计 算 机 科 学 丛 书

搜索引擎
信息检索实践

(美) W. Bruce Croft Donald Metzler Trevor Strohman 著 刘挺 秦兵 张宇 车万翔 译

Search Engines
Information Retrieval in Practice

机 械 工 业 出 版 社
China Machine Press

本书介绍了信息检索中的关键问题，以及这些问题如何影响搜索引擎的设计与实现，很好地兼顾了信息检索理论以及搜索引擎的设计、实现和使用中的知识面广度与深度问题，重点关注于那些对于实现搜索引擎组件以及组件背后的信息检索模型最重要的部分，以及网络上使用的搜索技术。本书适合作为高等院校计算机科学或计算机工程专业本科生或研究生的教材。

本书版权登记号：图字：01-2009-4501

图书在版编目（CIP）数据

搜索引擎：信息检索实践 /（美）克罗夫特（Croft, W. B.）等著；刘挺等译. —北京：机械工业出版社，2010.2

（计算机科学丛书）

书名原文：Search Engines: Information Retrieval in Practice

ISBN 978-7-111-28808-4

Ⅰ. 搜… Ⅱ. ① 克… ② 刘… Ⅲ. 互联网络—情报检索 Ⅳ. G354.4

中国版本图书馆CIP数据核字（2010）第014782号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：刘立卿

北京瑞德印刷有限公司印刷

2010年6月第1版第1次印刷

184mm × 260mm · 20印张

标准书号：ISBN 978-7-111-28808-4

定价：56.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991，88361066

购书热线：（010）68326294，88379649，68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

出版者的话

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

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

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

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

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

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

华章教育
华章科技图书出版中心

译 者 序

记得1996年，我在北京图书馆的电子阅览室里付费上机检索，查到了一些我感兴趣的英文文献的题录。当时感觉很兴奋，用电脑检索比手工卡片检索方便了许多，但实在无法想象，短短几年后信息检索技术就冲破了科技情报领域的局限，成为普通民众每天都要使用的信息搜索工具。是互联网这项人类历史上伟大的发明，是网上浩如烟海的信息给人类带来的挑战，推动了信息检索技术向海量、高效、多媒体、个性化等方向突飞猛进地发展。

在21世纪第二个“十年”到来的今天，越来越多的学者、研发人员、研究生和大学生们投入到搜索引擎的研究与实践中来，他们希望掌握搜索引擎背后的理论和技术，希望了解在搜索引擎设计和开发中的原则和经验，从而为搜索引擎在各个行业的应用以及更具颠覆性的下一代搜索引擎的研发创造条件。

本书恰好可以满足这些读者的迫切需求。本书英文版原作者W. Bruce Croft于1979年在英国剑桥大学获得计算机科学博士学位，同年加入University of Massachusetts（Amherst分校）计算机科学系，曾任系主任多年，目前担任该系的特聘教授。1992年，他创建了智能信息检索研究中心，其研究兴趣覆盖信息检索中的多个领域，包括检索模型、Web搜索、查询处理、跨语言检索和搜索架构等。他在1997年被选为ACM Fellow，2000年获得美国信息科学与技术学会的研究奖，2003年获得ACM SIGIR（ACM关于信息检索的特殊兴趣组）的Gerard Salton奖，Salton奖是信息检索领域最受关注的奖励。

从作者的简历中可以看到这是一位国际信息检索领域顶尖的学者，他的这本新书一经出版，就以其深入浅出，理论与实践密切结合，覆盖了诸多最新的搜索引擎技术等鲜明特色而获得了业内的普遍好评。机械工业出版社迅速出版了该书的英文版，同时委托我研究中心将其译为中文，以飨读者。

在本书的翻译过程中，秦兵教授付出的心血最多，她除了要翻译自己负责的部分以外，还要负责组织、校对等工作，她高度的责任心是本书得以顺利出版的关键。除四位主要译者外，李正华、伍大勇、郎君、赵妍妍、宋巍等多位高年级博士生也参与了部分章节的翻译工作。本书也得到了译者们的恩师李生教授的支持，并得到了他主持的国家自然科学基金重点项目“下一代信息检索研究”（编号60736044）的资助。

译者水平有限，书中疏漏在所难免，敬请读者批评指正。

译 者

2010年3月1日

于哈尔滨工业大学

前　言

本书综述了信息检索中的重要问题，并介绍了这些问题如何对搜索引擎的设计与实现产生影响。本书并不是按照相同的详细程度描述每个主题，相反，我们侧重于那些对于实现搜索引擎组件以及组件背后的信息检索模型最重要的部分。网络搜索引擎显然是一个重要的话题，我们主要覆盖了在网络上使用的搜索技术，但搜索引擎在其他场合中也有应用，这就是为什么我们重点强调各种搜索引擎背后的信息检索理论与概念的原因。

本书的目标读者群主要是计算机科学或计算机工程的本科生，但研究生也会发现本书是有用的，此外，本书也适合多数情报科学专业的学生。最后，无论读者是什么背景，通过阅读本书都可以对他们动手开发搜索引擎有所帮助。本书中涉及数学知识，但并不深奥。书中也有代码和程序设计的练习，但对于那些已经完成了基本计算机科学与程序设计课程的人来说，完全可以掌握。

每章末尾的练习使用了被称为Galago的基于Java的开源搜索引擎。Galago既是为本书所设计的，也借鉴了Lemur和Indri项目的经验。换句话说，这是一个功能齐全的能够支持真正应用的搜索引擎。许多编程练习都是针对Galago组件的使用、修改和扩展。

内容

在第1章，我们对信息检索及它与搜索引擎的关系做了一个高层次的回顾。在第2章，我们描述了搜索引擎的架构，这一章全面介绍搜索引擎的各个组件，但没有涉及细节问题。在第3章，我们关注于爬取、文档信息源和其他用于获取被检索信息的技术。第4章描述了文本的统计特征，以及用来处理和识别重要特征的技术，并为建立索引做准备。第5章描述了怎样为有效的搜索建立索引，以及怎样利用索引处理查询。在第6章，我们描述了怎样处理查询，并把它们转换为更好的形式，以表达用户的信息需求。

第7章介绍排序算法及以这些算法为基础的检索模型。这一章也概述了机器学习技术以及机器学习与信息检索、搜索引擎的关系。第8章描述了用于比较和调整搜索引擎的评价指标和性能指标。第9章覆盖了分类、过滤、聚类和垃圾信息处理等重要技术。社会化搜索一词用于描述一种搜索引擎的应用，这种应用涉及对内容进行标注或者回答问题的社区人群。第10章描述了为这些应用服务的搜索技术以及P2P搜索。最后，在第11章，我们介绍了一些高级搜索技术，这些技术能够从文档中捕获更多内容，而不只是简单地基于词进行计算，其中包括使用语言学特征、文档结构和非文本媒体的内容，如图像和音乐。

信息检索理论以及搜索引擎的设计、实现、评价和使用覆盖了太多的话题，以至于无法在一本书中做全部深入的描述。我们试图集中于最重要的主题，同时，对于这些充满挑战也非常有价值的课题，我们也兼顾到各个主题的方方面面。

补充材料

本书提供了一定范围的补充材料，这些材料既可以供那些选修这门课的学生使用，也可以供讲授这门课的教师使用，其中包括：

- 课程胶片（PDF或PPT格式）
- 每章末尾问题的参考答案（仅供教师使用）
- Galago搜索引擎

致谢

首先，也是最重要的，如果没有我们夫人们（Pam Aselton、Anne-Marie Strohman和ShelleyWang）的支持和鼓励，本书是无法完成的。Massachusetts Amherst大学提供了本书准备工作中的素材支持，并给Croft颁发了ContiFaculty Fellowship奖，这显著地提高了本书的成书进程。智能信息检索中心的员工们（Jean Joyce、Kate Moruzzi、Glenn Stowell和Andre Gauthier）在很多方面对我们帮助很大，中心的同事和学生们提供了充满激情的环境，使我们在这个领域的工作很有价值。很多人审阅了本书的部分章节，我们感谢他们给出的建议。最后，不得不提及我们的孩子（Doug、Eric、Evan和Natalie），谢谢他们带来的快乐。

Bruce Croft
Don Metzler
Trevor Strohman

目　录

第1章　搜索引擎和信息检索

“Helpmann先生，我很高兴进入信息检索领域。”

——Sam Lowry，《妙想天开》

1.1　什么是信息检索

撰写本书的目标是帮助人们理解、评价和比较搜索引擎（Search Engine），并进行改写以适应特殊的应用需要。对大多数人来说，在Web上搜索信息是一项日常活动。目前，搜索和通信是计算机最普遍的应用。公司和大学中的很多人试图改进搜索引擎，让人们以更简单更快速的方式找到正确的信息，这一点也不奇怪。这些人，无论他们自称为计算机科学家、软件工程师、信息科学家、搜索引擎优化者或其他什么称谓，都是在信息检索（Information Retrieval, IR）这个领域工作。因此，在进入搜索引擎内部开始具体的旅程以前，我们要先介绍一下背景。

Gerard Salton是信息检索领域的先驱，也是20世纪60年代到90年代信息检索领域的领袖人物之一，他在其经典教科书（Salton, 1968）中，给信息检索做出了以下定义。

信息检索是关于信息的结构、分析、组织、存储、搜索和检索的领域。

尽管在过去的40年中，对搜索的理解以及搜索技术都有了巨大的进步，但上述定义仍然非常合适、非常准确。“信息检索”一词含义非常宽泛，涵盖了很宽范围的信息类型和各种与搜索相关的应用。

从20世纪50年代开始，该领域的主要焦点一直是文本（text）和文本形式的文档（text document）。网页、电子邮件、学术论文、图书和新闻报道只是文档类型中的一部分。所有这些文档都有一定的结构，例如与科技期刊论文的内容相关联的标题、作者、日期和摘要信息等。当用于数据库记录时，这些结构由属性或域组成。文档和典型的数据库记录（例如银行账户记录或航班预定记录）最重要的区别在于，文档中的大部分信息以文本形式存放，文本是没有结构的。

为了解释这一区别，考虑账户记录中包含的两个典型属性：账号和当前余额。无论在格式（例如：用6位整数定义账号，用带有2位小数的实数描述余额）上，还是在意义上，这两个属性都被非常精确地定义。要比较这些属性的值是非常容易的，因此可以直接实现某个算法，识别出满足某个查询条件的记录，例如：“找出账号为321456的账户”，或者“找出余额大于5万美金的账户”。

现在考虑一个关于两个银行合并的新闻报道，该报道有一些属性，比如标题和新闻来源，但重要的内容是报道本身。在数据库系统中，这一关键信息一般会被存储在一个没有内部结构的单独的大属性中。大多数提交给网络搜索引擎的查询，如果跟这篇报道有关，则具有“银行合并”或“银行接管”这样的字眼。为了做这个搜索，我们必须设计出能够比较查询文字和报道文字的算法，并决定报道中是否包含被搜索的信息。定义一个词、句子、段落或者

整个新闻报道的意义，比定义一个账号要难得多，因此文本的比较并不容易。对人们比较文本的过程进行理解和建模，并设计计算机算法以便精确地执行这种比较，是信息检索的核心。

逐渐地，信息检索的应用包含了带有结构的多媒体文档、有意义的文本内容和其他媒体。常见的信息媒体包括图片、视频、音频（包括音乐和语音）。在某些应用中，例如在法律支持系统中，被扫描的文档图片也是重要的。这些媒体像文本一样，其内容都很难描述和比较。当前搜索非文本文档的技术依赖于对这些内容的文本描述，而不是这些媒体自身的内容，但对媒体内容的直接比较技术正在不断进步，例如图片的比较。

除了媒体多种多样以外，信息检索还包括一系列任务和应用。通常的搜索情景是：某人向搜索引擎输入一个查询，并从一个经过排序的文档列表中得到答案。尽管在万维网上进行的搜索是信息检索最常见的应用，不过搜索也在企业、政府和其他许多应用领域中扮演着重要的角色。垂直搜索（vertical search）是网络搜索的特殊形式，搜索被限制在特殊的主题上。企业搜索（enterprise search）是在散布在企业内部网中的大量计算机文件中寻找所需的信息。网页当然是分布式信息存储的一部分，但大多数信息将在邮件、报告、发言稿、数据表以及企业的结构化数据库中得到。桌面搜索（desktop search）是企业搜索的个人版，信息源是存储在一台个人电脑中的文件集合，包括那些被浏览过的邮件和网页。P2P搜索（peer-to-peer search）是在节点机或计算机构成的网络中搜寻信息，但没有任何集中式的控制。这种类型的搜索是从音乐文件的共享工具开始的，但也可以被用于任何有共同兴趣的社区，甚至在移动设备中共享位置信息。人们采用搜索和相关的信息检索技术发布广告，做智能分析、科学发现、卫生保健、客服支持和房地产投资等等。任何包含文本集合的应用或其他非结构化的信息，都需要进行组织和搜索。

基于用户查询的搜索（有时称为特殊搜索（ad hoc search），因为查询的范围巨大而且事先没有约定）并不是信息检索中研究的唯一的文本处理任务，其他任务包括过滤（filtering）、分类（classification）和问答（question answering）。过滤也称为跟踪，根据一个人的兴趣发现符合其兴趣的报道，并用邮件和其他机制报警。分类程序基于一套预先定义的标签或类别（比如Yahoo目录体系）给文档打上标记。问答系统与搜索很相似，但它的目标是处理更特殊的问题，例如“珠穆朗玛峰的高度是多少？”。问答系统的目标是从文本中发现明确的答案，而不是一个文档列表。表1-1对信息检索领域的一些特征和维度进行了总结。

表1-1 信息检索的维度

内容实例	应用实例	任务实例
文本	网络搜索	特殊搜索
图像	垂直搜索	过滤
视频	企业搜索	分类
扫描文档	桌面搜索	问答
音频	P2P搜索	
音乐		

1.2 重要问题

信息检索的研究者们把注意力集中在一些关键问题上，这些问题在20世纪60年代文本集合的大小为1.5MB时很重要，在当今能够处理数十亿网页的商业化网络搜索引擎时代，这些

问题仍然非常重要。其中一个关键问题是相关性（relevance）。相关性是信息检索中的基本概念，不严格地讲，一个相关文档包含了当一个人把查询发给搜索引擎后他要找的信息。尽管这个定义听上去有些简单，但在一个人判断一篇文档是否相关时，有许多因素会影响他的决策。在设计比较文本以及对文档进行排序的算法时，需要考虑这些因素。如果像数据库系统或者Unix中的文本查找工具那样，对查询和文档进行简单的比较，寻找精确的匹配，那结果的相关性一定很差。一个明显的原因是：语言可以用许多不同的方式，通常是用不同的词语，表达同一个概念，这在信息检索中称为词表不匹配问题（vocabulary mismatch problem）。

有必要区分话题相关（topical relevance）和用户相关（user relevance）这两个概念。如果一个文本与查询是话题相关的，就意味着两者有相同的话题。例如，关于堪萨斯州龙卷风的新闻报道，与查询“严重天气事件”是话题相关的。如果提这个问题的人以前曾经看过那篇报道，或者那篇报道是五年前发表的，或者那篇报道来自中国通讯社并用中文写成，那么他（通常称为用户）可能不认为那篇报道是相关的。因此，用户相关会考虑该报道的一些附加特性。

