine model. In *Proc. CIKM,* pp. 208-217. ACM Press. DOI: http://doi.acm.org/10.1145/1183614.1183648. [292]

Anderberg, Michael R. 1973. *Cluster analysis for applications*. Academic Press. [344]

Andoni, Alexandr, Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab Mirrokni. 2006. Locality-sensitive hashing using stable distributions. In *Nearest Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press. [291]

Anh, Vo Ngoc, Owen de Kretser, and Alistair Moffat. 2001. Vector-space ranking with effective early termination. In *Proc. SIGIR,* pp. 35-42. ACM Press. [137]

Anh, Vo Ngoc, and Alistair Moffat. 2005. Inverted index compression using word-aligned binary codes. *IR* 8(1):151-166. DOI: http://dx.doi.org/10.1023/B:INRT. 0000048490.99518.5c. [98]

Anh, Vo Ngoc, and Alistair Moffat. 2006a. Improved word-aligned binary compression for text indexing. *IEEE Transactions on Knowledge and Data Engineering* 18(6): 857-861. [98]

Anh, Vo Ngoc, and Alistair Moffat. 2006b. Pruned query evaluation using precomputed impacts. In *Proc. SIGIR,* pp. 372-379. ACM Press. DOI: http://doi.acm.org/ 10.1145/1148170.1148235. [137]

Anh, Vo Ngoc, and Alistair Moffat. 2006c. Structured index organizations for highthroughput text querying. In *Proc. SPIRE,* pp. 304-315. Springer. [138]

Apté, Chidanand, Fred Damerau, and Sholom M. Weiss. 1994. Automated learning of decision rules for text categorization. *TOIS* 12(1):233-251. [265]

Arthur, David, and Sergei Vassilvitskii. 2006. How slow is the K-means method? In *Proc. Symposium on Computational Geometry*, pp. 144-153. [345]

Arvola, Paavo, Marko Junkkari, and Jaana Kekäläinen. 2005. Generalized contextualization method for XML information retrieval. In *Proc. CIKM,* pp. 20.27. ACM Press. [199]

Aslam, Javed A., and Emine Yilmaz. 2005. A geometric interpretation and analysis of R-precision. In *Proc. CIKM,* pp. 664-671. ACM Press. [160]

Ault, Thomas Galen, and Yiming Yang. 2002. Information filtering in TREC-9 and TDT-3: A comparative analysis. *IR* 5(2-3):159-187. [292]

Badue, Claudine Santos, Ricardo A. Baeza-Yates, Berthier Ribeiro-Neto, and Nivio Ziviani. 2001. Distributed query processing using partitioned inverted files. In *Proc. SPIRE,* pp. 10.20. [420]

Baeza-Yates, Ricardo, Paolo Boldi, and Carlos Castillo. 2005. The choice of a damping function for propagating importance in link-based ranking. Technical report, Dipartimento di Scienze dell'Informazione, Università degli Studi diMilano. [439]

Baeza-Yates, Ricardo, and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison-Wesley. [xviii, 76, 97, 161, 368]

Bahle, Dirk, Hugh E.Williams, and Justin Zobel. 2002. Efficient phrase querying with an auxiliary index. In *Proc.*

SIGIR, pp. 215-221. ACM Press. [44]

Baldridge, Jason, andMiles Osborne. 2004. Active learning and the total cost of annotation. In *Proc. EMNLP*, pp. 9-16. [320]

Ball, G. H. 1965. Data analysis in the social sciences: What about the details? In *Proc. Fall Joint Computer Conference,* pp. 533-560. Spartan Books. [345]

Banko, Michele, and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proc. ACL*. [309]

Bar-Ilan, Judit, and Tatyana Gutman. 2005. How do search engines respond to some non-English queries? *Journal of Information Science* 31(1):13-28. [43]

Bar-Yossef, Ziv, and Maxim Gurevich. 2006. Random sampling from a search engine's index. In *Proc.WWW*, pp. 367-376. ACM Press. DOI: http://doi.acm.org/10.1145/ 1135777.1135833. [404]

Barroso, Luiz André, Jeffrey Dean, and Urs Hölzle. 2003. Web search for a planet: The Google cluster architecture. *IEEE Micro* 23(2):22-28. http://dx.doi.org/ 10.1109/MM.2003.1196112. [420]

Bartell, Brian Theodore. 1994. *Optimizing ranking functions: A connectionist approach to adaptive information retrieval.* PhD thesis, University of California at San Diego, La Jolla, CA. [138]

Bartell, Brian T., Garrison W. Cottrell, and Richard K. Belew. 1998. Optimizing similarity using multi-query relevance feedback. *JASIS* 49(8):742-761. [138]

Barzilay, Regina, and Michael Elhadad. 1997. Using lexical chains for text summarization. In *Workshop on Intelligent Scalable Text Summarization,* pp. 10.17. [161]

Bast, Holger, and Debapriyo Majumdar. 2005. Why spectral retrieval works. In *Proc. SIGIR,* pp. 11-18. ACM Press. DOI: http://doi.acm.org/10.1145/1076034.1076040. [384]

Basu, Sugato, Arindam Banerjee, and Raymond J. Mooney. 2004. Active semisupervision for pairwise constrained clustering. In *Proc. SIAM International Conference on Data Mining,* pp. 333-344. [344]

Beesley, Kenneth R. 1998. Language identifier: A computer program for automatic natural-language identification of on-line text. In *Languages at Crossroads: Proceedings of the Annual Conference of the American Translators Association,* pp. 47-54. [43]

Beesley, Kenneth R., and Lauri Karttunen. 2003. *Finite State Morphology.* CSLI Publications. [43]

Bennett, Paul N. 2000. *Assessing the calibration of naive Bayes' posterior estimates.* Technical Report CMU-CS-00-155, School of Computer Science, Carnegie Mellon University. [265]

Berger, Adam, and John Lafferty. 1999. Information retrieval as statistical translation. In *Proc. SIGIR,* pp. 222-229. ACM Press. [232]

Berkhin, Pavel. 2005. A survey on pagerank computing. *Internet Mathematics* 2(1):73-120. [439]

Berkhin, Pavel. 2006a. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics* 3(1):41-62. [439]

Berkhin, Pavel. 2006b. A survey of clustering data mining techniques. In Jacob Kogan, Charles Nicholas, and Marc Teboulle (eds.), *Grouping Multidimensional Data: Recent Advances in Clustering,* pp. 25-71. Springer. [343]

Berners-Lee, Tim, Robert Cailliau, Jean-Francois Groff, and Bernd Pollermann. 1992. World-Wide Web: The information universe. *Electronic Networking: Research, Applications and Policy* 1(2):74-82. URL: citeseer.ist.psu.edu/article/bernerslee92worldwide.html. [404]

Berry, Michael, and Paul Young. 1995. Using latent semantic indexing for multilanguage information retrieval. *Computers and the Humanities* 29(6):413-429. [384]

Berry,MichaelW., Susan T. Dumais, and GavinW. O'Brien. 1995. Using linear algebra for intelligent information retrieval. *SIAM Review* 37(4):573-595. [383]

Betsi, Stamatina, Mounia Lalmas, Anastasios Tombros, and Theodora Tsikrika. 2006. User expectations from XML element retrieval. In *Proc. SIGIR,* pp. 611-612. ACM Press. [199]

Bharat, Krishna, and Andrei Broder. 1998. A technique for measuring the relative size and overlap of public web search engines. *Computer Networks and ISDN Systems* 30 (1-7):379-388. DOI: http://dx.doi.org/10.1016/ S0169-7552(98)00127-5. [404]

Bharat, Krishna, Andrei Broder, Monika Henzinger, Puneet Kumar, and Suresh Venkatasubramanian. 1998. The connectivity server: Fast access to linkage information on the web. In *Proc. WWW,* pp. 469-477. [420]

Bharat, Krishna, Andrei Z. Broder, Jeffrey Dean, and Monika Rauch Henzinger. 2000. A comparison of techniques to find mirrored hosts on the WWW. *JASIS* 51(12): 1114-1122. URL: citeseer.ist.psu.edu/bharat99comparison.html. [404]

Bharat, Krishna, and Monika R. Henzinger. 1998. Improved algorithms for topic distillation in a hyperlinked environment. In Proc. SIGIR, pp. 104-111. ACM Press. URL: citeseer.ist.psu.edu/bharat98improved.html. [439]

Bishop, ChristopherM. 2006. *Pattern Recognition and Machine Learning.* Springer. [292]

Blair, David C., and M. E. Maron. 1985. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *CACM* 28(3):289-299. [177]

Blanco, Roi, and Alvaro Barreiro. 2006. TSP and cluster-based solutions to the reassignment of document identifiers. *IR* 9(4):499-517. [98]

Blanco, Roi, and Alvaro Barreiro. 2007. Boosting static pruning of inverted files. In *Proc. SIGIR*. ACM Press. [97]

Blandford, Dan, and Guy Blelloch. 2002. Index compression through document reordering. In *Proc. Data Compression Conference,* p. 342. IEEE Computer Society. [98]

Blei, David M., Andrew Y. Ng, andMichael I. Jordan. 2003. Latent Dirichlet allocation. *JMLR* 3:993-1022. [384]

Boldi, Paolo, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. 2002. Ubicrawler: A scalable fully distributed web crawler. In *Proc. Australian World Wide Web Conference.* URL: citeseer.ist.psu.edu/article/boldi03ubicrawler.html. [419]

Boldi, Paolo, Massimo Santini, and Sebastiano Vigna. 2005. PageRank as a function of the damping factor. In *Proc. WWW*. URL: citeseer.ist.psu.edu/boldi05pagerank.html. [439]

Boldi, Paolo, and Sebastiano Vigna. 2004a. Codes for the World-Wide Web. *Internet Mathematics* 2(4):405-427. [420]

Boldi, Paolo, and Sebastiano Vigna. 2004b. TheWebGraph framework I: Compression techniques. In *Proc. WWW,* pp. 595-601. ACM Press. [420]

Boldi, Paolo, and Sebastiano Vigna. 2005. Compressed perfect embedded skip lists for quick inverted-index lookups. In *Proc. SPIRE*. Springer. [44]

Boley, Daniel. 1998. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery* 2(4):325-344. DOI: http://dx.doi.org/10.1023/A:1009740529316. [368]

Borodin, Allan, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. 2001. Finding authorities and hubs from link structures on theWorldWideWeb. In *Proc. WWW,* pp. 415-429. [439]

Bourne, Charles P., and Donald F. Ford. 1961. A study of methods for systematically abbreviating English, words and names. *JACM* 8(4):538-552. DOI: http://doi. acm.org/10.1145/321088.321094. [60]

Bradley, Paul S., and UsamaM. Fayyad. 1998. Refining initial points for k-means clustering. In *Proc. ICML,* pp. 91-99. [345]

Bradley, Paul S., Usama M. Fayyad, and Cory Reina. 1998. Scaling clustering algorithms to large databases. In *Proc. KDD,* pp. 9-15. [345]

Brill, Eric, and Robert C. Moore. 2000. An improved error model for noisy channel spelling correction. In *Proc. ACL,* pp. 286-293. [60]

Brin, Sergey, and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proc. WWW,* pp. 107-117. [137, 419, 439]

Brisaboa, Nieves R., Antonio Fari ñ a, Gonzalo Navarro, and José R. Paramá. 2007. Lightweight natural language text compression. *IR* 10(1):1-33. [99]

Broder, Andrei. 2002. A taxonomy of web search. *SIGIR Forum* 36(2):3-10. DOI: http://doi.acm.org/10.1145/792550.792552. [404]

Broder, Andrei, S. Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. 2000. Graph structure in the web. *Computer Networks* 33(1):309-320. [404]

Broder, Andrei Z., Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. 1997. Syntactic clustering of the web. In *Proc. WWW,* pp. 391-404. [404]

Brown, EricW. 1995. *Execution Performance Issues in Full-Text Information Retrieval.* PhD thesis, University of Massachusets, Amherst. [137]

Buckley, Chris, James Allan, and Gerard Salton. 1994a. Automatic routing and ad-hoc retrieval using SMART: TREC 2. In *Proc. TREC*, pp. 45-55.