为了探讨相关性问题，研究者们提出了多种检索模型（retrieval model），并测试这些模型的作用。一个检索模型是对查询与文档匹配过程的形式化表示，它是排序算法（ranking algorithm）的基础，搜索引擎利用排序算法生成文档的有序列表。一个好的检索模型能够找到那些与提问者相关的文档。有些检索模型集中在话题相关性上，但一个部署在真实环境中的搜索引擎，必须使用包含了用户相关性的排序算法。

在信息检索中，检索模型的一个有趣的特点是，它们往往对文本的统计特征而不是语言结构建模。举例来说，这就意味着排序算法会更多地考虑词出现的数量，而不关心一个词是名词还是形容词。更先进的模型能够采用语言特征，但仍把语言特征的重要性放在第二位。对词频信息的使用始于20世纪50年代信息检索领域的另一位先驱H. P. Luhn。直到20世纪90年代，这种看待文本的视角才在计算机科学的其他领域（比如自然语言处理）流行起来。

信息检索的另一个核心问题是评价问题。由于文本排序的质量依赖于该文本与用户期望的匹配程度，因此有必要及早制定评价体系，以及获取评价数据、比较排序算法的实验步骤。在20世纪60年代，Cyril Cleverdon率先制定了评价方法，他使用的两种评价指标，准确率（precision）和召回率（recall），目前仍很流行。准确率是非常符合直觉的评价指标，它是检索出来的文档中相关文档所占比例。召回率是全部相关文档中被检索出来的文档比例。当使用召回率这个评价指标时，有一个假设：对于给定的查询，我们知道所有的相关文档。在网络搜索环境下，这样的假设显然是有问题的，但对于比较小的文档测试集合（test collection），这样的指标是有用的。一个信息检索实验的测试集合，由文本文档集合、典型查询样本和每个查询的相关文档列表（相关性判别（relevance judgment））组成。最知名的测试集是TREC（Text REtrieval Conference, http://trec.nist.gov/）评测会议提供的测试集。

检索模型和搜索引擎的评测是一个非常活跃的领域，目前的热点集中在使用大量的从用户交互中获得的日志数据，比如点击流（clickthrough）数据，点击流数据记录了在一个搜索过程中被点击的文档。点击流和其他日志数据与相关性有很大的关联，因此可以用它来对搜索进行评价，但是搜索引擎公司仍然使用相关性判别作为日志数据的补充，以便确保结果的有效性。

信息检索的第三个核心问题是注重用户和他们的信息需求（information need），这一原则

是显而易见的，因为对搜索的评价是以用户为中心的，即搜索引擎用户是搜索质量的终极判定者。这种理念引发了大量关于人们怎样与搜索引擎之间进行交互的研究，特别是开发帮助用户表达他们的信息需求的技术。信息需求是人们向搜索引擎发送查询的背后动因。与数据库系统的需求（比如某个银行账户的存款余额）相比，文本查询通常是用户实际需求的一种很糟糕的描述。一个单词查询如“猫”，可能表示“在哪儿能买到猫”或要查询百老汇爵士乐的信息。尽管很缺乏特指性，但在网络搜索中一个词的查询仍非常普遍。像查询建议（query suggestion）、查询扩展（query expansion）和相关反馈（relevance feedback）这些技术，使用交互的方式和上下文环境来优化初始的查询，以便产生更好的排序列表。

这些问题将贯穿本书，并进行非常细致的讨论。我们现在已具备充分的背景知识来讨论信息检索研究中的主要产品，即搜索引擎。

1.3 搜索引擎

搜索引擎是信息检索技术在大规模文本集合上的实际应用。网络搜索引擎是一个明显的例子，但如前所述，搜索引擎有许多不同的应用，比如桌面搜索或企业搜索。搜索引擎的出现已经有很多年了，例如MEDLINE系统是在线医学文献搜索系统，从20世纪70年代开始兴起。“搜索引擎”一词原来是指为文本搜索服务的特殊的硬件。从20世纪80年代中期开始，在描述用来比较查询和文档并生成文档排序结果的软件系统时，逐渐更多地使用“搜索引擎”一词，而不是“信息检索系统”。关于搜索引擎的内容比排序算法当然要丰富得多，我们将在下一章讨论这些系统的一般结构。

为了满足不同应用的需求，搜索引擎被设计成各种不同的结构。网络搜索引擎，比如Yahoo，必须能够捕获，或者说爬取（crawl）T级的数据，并对每天收到的全世界数以百万计的查询提供亚秒级的响应时间。企业搜索引擎，比如Autonomy，必须能够处理一个公司内部不同类型的信息源，使用与公司有关的特殊知识作为搜索和相关任务（如数据挖掘（data mining））的一部分。数据挖掘指从数据中自动发现有趣的结构，也包括聚类（clustering）技术。桌面搜索引擎，比如微软的Vista™搜索必须能够在人们制作和浏览新文档、网页和邮件时快速地合并，同时提供非常直观的界面来搜索这些非常异质的混合信息。

开源（open source）搜索引擎是另外一类重要的搜索系统，与商业搜索引擎有不同的设计目标。这样的系统很多，维基百科关于信息检索的页面有多个链接指向许多开源搜索引擎。三个特别有趣的系统是Lucene、Lemur和本书提供的Galago。Lucene是一个基于Java的流行的搜索引擎，它已经被用于大范围的商业应用中，其中使用的信息检索技术相对简单。Lemur是一个开源的工具包，包含基于Indri C++的搜索引擎，Lemur主要被信息检索研究者用来比较先进的搜索技术。Galago是一个基于Java的搜索引擎，它基于Lemur和Indri项目。本书的习题大量使用Galago。Calago的速度快、有自适应能力而且易于理解，它融合了非常有效的信息检索技术。

搜索引擎设计中的重要问题包括了信息检索中的各种问题：有效的排序算法、评价及用户交互。此外，在搜索引擎的部署过程中遇到的大规模数据的运行环境，还给搜索引擎带来了其他许多难题。这些难题中的首要问题是搜索引擎的性能，评价指标包括响应时间（response time）、查询吞吐量（query throughput）和索引速度（indexing speed）。响应时间是从发出一个查询请求到得到检索结果列表之间的延迟。吞吐量是在一个给定时间内能够处理

的查询数量。索引速度是为文本文档编排索引以便用于搜索的速率。为搜索引擎服务的索引设计是本书的重要主题。

另一个重要的性能指标是把新数据合并到索引中的速度。搜索应用往往要处理动态持续变化的信息。覆盖率（coverage）衡量现存信息（比如在一个企业信息环境中）有多少被索引和存储在搜索引擎中。新近性（recency）或时新性（freshness）衡量所存信息的年龄（age）。

搜索引擎可以用在小规模数据集上，比如桌面上的几百封邮件和文档，也可以用于极大规模的数据集，比如整个互联网。对某个应用可能只有很少的一些用户，也可能有成千上万的用户。对搜索引擎来说，可扩充性（scalability）很明显是一个重要问题。面向一个特定应用的设计应该考虑到数据量和用户量的增长。在1.1节，我们介绍了搜索引擎是怎样用于多种应用和多种任务的，为了完成这些任务，搜索引擎必须是可定制的（customizable）或者说是自适应的（adaptable）。这意味着搜索引擎的许多功能，比如排序算法、界面或索引策略，能够为满足新的应用需要而调整和适应。

特殊应用也会影响搜索引擎的设计，最好的例子是网络搜索中的垃圾信息（spam）。垃圾信息一般被视为非所要的邮件，但更一般地可以被定义为，为某种商业利益而制作的文档中误导的、不合适的或不相关的信息。有许多类型的垃圾信息，但搜索引擎必须处理的一种类型是文档中的垃圾词，这些词导致该文档能够在搜索引擎响应一些热门查询时被检索出来。由于垃圾索引（spamdexing）显著地导致搜索引擎排序质量的降低，网络搜索引擎的设计者不得不开发能够识别和删除这些垃圾文档的技术。图1-1总结了搜索引擎设计中涉及的主要问题。

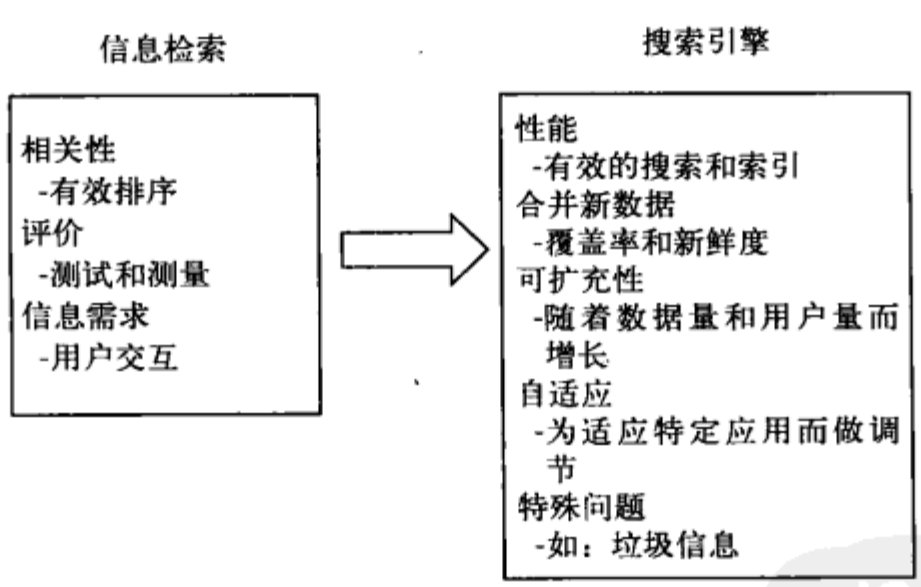


图1-1 搜索引擎设计及核心信息检索问题

根据信息检索和搜索引擎关系的讨论，我们现在来考虑计算机科学家和其他人在搜索引擎设计和使用中所扮演的角色。

1.4 搜索工程师

信息检索研究包括文本和语言的数学模型的建立、带有测试集合与用户的大规模环境的建立，以及大量学术论文的写作。由于这些原因，此类研究比较适合专业学者或研究实验室中的人们来做。这些人主要是受到过计算机科学训练的人，尽管信息科学、数学，甚至社会

科学和计算语言学也是需要的。那么谁来做搜索引擎呢？在很大程度上，还是同一种人，只不过应该更强调实践能力。计算机产业已经开始使用“搜索工程师”（search engineer）一词来指称这种类型的人。搜索工程师主要是受过计算机科学训练的人，多数有计算机系统或数据库的背景。令人惊讶的是，他们当中很少有人受过信息检索方面的训练，这也是撰写本书的动机之一。

搜索工程师的角色是什么？当然，那些在设计和实现新搜索引擎的主要的互联网搜索公司工作的人是搜索工程师，但搜索工程师中的大部分人是那些修改、扩充、维护和调整现有搜索引擎，以满足大范围商业应用需求的人。为搜索引擎设计和优化内容的人，以及实现垃圾信息处理的人也是搜索工程师。搜索工程师所开发的搜索引擎覆盖了上一节提到的全部搜索引擎，他们主要使用开源搜索引擎和企业搜索引擎做应用开发，但也最有效地使用桌面搜索引擎和网络搜索引擎。

在现代计算机应用中，搜索的重要性和普适性意味着，搜索工程已经成为计算机产业中一种重要的职业。然而，在计算机科学系中，只有非常少的教程从信息检索的视角给学生们补充搜索方面的知识。本书旨在帮助潜在的搜索工程师理解搜索，并掌握相关的工具。

参考文献和深入阅读

在每一章中，我们都提供一些论文和书籍的列表，这些资料对所讨论的主题提供了更为详细的论述。补充读物对于理解本书的内容来说并不是必需的，但如能参考这些资料，可以了解更多的背景，并在一些情况下获得更深入的理解，也可以接触到更高级的主题，其中会描述本书没有覆盖的技术和研究成果。

根据我们的观点，信息检索领域最经典的参考文献是Salton（1968;1983）和van Rijsbergen（1979）的三本书。Van Rijsbergen的书自从在网上发布以来就一直很流行，这三本书非常精彩地描述了信息检索早期（直到20世纪70年代末）的研究成果。根据计算机科学对信息检索领域的定义，Salton的早期著作特别重要。最近的书包括Baeza-Yates和Ribeiro-Neto（1999）和Manning等（2008）。

与本书覆盖的全部主题相关的研究论文，可以在美国计算机协会（ACM）信息检索特殊兴趣组（SIGIR）年会的论文集中找到。这些论文集可以从网上ACM数字图书馆得到。信息检索欧洲会议（ECIR）、信息与知识管理（CIKM）和网络搜索与数据挖掘会议（WSDM）的论文集中，也有一些好的论文。WSDM会议是从WWW会议派生出来的会议，包括了一些网络搜索的重要论文。TREC会议的论文集可以在线获得，其中介绍了许多来自不同的学术和工业组织的新技术。对TREC实验的综述，可以在Voorhees和Harman（2005）中得到。在数据库会议，如VLDB和SIGMOD中，与搜索相关的论文在不断增长；在语言技术的会议，如ACL和HLT（计算语言学和人类语言技术协会）以及机器学习等其他会议上，偶尔也有与搜索相关的文章。

练习

1.1 思考并写出若干用于网络搜索引擎的查询，确保这些查询的长度不同（比如不要全都是一个词）。尝试在某些查询中详细而准确地说明你要找什么信息。在两个商业网络搜索引擎中提交这些查询，并通过相关性判断比较前10个结果。写一个报告，至少回答以下问

题：结果的准确率如何？两个搜索引擎结果的重叠状况如何？其中一个搜索引擎明显比另一个好吗？如果是这样，好多少？短查询和长查询的效果相比如何？

1.2 站内搜索（site search）是另一个搜索引擎的常见应用。此时，搜索被限制在一个给定网站的页面里。请对站内搜索、网络搜索、垂直搜索和企业搜索进行对比分析。

1.3 列出5个你使用的网络服务或网站，这些网站使用搜索但并不是网络搜索引擎。描述搜索在这些服务中扮演的角色，同时描述其中的搜索是否基于数据库或grep风格的匹配技术，是否使用了某种类型的排序算法。

1.4 在网上尽可能多地查找开源搜索引擎、信息检索系统和相关技术的例子。给出对每个搜索引擎的简短描述，并总结它们之间的异同。

第2章　搜索引擎的架构

“你的第一个问题也许是最恰当的，你可能会或者可能不会意识到，它也是最不相关的。”

——设计师，《黑客帝国2：重装上阵》

2.1　什么是软件架构

在本章我们将描述搜索引擎的基本软件架构。对于软件架构的定义，尽管没有一个统一的规范，但软件架构通常包括软件组件、组件提供的接口以及各组件之间的联系。软件架构是在一个特殊的抽象层次用于描述系统的工具。UIMA[⊖]（Unstructured Information Management Architecture，非结构化信息管理架构）是一个软件架构的实例，该架构用于提供一个将搜索和相关语言技术组件整合在一起的标准。UIMA为组件定义了接口，使系统在增加处理文本和其他非结构化数据的技术的时候，变得更加简单。

搜索引擎的架构用于提供对于系统中重要的组件以及组件之间关系的高层次的描述。尽管架构中有些组件的确相当于Galago和其他搜索引擎的软件模块，但它并不是一个对于系统的代码级的描述。在本章及本书中我们都使用该软件架构，以便为讨论特定的技术提供背景。

架构的设计用于保证系统能够满足应用需求或目标。搜索引擎的两个主要目标是：

- 效果（质量）：对于一个用户查询，希望能够检索到最多的相关文档。
- 效率（速度）：尽可能快地处理用户的查询。

也许系统还有其他一些特殊目标，但这些也都属于效果问题或者效率问题（或者两者都有）。例如，搜索的文档集合发生了变化，如何确保搜索引擎系统对该变化做出快速的反应，就属于效果和效率问题。

搜索引擎的架构是由效果和效率这两个需求决定的。原因在于，用户需要一个有效率的系统。搜索引擎采用专门的、经过优化的数据结构，以达到快速检索的目的；用户需要高质量的结果，搜索引擎对文本进行深入的加工处理，并存储有助于改善结果相关性计算的文本统计学结果。

在接下来的章节中讨论的一些组件，已经沿用了几十年。事实证明，这种通用的设计对于检索效率和检索效果的折中起着重要的作用。在后续的章节中，会进一步详细讨论这些组件。

2.2　基本的构件

搜索引擎的组件主要提供两种功能，也就是我们所说的索引处理（indexing process）和查询处理（query process）。索引处理建立可查找的数据结构，查询处理使用这些数据结构和用户的查询生成一个排好序的文档列表。图2-1给出了索引处理的高级“构件”（building block）。这些主要的组件包括文本采集（text acquisition）、文本转换（text transformation）和索引创建（index creation）。

⊖ http://www.research.ibm.com/UIMA。

文本采集组件用于发现文档，并且使这些文档能够被搜索到。尽管有时候系统可以仅仅使用已有的文档集合，但文本采集通常需要通过爬行（crawling）或者扫描互联网、企业内部网、桌面或者其他信息源，来建立一个文档集合。除了将这些文档传递给索引处理中的下一个组件，文本采集组件还创建一个文档数据库，其中包含所有文档的文本和元数据（metadata）。元数据表示的不是文档的部分内容，而是关于一篇文档的信息，如文档类型（例如电子邮件、网页）、文档结构，或者其他的特征，如文档的长度。

文本转换组件将文档转换为索引项（index term）或者特征（feature）。顾名思义，索引项是文档的一部分，存储在索引表中并且用于搜索。最简单的索引项是一个词，但并不是每一个词都可以用于搜索。“特征”更普遍地是应用于机器学习领域中，是指文档的一部分，用于表达文档的内容，特征也可以用来描述索引项。其他类型的索引项或者特征，是诸如短语、人名、日期、网页中的超链接等等。索引项有时也简单地称作“词项”。索引整个文档集合的所有词项集合，称为索引词表（index vocabulary）。

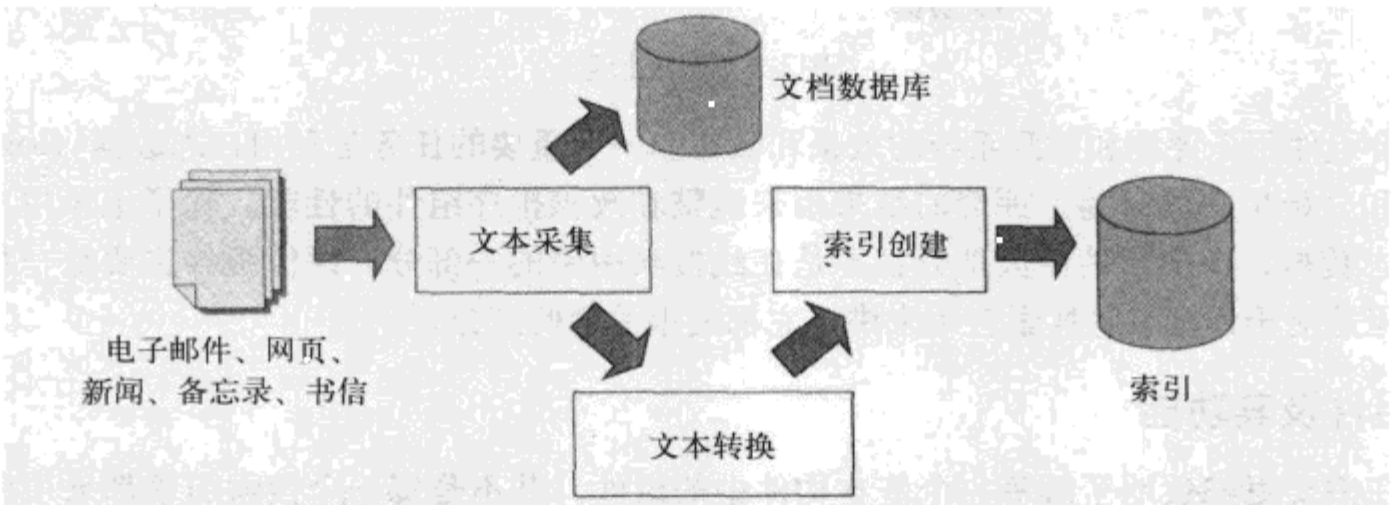


图2-1 索引处理

索引创建组件利用文本转换组件的输出结果，创建索引或者数据结构，以便于实现快速搜索。在一些应用系统中，文档的规模很大，索引的创建在时间和空间上都必须是高效的。当新的文档加入到文档集合中，索引表必须能够高效地更新（updated）。倒排索引（inverted index），有时也称为倒排文件（inverted file），是到目前为止搜索引擎使用得最普遍的索引形式。倒排索引中，每一个索引项都含有一个列表，列表中包含那些含有该索引项的所有文档。这是一种相对意义的倒排：针对一个文档集合，每一个文档都含有一个列表，列表中包含该文档含有的所有索引项。倒排索引的形式多种多样，而索引的使用形式是搜索引擎中的一个重要方面。

图2-2给出了查询处理中的构件。主要的组件包括：用户交互（user interaction）、排序（ranking）和评价（evaluation）。

用户交互组件提供了搜索用户和搜索引擎之间的接口。用户交互组件的一个功能是接收用户查询并将它转换为索引项；另一个功能是从搜索引擎得到一个排好序的文档列表，并将它重新组织成搜索结果显示给用户。例如，包括生成概括文档的摘要（snippet）。文档数据库是用于生成结果的一个信息源。最后，该组件还提供一些技术，用于完善用户的查询，以便它能够更好地反映用户需求的信息。

排序组件是搜索引擎系统的核心。它使用从用户交互组件得到的转换之后的查询，并且根据检索模型生成一个按照分值排好序的文档列表。排序必须是高效的，因为短时间内需要处

理大量的用户查询；排序也必须是高质量的，因为排序的质量决定着搜索引擎是否能够实现找到相关信息的目标。排序的效率依赖于索引技术，而排序的质量依赖于所采用的检索模型。

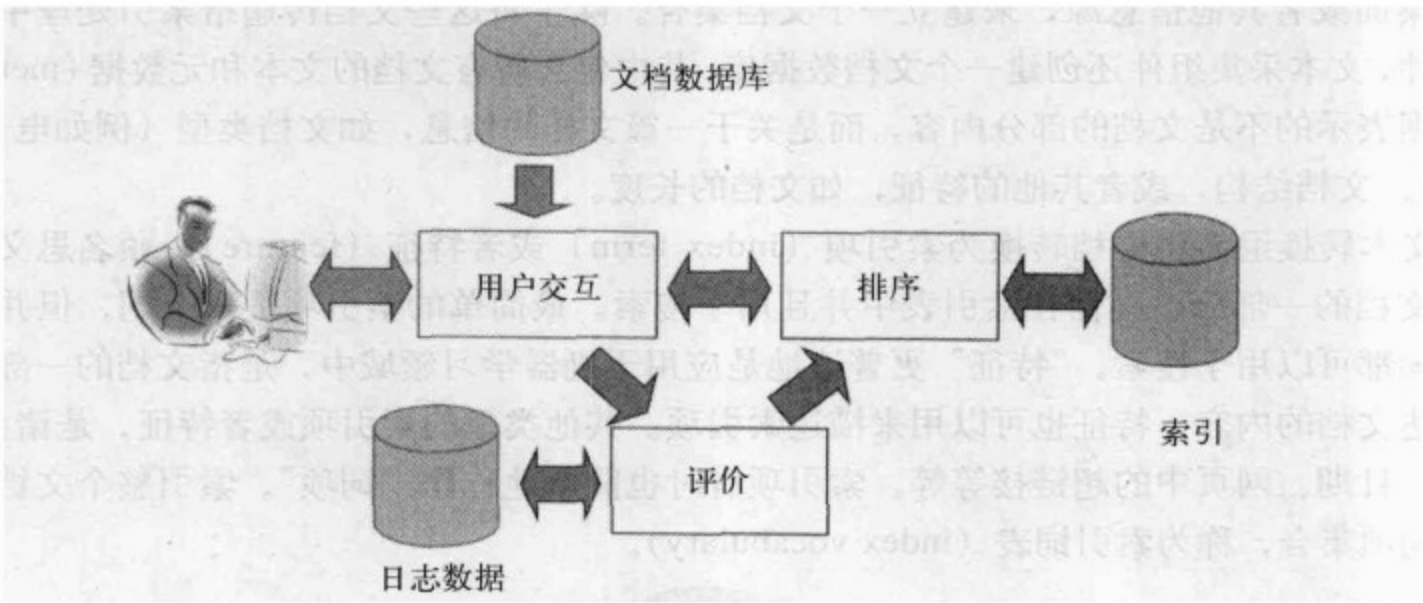


图2-2 查询处理

评价组件用于评测和监测系统的效果和效率。一个重要的任务是利用日志数据（log data）来记录和分析用户的行为。评价的结果用来调整和改善排序组件的性能。除了记录用户和系统的日志数据，评价组件中大部分都不是在线搜索引擎的一部分。评价组件主要是一种离线行为，但是对于任意一种搜索应用来讲，它都是很关键的部分。

2.3 组件及其功能

现在需要更细致地了解每一个基本构件中的组件。并不是每一个搜索引擎都采用所有这些组件，但是它们涵盖了大部分的搜索应用中我们认为最主要的功能。

2.3.1 文本采集

1. 爬虫

在一些搜索引擎应用系统中，爬虫（crawler）组件对于搜索引擎发现和抓取文档具有首要的责任。爬虫的类型有很多种，但最普遍的是网络爬虫。网络爬虫通过追踪网页上的超链接来找到并下载新的页面。尽管这听起来很简单，但是如何能够高效处理互联网上大量出现的新网页，而且如果上次爬虫抓取过的网页发生了变化，如何保证所抓取的页面是“时新的”，这对于网络爬虫的设计是一项极富挑战性的任务。网络爬虫的抓取任务可以限制在一个单独的站点，如一所大学的站点，以此为基础进行站内搜索（site search）。主题（focused）网络爬虫或者话题（topical）网络爬虫采用分类技术来限制所访问的网页是关于同一个主题的。对于垂直搜索（vertical search）或者话题搜索（topical search）的应用系统，比如一个搜索引擎只提供对医学信息的存取，可以采用主题网络爬虫或者话题网络爬虫。

对于企业搜索，爬虫适用于找到并更新所有与公司运营相关的文档和网页。企业文档爬虫（document crawler）跟踪网页上的超链接来发现外部的和内部的（限制在企业内部网）页面，而且还必须扫描公司的和个人的目录，来发现电子邮件、文档、讲稿、数据库以及其他的公司信息。文档爬虫还可以用于桌面搜索，虽然只是对用户个人的目录进行扫描。

2. 信息源

文档信息源（document feed）是一种存取实时文档流的机制。例如，新闻信息源是一个持续不断的新闻流及新闻的更新。爬虫必须能够发现新的文档，而与之相比，搜索引擎期望仅仅通过监测信息源，就能够抓取新的文档。对于新闻、博客、视频等这样一些互联网上的内容，RSS[㊀]是互联网上信息源采用的一个通用标准。RSS“阅读器”用于支持RSS信息源，RSS信息源都采用标准的XML[㊁]数据格式。与HTML[㊂]语言类似，XML是一种用于描述数据格式的语言。阅读器监测信息源，可以获取信息源更新的内容。广播和电视信息源也可以应用于搜索中，这种类型的“文档”中含有一些自动分段的音频和视频流，以及相关的隐藏字幕和语音。

3. 转换

由爬虫发现的文档或者由信息源提供的文档，通常都不是纯文本，它们的格式多种多样，如HTML、XML、Adobe PDF、Microsoft Word，或者是Microsoft PowerPoint等等。大多数的搜索引擎需要将这些文档转换成统一的文本格式和文档的元数据格式。在转换过程中，控制序列和具有特殊格式的非内容的数据，或者被删除，或者作为元数据进行记录。大部分对于HTML和XML文档的处理，可以看作是文本转换组件中的一部分。而对于其他格式的文档，转换过程是作为对该文档进行深入加工处理的基础。例如，PDF文档必须转换成文本。有很多实用程序可以完成这个转换过程，而不同的程序转换的精度是不同的。类似地，现在也有很实用程序，可以完成对Microsoft Office系列文档的转换。

另外一个常见的文本转换问题，是由文档中文本的编码方式引起的。ASCII[㊃]编码是对于文本中单字节字符编码的通用标准方案。ASCII编码采用7位或8位来表示128个或256个字符。然而，有些语言，如汉语，比英语含有更多的字符，并使用很多其他的编码方案进行编码。Unicode是一个通常使用16位进行编码的标准编码方案，可以表示出世界上绝大多数语言中使用的文字。实际应用中，在对不同语言的文档进一步深入处理之前，必须要保证它们使用统一的编码方案进行了转换。

4. 文档数据库

文档数据库用于管理大量的文档及与这些文档相关的结构化数据。从效率的角度来考虑，文档内容通常压缩之后进行保存。结构化数据包括文档的元数据，以及从文档中抽取出来的其他信息，如超链接和锚文本（anchor text，与超链接关联的文本）。关系数据库系统（relational database system）可以用来存储文档和元数据。然而，在一些实际应用中，可以使用更简单、更有效的存储系统，以便实现对大规模文档集的快速检索。

尽管原始文档在互联网上可以随时存取，但在企业数据库中文档数据库是必需的，可以为大量搜索引擎组件提供对文档内容的快速存取。如果搜索引擎必须从互联网上获取原始文档并重新对文档进行处理，那么，为检索回来的文档生成摘要将花费很长的时间。

㊀ RSS实际上是指具有相似名称（和具有相同首字母）的标准族，例如简易信息聚合（Really Simple Syndication）或者丰富站点摘要（Rich Site Summary）。

㊁ 可扩展标记语言（eXtensible Markup Language）。

㊂ 超文本标记语言（HyperText Markup Language）。

㊃ 美国信息交换标准码（American Standard Code for Information Interchange）。

2.3.2 文本转换

1. 解析器

解析组件负责处理文档中的文本词素（tokens）序列，以识别文档中的结构化元素，如标题、图表、超链接和页首文字等等。词素切分（tokenizing）是该项处理中的第一个重要步骤。有时候，词素和词是等同的。文档和查询中的文本必须以同样的方式转换为词素，这样它们之间可以相互比较。对于一个词素会得到多种结果，这会潜在地影响到检索，因此词素切分是一项很有意义的任务。例如，词素的简单定义是由空格分开的字母与数字构成的字符串。然而，这并没有告诉我们如何处理那些特殊的字符，如大写字母、连接符和单撇号。“apple”和“Apple”是一样的吗？“on-line”是一个词还是两个词？“O'Connor”中的单撇号可以看作和所有格是等价的吗？在有些语言中，词素切分的问题更加有趣，如中文，中文里没有像英文那样明显的词之间的分隔符。