Buckley, Chris, and Gerard Salton. 1995. Optimization of relevance feedback weights. In *Proc. SIGIR*, pp. 351-357. ACM Press. doi: http://doi.acm.org/10.1145/ 215206.215383. [292]

Buckley, Chris, Gerard Salton, and James Allan. 1994b. The effect of adding relevance information in a relevance feedback environment. In *Proc. SIGIR*, pp. 292-300. ACM Press. [170, 177]

Buckley, Chris, Amit Singhal, and Mandar Mitra. 1995. New retrieval approaches using SMART: TREC 4. In *Proc. TREC*. [172]

Buckley, Chris, and EllenM. Voorhees. 2000. Evaluating evaluation measure stability. In *Proc. SIGIR*, pp. 33-40. [160]

Burges, Chris, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proc. ICML*. [320]

Burges, Christopher J. C. 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery* 2(2):121-167. [318]

Burner, Mike. 1997. Crawling towards eternity: Building an archive of the World Wide Web. *Web Techniques Magazine* 2(5). [419]

Burnham, Kenneth P., and David Anderson. 2002. *Model Selection and Multi-Model Inference*. Springer. [345]

Bush, Vannevar. 1945. As we may think. *The Atlantic Monthly*. URL: www.theatlantic. com/doc/194507/bush. [16, 404]

Büttcher, Stefan, and Charles L. A. Clarke. 2005a. Indexing time vs. query time: Tradeoffs in dynamic information retrieval systems. In *Proc. CIKM*, pp. 317-318. ACM Press. DOI: http://doi.acm.org/10.1145/1099554. 1099645. [76]

Büttcher, Stefan, and Charles L. A. Clarke. 2005b. A security model for full-text file system search in multi-user environments. In *FAST*. URL: www.usenix.org/events/ fast05/tech/buettcher.html. [77]

Büttcher, Stefan, and Charles L. A. Clarke. 2006. A document-centric approach to static index pruning in text retrieval systems. In *Proc. CIKM*, pp. 182-189. ACM Press. DOI: http://doi.acm.org/10.1145/1183614. 1183644. [97]

Büttcher, Stefan, Charles L. A. Clarke, and Brad Lushman. 2006. Hybrid index maintenance for growing text collections. In *Proc. SIGIR*, pp. 356-363. ACM Press. DOI: http://doi.acm.org/10.1145/1148170.1148233. [76]

Cacheda, Fidel, Victor Carneiro, Carmen Guerrero, and Ángel Viña. 2003. Optimization of restricted searches in web directories using hybrid data structures. In *Proc. ECIR*, pp. 436-451. [344]

Callan, Jamie. 2000. Distributed information retrieval. In W. Bruce Croft (ed.), *Advances in information retrieval*, pp. 127-150. Kluwer. [76]

Can, Fazli, Ismail Sengör Altingövde, and Engin Demir. 2004. Efficiency and effectiveness of query processing in cluster-based retrieval. *Information Systems* 29(8): 697-717. DOI: http://dx.doi.org/10.1016/S0306-4379 (03)00062-0. [344]

Can, Fazli, and Esen A. Ozkarahan. 1990. Concepts and effectiveness of the covercoefficient- based clustering methodology for text databases. *ACM Trans. Database Syst.* 15(4):483-517. [344]

Cao, Guihong, Jian-Yun Nie, and Jing Bai. 2005. Integrating word relationships into language models. In *Proc. SIGIR*, pp. 298-305. ACM Press.

Cao, Yunbo, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. 2006. Adapting Ranking SVM to document retrieval. In *Proc. SIGIR*. ACM Press. [320]

Carbonell, Jaime, and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proc. SIGIR*, pp. 335-336. ACM Press. DOI: http://doi.acm.org/10.1145/ 290941.291025. [154]

Carletta, Jean. 1996. Assessing agreement on classification tasks: The kappa statistic. *Computational Linguistics* 22:249-254. [160]

Carmel, David, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, Yoelle S. Maarek, and Aya Soffer. 2001. Static index pruning for information retrieval systems. In *Proc. SIGIR*, pp. 43-50. ACM Press. DOI: http://doi.acm.org/ 10.1145/383952.383958. [97, 138]

Carmel, David, Yoelle S. Maarek,Matan Mandelbrod, Yosi Mass, andAya Soffer. 2003. Searching XML documents via XML fragments. In *Proc. SIGIR,* pp. 151-158. ACM Press. doi: http://doi.acm.org/10.1145/ 860435.860464. [199]

Caruana, Rich, and Alexandru Niculescu-Mizil. 2006. An empirical comparison of supervised learning algorithms. In *Proc. ICML*. [319]

Castro, R. M., M. J. Coates, and R. D. Nowak. 2004. Likelihood based hierarchical clustering. *IEEE Transactions in Signal Processing* 52(8):2308-2321. [368]

Cavnar, William B., and John M. Trenkle. 1994. N-gram-based text categorization. In *Proc. SDAIR,* pp. 161-175. [43]

Chakrabarti, Soumen. 2002. *Mining the Web: Analysis of Hypertext and Semi Structured Data.* Morgan Kaufman. [404]

Chakrabarti, Soumen, Byron Dom, David Gibson, Jon Kleinberg, Prabhakar Raghavan, and Sridhar Rajagopalan. 1998. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *Proc. WWW*. URL: citeseer.ist.psu.edu/chakrabarti98automatic.html. [439]

Chapelle, Olivier, Bernhard Schölkopf, and Alexander Zien (eds.). 2006. *Semi- Supervised Learning.* MIT Press. [319, 459]

Chaudhuri, Surajit, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. 2006. Probabilistic information retrieval approach for ranking of database query results. *ACM Trans. Database Syst.* 31(3):1134-1168. DOI: http://doi.acm.org/10.1145/ 1166074.1166085. [200]

Cheeseman, Peter, and John Stutz. 1996. Bayesian classification (AutoClass): Theory and results. In *Advances in Knowledge Discovery and Data Mining,* pp. 153-180. MIT Press. [345]

Chen, Hsin-Hsi, and Chuan-Jie Lin. 2000. A multilingual news summarizer. In *Proc. COLING,* pp. 159-165. [344]

Chen, Pai-Hsuen, Chih-Jen Lin, and Bernhard Schölkopf. 2005. A tutorial on *v*- support vector machines. *Applied Stochastic Models in Business and Industry* 21:111-136. [318]

Chiaramella, Yves, Philippe Mulhem, and Franck Fourel. 1996. A model for multimedia information retrieval. Technical Report 4-96, University of Glasgow. [198]

Chierichetti, Flavio, Alessandro Panconesi, Prabhakar Raghavan, Mauro Sozio, Alessandro Tiberi, and Eli Upfal. 2007. Finding near neighbors through cluster pruning. In *Proc. PODS*. [137]

Cho, Junghoo, and Hector Garcia-Molina. 2002. Parallel crawlers. In *Proc. WWW,* pp. 124-135. ACM Press. DOI: http://doi.acm.org/10.1145/511446.511464. [419]

Cho, Junghoo, Hector Garcia-Molina, and Lawrence Page. 1998. Efficient crawling through URL ordering. In *Proc. WWW,* pp. 161-172. [419]

Chu-Carroll, Jennifer, John Prager, Krzysztof Czuba, David Ferrucci, and Pablo Duboue. 2006. Semantic search via XML fragments: A high-precision approach to IR. In *Proc. SIGIR,* pp. 445-452. ACM Press. DOI: http://doi.acm.org/10.1145/ 1148170.1148247. [198]

Clarke, Charles L.A., Gordon V. Cormack, and Elizabeth A. Tudhope. 2000. Relevance ranking for one to three term queries. *IP&M* 36:291-311. [138]

Cleverdon, Cyril W. 1991. The significance of the Cranfield tests on index languages. In *Proc. SIGIR,* pp. 3-12. ACM Press. [159]

Coden, Anni R., Eric W. Brown, and Savitha Srinivasan (eds.). 2002. *Information Retrieval Techniques for Speech Applications.* Springer. [xviii]

Cohen, Paul R. 1995. *Empirical Methods for Artificial Intelligence.* MIT Press. [265]

Cohen, William W. 1998. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. ACM SIGMOD International Conference on Management of Data,* pp. 201-212. ACM Press. [200]

Cohen, William W., Robert E. Schapire, and Yoram Singer. 1998. Learning to order things. In *Proc. NIPS*. The MIT Press. url: citeseer.ist.psu.edu/article/ cohen98learning.html. [138]

Cohen, William W., and Yoram Singer. 1999. Context-sensitive learning methods for text categorization. *TOIS* 17(2):141-173. [312]

Comtet, Louis. 1974. *Advanced Combinatorics.* Reidel. [327]

Cooper, William S., Aitao Chen, and Fredric C. Gey. 1994. Full text retrieval based on probabilistic equations with coefficients fitted by logistic regression. In *Proc. TREC,* pp. 57-66. [138]

Cormen, Thomas H., Charles Eric Leiserson, and Ronald L. Rivest. 1990. *Introduction to Algorithms.* MIT Press. [10, 72, 367]

Cover, Thomas M., and Peter E. Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13(1):21-27. [292]

Cover, Thomas M., and Joy A. Thomas. 1991. *Elements of Information Theory.* Wiley. [98]

Crammer, Koby, and Yoram Singer. 2001. On the algorithmic implementation of multiclass kernel-based machines. *JMLR* 2:265-292. [319]

Creecy, Robert H., Brij M. Masand, Stephen J. Smith, and David L. Waltz. 1992. Trading MIPS and memory for knowledge engineering. *CACM* 35(8):48-64. DOI: http://doi.acm.org/10.1145/135226.135228. [291]

Crestani, Fabio, Mounia Lalmas, Cornelis J. Van Rijsbergen, and Iain Campbell. 1998. Is this document relevant?. . . probably: A survey of probabilistic models in information retrieval. *ACM Computing Surveys* 30(4):528-552. DOI: http:// doi.acm.org/10.1145/299917.299920. [216]

Cristianini, Nello, and John Shawe-Taylor. 2000. *Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge. [319]

Croft, W. Bruce. 1978. A file organization for cluster-based retrieval. In *Proc. SIGIR,* pp. 65-82. ACM Press. [344]

Croft, W. Bruce, and David J. Harper. 1979. Using probabilistic models of document retrieval without relevance information. *Journal of Documentation* 35(4):285-295. [122, 209]

Croft, W. Bruce, and John Lafferty (eds.). 2003. *Language Modeling for Information Retrieval.* Springer. [232]

Crouch, Carolyn J. 1988. A cluster-based approach to thesaurus construction. In *Proc. SIGIR,* pp. 309-320. ACM Press. DOI: http://doi.acm.org/10.1145/62437.62467. [345]

Cucerzan, Silviu, and Eric Brill. 2004. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proc. EmpiricalMethods in Natural Language Processing.* [60]

Cutting, Douglas R., David R. Karger, and Jan O. Pedersen. 1993. Constant interaction-time Scatter/Gather browsing of very large document collections. In *Proc. SIGIR,* pp. 126-134. ACM Press. [367]

Cutting, Douglas R., Jan O. Pedersen, David Karger, and John W. Tukey. 1992. Scatter/ Gather: A cluster-based approach to browsing large document collections. In *Proc. SIGIR,* pp. 318-329. ACM Press. [344, 367]

Damerau, Fred J. 1964. A technique for computer detection and correction of spelling errors. *CACM* 7(3):171-176. DOI: http://doi.acm.org/10.1145/363958.363994. [59]

Davidson, Ian, and Ashwin Satyanarayana. 2003. Speeding up k-means clustering by bootstrap averaging. In *ICDM 2003 Workshop on Clustering Large Data Sets.* [345]

Day, William H., and Herbert Edelsbrunner. 1984. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification* 1:1-24. [367]

de Moura, Edleno Silva, Gonzalo Navarro, Nivio Ziviani, and Ricardo Baeza-Yates. 2000. Fast and flexible word searching on compressed text. *TOIS* 18(2):113-139. DOI: http://doi.acm.org/10.1145/348751.348754. [99]

Dean, Jeffrey, and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Symposium on Operating System Design and Implementation.* [69, 76]

Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *JASIS* 41(6):391-407. [383]

del Bimbo, Alberto. 1999. *Visual Information Retrieval.* Morgan Kaufmann. [xviii]

Dempster, A.P., N.M. Laird, and D.B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B* 39: 1-38. [345]

Dhillon, Inderjit S. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proc. KDD,* pp. 269-274. [345, 368]

Dhillon, Inderjit S., and Dharmendra S. Modha. 2001. Concept decompositions for large sparse text data using clustering. *Machine Learning* 42(1/2):143-175. DOI: http://dx.doi.org/10.1023/A:1007612920971. [345]

Di Eugenio, Barbara, and Michael Glass. 2004. The kappa statistic: A second look. *Computational Linguistics* 30(1):95-101. DOI: http://dx.doi.org/10.1162/ 089120104773633402. [160]

Dietterich, Thomas G. 2002. Ensemble learning. In Michael A. Arbib (ed.), *The Handbook of Brain Theory and*

Neural Networks. 2nd edition. MIT Press. [319]

Dietterich, Thomas G., and Ghulum Bakiri. 1995. Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research 2: 263-286. [292]

Dom, Byron E. 2002. An information-theoretic external cluster-validity measure. In *Proc. UAI*. [344]

Domingos, Pedro. 2000. A unified bias-variance decomposition for zero-one and squared loss. In *Proc. National Conference on Artificial Intelligence and Proc. Confer ence Innovative Applications of Artificial Intelligence,* pp. 564-569. AAAI Press/The MIT Press. [292]

Domingos, Pedro, and Michael J. Pazzani. 1997. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* 29(2-3):103-130. URL: citeseer.ist.psu.edu/domingos97optimality.html. [265]

Downie, J. Stephen. 2006. The Music Information Retrieval Evaluation eXchange (MIREX). *D-Lib Magazine* 12(12). [xviii]

Duda, Richard O., Peter E. Hart, and David G. Stork. 2000. *Pattern Classification,* 2nd edition. Wiley-Interscience. [264, 343]

Dumais, Susan, John Platt, David Heckerman, and Mehran Sahami. 1998. Inductive learning algorithms and representations for text categorization. In *Proc. CIKM,* pp. 148-155. ACM Press. DOI: http://doi.acm.org/10.1145/288627.288651. [261, 306, 319]