文档结构通常由HTML、XML等标记语言来指定。HTML是用来指定网页结构的缺省标记语言。XML相对来说更加灵活，是许多实际应用系统中使用的数据交换格式。文档解析器使用标记语言中的句法（syntax）知识来识别文档的结构。

HTML和XML都使用标签（tag）来定义文档的元素（element）。例如：<h2>Search</h2>定义“Search”是HTML文档中的二级标题。词素切分时，标签和其他控制序列必须进行相应的处理。其他的文档类型，如电子邮件和讲稿，由指定的句法和方法来规范文档的结构，但大部分对这种文档的处理是在转换组件中删除或简化。

2. 停用词去除

停用词去除组件具有简单的任务，从那些成为索引项的词素序列中删除常用词。最常用的词是一些典型的功能（function）词，这些词有助于构成句子的结构，但对于描述文本所涵盖话题的贡献很小，如：“the”、“of”、“to”和“for”。由于它们的用处太普遍，去除这些词可以相当大程度地减少索引的大小。排序取决于所采用的检索模型，但停用词去除对于搜索引擎的效果没有任何影响，甚至可能会有所改善。尽管停用词去除有这些潜在的优点，但很难确定停用词表（stopword list）中应该包含多少个停用词，一些研究中使用的停用词表包含几百个停用词。使用停用词表的问题是，用户如果提交查询“to be or not to be”或“down under”，搜索引擎不可能返回搜索结果。为了避免这样的问题，搜索引擎系统在处理文本的时候，可以使用一个很小的停用词表（可能仅含有一个停用词“the”），但在对于查询文本进行处理的时候，则用一个较大的停用词表。

3. 词干提取

词干提取是另外一个单词级别的转换任务。词干提取组件（或词干提取器，stemmer）的任务是把同一个词干（stem）得到的派生词进行归类。例如，把“fish”、“fishes”、“fishing”可以归为一类。通过使用一个给定的词（如最短的词，上面的例子中是“fish”）来替换类中的每一个元素，可以进一步提高查询与文档中词之间匹配的可能性。事实上，词干提取对排序的效果通常只有很小的改善。类似于停用词去除，词干提取可以针对所有词进行、谨慎地针对少部分词进行或可以干脆不做。针对所有词进行词干提取可能会导致搜索问题。例如，用户提交查询“fishing”，系统检索回来的文档中包含的是“fish”的其他词形的词，这样的检索结果不是很恰当的。有些搜索引擎应用中，谨慎地对少部分词进行了词干提取，如只用

“s”来识别复数形式，或者在处理文档中的文本时不进行词干提取。词干提取的工作集中在对查询文本进行适当的词的变形。

与英语相比，一些语言，如阿拉伯语，具有更复杂的词汇形态（morphology），词干提取必然显得格外重要。阿拉伯语中，高质量的词干提取组件对于搜索效果有着重大的影响。与之相比，其他的语言如中文，词形的变化很少，在这些语言上进行词干提取不会产生任何效果。

4. 超链接的抽取和分析

在对文档进行解析的过程中，网页中的超链接和锚文本可以很容易地被识别并抽取出来。抽取意味着这些信息可以记录在文档数据库中，并且可以和文本内容分开索引。网络搜索引擎通过使用像PageRank（Brin&Page, 1998）这样的链接分析（link analysis）算法，广泛地利用超链接和锚文本这些信息。链接分析向搜索引擎提供一个页面的关注度，并且在一定程度上还向搜索引擎提供一个页面的权威度（authority，换句话说，该页面的重要度）。锚文本（anchor text）是网络链接上可以点击的文本，可以用来提高链接所指向网页的文本内容对用户的吸引力。对于有些类型的查询，这两个因素可以很大程度地改善网络搜索的效果。

5. 信息抽取

信息抽取用于识别更加复杂的索引项，而不是一个单独的词。这些索引项可能简单地是一个黑体、加粗的词，或者是题目中的词，但通常需要更多的附加计算。抽取句法特征，如名词短语，需要某种形式的句法分析和词性标注（part-of-speech tagging）。该领域的研究专注于抽取具有指定语义内容的特征，例如，命名实体（named entity）识别器，能够可靠地识别如人名、公司名称、日期和地名等信息。

6. 分类器

分类器组件为文档或文档中的部分内容识别出与类别相关的元数据，这覆盖了那些经常单独描述的功能。分类技术给文档分配事先定义好的类别标签，这些标签代表性地表达话题的类别，如“体育”、“政治”或“商业”。其他类型的分类技术的两个重要实例是，判别一个文档是否是垃圾文档，以及识别文档中的非内容部分，如广告。聚类技术用于在没有事先定义类别标签的基础上，将相关的文档聚集在一起。在排序或用户交互过程中，这些经过聚类的文档可以以多种方式使用。

2.3.3 索引的创建

1. 文档统计

文档统计组件的功能只是简单地汇总和记录词、特征及文档的统计信息。排序组件使用该信息来计算文档的分值。通常所需要的数据包括索引项在各文档中出现次数（词及更加复杂的特征）、索引项在文档中出现的位置、索引项在一组文档（如所有标记为“体育”的文档或者整个文档集合）中出现的次数，以及按照词素数量统计的文档长度。真正所需要的数据是由检索模型和排序算法来决定的。文档统计结果存储在查找表（lookup table）中，查找表是设计用于快速检索的一种数据结构。

2. 加权

索引项的权值（weight）反映了文档中词的相对重要性，并且用于为排序计算分值。权值

的形式是由检索模型来确定的。加权组件利用文档统计结果计算权值，并将权值存储在查找表中。权值的计算可以是查询处理的一部分，并且一些类型的权值需要关于查询的信息，但在索引过程中需要尽可能多的计算，这样可以提高查询处理的效率。

在过去的检索模型中，最普遍使用的一种加权方法称为 *tf.idf*。该方法有很多种变形，但是它们都基于索引项出现在一个文档中的次数或频率（词频，或者*tf*）以及索引项在整个文档集合中出现的频率（反文档频率，或者*idf*）两者的组合。*idf* 称为反文档频率，因为如果一个词素出现在少量的文档中，那么该词项被赋予较大的权值。*idf* 典型的计算公式是$\log N/n$，其中N是搜索引擎索引的文档数量总数，n是包含一个特定词项的文档数量。

3. 倒排

倒排（inversion）组件是索引处理的核心组件。它的任务是将文本转换组件传递过来的文档-词项信息流转换为词项-文档信息，以便于建立倒排索引。如何高效完成这项工作是极富挑战性的，不仅在索引初始创建时需要处理大量的文档，而且在爬虫或信息源得到新的文档时索引可以被及时更新。倒排索引设计用于快速的查询处理，并且在一定程度上依赖于所采用的排序算法。索引还被压缩以便于进一步提高效率。

4. 索引分派

索引分派组件将索引分发给多台计算机，很可能是网络中的多个站点。分布式处理是网络搜索引擎效率的基础。通过分派文档子集的索引表（文档分派，document distribution），索引和查询处理都可以并行（parallel）进行。分派词项子集的索引（词项分派，term distribution）还能够支持查询的并行处理。复制（replication）是分派的一种形式，索引表或部分索引表存储于多个站点，由此查询处理能够通过减少通信延迟进一步提高效率。对等搜索涉及较少的分布式组织形式，网络中的每一个节点维护自己的索引表和文档集合。

2.3.4 用户交互

1. 查询输入

查询输入组件为查询语言（query language）提供接口和解析器。在大多数网络搜索接口中使用的最简单的查询语言仅有少量的操作符（operator）。操作符是查询语言中的命令，用于指示文本需要进行特殊方式的处理。通常，通过限制文档中的文本与查询中的文本如何匹配，有助于使用户查询的意义更加清晰。例如，在一个简单的查询语言中使用引号操作符，该操作符指示引号中包围的词在文档中作为一个短语出现，而不是以单独的没有任何联系的词出现在文档中。然而，典型的网络查询中仅包含少量的关键词（keyword），而没有操作符。关键词只是一个简单的词，对于指定查询的话题来说是很重要的。因为大多数网络搜索引擎使用的排序算法是根据关键词查询设计的，对于那些含有较低比例关键词的较长的查询，效果并不好。例如，对于两个查询“搜索引擎”和“搜索引擎中使用的经典技术和数据结构是什么”，在网络搜索引擎中，前一个查询会比后一个查询得到更好的结果。搜索引擎设计中的一个挑战性的任务是，对于一系列的查询给出好的结果，对于更多规范的查询，给出更好的结果。

对于那些希望对搜索结果有更多的控制，或者对于使用搜索引擎的应用系统来说，可以使用更加复杂的查询语言。正如SQL查询语言并不是为数据库应用系统的用户（终端用户，the end user）设计的，同样，这些查询语言也不是为搜索应用的终端用户设计的。布尔型

(Boolean) 查询语言在信息检索中有着较长的历史。这种查询语言中使用的操作符包括 AND、OR和NOT，以及一些临近 (proximity) 操作符，用于指示词必须要在规定的距离内一起出现 (通常根据字数统计)。另外一些查询语言包括这些操作符和其他概率模型中的操作符，这些操作符的设计旨在认可与文档结构和内容相关联的特征规范。

2. 查询转换

查询转换组件包括一系列的技术，这些技术用于在生成排好序的文档之前和之后改善初始查询。最简单的处理涉及一些对文档进行文本转换的技术。在查询文本上，需要进行词素切分、停用词去除和词干提取这些工作，以生成与文档词项具有可比性的索引词。

拼写检查 (spell checking) 和查询建议 (query suggestion) 是查询转换中的技术，生成与用户初始查询相似的输出。在这两种情况下，向用户提供初始查询的一些候选查询，这些候选查询可能纠正了拼写错误或者是对用户所需信息的更规范的描述。这些技术通常会导致为网络应用搜集大量的查询日志 (query log)。查询扩展 (query expansion) 技术是对查询进行推荐或者增加一些额外的词项，但通常都是在对文档中词项的出现情况分析的基础上进行的。该分析通常是用不同的信息源，如整个文档集合、检索到的文档或者用户计算机上的文档。相关反馈 (relevance feedback) 是一种查询扩展技术，利用用户认为相关的文档中出现的词项对查询进行扩展。

3. 结果输出

结果输出组件负责对相关组件得到的排好序的文档的结果进行显示。可能包含的任务有生成网页摘要 (snippets) 来对检索到的文档内容进行概括；强调 (highlighting) 文档中重要的词和段落；对输出结果聚类以找到文档相关的类别；以及将相应的广告增加到结果显示中。在涉及多种语言的应用系统中，结果可能会被翻译成同一种的语言。

2.3.5 排序

1. 打分机制

打分组件也称为查询处理 (query processing)，在检索模型的基础上，使用排序算法来计算文档的分值。一些搜索引擎的设计者会明确说明系统所采用的检索模型。而另外一些搜索引擎的设计者，只讨论排序算法 (展示算法的所有细节)，所有的排序算法都是隐含地建立在某种检索模型基础上的。排序算法中使用的特征和权值，可能是凭经验 (empirically，通过测试和评估) 得到的，但必须与话题和用户相关，否则搜索引擎将不会工作。

研究者们提出了很多种检索模型及导出排序算法的方法，通过这些模型计算文档分值，基本的形式如下

$$\sum_i q_i \cdot d_i$$

公式用来计算词汇表中所有词项的总和，q_i是查询中第i个词项的权值，d_i是文档词项的权值。词项的权值依赖于所使用的特定的检索模型，通常类似于tf.idf权值计算方法。第7章中进一步详细讨论基于BM25和查询似然度 (query likelihood) 检索模型的排序算法。

另外，必须快速地计算并比较得到文档的分值，以便确定文档的排序，并将其传递给结果输出组件，这就是性能优化组件的任务。

2. 性能优化

性能优化涉及排序算法和相关联的索引表的设计，以降低系统的响应时间，提高查询吞吐量。对于一个给定的文档打分机制的形式，有很多种方式去计算这些分值，并且生成排好序的文档输出结果。例如，分值可以通过对某个查询词项存取索引表进行计算，计算该词项对文档分值的贡献度，将该贡献度的值添加到一个分值累加器中，然后存取下一个索引表。该方法称为term-at-a-time分值计算方法。另外一种方法是，对于所有的查询词项同时存取所有的索引表，通过在索引表中指针的移动来找到出现这些词项的某一个文档，以此来计算分值。在这种document-at-a-time分值计算方法中，可以快速地得到最终的文档分值，而不是每次累加一个词项的值。对这两种方法可以进一步的优化，以便大幅度地降低计算排序靠前的文档所需要的时间。安全的（safe）优化方式，能保证计算得到的分值和没有经过优化的分值相同。不安全的（unsafe）优化方式，有的时候计算速度更快，但不能保证结果和未经优化的结果相同。因此，谨慎地评价优化方式的影响是很重要的。

3. 分布式

索引是以分布式的形式给出的，那么排序也可以采用分布式。查询代理（query broker）描述了如何在网络中将多个用户查询分派给不同的处理器，并且负责将各处理器返回的结果整合到一起，形成给定查询的排好序的结果列表。代理的操作过程与索引的分派形式有关。缓存（caching）是另外一种分布式的形式，索引或者前一个查询得到的排好序的文档结果列表，都可以保留在本地内存中。如果大量的用户都提交相同的查询或索引项，那么本地内存中保留的这些信息可以被重复使用，从而节省很多时间。

2.3.6 评价

1. 日志

对于调整和改善搜索引擎系统的效果和效率来讲，用户查询及用户与搜索引擎交互的日志是很有价值的信息源。用户的查询日志可以用于拼写检查、相关查询词推荐、查询缓存及其他任务，例如为找到与用户搜索内容相关的广告提供帮助。搜索引擎返回的结果列表中的文档如果被用户点击浏览，那么该文档很可能是与用户查询相关的文档。这意味着用户对文档的点击日志（点击数据）和驻留时间（dwell time，用户阅读一篇文档所花费的时间）等信息可以用来评价和训练排序算法。

2. 排序分析

对于大量的（查询-文档）对，给定日志数据和显示的相关判定，可以对排序算法的效果进行评估，并与候选的算法进行比较。这是改善搜索引擎性能的关键部分，并且可以为应用系统选择合适的参数值。可以采用多样化的评价手段，并可以选择这些方法来评价应用系统的输出结果。对输出结果的评价侧重于排序靠前的文档的质量，而不是整个结果列表的文档质量。

3. 性能分析

性能分析组件专注于监测和改善系统的整体性能，排序分析组件以同样的方式监测系统的效果。可以采用多种多样的评价方法来进行系统的性能分析，如响应时间、吞吐量，但是评价方法的使用还依赖于具体的应用。例如，对于一个分布式搜索引擎系统来说，除了上述的评价方法之外，还要检测网络的使用情况和效率。模拟是与性能分析等效的方法，真实的

网络、处理器、存储设备及数据，都可以使用参数可调的数学模型来代替。

2.4 搜索引擎是如何工作的

现在了解了一个搜索引擎中各组件的名称和它的功能。但是，对于这些组件在实际中如何执行它们的功能，没有进行更多的阐述。这是本书的后续章节中将讨论的内容。在每一章中，都深入阐述了一个或多个组件是如何工作的。如果在阅读了相应的章节之后，你仍然不理解所描述的组件，就可以阅读每章后面给出的参考文献，或者学习Galago的代码。Galago的代码是本书中的想法的具体实现。

参考文献和深入阅读

关于组件描述中提到的技术和模型的详细参考文献，会在相应的章节中给出。关于搜索引擎架构的参考文献为数不多。在数据库教科书中，Elmasri和Navathe（2006）描述了数据库的系统架构和相关的查询语言，将它们和本章中讨论的搜索引擎系统架构进行比较是很有意思的。它们在高层次的组件是有些相似的，然而数据库系统着重于结构化的数据和精确匹配，而不像搜索引擎，重点在于处理文本和排序算法。因此，大多数的组件是不同的。

Brin&Page（1998）是一篇经典的关于搜索引擎架构的研究论文。Callan等（1992）发表了另外一篇关于早期的通用搜索引擎（Inquery）的概述。Hatcher和Gospodnetic（2004）的论文中全面描述了Lucene的架构和组件。

练习

2.1 在Galago的代码中，找出一些本章描述的搜索引擎组件的实例。

2.2 文档过滤是一种应用系统，该系统中存储了大量的用户查询和用户需求文档，将来自信息源的每一个文档与这些用户需求文档进行比较。那些与用户需求文档足够相似的文档通过电子邮件或其他方式推送给该用户。请对过滤引擎的架构进行描述，并阐述它和搜索引擎的不同之处。

2.3 如果用户点击了搜索引擎返回的结果列表中的一个特定的文档，告诉搜索引擎去检索与用户点击的文档相似的那些文档，那么这种查询称为more-like-this查询。为了回答这种类型的用户查询，需要使用哪些低层次的组件？这些组件的使用顺序如何？

第3章　信息采集和信息源

“你不要再干预我的事情了。”

——Doc Ock，《蜘蛛侠2》

3.1　确定搜索的内容

本书是关于构建搜索引擎系统的详细描述，从排序使用的算法到查询处理的方法。虽然我们重点关注的是那些使搜索引擎能够工作的技术，而且更好的技术能够使搜索引擎具有更好的性能，但是文档集合中的信息才使搜索引擎变得更加有用，换言之，如果搜索引擎中没有存储合适的文档，就没有一种搜索技术能够找到相关的信息。

本节的标题隐含地回答了这样一个问题，“我们想要搜索什么？”。简单的答案是：一切你所能搜索的信息。虽然一些最好的文档回答了更多的问题，但每一个文档至少回答了一个问题（例如，“Now where was that document again?”）。每次搜索引擎增加不同的文档，它能够回答的问题数目也随之增加。另一方面，为了找到最好的结果显示给用户添加一些低质量的文档会增加排序处理的负担。然而，对于网络搜索引擎来说，即使系统存储着几十亿没太大用处的低质量文档，也能显示出搜索引擎的成功之处。

即便是有用的文档，经过一段时间之后也会变得用处不大，尤其是新闻和财经信息。例如，大部分人想要知道的是当天股票市场的报道，只有少数的人会关心昨天的市场发生了什么。遗憾的是，我们经常会在搜索结果列表中发现过期的网页和链接。搜索引擎除了收集那些旧的资料外，还含有尽可能多的近期信息，这时搜索引擎的效果是最好的。

本章介绍了找到要搜索的文档的相关技术，无论这些文档是在互联网上、文件服务器上、计算机的硬盘上或者是在电子邮件程序中。我们将讨论存储文档以及如何保持这些文档是最新的策略。接下来，将讨论如何从文件中获取数据，以及浏览中存在的问题，如字符编码、过时的文件格式、重复文档以及文本噪声。通过本章的学习，你将能够牢固地掌握如何为搜索引擎准备文档数据，为建立索引做准备。

3.2　网络信息爬取

为了建立一个能够对网页进行搜索的搜索引擎系统，首先需要对你希望搜索的网页有一个备份。与后面我们将要考虑的一些其他类型的文本资源不同，网页很容易进行备份，因为通过互联网用浏览器可以将网页抓取到本地。这解决了获取要搜索的信息的一个主要问题：如何将数据从它存储的地方取出来交给搜索引擎。

自动地发现并下载网页称为爬取（crawling），下载网页的程序称为网络爬虫[⊖]（web crawler）。对于网页的采集，存在很多特殊的问题。最大的问题是互联网的规模非常庞大，互联网上至少有上百亿的网页。上句话中提到的“至少”，是因为没有人能够确定互联网上网页

⊖　爬取有时候也指spidering，爬虫有时也称为蜘蛛（spider）。

的具体数目。即便可以精确地计算当天存在的网页数量，但随着网页不断地产生，这个数字很快就会失效。当一个用户每次增加一个新的博客帖子或者上传一个照片的时候，就创建了不同的网页。对于大多数的组织来说，即便是互联网上的大部分网页，也没有足够的存储空间来存储。但对于拥有大量资源的网络搜索提供者来讲，必须能够不断地下载新的内容，以保持文档集合的内容是最新的。

另一个问题是，互联网上的网页通常不受搜索引擎数据库创建者的控制。即便知道想要从www.company.com拷贝所有的网页，也没有简单的方法能够知道这个站点有多少个页面。该站点的拥有者可能会不希望你拷贝其中的一些数据，而且如果你很快或者很频繁地拷贝站点上的数据，就会引起站点拥有者的愤怒。有些你想要拷贝的数据可能只能通过在表单中填写需求才能获取，这很难实现自动处理。

3.2.1 抓取网页

互联网上的每一个网页都有自己唯一的统一资源定位器（uniform resource locator），或URL。用于描述网页的URL由三部分组成：协议方案、主机名、资源名（如图3-1所示）。网页存储在网络服务器上，使用超文本传输协议（Hypertext Transfer Protocol, HTTP）来和客户端软件交换信息。因此，互联网上使用的绝大多数URL都以http开始，指出该URL表示的资源可以使用HTTP协议进行抓取。接下来的主机名（hostname），是保存该网页的网络服务器的计算机名。图中，计算机的名字是 www.cs.umass.edu，是马萨诸塞大学计算机系的一台计算机。该URL指向这台计算机上的一个页面/csinfo/people.html。

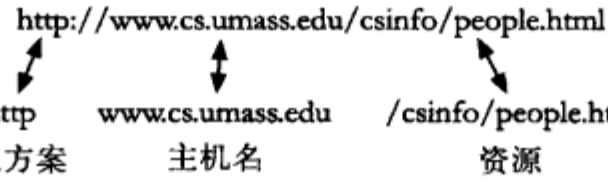


图3-1 统一资源定位器（URL）切分为三个部分

网络浏览器和网络爬虫是两种不同的网络客户端，但都以相同的方式来获取网页。首先，客户端程序连接到一个域名系统（domain name system, DNS）服务器上。DNS服务器将主机名转换成IP（internet protocol）地址。典型的IP地址为32位二进制数，但现在有些网络使用128位的IP地址。接下来，客户端程序试着连接具有该IP地址的服务器。服务器上可能有多个不同程序在运行，每个程序都在监听网络以发现新的连接，各程序监听不同的端口（port）。端口是一个16位的数字，用来辨识不同的服务。除非在URL中指定了其他的端口，否则对网页的请求通常都发送到80端口。

一旦建立了连接，客户端程序发送一个HTTP请求给网络服务器，以请求一个页面。最常见的HTTP请求是GET请求，例如：

```
GET /csinfo/people.html HTTP/1.0
```

该命令请求服务器使用HTTP协议规范的1.0版本，将页面 /csinfo/people.html 返回给客户端。服务器在发送一个简短的头部信息之后，将该文件的内容返回给客户端。如果客户端需要更多的页面，可以发送其他的请求；否则，客户端关闭该连接。

客户端程序也可以通过使用POST请求获取网页。除了POST请求可以向服务器发送额外的请求信息之外，它和GET请求类似。习惯上，GET请求用于抓取已经在服务器上存在的数

据，而POST请求用于告诉服务器一些事情。当你点击一个按钮购买商品或者对网页进行编辑时，就可以使用POST请求。如果运行一个网络爬虫，这个惯例是很有用的。由于只是发送了一个GET请求，就有助于确保网络爬虫不会在无意间去购买一件商品。

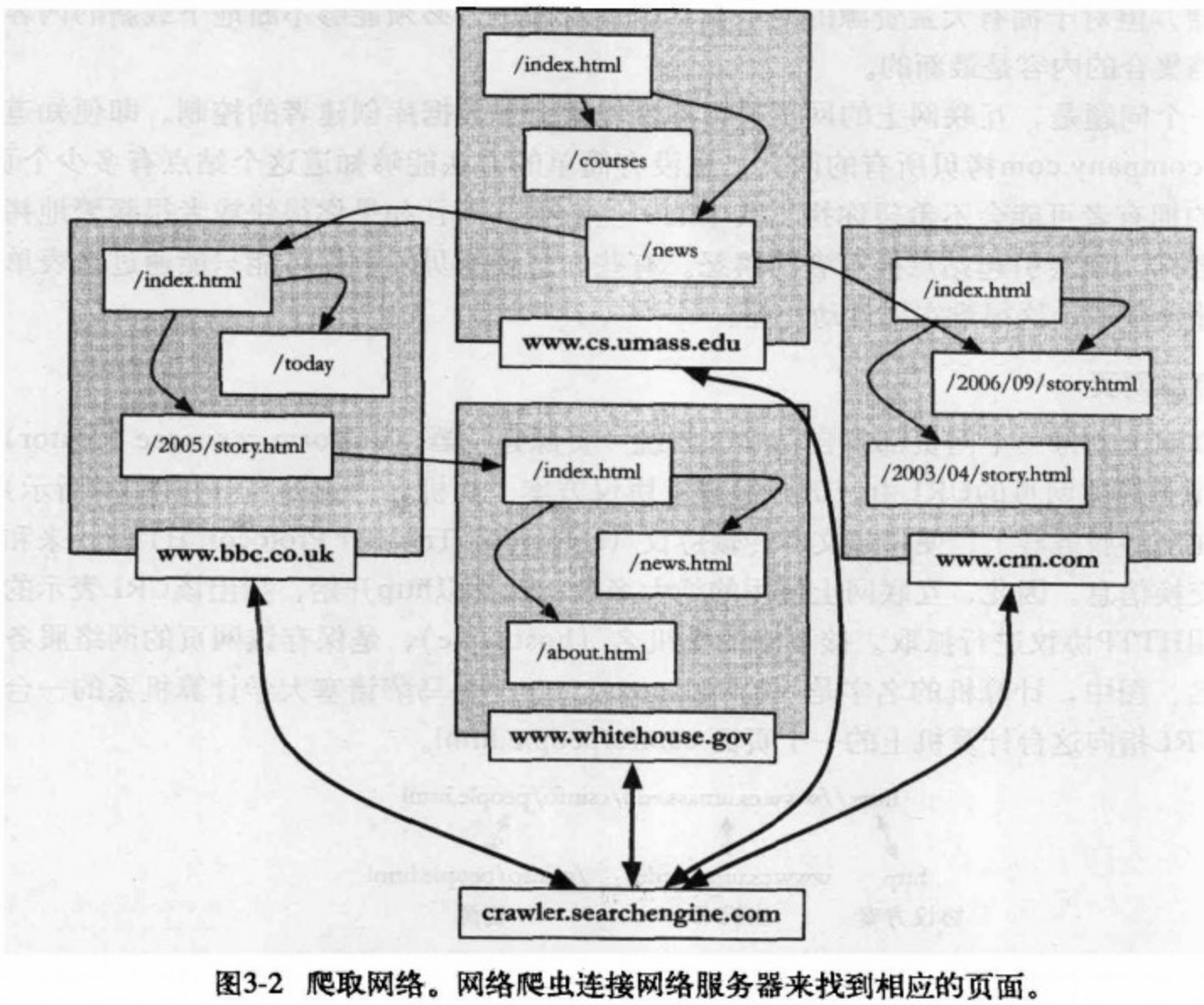


图3-2 爬取网络。网络爬虫连接网络服务器来找到相应的页面。
页面可能指向同一个服务器或不同服务器上的其他页面

3.2.2 网络爬虫

图3-2从一个简单的网络爬虫的角度，给出了网络信息采集的图示。网络爬虫有两个任务：下载页面和发现URL。

网络爬虫的工作从一个种子（seed）集合开始，种子集合是作为参数传递给网络爬虫的一个URL的集合。这些种子被添加到URL请求队列中。网络爬虫从请求队列中取出URL地址，开始抓取页面的任务。一旦下载了一个页面，就对该页面进行解析，找到链接标签，其中可能会含有用于抓取的URL地址。如果网络爬虫发现一个以前没有遇到过的URL地址，则将该地址添加到请求队列或frontier中去。frontier可以是一个标准的队列，或者是经过排序的队列，以便使重要的页面置于队列的前面。这个过程重复进行，直到网络爬虫用尽存储页面的磁盘空间或者请求队列为空。

如果网络爬虫只使用一个单一的线程，效率将会很低。可以看到，网络爬虫大量的时间都消耗在等待响应上：等待DNS服务器的响应，然后等待与服务器的连接被确认，接下来还要等待从服务器发送的网页数据。在等待的过程中，网络爬虫所使用的机器的CPU是处于空闲状态的，而且这时也没有利用到网络连接。为了提升效率，网络爬虫使用多个线程，一次

抓取几百个网页。

一次抓取几百个网页，对于使用网络爬虫的人来说是一件好事，但对于网络服务器的所有者来说却未必是件好事。设想一下，请求队列在实际中是如何工作的。当抓取到一个页面后，比如 www.company.com，该页面被解析，页面中所有的链接都添加到请求队列中。接下来，网络爬虫会试图一次抓取所有这些链接指向的页面。假如该服务器 www.company.com 的性能并不是很强大，它将花费所有的时间来处理网络爬虫的请求，而不会去处理真实用户的请求。网络爬虫的这种行为会导致网络服务器管理员非常生气。

为了避免这个问题，网络爬虫使用了礼貌策略（politeness policy）。有礼貌的网络爬虫不会在特定的网络服务器上一次抓取多个页面。另外，在同一个网络服务器上的两次请求之间，网络爬虫至少等待几秒钟，有时也可以是几分钟的时间。这样就能够使网络服务器将大多数的时间花费在处理真实用户的请求上。为了支持该策略，请求队列在逻辑上为每个网络服务器划分出一个单独的队列。在任何时间，绝大多数这些单独的队列都会被禁止进行爬取，因为网络爬虫已经在近期从相应的服务器上抓取了页面。对于那些在礼貌策略规定的时间窗口内未被存取的队列，网络爬虫可以自由地读取页面请求。

当使用礼貌策略的时间窗口时，请求队列必须很大，以获得好的性能。假设网络爬虫可以在每秒内抓取100个页面，它的礼貌策略规定每30秒内从特定的网络服务器上抓取的页面数目不能多于1个。为了得到较高的吞吐量，该网络爬虫需要在请求队列中有至少来自3000个不同网络服务器的URL。有些URL会来自同一台服务器，网络爬虫达到吞吐量峰值之前，请求队列中需要含有几万个URL。

```
User-agent: *
Disallow: /private/
Disallow: /confidential/
Disallow: /other/
Allow: /other/public/

User-agent: FavoredCrawler
Disallow:

Sitemap: http://mysite.com/sitemap.xml.gz
```