Dumais, Susan T. 1993. Latent semantic indexing (LSI) and TREC-2. In *Proc. TREC,* pp. 105-115. [382, 383]

Dumais, Susan T. 1995. Latent semantic indexing (LSI): TREC-3 report. In *Proc. TREC,* pp. 219-230. [382, 383]

Dumais, Susan T., and Hao Chen. 2000. Hierarchical classification of Web content. In *Proc. SIGIR,* pp. 256-263. ACM Press. [319]

Dunning, Ted. 1993. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics* 19(1):61-74. [265]

Dunning, Ted. 1994. Statistical identification of language. Technical Report 94-273, Computing Research Laboratory, New Mexico State University. [43]

Eckart, Carl, and Gale Young. 1936. The approximation of a matrix by another of lower rank. *Psychometrika* 1:211-218. [383]

El-Hamdouchi, Abdelmoula, and PeterWillett. 1986. Hierarchic document classification using Ward's clustering method. In *Proc. SIGIR,* pp. 149-156. ACM Press. DOI: http://doi.acm.org/10.1145/253168.253200. [367]

Elias, Peter. 1975. Universal code word sets and representations of the integers. *IEEE Transactions on Information Theory* 21(2):194-203. [98]

Eyheramendy, Susana, David Lewis, and David Madigan. 2003. On the Naive Bayes model for text categorization. In *Proc. International Workshop on Artificial Intelligence and Statistics.* Society for Artificial Intelligence and Statistics. [265]

Fallows, Deborah, 2004. The internet and daily life. URL: www.pewinternet.org/pdfs/PIP_ Internet_and_Daily_ Life.pdf. Pew/Internet and American Life Project. [xv]

Fayyad, Usama M., Cory Reina, and Paul S. Bradley. 1998. Initialization of iterative refinement clustering algorithms. In *Proc. KDD,* pp. 194-198. [345]

Fellbaum, Christiane D. 1998. *WordNet - An Electronic Lexical Database.* MIT Press. [177]

Ferragina, Paolo, and Rossano Venturini. 2007. Compressed permuterm indexes. In *Proc. SIGIR*. ACM Press. [59]

Forman, George. 2004. A pitfall and solution in multi-class feature selection for text classification. In *Proc. ICML*. [265]

Forman, George. 2006. Tackling concept drift by temporal inductive transfer. In *Proc. SIGIR,* pp. 252-259. ACM Press. DOI: http://doi.acm.org/10.1145/1148170.1148216. [265]

Forman, George, and Ira Cohen. 2004. Learning from little: Comparison of classifiers given little training. In *Proc.* pp. 161-172. [308]

Fowlkes, Edward B., and Colin L. Mallows. 1983. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association* 78(383):553-569. URL: www.jstor.org/view/01621459/di985957/98p0926l/0. [368]

Fox, Edward A., and Whay C. Lee. 1991. *FAST-INV: A fast algorithm for building large inverted files.* Technical report, Virginia Polytechnic Institute & State University, Blacksburg, VA, USA. [76]

Fraenkel, Aviezri S., and Shmuel T. Klein. 1985. Novel compression of sparse bit-strings - Preliminary report. In *Combinatorial Algorithms on Words, NATO ASI Series Vol F12,* pp. 169-183. Springer. [98]

Frakes, William B., and Ricardo Baeza-Yates (eds.). 1992. *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall. [451, 461]

Fraley, Chris, and Adrian E. Raftery. 1998. How many clusters? Which clustering method? Answers via model-based cluster analysis. *Computer Journal* 41(8):578-588. [345]

Friedl, Jeffrey E. F. 2006. *Mastering Regular Expressions,* 3rd edition. O'Reilly. [17]

Friedman, Jerome H. 1997. On bias, variance, 0/1.loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery* 1(1):55-77. [265, 292]

Friedman, Nir, and Moises Goldszmidt. 1996. Building classifiers using bayesian networks. In *Proc. National Conference on Artificial Intelligence,* pp. 1277-1284. [213]

Fuhr, Norbert. 1989. Optimum polynomial retrieval functions based on the probability ranking principle. *TOIS* 7(3):183-204. [138]

Fuhr, Norbert. 1992. Probabilistic models in information retrieval. *Computer Journal* 35 (3):243-255. [216, 320]

Fuhr, Norbert, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas (eds.). 2003a. *INitiative for the Evaluation of XML Retrieval (INEX). Proc. First INEX Workshop.* ERCIM. [198]

Fuhr, Norbert, and Kai Großjohann. 2004. XIRQL: An XML query language based on information retrieval concepts. *TOIS* 22(2):313-356. URL: http://doi.acm.org/ 10.1145/984321.984326. [198]

Fuhr, Norbert, and Mounia Lalmas. 2007. Advances in XML retrieval: The INEX initiative. In *Proc. International Workshop on Research Issues in Digital Libraries.* [198]

Fuhr, Norbert, Mounia Lalmas, Saadia Malik, and Gabriella Kazai (eds.). 2006. *Advances in XML Information Retrieval and Evaluation, 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005.* Springer. [198]

Fuhr, Norbert, Mounia Lalmas, Saadia Malik, and Zoltán Szlávik (eds.). 2005. *Advances in XML Information Retrieval, Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004.* Springer. [198, 460, 465]

Fuhr, Norbert, Mounia Lalmas, and Andrew gTrotman (eds.). 2007. Comparative Evaluation of XML Information Retrieval Systems, *5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006*. Springer. [198, 456, 458]

Fuhr, Norbert, Saadia Malik, and Mounia Lalmas (eds.). 2003b. *INEX 2003 Workshop Proceedings.* URL: http://inex.is.informatik.uni-duisburg.de:2003/proceedings.pdf. [198, 451, 458]

Fuhr, Norbert, and Ulrich Pfeifer. 1994. Probabilistic information retrieval as a combination of abstraction, inductive learning, and probabilistic assumptions. *TOIS* 12 (1):92-115. DOI: http://doi.acm.org/10.1145/ 174608.174612. [138]

Fuhr, Norbert, and Thomas Rölleke. 1997. A probabilistic relational algebra for the integration of information retrieval and database systems. *TOIS* 15(1):32-66. DOI: http://doi.acm.org/10.1145/239041.239045. [200]

Gaertner, Thomas, John W. Lloyd, and Peter A. Flach. 2002. Kernels for structured data. In *International Conference on Inductive Logic Programming,* pp. 66-83. [319]

Gao, Jianfeng, Mu Li, Chang-Ning Huang, and Andi Wu. 2005. Chinese word segmentation and named entity recognition: A pragmatic approach. *Computational Linguistics* 31(4):531-574. [43]

Gao, Jianfeng, Jian-Yun Nie, Guangyuan Wu, and Guihong Cao. 2004. Dependence language model for information retrieval. In *Proc. SIGIR,* pp. 170.177. ACM Press.

Garcia, Steven, Hugh E.Williams, and Adam Cannane. 2004. Access-ordered indexes. In *Proc. Australasian conference on Computer science,* pp. 7-14. [137]

Garcia-Molina, Hector, Jennifer Widom, and Jeffrey D. Ullman. 1999. *Database System Implementation. Prentice-Hall.* [77]

Garfield, Eugene. 1955. Citation indexes to science: A new dimension in documentation through association of ideas. *Science* 122:108-111. [439]

Garfield, Eugene. 1976. The permuterm subject index: An autobiographic review. *JASIS* 27(5-6):288-291. [59]

Geman, Stuart, Elie Bienenstock, and René Doursat. 1992. Neural networks and the bias/variance dilemma. *Neural*

Computation 4(1):1-58. [292]

Geng, Xiubo, Tie-Yan Liu, Tao Qin, and Hang Li. 2007. Feature selection for ranking. In *Proc. SIGIR,* pp. 407-414. ACM Press. [320]

Gerrand, Peter. 2007. Estimating linguistic diversity on the internet: A taxonomy to avoid pitfalls and paradoxes. *Journal of Computer-Mediated Communication* 12(4). URL: http://jcmc.indiana.edu/ vol12/issue4/ gerrand.html. article 8. [29]

Gey, Fredric C. 1994. Inferring probability of relevance using the method of logistic regression. In *Proc. SIGIR,* pp. 222-231. ACM Press. [320]

Ghamrawi, Nadia, and Andrew McCallum. 2005. Collective multi-label classification. In *Proc. CIKM,* pp. 195-200. ACM Press. DOI: http://doi.acm.org/10.1145/1099554. 1099591. [292]

Glover, Eric, David M. Pennock, Steve Lawrence, and Robert Krovetz. 2002a. Inferring hierarchical descriptions. In *Proc. CIKM,* pp. 507-514. ACM Press. DOI: http://doi.acm.org/10.1145/584792.584876. [368]

Glover, Eric J., Kostas Tsioutsiouliklis, Steve Lawrence, David M. Pennock, and Gary W. Flake. 2002b. Using web structure for classifying and describing web pages. In *Proc. WWW,* pp. 562-569. ACM Press. doi: http://doi.acm.org/10.1145/ 511446.511520. [367]

Gövert, Norbert, and Gabriella Kazai. 2003. Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002. In Fuhr et al. (2003b), pp. 1-17. URL: http://inex.is.informatik.uni-duisburg.de:2003/ proceedings.pdf. [198]

Grabs, Torsten, and Hans-Jörg Schek. 2002. Generating vector spaces on-the-fly for flexible XML retrieval. In *XML and Information Retrieval Workshop at SIGIR 2002.* [199]

Greiff, Warren R. 1998. A theory of term weighting based on exploratory data analysis. In *Proc. SIGIR,* pp. 11-19. ACM Press. [209]

Grinstead, CharlesM., and J. Laurie Snell. 1997. *Introduction to Probability,* 2nd edition. American Mathematical Society. URL: www.dartmouth.edu/~chance/teaching aids/ books articles/probability book/amsbook. mac.pdf. [216]

Grossman, David A., and Ophir Frieder. 2004. *Information Retrieval: Algorithms and Heuristics,* 2nd edition. Springer. [xviii, 76, 200]

Gusfield, Dan. 1997. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press. [60]

Hamerly, Greg, and Charles Elkan. 2003. Learning the *k* in *k*-means. In *NIPS.* URL: http://books.nips.cc/ papers/files/nips16/NIPS2003_AA36.pdf . [345]

Han, Eui-Hong, and George Karypis. 2000. Centroid-based document classification: Analysis and experimental results. In *PKDD,* pp. 424-431. [291]

Hand, David J. 2006. Classifier technology and the illusion of progress. *Statistical Science* 21:1-14. [265]

Hand, David J., and Keming Yu. 2001. Idiot's Bayes: Not so stupid after all. *International Statistical Review* 69(3):385-398. [265]

Harman, Donna. 1991. How effective is suffixing? *JASIS* 42:7-15. [43]

Harman, Donna. 1992. Relevance feedback revisited. In *Proc. SIGIR,* pp. 1-10. ACM Press. [170, 177]

Harman, Donna, Ricardo Baeza-Yates, Edward Fox, and W. Lee. 1992. Inverted files. In Frakes and Baeza-Yates (1992), pp. 28-43. [76]

Harman, Donna, and Gerald Candela. 1990. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *JASIS* 41(8):581-589. [76]

Harold, Elliotte Rusty, and Scott W. Means. 2004. *XML in a Nutshell,* 3rd edition. O'Reilly. [198]

Harter, Stephen P. 1998. Variations in relevance assessments and the measurement of retrieval effectiveness. *JASIS* 47:37-49. [160]

Hartigan, J. A., andM. A.Wong. 1979. A K-means clustering algorithm. *Applied Statistics* 28:100.108. [345]

Hastie, Trevor, Robert Tibshirani, and Jerome H. Friedman. 2001. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer. [264, 265, 291, 292, 319]

Hatzivassiloglou, Vasileios, Luis Gravano, and Ankineedu Maganti. 2000. An investigation of linguistic features and clustering algorithms for topical document clustering. In *Proc. SIGIR,* pp. 224-231. ACM Press. DOI:

http://doi.acm.org/10.1145/ 345508.345582. [344]

Haveliwala, Taher. 2003. Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering* 15(4): 784-796. URL: citeseer.ist.psu.edu/article/ haveliwala03topicsensitive.html. [439]

Haveliwala, Taher H. 2002. Topic-sensitive PageRank. In *Proc. WWW*. URL: citeseer.ist.psu.edu/haveliwala02 topicsensitive.html. [439]

Hayes, Philip J., and Steven P. Weinstein. 1990. CONSTRUE/TIS: A system for content-based indexing of a database of news stories. In *Proc. Conference on Innovative Applications of Artificial Intelligence,* pp. 49-66. [308]

Heaps, Harold S. 1978. *Information Retrieval: Computational and Theoretical Aspects.* Academic Press. [97]

Hearst,Marti A. 1997. TextTiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics* 23(1):33-64. [199]

Hearst, Marti A. 2006. Clustering versus faceted categories for information exploration. *CACM* 49(4):59-61. doi: http://doi.acm.org/10.1145/1121949.1121983. [344]

Hearst, Marti A., and Jan O. Pedersen. 1996. Reexamining the cluster hypothesis. In *Proc. SIGIR,* pp. 76-84. ACM Press. [344]

Hearst, Marti A., and Christian Plaunt. 1993. Subtopic structuring for full-length document access. In *Proc. SIGIR,* pp. 59-68. ACM Press. DOI: http://doi.acm.org/ 10.1145/160688.160695. [199]

Heinz, Steffen, and Justin Zobel. 2003. Efficient single-pass index construction for text databases. *JASIST* 54(8):713-729. DOI: http://dx.doi.org/10.1002/asi.10268. [76]

Heinz, Steffen, Justin Zobel, and Hugh E. Williams. 2002. Burst tries: A fast, efficient data structure for string keys. *TOIS* 20(2):192-223. DOI: http://doi.acm. org/10.1145/506309.506312. [77]

Henzinger, Monika R., Allan Heydon, MichaelMitzenmacher, andMarc Najork. 2000. On near-uniform URL sampling. In *Proc. WWW,* pp. 295-308. North-Holland. DOI: http://dx.doi.org/10.1016/S1389-1286(00) 00055-4. [404]

Herbrich, Ralf, Thore Graepel, and Klaus Obermayer. 2000. Large margin rank boundaries for ordinal regression. *In Advances in Large Margin Classifiers*, pp. 115-132. MIT Press. [320]

Hersh,William, Chris Buckley, T. J. Leone, and David Hickam. 1994. OHSUMED: An interactive retrieval evaluation and new large test collection for research. In *Proc. SIGIR,* pp. 192-201. ACM Press. [160]

Hersh, William R., Andrew Turpin, Susan Price, Benjamin Chan, Dale Kraemer, Lynetta Sacherek, and Daniel Olson. 2000a. Do batch and user evaluation give the same results? In *Proc. SIGIR,* pp. 17-24.