图3-3 robots.txt文件样例

有些网络服务器管理员反对对他们的数据进行任何拷贝，即使对该站点的信息采集速度很慢，也会惹怒他们。这时候，管理员可以在他们的网络服务器上保存一个称为robots.txt的文件。图3-3给出了robots.txt文件的样例。该文件由 User-agent 开始，由多个命令块组成。User-agent这一行标识一个网络爬虫或者一组网络爬虫受以下的规则约束。这一行的后面是Allow和Disallow规则，规定哪些资源允许该爬虫进行存取。图中，第一个命令块指出除了以 /other/public/开始的目录外，所有的网络爬虫都不允许存取。以 /private/、/confidential/ 或 /other/ 开始的目录。第二个命令块指出，名为 FavoredCrawler 的网络爬虫拥有自己的规则集：允许它复制服务器上所有的内容。

样例中的最后一个命令块是一个可选的 Sitemap：指令，将在本节的后面进行讨论。

```
procedure CrawlerThread(frontier)
    while not frontier.done() do
        website ← frontier.nextSite()
        url ← website.nextURL()
        if website.permitsCrawl(url) then
            text ← retrieveURL(url)
            storeDocument(url, text)
            for each url in parse(text) do
                frontier.addURL(url)
            end for
        end if
        frontier.releaseSite(website)
    end while
end procedure
```

图3-4 信息采集线程的实现

图3-4给出了一个信息采集线程的实现，使用了我们已了解的网络爬虫构件。假设frontier已经用一

些种子URL进行了初始化。信息采集线程首先从frontier中获取一个网站。然后，网络爬虫从该网站队列中识别出下一个URL地址。在permitsCrawl中，网络爬虫根据该网站的robots.txt文件检查这个URL是否允许进行信息采集。如果允许，网络爬虫使用 retriveURL 来抓取文档的内容。这是该循环中最耗时的部分，爬虫线程在此处会阻塞几秒钟。一旦将文本抓取回来，storeDocument就将文档中的文本存储到文档数据库中（在本章后面进行讨论）。对文本的内容进行分析，会发现其他的URL，将新发现的URL添加到frontier中，由frontier负责将它们添加到相应的网站队列中去。所有这些工作做完之后，website对象被返回给frontier，在合理的时间内不将该站点分配给其他的线程，以此来实施它的礼貌策略。在实际的网络爬虫中，计时器在文档被抓取回来之后会立即启动，因为文本分析和存储文档会花费很长的时间。

3.2.3 时新性

互联网上的网页在不断地增加、删除和修改。为了准确地查看网页中的内容，网络爬虫必须不断地对它已经爬取过的网页进行访问，看它们是否发生了变化，以保持文档集合的时新性（freshness）。陈旧的备份是与时新的备份相反的，意味着这个备份已经不再能够反映真实的网页内容了。

客户请求：

```
HEAD /csinfo/people.html HTTP/1.1
Host: www.cs.umass.edu
```

服务器响应：

```
HTTP/1.1 200 OK
Date: Thu, 03 Apr 2008 05:17:54 GMT
Server: Apache/2.0.52 (CentOS)
Last-Modified: Fri, 04 Jan 2008 15:28:39 GMT
ETag: "239c33-2576-2a2837c0"
Accept-Ranges: bytes
Content-Length: 9590
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

图3-5 HTTP HEAD请求和服务器响应

HTTP协议有一个称为HEAD的特殊请求，使得检查页面是否发生变化更加容易了。HEAD请求只返回页面的头部信息，而不是页面内容。图3-5给出了HEAD请求和响应的实例。Last-Modified的值指出了页面内容最后一次发生变化的时间。可以看到，在对GET请求进行响应时，这个日期信息也随着响应一起发送。这就允许网络爬虫对上一次从GET请求接收到的日期与HEAD请求中得到的Last-Modified的值进行比较。

HEAD请求降低了页面检查的开销，但并没有消除这个开销。每分钟对每个页面进行检查是不可能的。这不仅令服务器管理员产生反感，而且会导致网络爬虫和网络连接的负载大量增加。

值得庆幸的是，绝大多数的网页并不是每几分钟就进行更新的。其中的一些的确更新得很频繁，如新闻站点。而其他的站点几乎很少发生变化，如个人主页。即便是相同的页面类型，更新的比率也有很大的不同。例如，有的博客一天内更新很多次，而其他的博客几个月才会更新一次。不断地去检查那些很少更新的站点是无益的。因此，网络爬虫的一个任务就是评估每个页面的变化比率。经过一段时间，该数据可以用于估计每个网页的变化频率。

网络爬虫不可能在每个页面发生变化时都立即对它进行更新，它需要有一些度量方式来

评估采集信息的时新性。本章中已经将时新性用作一般性术语，但是时新性也是一种度量的名字。在时新性的度量下，如果所采集的信息是一个网页最近的备份，那么这个页面就是时新的，否则就是陈旧的。因此，时新性是指所抓取的页面中当前为新网页的比例。

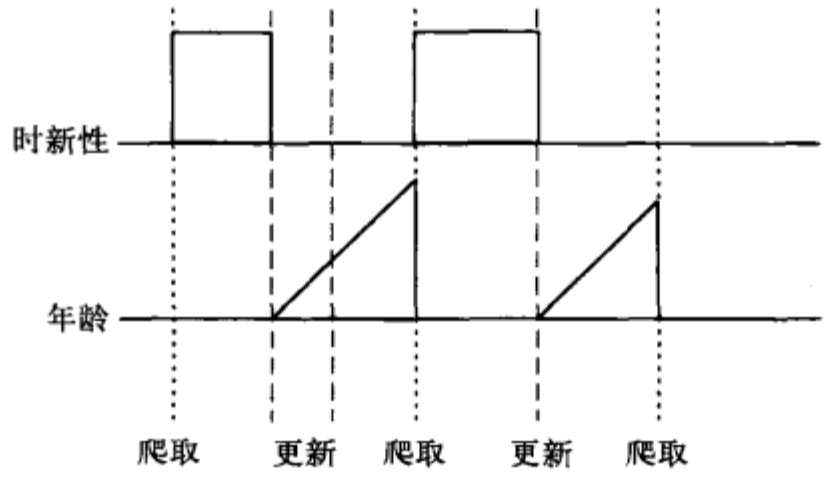


图3-6 一个页面的年龄和时新性随时间的变化

保持时新性看起来好像确实是我们想要做的，但对于时新性的优化会得到意想不到的结果。假设http://www.example.com 是一个众所周知的网站，每隔几分钟它的首页就会有细小的变化。除非你的网络爬虫不停地选择该站点进行网页抓取，否则你所得到的通常是该页面的一份过时的备份。如果你想对时新性进行优化，正确的方法就是彻底停止对该站点的信息采集。如果该页面一向都不是时新的，那么对时新性的值是没有帮助的。取而代之的是，可以将网络爬虫资源分配给那些很少发生变化的页面。

当然，如果你确定要为了时新性对网络爬虫进行优化，这通常会引起用户的反感。因为他们要浏览 http://www.example.com 的页面，但是发现你所引用的该页面的备份过时了几个月，这会使他们产生困惑。

年龄（age）是可以使用的更好的度量方法。在图3-6中可以看到年龄和时新性的不同。从图中的上半部分可以看到，当页面被爬虫采集回来的时候，就立即成为时新的。但是一旦页面发生了变化，所采集回来的网页就成为过时的了。在年龄的度量下，页面的年龄为0，除非该页面发生了变化。这时，页面的年龄逐渐增加，直到该页面被爬虫再次采集，页面的年龄又回到0。

假设一个页面的变化频率为λ，这意味着我们期望该页面在一天的周期内变化λ次。我们可以计算出一个页面从上一次采集t天之后该页面的年龄期望值：

$$Age(\lambda, t) = \int_0^t P(\text{在时间为}x\text{时页面发生变化})(t-x)\,dx$$

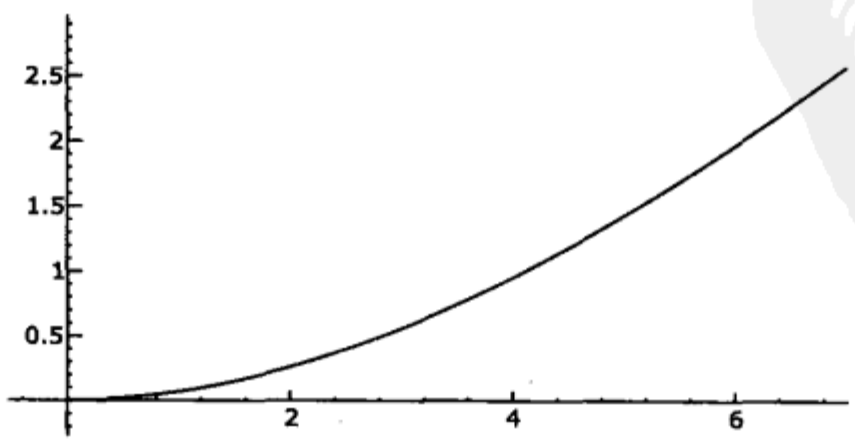


图3-7 一个页面的平均变化频率λ=1/7（一个星期）时，该页面的年龄期望值

表达式（$t-x$）表示的是年龄：假定页面在时间t被采集，在时间x时页面发生了变化。将该表达式与页面在时间x发生变化的概率相乘。研究表明，网页的更新一般遵循泊松分布，这意味着页面下一次更新的时间受指数分布（Cho & Garcia-Molina, 2003）支配。将其插入到表达式P（在时间为x时页面发生变化）中得到：

$$\text{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t-x)\mathrm{d}x$$

图3-7中给出了该表达式的曲线，此时λ=1/7，也就是页面大约每周变化一次。注意，为什么年龄的期望值是从0开始的，而且该曲线在开始时上升得比较缓慢？这时因为页面在第一天时不可能发生变化。随着天数的增加，页面发生变化的概率也随之增大。在周末的时候，该页面年龄的期望值大约为2.6天。也就是说，如果爬虫每周采集一次页面，而且集合中的每个页面平均更新时间为一周，那么在爬虫再次启动采集网页之前，索引中页面的平均年龄为2.6天。

年龄的二阶导数函数值通常是正的。也就是说，图中的曲线不仅是上升的，而且曲线的增长率也是呈上升趋势的。这个正数值还意味着网页的年龄越大，不采集它的代价也越大。对时新性度量的优化可以得出，有时候对一个页面不进行采集是节省开销的，但对年龄这种度量的优化却不能得到这样的结论。

3.2.4 面向主题的信息采集

有些用户会希望搜索引擎关注某一个特定话题的信息。例如，对于一个与电影有关的站点，用户可能会希望通过使用搜索引擎给他们带来更多的电影方面的信息。如果能够正确地构建这种类型的垂直搜索（vertical search），那么与通用搜索相比，它能够提供更高的准确率，因为文档集合中没有无关信息。垂直搜索中的计算开销也会比全网搜索的开销低很多，因为垂直搜索中的文档集合规模相对来讲要小很多。

为垂直搜索抓取网页的最精确的方式是，首先对整个互联网的网页进行采集，然后扔掉所有那些不相关的页面。这种策略需要大量的磁盘存储空间和带宽，而最终大多数的网页都会被筛选掉。

相对来说代价较低的方法是面向主题（focused）或话题（topical）的信息采集。主题爬虫试图只下载那些和一个特定主题相关的页面。主题爬虫依据这样一个事实：一个话题的页面中含有指向同一话题的其他页面的链接。如果这个依据成立的话，主题爬虫可以从一个特定话题的页面开始信息采集，接下来对跟踪这个根页面中的链接采集该话题的所有页面。在实际应用中，一个特定话题的多个权威的页面都被用做种子页面。

主题爬虫需要一些自动工具来判断一个页面是否与某个特定的主题相关。第9章中将会介绍文本分类技术，它可以用做自动判断的工具。一个页面被下载之后，爬虫使用分类器来确定该页面是否与给定的主题相关。如果是相关的，则保留该页面，而该页面中的超链接则用于发现其他相关的站点。页面中的外出链接上的锚文本是主题相关性判定的很重要的线索。还有，一些页面比其他页面含有更多的与话题相关的超链接。在对一个页面中的超链接进行访问的时候，爬虫可以继续跟踪下载页面的话题相关性，以此来确定是否下载其他类似的页面。锚文本数据和与页面超链接话题相关的数据可以结合在一起，用于确定爬虫接下来要采集的页面。

3.2.5 深层网络

并不是互联网中的所有站点都易于进行爬取和导航的。那些网络爬虫很难找到的站点统

称为深层网络（deep Web，也称为隐藏网络，hidden Web）。尽管深层网络的规模很难精确地评估，但一些研究估计它的规模超过传统索引的网络规模一百倍。

属于深层网络的大多数站点都可以归为如下三类：

- 私人站点（private site）。这些站点是倾向于隐私内容的，没有任何指向它的链接，或者在使用该站点之前，需要使用有效的账号进行注册。虽然一些新闻出版商仍然希望他们的内容被主要的搜索引擎索引，但是这类站点通常要求阻止爬虫对页面进行存取。
- 表单结果（form result）。这些站点通常在向表单中填写数据之后才能进入。例如，销售机票的站点，通常在页面的入口处会询问旅行的信息。在你递交了本次旅行的信息之后，航班信息才会显示出来。尽管你可能会希望使用搜索引擎找到航班时刻表，但大多数的爬虫都不可能越过这个表单获取航班时刻表的信息。
- 脚本页面（scripted page）。是使用JavaScript、Flash或其他客户端语言的页面。如果一个链接并不是以HTML语言给出的，而是通过在浏览器中运行JavaScript生成的，爬虫需要在该页面上执行JavaScript才能找到这个链接。虽然在技术上这是可行的，但执行JavaScript会很大程度地影响爬虫抓取页面的速度，并增加系统的复杂性。

有时候人们会对静态页面（static page）和动态页面（dynamic page）进行区分。静态页面是指存储在网络服务器上并在浏览器上不需要修改即可显示的文件，而动态页面可能会是在网络服务器或客户端上执行代码的结果。通常认为静态网页易于采集，而动态网页不易于抓取。然而，事实上并不是这样。一些动态生成页面的站点也易于采集，维基百科就是很好的例子。其他拥有静态页面的站点也不易于采集，因为它们需要通过填写表单才能对页面进行采集。

与私人站点不同的是，使用表单结果和脚本页面站点的管理员通常希望他们的站点被搜索引擎索引。在这两个类别中，脚本页面很容易处理。站点管理员通常只需要对页面进行轻微的改动，超链接就可以在服务器端通过代码生成，而不需要在浏览器中利用代码生成。虽然可能会花费一些时间，但爬虫也可以运行JavaScript及Flash。

最难的问题是那些使用表单结果的站点。通常，这些站点是那些变化的数据的仓库，通过表单将一个用户查询发送给数据库系统。如果数据库中包含几百万条记录，该站点会向搜索引擎的爬虫提供几百万的超链接。将上百万的链接添加到站点的首页中显然是不可行的。另外一个选择就是让爬虫来猜测需要在表单中输入什么样的数据，但很难选择好的表单输入。即便是猜测得比较准确，这种方法也不可能得到所有的隐藏数据。

3.2.6 网站地图

正如在前两节所看到的，信息采集中出现的最大的问题是网站管理员不能很好地将网站的信息告诉爬虫程序。在3.2.3节中了解到爬虫必须去猜测页面什么时候会更新的原因是由于轮询机制开销很大。在3.2.5节中了解到网站管理员们有时候希望一些数据被搜索引擎抓取，但是却做不到，因为没有合理的地方来存储这些超链接。网站地图（sitemap）可以用来解决所有这些问题。

robots.txt文件中含有一个对网站地图的引用，如图3-8所示。网站地图中含有一个URL的列表以及与这些URL相关的数据，如修改的时间和修改的频率。

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.company.com/</loc>
    <lastmod>2008-01-15</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.7</priority>
  </url>
  <url>
    <loc>http://www.company.com/items?item=truck</loc>
    <changefreq>weekly</changefreq>
  </url>
  <url>
    <loc>http://www.company.com/items?item=bicycle</loc>
    <changefreq>daily</changefreq>
  </url>
</urlset>
```