Hersh, William R., Andrew Turpin, Susan Price, Dale Kraemer, Daniel Olson, Benjamin Chan, and Lynetta Sacherek. 2001. Challenging conventional assumptions of automated information retrieval with real users: Boolean searching and batch retrieval evaluations. *IP&M* 37(3):383-402.

Hersh,William R., Andrew Turpin, Lynetta Sacherek, Daniel Olson, Susan Price, Benjamin Chan, and Dale Kraemer. 2000b. Further analysis of whether batch and user evaluations give the same results with a question-answering task. In *Proc. TREC.*

Hiemstra, Djoerd. 1998. A linguistically motivated probabilistic model of information retrieval. In *Proc. ECDL,* pp. 569-584.

Hiemstra, Djoerd. 2000. A probabilistic justification for using tf.idf term weighting in information retrieval. *International Journal on Digital Libraries* 3(2):131-139.

Hiemstra, Djoerd, and Wessel Kraaij. 2005. A language-modeling approach to TREC. In Voorhees and Harman (2005), pp. 373-395. [232, 233]

Hirai, Jun, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. 2000. WebBase: A repository of web pages. In *Proc. WWW,* pp. 277-293. [419]

Hofmann, Thomas. 1999a. Probabilistic Latent Semantic Indexing. In *UAI.* url: citeseer.ist.psu.edu/hofmann99 probabilistic.html.

Hofmann, Thomas. 1999b. Probabilistic Latent Semantic Indexing. In *Proc. SIGIR,* pp. 50.57. ACM Press. url: citeseer.ist.psu.edu/article/hofmann99probabilistic.html.

Hollink, Vera, Jaap Kamps, Christof Monz, and Maarten de Rijke. 2004. Monolingual document retrieval for

European languages. *IR* 7(1):33-52. [43]

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2000. *Introduction to Automata Theory, Languages, and Computation,* 2nd edition. Addison Wesley. [17]

Huang, Yifen, and Tom M. Mitchell. 2006. Text clustering with extended user feedback. In *Proc. SIGIR,* pp. 413-420. ACM Press. DOI: http://doi.acm.org/10.1145/ 1148170.1148242. [345]

Hubert, Lawrence, and Phipps Arabie. 1985. Comparing partitions. *Journal of Classification* 2:193-218. [344]

Hughes, Baden, Timothy Baldwin, Steven Bird, Jeremy Nicholson, and Andrew MacKinlay. 2006. Reconsidering language identification for written language resources. In *International Conference on Language Resources and Evaluation,* pp. 485-488. [43]

Hull, David. 1993. Using statistical testing in the evaluation of retrieval performance. In *Proc. SIGIR,* pp. 329-338. ACM Press. [159]

Hull, David. 1996. Stemming algorithms - A case study for detailed evaluation. *JASIS* 47(1):70.84. [43]

Ide, E. 1971. New experiments in relevance feedback. In Salton (1971b), pp. 337-354. [177]

Indyk, Piotr. 2004. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke (eds.), *Handbook of Discrete and Computational Geometry,* 2nd edition. pp. 877-892. Chapman and Hall/CRC Press. [291]

Ingwersen, Peter, and Kalervo Järvelin. 2005. *The Turn: Integration of Information Seeking and Retrieval in Context.* Springer. [xviii]

Ittner, David J., David D. Lewis, and David D. Ahn. 1995. Text categorization of low quality images. In *Proc. Annual Symposium on Document Analysis and Information Retrieval,* pp. 301-315. [291]

Iwayama, Makoto, and Takenobu Tokunaga. 1995. Cluster-based text categorization: A comparison of category search strategies. In *Proc. SIGIR,* pp. 273-280. ACM Press. [291]

Jackson, Peter, and Isabelle Moulinier. 2002. *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization.* John Benjamins. [307]

Jacobs, Paul S., and Lisa F. Rau. 1990. SCISOR: Extracting information from on-line news. *CACM* 33:88-97. [308]

Jain, Anil, M. Narasimha Murty, and Patrick Flynn. 1999. Data clustering: A review. *ACM Computing Surveys* 31(3):264-323. [367]

Jain, Anil K., and Richard C. Dubes. 1988. *Algorithms for Clustering Data*. Prentice-Hall. [367]

Jardine, N., and C. J. van Rijsbergen. 1971. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval* 7:217-240. [344]

Järvelin, Kalervo, and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *TOIS* 20(4):422-446. [160]

Jeh, Glen, and JenniferWidom. 2003. Scaling personalized web search. In *Proc.WWW*, pp. 271-279. ACM Press. [439]

Jensen, Finn V., and Finn B. Jensen. 2001. *Bayesian Networks and Decision Graphs.* Springer. [215]

Jeong, Byeong-Soo, and Edward Omiecinski. 1995. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems* 6(2): 142-153. [419]

Ji, Xiang, and Wei Xu. 2006. Document clustering with prior knowledge. In *Proc. SIGIR,* pp. 405-412. ACM Press. doi: http://doi.acm.org/10.1145/1148170.1148241. [345]

Jing, Hongyan. 2000. Sentence reduction for automatic text summarization. In *Proc. Conference on applied natural language processing,* pp. 310.315. [161]

Joachims, Thorsten. 1997. A probabilistic analysis of the Rocchio algorithm with tfidf for text categorization. In *Proc. ICML,* pp. 143-151. Morgan Kaufmann. [291]

Joachims, Thorsten. 1998. Text categorization with support vector machines: Learning with many relevant features. In *Proc. ECML,* pp. 137-142. Springer. [261, 306, 307]

Joachims, Thorsten. 1999. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola (eds.), *Advances in Kernel Methods-Support Vector Learning.* MIT Press. [319]

Joachims, Thorsten. 2002a. *Learning to Classify Text Using Support Vector Machines*. Kluwer. [306, 307, 319]

Joachims, Thorsten. 2002b. Optimizing search engines using clickthrough data. In *Proc. KDD,* pp. 133-142. [161, 170, 320]

Joachims, Thorsten. 2006a. Training linear SVMs in linear time. In *Proc. KDD,* pp. 217-226. ACM Press. DOI: http://doi.acm.org/10.1145/1150402.1150429. [265, 302, 319]

Joachims, Thorsten. 2006b. Transductive support vector machines. In Chapelle et al. (2006), pp. 105-118. [320]

Joachims, Thorsten, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. 2005. Accurately interpreting clickthrough data as implicit feedback. In *Proc. SIGIR,* pp. 154-161. ACM Press. [161, 170]

Johnson, David, Vishv Malhotra, and Peter Vamplew. 2006. More effective web search using bigrams and trigrams. *Webology* 3(4). URL: www.webology.ir/2006/v3n4/a35.html. [44]

Jurafsky, Dan, and James H.Martin. 2008. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition,* 2nd edition. Prentice-Hall. [xviii]

Käki, Mika. 2005. Findex: Search result categories help users when document ranking fails. In *Proc. SIGCHI Conference on Human Factors in Computing Systems,* pp. 131-140. ACM Press. DOI: http://doi.acm.org/10.1145/1054972.1054991. [344, 368]

Kammenhuber, Nils, Julia Luxenburger, Anja Feldmann, and Gerhard Weikum. 2006. Web search clickstreams. In *ACM SIGCOMM on Internet Measurement,* pp. 245-250. ACM Press. [44]

Kamps, Jaap, Maarten de Rijke, and Börkur Sigurbjörnsson. 2004. Length normalization in XML retrieval. In *Proc. SIGIR,* pp. 80.87. ACM Press. DOI: http:// doi.acm.org/10.1145/1008992.1009009. [199]

Kamps, Jaap, Maarten Marx, Maarten de Rijke, and Börkur Sigurbjörnsson. 2006. Articulating information needs in XML query languages. *TOIS* 24(4):407-436. DOI: http://doi.acm.org/10.1145/1185877.1185879. [198]

Kamvar, Sepandar D., Dan Klein, and Christopher D. Manning. 2002. Interpreting and extending classical agglomerative clustering algorithms using a model-based approach. In *Proc. ICML,* pp. 283-290. Morgan Kaufmann. [368]

Kannan, Ravi, Santosh Vempala, and Adrian Vetta. 2000. On clusterings - Good, bad and spectral. In *Proc. Annual Symposium on Foundations of Computer Science,* p. 367. IEEE Computer Society. [368]

Kaszkiel, Marcin, and Justin Zobel. 1997. Passage retrieval revisited. In *Proc. SIGIR,* pp. 178-185. ACM Press. DOI: http://doi.acm.org/10.1145/258525.258561. [199]

Kaufman, Leonard, and Peter J. Rousseeuw. 1990. *Finding groups in data.* Wiley. [345]

Kazai, Gabriella, and Mounia Lalmas. 2006. eXtended cumulated gain measures for the evaluation of content-oriented XML retrieval. *TOIS* 24(4):503-542. DOI: http://doi.acm.org/10.1145/1185883. [199]

Kekäläinen, Jaana. 2005. Binary and graded relevance in IR evaluations - Comparison of the effects on ranking of IR systems. *IP&M* 41:1019-1033. [160]

Kekäläinen, Jaana, and Kalervo Järvelin. 2002. Using graded relevance assessments in IR evaluation. *JASIST* 53(13):1120.1129. [160]

Kemeny, John G., and J. Laurie Snell. 1976. *Finite Markov Chains.* Springer. [439]

Kent, Allen, Madeline M. Berry, Fred U. Luehrs, Jr., and J. W. Perry. 1955. Machine literature searching VIII. Operational criteria for designing information retrieval systems. *American Documentation* 6(2):93-101. [159]

Kernighan, Mark D., Kenneth W. Church, and William A. Gale. 1990. A spelling correction program based on a noisy channel model. In *Proc. ACL,* pp. 205-210. [60]

King, Benjamin. 1967. Step-wise clustering procedures. *Journal of the American Statistical Association* 69:86-101. [367]

Kishida, Kazuaki, Kuang Hua Chen, Sukhoon Lee, Kazuko Kuriyama, Noriko Kando, Hsin-Hsi Chen, and Sung Hyon Myaeng. 2005. Overview of CLIR task at the fifth NTCIR workshop. In *Proc. NTCIR Workshop Meeting on Evaluation of Information Access Technologies: Information Retrieval, Question Answering and Cross-Lingual Information Access*. National Institute of Informatics. [43]

Klein, Dan, and Christopher D. Manning. 2002. Conditional structure versus conditional estimation in NLP models. In *Proc. Empirical Methods in Natural Language Processing,* pp. 9-16. [308]

Kleinberg, Jon M. 1997. Two algorithms for nearest-neighbor search in high dimensions. In *Proc. Annual ACM Symposium on Theory of Computing,* pp. 599-608. ACM Press. DOI: http://doi.acm.org/10.1145/258533.258653. [291]

Kleinberg, Jon M. 1999. Authoritative sources in a hyperlinked environment. *JACM* 46(5):604-632. URL: citeseer.ist.psu.edu/article/kleinberg98authoritative.html. [439]

Kleinberg, Jon M. 2002. An impossibility theorem for clustering. In *Proc. NIPS*. [345]

Knuth, Donald E. 1997. *The Art of Computer Programming, Volume 3: Sorting and Searching,* 3rd edition. Addison-Wesley. [59]

Ko, Youngjoong, Jinwoo Park, and Jungyun Seo. 2004. Improving text categorization using the importance of sentences. *IP&M* 40(1):65-79. [313]

Koenemann, Jürgen, and Nicholas J. Belkin. 1996. A case for interaction: A study of interactive information retrieval behavior and effectiveness. In *Proc. SIGCHI Conference on Human Factors in Computing Systems,* pp. 205-212. ACM Press. DOI: http://doi.acm.org/10.1145/238386.238487. [177]

Kołcz, Aleksander, Vidya Prabakarmurthi, and Jugal Kalita. 2000. Summarization as feature selection for text categorization. In *Proc. CIKM,* pp. 365-370. ACM Press. [313]

Kołcz, Aleksander, and Wen-Tau Yih. 2007. Raising the baseline for high-precision text classifiers. In *Proc. KDD.* [265]

Koller, Daphne, and Mehran Sahami. 1997. Hierarchically classifying documents using very few words. In *Proc. ICML,* pp. 170.178. [319]

Konheim, Alan G. 1981. *Cryptography: A Primer.* John Wiley & Sons. [43]

Korfhage, Robert R. 1997. *Information Storage and Retrieval.* Wiley. [161]

Kozlov,M. K., S. P. Tarasov, and L. G. Khachiyan. 1979. Polynomial solvability of convex quadratic programming. *Soviet Mathematics Doklady* 20:1108-1111. Translated from original in *Doklady Akademiia Nauk SSR,* 228 (1979). [302]

Kraaij, Wessel, and Martijn Spitters. 2003. Language models for topic tracking. In W. B. Croft and J. Lafferty (eds.), *Language Modeling for Information Retrieval,* pp. 95-124. Kluwer. [231]

Kraaij,Wessel, Thijs Westerveld, and Djoerd Hiemstra. 2002. The importance of prior probabilities for entry page search. In *Proc. SIGIR,* pp. 27-34. ACM Press. [233]

Krippendorff, Klaus. 2003. *Content Analysis: An Introduction to its Methodology.* Sage. [160]

Krovetz, Bob. 1995. *Word sense disambiguation for large text databases.* PhD thesis, University of Massachusetts Amherst. [43]

Kukich, Karen. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys* 24(4):377-439. DOI: http://doi.acm.org/10.1145/146370.146380. [59]

Kumar, Ravi, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. 1999. Trawling the Web for emerging cyber-communities. *Computer Networks* 31(11-16): 1481-1493. URL: citeseer.ist.psu.edu/kumar99 trawling.html. [404]

Kumar, S. Ravi, Prabhakar Raghavan, Sridhar Rajagopalan, Dandapani Sivakumar, Andrew Tomkins, and Eli Upfal. 2000. The Web as a graph. In *Proc. PODS,* pp. 1-10. ACM Press. URL: citeseer.ist.psu.edu/article/ kumar00web.html. [404]

Kupiec, Julian, Jan Pedersen, and Francine Chen. 1995. A trainable document summarizer. In *Proc. SIGIR,* pp. 68-73. ACM Press. [160]

Kurland, Oren, and Lillian Lee. 2004. Corpus structure, language models, and ad hoc information retrieval. In *Proc. SIGIR,* pp. 194-201. ACM Press. DOI: http://doi.acm.org/10.1145/1008992.1009027. [344]

Lafferty, John, and Chengxiang Zhai. 2001. Document language models, query models, and risk minimization for information retrieval. In *Proc. SIGIR,* pp. 111-119. ACM Press. [231]

Lafferty, John, and Chengxiang Zhai. 2003. Probabilistic relevance models based on document and query generation. In W. Bruce Croft and John Lafferty (eds.), *Language Modeling and Information Retrieval.* Kluwer. [233]

Lalmas, Mounia, Gabriella Kazai, Jaap Kamps, Jovan Pehcevski, Benjamin Piwowarski, and Stephen E. Robertson. 2007. INEX 2006 evaluation measures. In Fuhr et al. (2007), pp. 20.34. [199]

Lalmas, Mounia, and Anastasios Tombros. 2007. Evaluating XML retrieval effectiveness at INEX. *SIGIR Forum* 41(1):40.57. DOI: http://doi.acm.org/10.1145/ 1273221.1273225. [198]

Lance, G. N., and W. T. Williams. 1967. A general theory of classificatory sorting strategies 1. Hierarchical systems. *Computer Journal* 9(4):373-380. [367]

Langville, Amy, and Carl Meyer. 2006. *Google's PageRank and Beyond: The Science of Search Engine Rankings.*

Princeton University Press. [439]

Larsen, Bjornar, and Chinatsu Aone. 1999. Fast and effective text mining using linear-time document clustering. In *Proc. KDD,* pp. 16-22. ACM Press. DOI: http://doi.acm.org/10.1145/312129.312186. [367, 368]

Larson, Ray R. 2005. A fusion approach to XML structured document retrieval. *IR* 8 (4):601-629. DOI: http://dx.doi.org/10.1007/s10791-005-0749-0. [199]

Lavrenko, Victor, and W. Bruce Croft. 2001. Relevance-based language models. In *Proc. SIGIR,* pp. 120-127. ACM Press. [231]

Lawrence, Steve, and C. Lee Giles. 1998. Searching the World Wide Web. *Science* 280 (5360):98-100. URL: citeseer.ist.psu.edu/lawrence98searching.html. [404]

Lawrence, Steve, and C. Lee Giles. 1999. Accessibility of information on the web. *Nature* 500:107-109. [404]

Lee, Whay C., and Edward A. Fox. 1988. *Experimental comparison of schemes for interpreting Boolean queries.* Technical Report TR-88-27, Computer Science, Virginia Polytechnic Institute and State University. [17]

Lempel, Ronny, and Shlomo Moran. 2000. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Computer Networks* 33(1-6):387-401. URL: citeseer.ist.psu.edu/lempel00stochastic.html. [440]

Lesk, Michael. 1988. Grab - Inverted indexes with low storage overhead. *Computing Systems* 1:207-220. [76]

Lesk, Michael. 2004. *Understanding Digital Libraries,* 2nd edition. Morgan Kaufmann. [xviii]

Lester, Nicholas, Alistair Moffat, and Justin Zobel. 2005. Fast on-line index construction by geometric partitioning. In *Proc. CIKM,* pp. 776-783. ACM Press. DOI: http://doi.acm.org/10.1145/1099554.1099739. [76]

Lester, Nicholas, Justin Zobel, and Hugh E. Williams. 2006. Efficient online index maintenance for contiguous inverted lists. *IP&M* 42(4):916-933. doi: http://dx.doi. org/10.1016/j.ipm.2005.09.005 . [76]

Levenshtein, Vladimir I. 1965. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission* 1:8-17. [59]

Lew, Michael S. 2001. *Principles of Visual Information Retrieval.* Springer. [xviii]

Lewis, David D. 1995. Evaluating and optimizing autonomous text classification systems. In *Proc. SIGIR*. ACM Press. [265]

Lewis, David D. 1998. Naive (Bayes) at forty: The independence assumption in information retrieval. In *ECML,* pp. 4-15. Springer. [265]

Lewis, David D., and Karen Spärck Jones. 1996. Natural language processing for information retrieval. *CACM* 39(1):92-101. DOI: http://doi.acm.org/10.1145/ 234173.234210. [xviii]

Lewis, David D., and Marc Ringuette. 1994. A comparison of two learning algorithms for text categorization. In *SDAIR,* pp. 81-93. [265]

Lewis, David D., Robert E. Schapire, James P. Callan, and Ron Papka. 1996. Training algorithms for linear text classifiers. In *Proc. SIGIR,* pp. 298-306. ACM Press. DOI: http://doi.acm.org/10.1145/243199.243277. [292]

Lewis, David D., Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A new benchmark collection for text categorization research. *JMLR* 5:361-397. [77, 265]

Li, Fan, and Yiming Yang. 2003. A loss function analysis for classification methods in text categorization. In *Proc. ICML,* pp. 472-479. [261, 319]

Liddy, Elizabeth D. 2005. Automatic document retrieval. In *Encyclopedia of Language and Linguistics, 2nd edition.* Elsevier.

List, Johan, Vojkan Mihajlovic, Georgina Ramírez, Arjen P. Vries, Djoerd Hiemstra, and Henk Ernst Blok. 2005. TIJAH: Embracing IR methods in XML databases. *IR* 8 (4):547-570. DOI: http://dx.doi.org/10.1007/ s10791-005-0747-2. [199]

Lita, Lucian Vlad, Abe Ittycheriah, Salim Roukos, and Nanda Kambhatla. 2003. tRuEcasIng. In *Proc. ACL,* pp. 152-159. [43]

Littman, Michael L., Susan T. Dumais, and Thomas K. Landauer. 1998. Automatic cross-language information retrieval using latent semantic indexing. In Gregory Grefenstette (ed.), *Cross Language Information Retrieval.* Kluwer. URL: citeseer.ist.psu.edu/littman98automatic.html. [384]

Liu, Tie-Yan, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. 2005. Support vector machines classification with very large scale taxonomy. *ACM SIGKDD Explorations* 7(1):36-43. [319]

Liu, Xiaoyong, and W. Bruce Croft. 2004. Cluster-based retrieval using language models. In *Proc. SIGIR,* pp.

186-193. ACM Press. DOI: http://doi.acm.org/10.1145/ 1008992.1009026. [233, 323, 344]

Lloyd, Stuart P. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28(2):129-136. [345]

Lodhi, Huma, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *JMLR* 2:419-444. [319]

Lombard, Matthew, Cheryl C. Bracken, and Jennifer Snyder-Duch. 2002. Content analysis in mass communication: Assessment and reporting of intercoder reliability. *Human Communication Research* 28:587-604. [160]

Long, Xiaohui, and Torsten Suel. 2003. Optimized query execution in large search engines with global page ordering. In *Proc. VLDB.* URL: citeseer.ist.psu.edu/ long03optimized.html. [137]

Lovins, Julie Beth. 1968. Development of a stemming algorithm. *Translation and Computational Linguistics* 11(1):22-31. [31]

Lu,Wei, Stephen E. Robertson, and Andrew MacFarlane. 2007. CISR at INEX 2006. In Fuhr et al. (2007), pp. 57-63. [199]

Luhn, Hans Peter. 1957. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development* 1(4):309-317. [122]

Luhn, Hans Peter. 1958. The automatic creation of literature abstracts. *IBM Journal of Research and Development* 2(2):159-165, 317. [122]

Luk, Robert W. P., and Kui-Lam Kwok. 2002. A comparison of Chinese document indexing strategies and retrieval models. *ACM Transactions on Asian Language Information Processing* 1(3):225-268. [43]

Lunde, Ken. 1998. *CJKV Information Processing.* O'Reilly. [43]

MacFarlane, A., J.A. McCann, and S.E. Robertson. 2000. Parallel search using partitioned inverted files. In *Proc. SPIRE,* pp. 209-220. [419]

MacQueen, James B. 1967. Some methods for classification and analysis of multivariate observations. In *Proc. Berkeley Symposium on Mathematics, Statistics and Probability,* pp. 281-297. University of California Press. [345]

Manning, Christopher D., and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing.* MIT Press. [xviii, 37, 97, 264, 342]

Maron, M. E., and J. L. Kuhns. 1960. On relevance, probabilistic indexing, and information retrieval. *JACM* 7(3):216-244. [216, 265]

Mass, Yosi, Matan Mandelbrod, Einat Amitay, David Carmel, Yöelle S. Maarek, and Aya Soffer. 2003. JuruXML-An XML retrieval system at INEX'02. In Fuhr et al. (2003b), pp. 73-80. URL: http://inex.is.informatik.uni-duisburg.de:2003/proceedings.pdf. [199]

McBryan, Oliver A. 1994. GENVL and WWWW: Tools for Taming the Web. In *Proc. WWW*. URL: citeseer.ist .psu.edu/mcbryan94genvl.html. [404, 439]

McCallum, Andrew, and Kamal Nigam. 1998. A comparison of event models for Naive Bayes text classification. In *Working Notes of the 1998 AAAI/ICML Workshop on Learning for Text Categorization,* pp. 41-48. [265]

McCallum, Andrew, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. 1998. Improving text classification by shrinkage in a hierarchy of classes. In *Proc. ICML,* pp. 359-367. Morgan Kaufmann. [319]

McCallum, Andrew Kachites. 1996. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. URL: www.cs.cmu.edu/~mccallum/bow. [289]

McKeown, Kathleen, and Dragomir R. Radev. 1995. Generating summaries of multiple news articles. In *Proc. SIGIR,* pp. 74-82. ACM Press. DOI: http://doi.acm.org/ 10.1145/215206.215334. [368]

McKeown, Kathleen R., Regina Barzilay, David Evans, Vasileios Hatzivassiloglou, Judith L. Klavans, Ani Nenkova, Carl Sable, Barry Schiffman, and Sergey Sigelman. 2002. Tracking and summarizing news on a daily basis with Columbia's Newsblaster. In *Proc. Human Language Technology Conference.* [323, 344]

McLachlan, Geoffrey J., and Thiriyambakam Krishnan. 1996. *The EM Algorithm and Extensions.* JohnWiley & Sons. [345]

Meadow, Charles T., Donald H. Kraft, and Bert R. Boyce. 1999. *Text Information Retrieval Systems.* Academic Press.