图3-8 网站地图文件的实例

在网站地图文件的实例中有三个URL的记录。每一个记录中，在loc标签中含有一个URL。changefreq标签标识该资源可能经过多长时间发生变化。第一个记录中含有一个las tmod标签，指出上一次发生变化的时间。第一个记录中还包含一个 priority 标签，值为0.7，高于平均值0.5。这个值的目的是告诉爬虫程序，在这个网站中该页面比其他页面更加重要。

为什么网站管理员对于创建一个网站地图会感到困惑？一个原因是为了告诉搜索引擎那些用别的方法可能找不到的页面。看一下网站地图文件实例中的第二个和第三个URL记录，假设它们是通过表单结果生成的页面。在这个网站里可能没有任何链接指向这些页面；而用户必须通过搜索框得到它们。比较简单的网络爬虫不会向搜索框中填写任何数据（尽管有些高级的网络爬虫可能会这样做），这样，这些页面对搜索引擎来说是不可见的。网站地图允许爬虫程序找到这些隐藏的内容。

网站地图还可以使爬虫程序知道页面的修改时间。在关于页面的时新性讨论中，我们提到爬虫程序必须去猜测页面什么时候可能会发生变化。changefreq 标签给爬虫程序提供了一个暗示，告诉它什么时候去对页面是否发生变化进行检查，las tmod 标签则告诉什么时候页面发生了变化。这有助于在不牺牲页面时新性的条件下，减少爬虫发送给网站的请求数量。

3.2.7 分布式信息采集

对单个的网站进行信息采集时，使用一台计算机就足够了。然而，对整个网络进行信息采集则需要多台计算机。为什么不能用一台单独的计算机进行信息采集呢？主要考虑三个方面的原因。

使用多台计算机的一个原因是，将爬虫程序放在采集信息的网站附近。长距离网络连接会具有较低的吞吐量（每秒钟只备份少量的字节）和较长的延迟时间（字节数据需要经过更长距离的传输）。降低吞吐量和增大延迟时间，会使得对每个页面的请求时间也随之增加。随着吞吐量下降和延迟时间的增加，爬虫必须开放更多的连接，以便以同样的速度对页面进行备份。

例如，假设一个爬虫程序有一个传输率为 1MB/s 的网络连接。网页的平均大小为20K，

那么爬虫程序每秒钟可以备份50个页面。如果被采集信息的网站和爬虫程序所在的计算机距离很近，那么数据传输率可能会达到1MB/s。然而，网站开始传输数据需要80ms的时间，因为在打开连接和发送请求之间有传输延迟。假设每次请求需要100ms的时间（80ms的延迟时间，20ms的数据传输时间）。用50乘以100ms，可以看到，在传输50个页面时需要5秒钟的时间，其中包括延迟等待时间。这意味着，在一秒钟内传输50个页面需要使用5个连接。如果被采集信息的网站和爬虫程序所在的计算机距离很远，网站的平均吞吐量为100K/s，500ms的延迟时间，那么处理每一个请求需要600ms。50 × 600ms=30s。这时，1秒钟内要传输50个页面需要30个连接。

使用多台计算机用于信息采集的另外一个原因是，为了减少爬虫需要记住的网站数目。爬虫程序必须记住所有它曾经进行过信息采集的URL，以及队列中存放的即将进行采集的URL。这些URL必须易于存取，因为采集到的每个页面都含有新的链接，需要将这些新的链接添加到采集队列中。爬虫程序中的队列不能含有重复的URL或已经采集过的页面的URL，每一个新的URL必须与队列中的所有元素进行检查，并与已经采集过的所有URL进行检查。这个用于查找的数据结构需要存放在RAM中，否则，计算机采集页面的速度将严重受到限制。将信息采集的任务分摊给多台计算机，可以降低存储和检查的开销。

使用多台计算机用于信息采集的第三个原因是，可以使用大量的计算资源（computing resources），包括用于分析的CPU资源，用于页面采集的网络带宽。对网络中的大部分页面进行采集，仅使用单独的一台计算机需要处理大量的工作。

分布式网络爬虫更像单独一台计算机上的爬虫程序，但它拥有多个URL队列，而不是一个。分布式网络爬虫使用散列函数将URL分配给进行信息采集的计算机。当一个爬虫程序看到了一个新的URL，就对该地址计算它的散列值，以确定该地址由哪台进行信息采集的计算机负责。将这些URL分批进行汇总，然后周期性地发送，以减少每次发送一个URL的网络开销。

散列函数只需要对URL中的主机部分进行计算。这样，可以将某一主机下的所有URL分配给单独的一个爬虫程序。虽然有些主机与其他的主机相比含有更多的页面，这可能会增加爬虫程序之间URL分配的不平衡性，但礼貌规则要求在同一主机上的不同URL抓取之间需要时间延迟。对于同一主机上的所有URL使用同一台计算机进行信息采集，会更加易于支持礼貌规则中需要的时间延迟。另外，我们期望 domain.com 上的网站有大量的链接指向 domain.com 上的其他页面。通过将 domain.com 分配给单独的一个爬虫程序，可以减少在多台用于信息采集的计算机之间交换URL的数量。

3.3 文档和电子邮件的信息采集

尽管互联网是一个巨大的信息资源，但大量的数字信息并没有存储在网站上。本节中，我们将考虑那些普通的台式计算机上可能会发现的信息，如电子邮件、字处理文档、讲稿或电子表格。这些信息可以通过桌面搜索（desktop search）工具搜索到。在公司和组织机构中，企业搜索（enterprise search）会利用文件服务器或员工个人计算机上的文档，以及局域网内的网页。

当我们面对台式计算机上的数据时，互联网信息采集中的一些问题在这里发生了变化。在互联网信息采集中，仅仅找到数据就是很艰难的任务。在台式计算机中，感兴趣的数据都是以熟悉的组织形式存放在一个文件系统中。由于文件系统中含有易于查找的目录，在磁盘

中找到所有的文件并不是很困难的。从某种方式上看，一个文件系统更像是一台网络服务器，但它具有自动生成的网站地图。

然而，在采集桌面数据的时候，会遇到一些特殊的问题。首先，涉及的问题是更新速度。在桌面搜索系统中，用户要求搜索结果是基于文件的当前内容的。这意味着，要能够搜索到刚刚接收到的电子邮件，要能够搜索到一个刚刚被保存的文档。这和互联网搜索的预期结果是不同的，互联网搜索中，可以容忍信息采集延迟几个小时或者几天。每隔几秒钟就对文件系统进行信息采集是不切实际的，但是现代文件系统能够直接向爬虫程序发送变化通知，这样它们可以立即对新的文件进行备份。文件服务器中的远程文件系统通常不提供这种类型的变化通知，因此它们必须像网络服务一样被爬取。

磁盘空间是另外一个涉及的问题。对于网络爬虫，我们假定需要对找到的每一个文档保留一个备份。对于桌面系统来说并不是这样，桌面系统中文件已经在本地进行存储了，而且如果大部分磁盘空间都被索引器占有的话，用户会很不满意。这时，桌面爬虫会将文档读入内存中，并且将它直接发送给索引器。第5章中会对索引进行进一步的讨论。

由于网站都是通过网络浏览器查看的，大多数的网络内容都是以HTML格式存储的。另一方面，每一个桌面程序，如字处理器、讲稿设计工具、电子邮件程序等等，都有自己的文件格式。因此，仅仅找到这些文件是不够的，最终需要将它们转换成索引器能够理解的统一格式。在3.5节中，我们会再次讨论这种转换问题。

最后，可能也是最重要的，采集桌面数据需要重点关注隐私问题。桌面系统中，多个用户可以使用不同的账户登录，用户A不能通过搜索发现用户B账户中的电子邮件。当我们把网络文件系统的信息采集与企业内部网的信息采集同等看待时，这个问题显得特别地重要。文件存取权限必须伴随着采集到的数据进行记录，而且必须保持是最新的。

3.4 文档信息源

通常来讲，对于互联网信息采集或桌面信息采集，我们假定所有的文档都可以在任意的时间被创建或修改。然而，一些文档是出版物（published），也就是说，这些文档在某个时间被创建，并且几乎不再更新。新闻文章、博客帖子、通讯稿和电子邮件，都是属于出版物这种类型的文档。绝大多数对时间不敏感的信息都属于出版物。

每一个出版物文档都有一个与它关联的时间，来自同一个源头的出版物文档可以在一个文档信息源（document feed）的序列中得到。爬虫对文档信息源非常感兴趣，因为它可以通过仅检查信息源就发现所有的新文档。

我们可以区分两种类型的文档信息源，push和pull。当有新的文档产生时，push类型信息源会向订阅者发出通知。这类似于一台电话，当有来电时可以通知你，并不需要不停地查看电话以便知道是否有人打来电话。pull类型信息源需要订阅者周期地查看是否有新文档。这类似于为了查看是否有新邮件而对邮箱进行检查。来自商业新闻的新闻信息源通常属于push类型信息源，但对于那些无偿服务，pull类型信息源的使用更加普遍。在本节中，我们主要关注pull类型信息源。

pull类型信息源最常见的形式称为RSS。RSS至少有三个定义：简易信息聚合、RDF站点摘要或丰富站点摘要。毫无疑问，RSS有许多种略为不同的实现，并存在相似但不同的格式，称为Atom联合格式（Atom Syndication Format）。由于一个想法很快地被开发者接受，但他们

并不满意单独的一个标准，因此导致了标准的扩展。

图3-9给出了http://www.search-engine-news.org网站RSS源的一个示例。这个信息源包括两个内容：一个是关于即将召开的SIGIR会议的，另一个是关于一本教科书的。每个记录中含有一个时间，该时间指出该项记录所指示内容的出版时间。另外，在RSS源开始的地方，有一个ttl标签，表示存活时间（time to live），该时间以分钟为单位。该标签及其中的内容表示它的内容只缓存60分钟，超过一个小时的信息即被看作是过时的信息。这就给爬虫程序一个提示，这个信息源文件需要多久爬取一次。

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Search Engine News</title>
    <link>http://www.search-engine-news.org/</link>
   <description>News about search engines.</description>
    <language>en-us</language>
    <pubDate>Tue, 19 Jun 2008 05:17:00 GMT</pubDate>
    <ttl>60</ttl>

    <item>
      <title>Upcoming SIGIR Conference</title>
      <link>http://www.sigir.org/conference</link>
     <description>The annual SIGIR conference is coming!
        Mark your calendars and check for cheap
        flights.</description>
      <pubDate>Tue, 05 Jun 2008 09:50:11 GMT</pubDate>
      <guid>http://search-engine-news.org#500</guid>
    </item>