Meilă, Marina. 2005. Comparing clusterings - An axiomatic view. In *Proc. ICML*. [345]

Melnik, Sergey, Sriram Raghavan, Beverly Yang, and Hector Garcia-Molina. 2001. Building a distributed full-text index for the web. In *Proc.WWW*, pp. 396-406.ACM Press. DOI: http://doi.acm.org/10.1145/371920.372095. [76]

Mihajlović, Vojkan, Henk Ernst Blok, Djoerd Hiemstra, and Peter M. G. Apers. 2005. Score region algebra: Building a transparent XML-R database. In *Proc. CIKM,* pp. 12-19. ACM Press. DOI: http://doi.acm.org/ 10.1145/1099554.1099560. [199]

Miller, David R. H., Tim Leek, and Richard M. Schwartz. 1999. A hidden Markov model information retrieval system. In *Proc. SIGIR,* pp. 214-221. ACM Press.

Minsky, Marvin Lee, and Seymour Papert (eds.). 1988. *Perceptrons: An Introduction to Computational Geometry.* MIT Press. Expanded edition. [292]

Mitchell, Tom M. 1997. *Machine Learning*. McGraw-Hill. [264]

Moffat, Alistair, and Timothy A. H. Bell. 1995. In situ generation of compressed inverted files. *JASIS* 46(7):537-550. [76]

Moffat, Alistair, and Lang Stuiver. 1996. Exploiting clustering in inverted file compression. In *Proc. Conference on Data Compression,* pp. 82-91. IEEE Computer Society. [98]

Moffat, Alistair, and Justin Zobel. 1992. Parameterised compression for sparse bitmaps. In *Proc. SIGIR,* pp. 274-285. ACM Press. DOI: http://doi.acm.org/10.1145/ 133160.133210. [98]

Moffat, Alistair, and Justin Zobel. 1996. Self-indexing inverted files for fast text retrieval. *TOIS* 14(4):349-379. [44]

Moffat, Alistair, and Justin Zobel. 1998. Exploring the similarity space. *SIGIR Forum* 32(1). [123]

Mooers, Calvin. 1961. From a point of view of mathematical etc. techniques. In R. A. Fairthorne (ed.), *Towards Information Retrieval,* pp. xvii.xxiii. Butterworths. [17]

Mooers, Calvin E. 1950. Coding, information retrieval, and the rapid selector. *American Documentation* 1(4):225-229. [16]

Moschitti, Alessandro. 2003. A study on optimal parameter tuning for Rocchio text classifier. In *Proc. ECIR,* pp. 420.435. [292]

Moschitti, Alessandro, and Roberto Basili. 2004. Complex linguistic features for text classification: A comprehensive study. In *Proc. ECIR,* pp. 181-196. [319]

Murata, Masaki, Qing Ma, Kiyotaka Uchimoto, Hiromi Ozaku, Masao Utiyama, and Hitoshi Isahara. 2000. Japanese probabilistic information retrieval using location and category information. In *Proc. International Workshop on Information Retrieval With Asian Languages,* pp. 81-88. URL: http://portal.acm.org/citation.
cfm? doid= 355214.355226. [312]

Muresan, Gheorghe, and David J. Harper. 2004. Topic modeling for mediated access to very large document collections. *JASIST* 55(10):892-910. DOI: http://dx.doi. org/10.1002/asi.20034. [344]

Murtagh, Fionn. 1983. A survey of recent advances in hierarchical clustering algorithms. *Computer Journal* 26(4):354-359. [367]

Najork, Marc, and Allan Heydon. 2001. *High-performance web crawling.* Technical Report 173, Compaq Systems Research Center. [419]

Najork, Marc, and Allan Heydon. 2002. High-performance web crawling. In Panos Pardalos, James Abello and Mauricio Resende (eds.), *Handbook of Massive Data Sets,* chapter 2. Kluwer. [419]

Navarro, Gonzalo, and Ricardo Baeza-Yates. 1997. Proximal nodes: A model to query document databases by content and structure. *TOIS* 15(4):400.435. DOI: http://doi.acm.org/10.1145/263479.263482. [200]

Newsam, Shawn, Sitaram Bhagavathy, and B. S. Manjunath. 2001. Category-based image retrieval. In *IEEE International Conference on Image Processing, Special Session on Multimedia Indexing, Browsing and Retrieval,* pp. 596-599. [164]

Ng, Andrew Y., and Michael I. Jordan. 2001. On discriminative vs. generative classifiers: A comparison of logistic regression and Naive Bayes. In *NIPS,* pp. 841-848. URL: www-2.cs.cmu.edu/Groups/NIPS/NIPS2001/papers/ psgz/AA28.ps.gz. [265, 308]

Ng, Andrew Y.,Michael I. Jordan, and Yair Weiss. 2001a. On spectral clustering: Analysis and an algorithm. In *Proc. NIPS,* pp. 849-856. [368]

Ng, Andrew Y., Alice X. Zheng, and Michael I. Jordan. 2001b. Link analysis, eigenvectors and stability. In *Proc. IJCAI,* pp. 903-910. URL: citeseer.ist.psu.edu/ng01link.html. [439, 440]

Nigam, Kamal, Andrew McCallum, and Tom Mitchell. 2006. Semi-supervised text classification using EM. In Chapelle et al. (2006), pp. 33-56. [320]

Ntoulas, Alexandros, and Junghoo Cho. 2007. Pruning policies for two-tiered inverted index with correctness guarantee. In *Proc. SIGIR,* pp. 191-198. ACM Press. [97]

Oard, Douglas W., and Bonnie J. Dorr. 1996. *A survey of multilingual text retrieval.* Technical Report UMIACS-TR-96-19, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA. [xviii]

Ogilvie, Paul, and Jamie Callan. 2005. Parameter estimation for a simple hierarchical generative model for XML retrieval. In *Proc. INEX,* pp. 211-224. DOI: http://dx.doi.org/10.1007/11766278 16. [199]

ÖKeefe, Richard A., and Andrew Trotman. 2004. The simplest query language that could possibly work. In Fuhr et al. (2005), pp. 167-174. [199]

Osiński, Stanisław, and Dawid Weiss. 2005. A concept-driven algorithm for clustering search results. *IEEE Intelligent Systems* 20(3):48-54. [368]

Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1998. The Page Rank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project. URL: citeseer.ist.psu.edu/page98pagerank.html. [439]

Paice, Chris D. 1990. Another stemmer. *SIGIR Forum* 24(3):56-61. [31]

Papineni, Kishore. 2001. Why inverse document frequency? In *North American Chapter of the Association for Computational Linguistics,* pp. 1-8. [122]

Pavlov, Dmitry, Ramnath Balasubramanyan, Byron Dom, Shyam Kapur, and Jignashu Parikh. 2004. Document preprocessing for naive Bayes classification and clustering with mixture of multinomials. In *Proc. KDD,* pp. 829-834. [265]

Pelleg, Dan, and Andrew Moore. 1999. Accelerating exact k-means algorithms with geometric reasoning. In *Proc. KDD,* pp. 277-281. ACM Press. doi: http://DOI.acm. org/10.1145/312129.312248. [345]

Pelleg, Dan, and Andrew Moore. 2000. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proc. ICML,* pp. 727-734. Morgan Kaufmann. [345]

Perkins, Simon, Kevin Lacker, and James Theiler. 2003. Grafting: Fast, incremental feature selection by gradient descent in function space. *JMLR* 3:1333-1356. [265]

Persin,Michael. 1994. Document filtering for fast ranking. In *Proc. SIGIR,* pp. 339-348. ACM Press. [137]

Persin,Michael, Justin Zobel, and Ron Sacks-Davis. 1996. Filtered document retrieval with frequency-sorted indexes. *JASIS* 47(10):749-764. [137]

Peterson, James L. 1980. Computer programs for detecting and correcting spelling errors. *CACM* 23(12):676-687. doi: http://doi.acm.org/10.1145/359038.359041. [59]

Picca, Davide, Benoît Curdy, and François Bavaud. 2006. Non-linear correspondence analysis in text retrieval: A kernel view. In *Proc. JADT*. [283]

Pinski, Gabriel, and Francis Narin. 1976. Citation influence for journal aggregates of scientific publications: Theory, with application to the literature of Physics. *IP&M* 12:297-326. [439]

Pirolli, Peter L. T. 2007. *Information Foraging Theory: Adaptive Interaction With Information.* Oxford University Press. [344]

Platt, John. 2000. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans (eds.), *Advances in Large Margin Classifiers,* pp. 61-74. MIT Press. [298]

Ponte, Jay M., and W. Bruce Croft. 1998. A language modeling approach to information retrieval. In *Proc. SIGIR,* pp. 275-281. ACM Press. [227, 228, 229]

Popescul, Alexandrin, and Lyle H. Ungar. 2000. Automatic labeling of document clusters. Unpublished. [367]

Porter, Martin F. 1980. An algorithm for suffix stripping. *Program* 14(3):130.137. [31]

Pugh, William. 1990. Skip lists: A probabilistic alternative to balanced trees. *CACM* 33(6):668-676. [44]

Qin, Tao, Tie-Yan Liu, Wei Lai, Xu-Dong Zhang, De-Sheng Wang, and Hang Li. 2007. Ranking with multiple

hyperplanes. In *Proc. SIGIR*. ACM Press. [320]

Qiu, Yonggang, and H.P. Frei. 1993. Concept based query expansion. In *Proc. SIGIR,* pp. 160.169. ACM Press. [177]

R Development Core Team. 2005. *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. url: www.Rproject. org . ISBN 3-900051-07-0. [342, 368]

Radev, Dragomir R., Sasha Blair-Goldensohn, Zhu Zhang, and Revathi Sundara Raghavan. 2001. Interactive, domain-independent identification and summarization of topically related news articles. In *Proc. European Conference on Research and Advanced Technology for Digital Libraries,* pp. 225-238. [344]

Rahm, Erhard, and Philip A. Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDB Journal* 10(4):334-350. URL: citeseer.ist.psu.edu/ rahm01survey.html. [198]

Rand, William M. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association* 66(336):846-850. [344]

Rasmussen, Edie. 1992. Clustering algorithms. In Frakes and Baeza-Yates (1992), pp. 419-442. [344]

Rennie, Jason D., Lawrence Shih, Jaime Teevan, and David R. Karger. 2003. Tackling the poor assumptions of naive Bayes text classifiers. In *Proc. ICML,* pp. 616-623. [265]

Ribeiro-Neto, Berthier, Edleno S.Moura,Marden S. Neubert, and Nivio Ziviani. 1999. Efficient distributed algorithms to build inverted files. In *Proc. SIGIR,* pp. 105-112. ACM Press. DOI: http://doi.acm.org/10.1145/ 312624.312663. [76]

Ribeiro-Neto, Berthier A., and Ramurti A. Barbosa. 1998. Query performance for tightly coupled distributed digital libraries. In *ACM Conference on Digital Libraries,* pp. 182-190. [420]

Rice, John A. 2006. *Mathematical Statistics and Data Analysis.* Duxbury Press. [91, 216, 256]

Richardson, M., A. Prakash, and E. Brill. 2006. Beyond PageRank: machine learning for static ranking. In *Proc. WWW,* pp. 707-715. [320]

Riezler, Stefan, Alexander Vasserman, Ioannis Tsochantaridis, Vibhu Mittal, and Yi Liu. 2007. Statistical machine translation for query expansion in answer retrieval. In *Proc. ACL,* pp. 464-471. Association for Computational Linguistics. URL: www.aclweb.org/anthology/P/P07/P07-1059. [177]

Ripley, B. D. 1996. *Pattern Recognition and Neural Networks.* Cambridge University Press. [204, 216]

Robertson, Stephen. 2005. How Okapi came to TREC. In Voorhees and Harman (2005), pp. 287-299. [216]

Robertson, Stephen, Hugo Zaragoza, and Michael Taylor. 2004. Simple BM25 extension to multiple weighted fields. In *Proc. CIKM,* pp. 42-49. ACM Press. DOI: http://doi.acm.org/10.1145/1031171.1031181. [217]

Robertson, Stephen E., and Karen Spärck Jones. 1976. Relevance weighting of search terms. *JASIS* 27:129-146. [122, 216]

Rocchio, J. J. 1971. Relevance feedback in information retrieval. In Salton (1971b), pp. 313-323. [166, 177, 291]

Roget, P. M. 1946. *Roget's International Thesaurus.* Thomas Y. Crowell. [177]

Rosen-Zvi, Michal, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. 2004. The author-topic model for authors and documents. In *Proc. UAI,* pp. 487-494. AUAI Press. [384]

Ross, Sheldon. 2006. *A First Course in Probability.* Pearson Prentice-Hall. [91, 216]

Rusmevichientong, Paat, David M. Pennock, Steve Lawrence, and C. Lee Giles. 2001. Methods for sampling pages uniformly from the world wide web. In *Proc. AAAI Fall Symposium on Using Uncertainty Within Computation,* pp. 121-128. URL: citeseer.ist.psu.edu/rusmevichientong01methods.html. [404]

Ruthven, Ian, and Mounia Lalmas. 2003. A survey on the use of relevance feedback for information access systems. *Knowledge Engineering Review* 18(1). [177]

Sahoo, Nachiketa, Jamie Callan, Ramayya Krishnan, George Duncan, and Rema Padman. 2006. Incremental hierarchical clustering of text documents. In *Proc. CIKM,* pp. 357-366. ACM Press. DOI: http://doi.acm.org/10.1145/1183614.1183667. [368]

Sakai, Tetsuya. 2007. On the reliability of information retrieval metrics based on graded relevance. *IP&M* 43(2):531-548. [160]

Salton, Gerard. 1971a. Cluster search strategies and the optimization of retrieval effectiveness. In *The SMART Retrieval System - Experiments in Automatic Document Processing* Salton (1971b), pp. 223-242. [323, 344]

Salton, Gerard (ed.). 1971b. *The SMART Retrieval System - Experiments in Automatic Document Processing.*

Prentice-Hall. [122, 159, 177, 453, 461, 462]

Salton, Gerard. 1975. *Dynamic information and library processing.* Prentice-Hall. [344]

Salton, Gerard. 1989. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* AddisonWesley. [43, 177]

Salton, Gerard. 1991. The Smart project in automatic document retrieval. In *Proc. SIGIR,* pp. 356-358. ACM Press. [159]

Salton, Gerard, James Allan, and Chris Buckley. 1993. Approaches to passage retrieval in full text information systems. In *Proc. SIGIR,* pp. 49-58. ACM Press. DOI: http://doi.acm.org/10.1145/160688.160693. [199]

Salton, Gerard, and Chris Buckley. 1987. *Term weighting approaches in automatic text retrieval.* Technical report, Cornell University, Ithaca, NY. [122]

Salton, Gerard, and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *IP&M* 24(5):513-523. [123]

Salton, Gerard, and Chris Buckley. 1990. Improving retrieval performance by relevance feedback. *JASIS* 41(4):288-297. [177]

Saracevic, Tefko, and Paul Kantor. 1988. A study of information seeking and retrieving. II: Users, questions and effectiveness. *JASIS* 39:177-196. [159]

Saracevic, Tefko, and Paul Kantor. 1996. A study of information seeking and retrieving. III: Searchers, searches, overlap. *JASIS* 39(3):197-216. [159]

Savaresi, Sergio M., and Daniel Boley. 2004. A comparative analysis on the bisecting K-means and the PDDP clustering algorithms. *Intelligent Data Analysis* 8(4):345-362. [368]

Schamber, Linda,Michael Eisenberg, andMichael S. Nilan. 1990. A re-examination of relevance: toward a dynamic, situational definition. *IP&M* 26(6):755-776. [160]

Schapire, Robert E. 2003. The boosting approach to machine learning: An overview. In D. D. Denison, M. H. Hansen, C. Holmes, B. Mallick, and B. Yu (eds.), *Nonlinear Estimation and Classification.* Springer. [319]

Schapire, Robert E., and Yoram Singer. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning* 39(2/3):135-168. [319]

Schapire, Robert E., Yoram Singer, and Amit Singhal. 1998. Boosting and Rocchio applied to text filtering. In *Proc. SIGIR,* pp. 215-223. ACM Press. [291, 292]

Schlieder, Torsten, and Holger Meuss. 2002. Querying and ranking XML documents. *JASIST* 53(6):489-503. DOI: http://dx.doi.org/10.1002/asi.10060. [199]

Scholer, Falk, Hugh E.Williams, John Yiannis, and Justin Zobel. 2002. Compression of inverted indexes for fast query evaluation. In *Proc. SIGIR,* pp. 222-229. ACM Press. DOI: http://doi.acm.org/10.1145/564376.564416. [98]

Schölkopf, Bernhard, and Alexander J. Smola. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press. [319]

Schütze, Hinrich. 1998. Automatic word sense discrimination. *Computational Linguistics* 24(1):97-124. [176, 177]

Schütze, Hinrich, David A. Hull, and Jan O. Pedersen. 1995. A comparison of classifiers and document representations for the routing problem. In *Proc. SIGIR,* pp. 229-237. ACM Press. [177, 265, 292]

Schütze, Hinrich, and Jan O. Pedersen. 1995. Information retrieval based on word senses. In *Proc. SDAIR,* pp. 161-175. [345]

Schütze, Hinrich, and Craig Silverstein. 1997. Projections for efficient document clustering. In *Proc. SIGIR,* pp. 74-81. ACM Press. [344, 383]

Schwarz, Gideon. 1978. Estimating the dimension of a model. *Annals of Statistics* 6(2): 461-464. [345]

Sebastiani, Fabrizio. 2002. Machine learning in automated text categorization. *ACM Computing Surveys* 34(1):1-47. [264]

Shawe-Taylor, John, and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis.* Cambridge University Press. [319]

Shkapenyuk, Vladislav, and Torsten Suel. 2002. Design and implementation of a high-performance distributed web crawler. In *Proc. International Conference on Data Engineering.* URL: citeseer.ist.psu.edu/shkapenyuk02 design.html. [419]

Siegel, Sidney, and N. John Castellan, Jr. 1988. *Nonparametric Statistics for the Behavioral Sciences,* 2nd edition. McGraw-Hill. [160]

Sifry, Dave, 2007. The state of the Live Web, April 2007. URL: http://technorati.com/ weblog/2007/04/328.html. [29]

Sigurbjörnsson, Börkur, Jaap Kamps, and Maarten de Rijke. 2004. Mixture models, overlap, and structural hints in XML element retrieval. In *Proc. INEX,* pp. 196-210. [199]

Silverstein, Craig, Monika Rauch Henzinger, Hannes Marais, and Michael Moricz. 1999. Analysis of a very large web search engine query log. *SIGIR Forum* 33(1): 6-12. [44]

Silvestri, Fabrizio. 2007. Sorting out the document identifier assignment problem. In *Proc. ECIR,* pp. 101-112. [98]

Silvestri, Fabrizio, Raffaele Perego, and Salvatore Orlando. 2004. Assigning document identifiers to enhance compressibility of web search engines indexes. In *Proc. ACM Symposium on Applied Computing,* pp. 600-605. [98]

Sindhwani, V., and S. S. Keerthi. 2006. Large scale semi-supervised linear SVMs. In *Proc. SIGIR,* pp. 477-484. [320]

Singhal, Amit, Chris Buckley, and Mandar Mitra. 1996a. Pivoted document length normalization. In *Proc. SIGIR,* pp. 21-29. ACM Press. URL: citeseer.ist.psu.edu/ singhal96pivoted.html. [122]

Singhal, Amit, Mandar Mitra, and Chris Buckley. 1997. Learning routing queries in a query zone. In *Proc. SIGIR,* pp. 25-32. ACM Press. [177]

Singhal, Amit, Gerard Salton, and Chris Buckley. 1995. *Length normalization in degraded text collections.* Technical report, Cornell University, Ithaca, NY. [123]

Singhal, Amit, Gerard Salton, and Chris Buckley. 1996b. Length normalization in degraded text collections. In *Proc. SDAIR,* pp. 149-162. [123]

Singitham, Pavan Kumar C., Mahathi S. Mahabhashyam, and Prabhakar Raghavan. 2004. Efficiency-quality tradeoffs for vector score aggregation. In *Proc. VLDB,* pp. 624-635. URL: http://www.vldb.org/conf/ 2004/RS17P1.PDF. [137, 344]

Smeulders, Arnold W. M., Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. 2000. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.* 22(12):1349-1380. DOI: http://dx.doi.org/ 10.1109/34.895972. [xviii]

Sneath, Peter H.A., and Robert R. Sokal. 1973. Numerical Taxonomy: *The Principles and Practice of Numerical Classification.* W.H. Freeman. [367]

Snedecor, George Waddel, and William G. Cochran. 1989. *Statistical Methods.* Iowa State University Press. [265]

Somogyi, Zoltan. 1990. *The Melbourne University bibliography system.* Technical Report 90/3, Melbourne University, Parkville, Victoria, Australia. [76]

Song, Ruihua, Ji-Rong Wen, and Wei-Ying Ma. 2005. *Viewing term proximity from a different perspective.* Technical Report MSR-TR-2005-69, Microsoft Research. [138]

Sornil, Ohm. 2001. *Parallel Inverted Index for Large-Scale, Dynamic Digital Libraries.* PhD thesis, Virginia Tech. URL: http://scholar.lib.vt.edu/theses/available/etd-02062001-114915/. [420]

Spärck Jones, Karen. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* 28(1):11-21. [122]

Spärck Jones, Karen. 2004. Language modelling's generative model: Is it rational? MS, Computer Laboratory, University of Cambridge. URL: http://www.cl.cam.ac. uk/~ksj21/ langmodnote4.pdf. [233]

Spärck Jones, Karen, S.Walker, and Stephen E. Robertson. 2000. A probabilistic model of information retrieval: Development and comparative experiments. *IP&M* 36(6): 779-808, 809-840. [214, 215, 216]

Spink, Amanda, and Charles Cole (eds.). 2005. *New Directions in Cognitive Information Retrieval.* Springer. [161]

Spink, Amanda, Bernard J. Jansen, and H. Cenk Ozmultu. 2000. Use of query reformulation and relevance feedback by Excite users. *Internet Research: Electronic Networking Applications and Policy* 10(4):317-328. URL: http://ist.psu.edu/faculty_ pages/jansen/ academic/pubs/internetresearch2000.pdf. [170]

Sproat, Richard, and Thomas Emerson. 2003. The first international Chinese word segmentation bakeoff. In *SIGHAN Workshop on Chinese Language Processing.* [43]

Sproat, Richard, William Gale, Chilin Shih, and Nancy Chang. 1996. A stochastic finite-state word-segmentation

algorithm for Chinese. *Computational Linguistics* 22 (3):377-404. [43]

Sproat, Richard William. 1992. *Morphology and Computation.* MIT Press. [43]

Stein, Benno, and Sven Meyer zu Eissen. 2004. Topic identification: Framework and application. In *Proc. International Conference on Knowledge Management.* [367]

Stein, Benno, Sven Meyer zu Eissen, and Frank Wißbrock. 2003. On cluster validity and the information need of users. In *Proc. Artificial Intelligence and Applications.* [344]

Steinbach, Michael, George Karypis, and Vipin Kumar. 2000. A comparison of document clustering techniques. In *KDD Workshop on Text Mining.* [368]

Strang, Gilbert (ed.). 1986. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press. [383]

Strehl, Alexander. 2002. *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining.* PhD thesis, The University of Texas at Austin. [344]

Strohman, Trevor, and W. Bruce Croft. 2007. Efficient document retrieval in main memory. In *Proc. SIGIR,* pp. 175-182. ACM Press. [44]

Swanson, Don R. 1988. Historical note: Information retrieval and the future of an illusion. *JASIS* 39(2):92-98. [159, 177]

Tague-Sutcliffe, Jean, and James Blustein. 1995. A statistical analysis of the TREC-3 data. In *Proc. TREC,* pp. 385-398. [160]

Tan, Songbo, and Xueqi Cheng. 2007. Using hypothesis margin to boost centroid text classifier. In *Proc. ACM Symposium on Applied Computing,* pp. 398-403. ACM Press. DOI: http://doi.acm.org/10.1145/1244002.1244096. [291]

Tannier, Xavier, and Shlomo Geva. 2005. XML retrieval with a natural language interface. In *Proc. SPIRE,* pp. 29-40. [200]

Tao, Tao, XuanhuiWang, Qiaozhu Mei, and ChengXiang Zhai. 2006. Language model information retrieval with document expansion. In *Proc. Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics,* pp. 407-414. [233]

Taube, Mortimer, and Harold Wooster (eds.). 1958. *Information Storage and Retrieval: Theory, Systems, and Devices.* Columbia University Press. [16]

Taylor, Michael, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. 2006. Optimisation methods for ranking functions with multiple parameters. In *Proc. CIKM.* ACM Press. [320]

Teh, Yee Whye, Michael I. Jordan, Matthew J. Beal, and David M. Blei. 2006. Hierarchical Dirichlet processes. *Journal of the American Statistical Association* 101(476): 1566-1581. [384]

Theobald, Martin, Holger Bast, Debapriyo Majumdar, Ralf Schenkel, and Gerhard Weikum. 2008. TopX: Efficient and versatile top-*k* query processing for semistructured data. *VLDB Journal* 17(1):81-115. [199]

Theobald, Martin, Ralf Schenkel, and Gerhard Weikum. 2005. An efficient and versatile query engine for TopX search. In *Proc. VLDB,* pp. 625-636. VLDB Endowment. [199]

Tibshirani, Robert, GuentherWalther, and Trevor Hastie. 2001. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society Series B* 63:411-423. [345]

Tishby, Naftali, and Noam Slonim. 2000. Data clustering byMarkovian relaxation and the information bottleneck method. In *Proc. NIPS,* pp. 640.646. [345]

Toda, Hiroyuki, and Ryoji Kataoka. 2005. A search result clustering method using informatively named entities. In *Proc. Annual ACM International Workshop on Web Information and Data Management,* pp. 81-86. ACM Press. DOI: http://doi.acm.org/ 10.1145/1097047.1097063. [344]

Tomasic, Anthony, and Hector Garcia-Molina. 1993. Query processing and inverted indices in shared-nothing document information retrieval systems. *VLDB Journal* 2 (3):243-275. [419]

Tombros, Anastasios, and Mark Sanderson. 1998. Advantages of query biased summaries in information retrieval. In *Proc. SIGIR,* pp. 2-10. ACM Press. DOI: http://doi.acm.org/10.1145/290941.290947. [161]

Tombros, Anastasios, Robert Villa, and C. J. van Rijsbergen. 2002. The effectiveness of query-specific hierarchic clustering in information retrieval. *IP&M* 38(4):559-582. DOI: http://dx.doi.org/10.1016/S0306-4573(01)00048-6. [344]

Tomlinson, Stephen. 2003. Lexical and algorithmic stemming compared for 9 European languages with

Hummingbird Searchserver at CLEF 2003. In *Proc. Cross- Language Evaluation Forum,* pp. 286-300. [43]

Tong, Simon, and Daphne Koller. 2001. Support vector machine active learning with applications to text classification. *JMLR* 2:45-66. [320]

Toutanova, Kristina, and Robert C. Moore. 2002. Pronunciation modeling for improved spelling correction. In *Proc. ACL,* pp. 144-151. [60]

Treeratpituk, Pucktada, and Jamie Callan. 2006. An experimental study on automatically labeling hierarchical clusters using statistical features. In *Proc. SIGIR,* pp. 707-708. ACM Press. DOI: http://doi.acm.org/10.1145/1148170.1148328. [368]

Trotman, Andrew. 2003. Compressing inverted files. *IR* 6(1):5-19. DOI: http://dx.doi. org/10.1023/A:1022949613039. [98]

Trotman, Andrew, and Shlomo Geva. 2006. Passage retrieval and other XML-retrieval tasks. In *SIGIR 2006 Workshop on XML Element Retrieval Methodology,* pp. 43-50. [199]

Trotman, Andrew, Shlomo Geva, and Jaap Kamps (eds.). 2007. *Proc. SIGIR* 2007Workshop on Focused Retrieval. University of Otago, Dunedin, New Zealand. [199]

Trotman, Andrew, Nils Pharo, and Miro Lehtonen. 2006. XML-IR users and use cases. In *Proc. INEX,* pp. 400.412. [198]

Trotman, Andrew, and Börkur Sigurbjörnsson. 2004. Narrowed Extended XPath I (NEXI). In Fuhr et al. (2005), pp. 16-40. DOI: http://dx.doi.org/10.1007/11424550 2. [199]

Tseng, Huihsin, Pichuan Chang, Galen Andrew, Daniel Jurafsky, and Christopher Manning. 2005. A conditional random field word segmenter. In *SIGHAN Workshop on Chinese Language Processing.* [43]

Tsochantaridis, Ioannis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. 2005. Large margin methods for structured and interdependent output variables. *JMLR* 6:1453-1484. [319]

Turpin, Andrew, andWilliam R. Hersh. 2001. Why batch and user evaluations do not give the same results. In *Proc. SIGIR,* pp. 225-231.

Turpin, Andrew, and William R. Hersh. 2002. User interface effects in past batch versus user experiments. In *Proc. SIGIR,* pp. 431-432.

Turpin, Andrew, Yohannes Tsegay, David Hawking, and Hugh E.Williams. 2007. Fast generation of result snippets in web search. In *Proc. SIGIR,* pp. 127-134 .ACMPress. [161]

Turtle, Howard. 1994. Natural language vs. Boolean query evaluation: A comparison of retrieval performance. In *Proc. SIGIR,* pp. 212-220. ACM Press. [15]

Turtle, Howard, and W. Bruce Croft. 1989. Inference networks for document retrieval. In *Proc. SIGIR,* pp. 1-24. ACM Press. [215]

Turtle, Howard, and W. Bruce Croft. 1991. Evaluation of an inference network-based retrieval model. *TOIS* 9(3):187-222. [215]

Turtle, Howard, and James Flood. 1995. Query evaluation: strategies and optimizations. *IP&M* 31(6):831-850. DOI: http://dx.doi.org/10.1016/0306-4573(95)00020-H. [123]

Vaithyanathan, Shivakumar, and Byron Dom. 2000. Model-based hierarchical clustering. In *Proc. UAI,* pp. 599-608. Morgan Kaufmann. [368]

van Rijsbergen, C. J. 1979. *Information Retrieval,* 2nd edition. Butterworths. [159, 198, 203, 213, 216]

van Rijsbergen, C. J. 1989. Towards an information logic. In *SIGIR,* pp. 77-86. ACM Press. DOI: http://doi.acm.org/10.1145/75334.75344. [xviii]

van Zwol, Roelof, Jeroen Baas, Herre van Oostendorp, and Frans Wiering. 2006. Bricks: The building blocks to tackle query formulation in structured document retrieval. In *Proc. ECIR,* pp. 314-325. [200]

Vapnik, Vladimir N. 1998. *Statistical Learning Theory.*Wiley-Interscience. [319]

Vittaut, Jean-Nöel, and Patrick Gallinari. 2006. Machine learning ranking for structured information retrieval. In *Proc. ECIR,* pp. 338-349. [199]

Voorhees, Ellen M. 1985a. The cluster hypothesis revisited. In *Proc. SIGIR,* pp. 188-196. ACM Press. [344]

Voorhees, Ellen M. 1985b. *The effectiveness and efficiency of agglomerative hierarchic clustering in document retrieval.* Technical Report TR 85-705, Cornell. [367]

Voorhees, Ellen M. 2000. Variations in relevance judgments and the measurement of retrieval effectiveness. *IP&M*

36:697-716. [160]

Voorhees, EllenM., and Donna Harman (eds.). 2005. *TREC: Experiment and Evaluation in Information Retrieval.* MIT Press. [159, 453, 461]

Wagner, Robert A., and Michael J. Fischer. 1974. The string-to-string correction problem. *JACM* 21(1):168-173. DOI: http://doi.acm.org/10.1145/321796.321811. [59]

Ward Jr., J. H. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association* 58:236-244. [367]

Wei, Xing, and W. Bruce Croft. 2006. LDA-based document models for ad-hoc retrieval. In *Proc. SIGIR,* pp. 178-185. ACM Press. DOI: http://doi.acm.org/10.1145/ 1148170.1148204. [384]

Weigend, Andreas S., Erik D.Wiener, and Jan O. Pedersen. 1999. Exploiting hierarchy in text categorization. *IR* 1(3):193-216. [319]

Weston, Jason, and Chris Watkins. 1999. Support vector machines for multi-class pattern recognition. In *Proc. European Symposium on Artificial Neural Networks,* pp. 219-224. [319]

Williams, Hugh E., and Justin Zobel. 2005. Searchable words on the web. *International Journal on Digital Libraries* 5(2):99-105. DOI: http://dx.doi.org/10.1007/s00799-003-0050- z. [97]

Williams, Hugh E., Justin Zobel, and Dirk Bahle. 2004. Fast phrase querying with combined indexes. *TOIS* 22(4):573-594. [41]

Witten, Ian H., and Timothy C. Bell. 1990. Source models for natural language text. *International Journal Man-Machine Studies* 32(5):545-579. [97]

Witten, Ian H., and Eibe Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques,* 2nd edition. Morgan Kaufmann. [342]

Witten, Ian H., Alistair Moffat, and Timothy C. Bell. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images,* 2nd edition. Morgan Kaufmann. [76, 97, 98]

Wong, S. K. Michael, Yiyu Yao, and Peter Bollmann. 1988. Linear structure in information retrieval. In *Proc. SIGIR,* pp. 219-232. ACM Press. [320]

Woodley, Alan, and Shlomo Geva. 2006. NLPX at INEX 2006. In *Proc. INEX,* pp. 302-311. [200]

Xu, Jinxi, andW. Bruce Croft. 1996. Query expansion using local and global document analysis. In *Proc. SIGIR,* pp. 4-11. ACM Press. [177]

Xu, Jinxi, and W. Bruce Croft. 1999. Cluster-based language models for distributed retrieval. In *Proc. SIGIR,* pp. 254-261. ACM Press. DOI: http://doi.acm. org/10.1145/312624.312687. [344]

Yang, Hui, and Jamie Callan. 2006. Near-duplicate detection by instance-level constrained clustering. In *Proc. SIGIR,* pp. 421-428. ACM Press. DOI: http://doi. acm.org/10.1145/1148170.1148243. [344]

Yang, Yiming. 1994. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *Proc. SIGIR,* pp. 13-22. ACM Press. [291]

Yang, Yiming. 1999. An evaluation of statistical approaches to text categorization. *IR* 1:69-90. [319]

Yang, Yiming. 2001. A study of thresholding strategies for text categorization. In *Proc. SIGIR,* pp. 137-145. ACM Press. DOI: http://doi.acm.org/10.1145/383952.383975. [292]

Yang, Yiming, and Bryan Kisiel. 2003. Margin-based local regression for adaptive filtering. In *Proc. CIKM,* pp. 191-198. ACM Press. DOI: http://doi.acm.org/10.1145/ 956863.956902. [292]

Yang, Yiming, and Xin Liu. 1999. A re-examination of text categorization methods. In *Proc. SIGIR,* pp. 42-49. ACM Press. [265, 319]

Yang, Yiming, and Jan Pedersen. 1997. Feature selection in statistical learning of text categorization. In *Proc. ICML*. [265]

Yue, Yisong, Thomas Finley, Filip Radlinski, and Thorsten Joachims. 2007. A support vector method for optimizing average precision. In *Proc. SIGIR*. ACM Press. [320]

Zamir, Oren, and Oren Etzioni. 1999. Grouper: A dynamic clustering interface to web search results. In *Proc. WWW,* pp. 1361-1374. Elsevier North-Holland. DOI: http://dx.doi.org/10.1016/S1389-1286(99)00054-7. [344, 368]

Zaragoza, Hugo, Djoerd Hiemstra, Michael Tipping, and Stephen Robertson. 2003. Bayesian extension to the language model for ad hoc information retrieval. In *Proc. SIGIR,* pp. 4-9. ACM Press. [232]

Zavrel, Jakub, Peter Berck, and Willem Lavrijssen. 2000. Information extraction by text classification: Corpus mining for features. In *Proc. Workshop Information Extraction meets Corpus Linguistics.* url: http://www.cnts.ua.ac.be/Publications/2000/ZBL00 . Held in conjunction with LREC-2000. [292]

Zha, Hongyuan, Xiaofeng He, Chris H. Q. Ding, Ming Gu, and Horst D. Simon. 2001. Bipartite graph partitioning and data clustering. In *Proc. CIKM,* pp. 25-32. ACM Press. [345, 368]

Zhai, Chengxiang, and John Lafferty. 2001a. Model-based feedback in the language modeling approach to information retrieval. In *Proc. CIKM.* ACM Press. [231]

Zhai, Chengxiang, and John Lafferty. 2001b. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proc. SIGIR,* pp. 334-342. ACM Press. [232]

Zhai, ChengXiang, and John Lafferty. 2002. Two-stage language models for information retrieval. In *Proc. SIGIR,* pp. 49-56. ACM Press. DOI: http://doi.acm.org/ 10.1145/564376.564387. [233]

Zhang, Jiangong, Xiaohui Long, and Torsten Suel. 2007. Performance of compressed inverted list caching in search engines. In *Proc. CIKM.* ACM Press. [98]

Zhang, Tong, and Frank J. Oles. 2001. Text categorization based on regularized linear classification methods. *IR* 4(1):5-31. URL: citeseer.ist.psu.edu/zhang00text.html. [319]

Zhao, Ying, and George Karypis. 2002. Evaluation of hierarchical clustering algorithms for document datasets. In *Proc. CIKM,* pp. 515-524. ACM Press. doi: http://DOI.acm.org/10.1145/584792.584877. [367]

Zipf, George Kingsley. 1949. *Human Behavior and the Principle of Least Effort.* Addison- Wesley. [97]

Zobel, Justin. 1998. How reliable are the results of large-scale information retrieval experiments? In *Proc. SIGIR,* pp. 307-314. [160]

Zobel, Justin, and Philip Dart. 1995. Finding approximate matches in large lexicons. *Software Practice and Experience* 25(3):331-345. URL: citeseer.ifi.unizh.ch/ zobel95finding.html . [60]

Zobel, Justin, and Philip Dart. 1996. Phonetic string matching: Lessons from information retrieval. In *Proc. SIGIR,* pp. 166-173. ACM Press. [60]

Zobel, Justin, and Alistair Moffat. 2006. Inverted files for text search engines. *ACM Computing Surveys* 38(2). [17, 76, 98, 122]

Zobel, Justin, Alistair Moffat, RossWilkinson, and Ron Sacks-Davis. 1995. Efficient retrieval of partial documents. *IP&M* 31(3):361-377. DOI: http://dx.doi.org/10.1016/0306 4573(94)00052-5. [199]

Zukowski, Marcin, Sandor Heman, Niels Nes, and Peter Boncz. 2006. Super-scalar RAM-CPU cache compression. In *Proc. International Conference on Data Engineering,* p. 59. IEEE Computer Society. DOI: http://dx.doi.org/ 10.1109/ICDE.2006.150. [98]

译名对照索引

D

E

带格式的: 葡萄牙语(巴西)

M

N

O

P

Q

R

S

T

U

V

W

X

Z