    <item>
      <title>New Search Engine Textbook</title>
      <link>http://www.cs.umass.edu/search-book</link>
      <description>A new textbook about search engines
        will be published soon.</description>
      <pubDate>Tue, 05 Jun 2008 09:33:01 GMT</pubDate>
      <guid>http://search-engine-news.org#499</guid>
    </item>
  </channel>
</rss>
```

图3-9 RSS 2.0源的实例

RSS源可以像网页一样被存取，向保存它们的网络服务器发送 HTTP GET请求。由此，前面我们所讨论的信息采集技术也可以应用于RSS源，例如使用HTTP HEAD请求检测RSS源何时发生了变化。

从信息采集的角度来看，与传统页面相比，文档信息源具有很多的优点。信息源对数据给出了合理的结构，甚至超越了网站地图，网络信息源隐含着数据记录之间的某种关联。很容易就可以对信息源进行分析，而且和网站地图类似，信息源中包括详细的时间信息，还包括对每一个页面的描述域（描述域有的时候包括URL指向页面的全部文本）。最重要的是，类似于网站地图，信息源提供了一个单独的位置用于查找新的数据，而不是为了找到新的文档

必须爬取整个站点。

3.5 转换问题

搜索引擎用于在文本中进行搜索。遗憾的是，计算机中文本是以几百种相互之间不兼容的格式进行存储的。标准的文本格式包括原始文本、RTF、HTML、XML、Word、ODF（开放文档格式）以及PDF（便携式文档格式）。还有几十种不常用的字处理器使用它们自己的文件格式。但是，文本文档并不是需要搜索的唯一类型的文档；其他类型的文件中也含有重要的文本，如PowerPoint讲演稿和Excel电子表格。除了所有的这些文档格式，人们通常还希望搜索一些旧的文档，也就是说搜索引擎可能还需要支持过时的文件格式。对于商业化的搜索引擎来说，支持上百种文件类型是很常见的。

处理一个新的文件格式最常见的方式是使用一个转换工具，将文档中的内容转换成标签文本格式，如HTML或XML。这些格式很容易解析，并支持一些重要的格式信息（如字体大小）。在主要的网络搜索引擎中，可以看到这种文本格式。如果打算搜索PDF文档，但当你点击搜索结果下方的“Cached”链接，会得到对于该文档搜索引擎给出的视图，该视图通常是将原始文本转换为HTML的结果。对于一些文档类型，如PowerPoint，搜索引擎缓存的版本几乎是不可读的。幸运的是，可读性并不是搜索引擎关注的重点。搜索引擎涉及的主要问题是，将数据备份到搜索引擎中，以便能够建立索引并被检索到。然而，将数据转换成为HTML格式有一个优点：为了查看一个文档，用户不需要有专用的能够识别该文档文件格式的应用程序。对于过时的文件格式，这一点很重要。

文档可以转换为纯文本，而不是HTML或XML，但这么做会将一些重要的信息从文档中剥离出去，如标题、字体大小等，这些信息对索引器来讲是很有价值的。正如我们将要看到的，标题和黑体文本中，可能含有准确描述文档内容的词，在计算分值的时候我们希望对这些词进行特殊的处理。只有对格式信息进行了准确的转换，索引器才能抽取出这些重要的信息。

字符编码

由于字符编码（character encoding）的问题，即便是HTML文件之间也不能很好地互相兼容。在纸面上你所看到的文本是一系列的图形，称为字母（letter）或符号（glyph）。当然，计算机中的文件是一个二进制流，而不是图形的集合。字符编码就是在二进制位和符号之间的一个映射。对于英文来讲，基本的字符编码采用的是1963年就开始使用的ASCII编码。ASCII编码使用7位二进制数对128个字母、数字、特殊字符及控制符进行编码，另外附加1位以字节方式进行存储。对于具有26个字母的英文来讲，这个编码方案很精巧。但是世界上有许多种语言，有些语言有相当多的符号。例如，汉语中有40 000多个汉字，常用字也超过3000个。用于东亚语言的CJK（中文-日文-韩文）语系，制定了很多不同的2字节编码标准。其他的语言，如北印度语或者阿拉伯语，也有很多不同的编码方案。即便是英文，也不是所有的编码都一致。例如，在大型主机上使用的EBCDIC编码方案与个人计算机上使用的ASCII编码方案完全不同。

在处理中文和阿拉伯语的复杂字符集上，计算机行业的进展很缓慢。直到最近，代表性的方法是使用不同的特定语言编码，有时称为代码页（code page）。每种编码方案中的开始

128个数值都保留给英文的字母、标点和数字。大于128的数字都映射到从希伯来语到阿拉伯语中的符号，然而，如果对每种语言使用不同的编码方案，在同一个文档中就不能同时使用希伯来语和日语。同时，文档也不再是自描述的了。只将数据存放在文本文件中是不够的，必须同时记录它所使用的编码。

为了解决混乱的编码问题，人们制定了Unicode编码方案。Unicode是从数字到符号的一个映射，该方案中试图包含所有语言中的所有符号。这样就解决了在单个文件中使用多种语言的问题。遗憾的是，它并没有完全解决二进制编码的问题，因为Unicode是数字到符号之间的映射，而不是二进制位到符号之间的映射。结果导致会有很多种方式将Unicode中的数字转换成符号。其中一些普遍使用的方式包括UTF-8、UTF-16、UTF-32和UCS-2（不推荐该方法）。

编码方案的发展起源于对兼容性和节省空间的需求。用UTF-8对英文文本进行编码时，与ASCII编码方案是一致的。每个ASCII字母只需要一个字节。然而，一些繁体汉字会需要四个字节。为了能够对大多数的语言文字进行编码，也为了减少对西方语言进行编码的长度，折中方案是每个字符用可变的字节数表示，但这样会使得计算字符串中某个字符的数目或者自动跳转到字符串中的随机位置变得更加困难。相比之下，UTF-32（也就是UCS-4）对每个字符都使用4个字节进行编码。这使得在UTF-32编码的字符串中，跳转到第20个字符变得更加容易，即跳转到第80个字节处开始读取。遗憾的是，UTF-32编码的字符串和过去使用ASCII编码的软件不兼容，并且UTF-32编码的文件比UTF-8编码的文件需要四倍的空间进行存储。因此，一些应用程序对国际化文本进行编码的时候使用UTF-32（随机存取显得很重要），但将文本进行存储的时候使用UTF-8编码方案。

表3-1 UTF-8 编码

十进制	十六进制	编码形式
0～127	0～7F	0xxxxxxx
128～2047	80～7FF	110xxxxx 10xxxxxx
2048～55295	800～D7FF	1110xxxx 10xxxxxx 10xxxxxx
55296～57343	D800～DFFF	未定义
57344～65535	E000～FFFF	1110xxxx 10xxxxxx 10xxxxxx
65536～1114111	10000～10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

表3-1给出了UTF-8的编码表。表中左边的列表示十进制数值的范围，最右边的列表示这些十进制数值以二进制方式编码的形式。字符x表示二进制数字。例如，希腊字母π在Unicode中对应数字960，对应的二进制数为 00000011 11000000（十六进制数为3C0）。表中的第二行告诉我们，这个字符在UTF-8中需要用两个字节进行编码。3C0的高五位二进制数落在第一个字节中，接下来的6位落在第二个字节中。该字符最终的UTF-8编码为11001111 10000000（对应的十六进制为CF80）。其中加粗的二进制位与表中的相同，表中的x字符对应的位由Unicode数字的二进制数值进行填充。

3.6 存储文档

为了对文档进行索引，文档转换为统一的格式之后需要对其进行存储。最简单的文档存储是不对文档进行存储，对于一些应用，这样做是可取的。例如，在桌面搜索中，文档已经

存储在文件系统中了，不需要再备份到其他地方。当信息采集进程运行的时候，它可以将经过转换的文档直接发送给索引进程。由于不需要存储文档转换的中间结果，桌面搜索系统就可以节省磁盘空间，这改善了索引需要的延迟。

绝大多数的搜索引擎都需要将文档存储在某个地方。为了对搜索结果创建网页摘要[⊖]，就需要对文档中的文本进行快速的存取。为用户提供网页文本的摘要，是为了使用户不需要点击相应的链接，就能够知道检索到的文档的主要内容。

网页摘要并不是必需的，还有其他的一些原因也需要对每一个文档都保留一个备份。从CPU和网络的负载来看，文档采集的代价是很高的。可将文档进行备份，当下次再建立索引的时候，就不需要再次抓取它们了，这对于降低CPU和网络负载是很有意义的。可以保存那些曾抓取过的文档，允许在爬虫程序中使用HEAD请求以节省带宽，或者只爬取索引页面的一个子集。

最后，文档存储系统是信息抽取（见第4章）的起点。网络搜索引擎中最常见的一种信息抽取类型是，从超链接中提取锚文本，将锚文本和文档一起存储。也可能会使用其他类型的信息抽取，如在文档中识别人名和地名。如果在搜索中使用信息抽取，那么文档存储系统需要能够支持对文档数据的修改。

接下来，我们讨论对文档存储系统的一些基本需求，包括随机存取、压缩和更新，并讨论使用数据库系统或定制的存储系统（如BigTable）的相对优点。

3.6.1 使用数据库系统

如果你以前使用过关系数据库，你可能会认为数据库很适合用户存储文档数据。实际上，对于一些应用系统来说，数据库是存储文档的好地方。对于具有很多碎片的数据，如网页，数据库考虑到了很多细节的问题，并且易于今后的更新。绝大多数的数据库可以作为网络服务器运行，这使得文档可以在网络上应用。由于具有这样的特点，单独的一台计算机可以用于生成文档摘要，同时其他的一些计算机可以处理用户查询。数据库系统中含有一些有用的输入和分析工具，这使得对文档集合的管理更加简单。

一些网络搜索引擎公司不愿意谈论它们的技术细节。然而，主要的搜索引擎中，很少使用传统的关系数据库来存储文档。一个问题是，大量的文档数据会压垮关系数据库系统。一些数据库经销商希望数据库服务器使用最昂贵的磁盘系统，但对于这样的文档集合规模来讲是不实际的。在本节的最后，我们讨论关系数据库的另外一种方案，那里会涉及上面关注的这些问题。

3.6.2 随机存取

要对文档进行快速检索，以便对一个搜索的结果生成网页摘要，文档存储需要能够支持随机存取。然而，与关系数据库相比，只需要一个相对简单的查找标准。我们需要一个数据库，使我们能够根据URL得到文档的内容。

最简单的处理这种查找的方式是使用散列。对URL使用散列函数可以得到一个数值，利用这个数值可以找到所需要的数据。对于小规模的系统，散列函数可以告诉我们哪一个文件

⊖ 在第6章中将讨论网页摘要的生成。

含有这个文档。对于大规模的系统，散列函数可以告诉我们哪一个服务器含有这个文档。一旦文档的位置限定在一个单独的文件中时，就可以使用B-Tree或者排序的数据结构，找到文档数据在文件中的偏移位置。

3.6.3 压缩和大规模文件

不管应用系统是否需要对文档进行随机存取，文档存储系统都需要大规模的文件和压缩技术。

尽管对于用户来讲，文档看上去可能会很长，但对于现代计算机来讲，这样的文档规模是很小的。例如，本章中的内容大约有10 000字左右，存储这些内容大约需要70K的磁盘空间。比网页的平均大小要大很多，但现代的磁盘系统传输70K的数据大约仅需要1个毫秒。然而，为了对文件进行读取，磁盘系统可能会需要10毫秒的时间来寻找这个文件。这就是为什么文档在它自己的文件中存放不是一个好的想法；打开这些零散的小文件来读取文档的内容，需要大量的时间开销。一个较好的解决方案是，将多个文档存储在一个单独的文件中，但这样做的结果会使文件变得很大，以至于在网络上传输该文件的时间开销会远远大于搜索文档的时间。文件的合理大小应该是几百兆字节。通过将多个文档在文件中一起存放，索引器可以将大量的时间用于数据内容的读取上，而不是文档内容的查找上。

在Galago搜索引擎中，含有三种混合文档格式的解析器：ARC、TREC文本和TREC Web。在每种格式中，大量的文档都存储在同一个文件中，各文档之间以一个较短的文档元数据区域进行分隔。图3-10给出了TREC Web格式。每个文档块以 <DOC> 标签开始，并以 </DOC> 标签结束。在文档的开始部分，<DOCHDR> 标签部分含有页面请求的信息，如它的URL、页面采集的日期以及网络服务器返回的HTTP头部信息。每个文档记录还包括一个<DOCNO>部分，该部分包括该文档的一个唯一的识别符。

大规模的文件对于从磁盘传输数据是有很大意义的，但减少对文档集合的总体存储需求也有着很明显的优势。幸运的是，人们书写的文本通常都有较大的冗余性。例如，字母 q 的后面通常会跟随着字母 u。香农（1951）指出，对于英语是母语的人，能够在英文文本中猜出下一个字母的准确率为69%。HTML和XML中的标签具有更大的冗余性。针对冗余性采用压缩（compression）技术，可以使文件在不丢失任何信息的基础上变得更小。我们将在第5章中介绍文档的压缩技术，这样做的部分原因是由于对于索引的压缩相当特别。目前，文本压缩技术仍有大量的研究，常用的算法像DEFLATE（Deutsch，1996）和LZW（Welch，1984），能够对HTML和XML文本压缩80%。空间的节省降低了存储大量文档的开销。由于有时候只需要读取少量的字节，这样也降低了从磁盘中读取文档的时间开销。

对于大块的数据，压缩的效果较好，这样就使得大规模的文件中可以存放数量较多的文档。然而，把这个文件作为单独的一个数据块进行压缩是不必要的。绝大多数的压缩方法都不支持随机存取，因此每一个数据块会被相继地解压缩。如果希望能够对数据进行随机存取，较好的方式是考虑对小数据块进行压缩，一个文档可以是一个数据块，或者少量的文档组成一个数据块。小数据块降低了压缩率（节省空间的数量），却能够降低请求等待的时间。

```
<DOC>
<DOCNO>WTX001-B01-10</DOCNO>
<DOCHDR>
http://www.example.com/test.html 204.244.59.33 19970101013145 text/html 440
HTTP/1.0 200 OK
Date: Wed, 01 Jan 1997 01:21:13 GMT
Server: Apache/1.0.3
Content-type: text/html
Content-length: 270
Last-modified: Mon, 25 Nov 1996 05:31:24 GMT
</DOCHDR>
<HTML>
<TITLE>Tropical Fish Store</TITLE>
Coming soon!
</HTML>
</DOC>
<DOC>
<DOCNO>WTX001-B01-109</DOCNO>
<DOCHDR>
http://www.example.com/fish.html 204.244.59.33 19970101013149 text/html 440
HTTP/1.0 200 OK
Date: Wed, 01 Jan 1997 01:21:19 GMT
Server: Apache/1.0.3
Content-type: text/html
Content-length: 270
Last-modified: Mon, 25 Nov 1996 05:31:24 GMT
</DOCHDR>
<HTML>
<TITLE>Fish Information</TITLE>
This page will soon contain interesting
information about tropical fish.
</HTML>
</DOC>
```

图3-10 TREC Web混合文档格式中的文本内容实例

3.6.4 更新

当爬虫程序抓取到了文档的新的版本时，需要对文档数据库进行更新。可以采用的一种方式是，将爬虫抓取来的新的、变化了的文档与文档数据库中那些没有发生变化的文档数据进行合并，重新生成一个新的文档存储。但如果文档数据的变化较小，与仅在相应的位置对数据进行更新相比，这个合并过程的开销会更大。

更新的另外一个重要原因是处理锚文本。图3-11给出了HTML链接标签中的锚文本的实例。图中的HTML代码在网络浏览器中表示为超链接，超链接上有锚文本 Example website，当点击这个超链接的时候，会将用户带到所指向的http://example.com网站上。锚文本是一个很重要的特征，因为它对目标网页内容给出了精确的摘要。如果这个链接来自另外一个不同的网站，我们仍然会相信摘要信息是准确的，这同样有助于对文档进行排序（见第4章和第7章）。

```
<a href="http://example.com" >Example website</a>
```

图3-11 具有锚文本的链接

由于锚文本需要和目标网页相关联，因此准确收集锚文本是很困难的。简单的方法是，使用一个支持更新的数据库。当发现一个文档中含有锚文本时，我们找到目标网页的记录，并对记录中的锚文本进行更新。当对文档进行索引的时候，锚文本也可以一起进行索引。

3.6.5 BigTable

虽然数据库可以履行文档数据库的职责，但规模庞大的文档集合需要特殊的文档存储系统。BigTalbe是一个众所周知的文档存储系统（Chang et al. 2006）。至少有两个开源项目都使用了类似的方法，但BigTable是内部使用的一个运行系统。在接下来的段落中，将看到BigTable的架构，了解文档存储是如何影响它的设计的。

在开始用于存储网页时，BigTable是一个分布式数据库系统。BigTable实际上的确是一个很庞大的表；它的规模可以超过1PB（1024TB），但每一个数据库只含有一个表。每个表被分割成为小块，称为tablet，通过上千台的机器提供支持（见图3-12）。

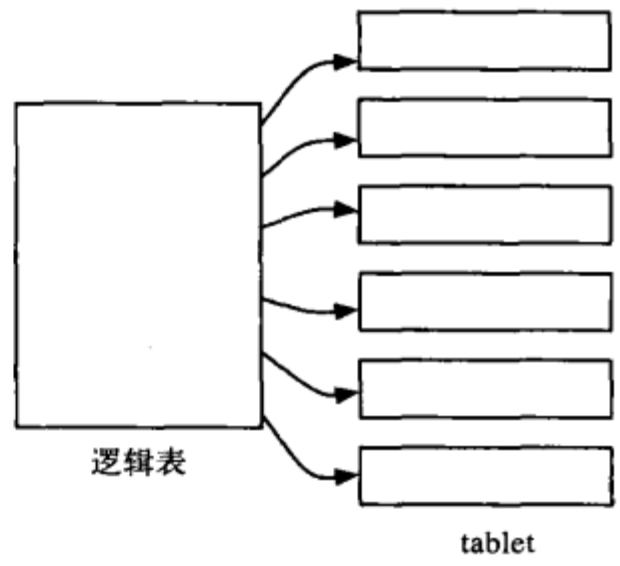


图3-12 BigTable在一个逻辑表中存储数据，该表被分割成一些更小的tablet

如果你了解关系数据库，就会使用过SQL（Structured Query Language，结构化查询语言）。SQL支持用户书写一些复杂的、大计算量的查询，数据库系统的任务之一就是优化这些对查询的处理，尽可能快地完成这些处理过程。由于其中的一些查询可能需要花费很长的时间才能完成，所以一些大型的关系数据库采用了复杂的锁定系统，以保证大量用户同时对数据库进行读或写操作时，数据库不会崩溃。将众多的用户互相隔离开是一个很困难的任务，很多文章和书籍都讨论了如何能更好地使用户相互分离。

BigTable方法有很大的不同。没有查询语言，也就没有复杂的查询，只有 row-level事务，从关系数据库的标准来看，这是很简单的任务。然而，如果采用价格低廉的计算机，这个简单的模型会使BigTable达到很大的数据规模，尽管这样的计算机易于出现故障。

BigTable的实现中，绝大部分的任务是关于故障恢复的。tablet是逻辑表中很小的部分，复制到另外一个文件系统中进行存储，该文件系统可以被BigTable中所有的tablet服务器存取。BigTable中tablet的变化记录在事务日志中，日志存储在一个共享的文件系统中。如果某个tablet服务器死机了，另外一台服务器可以立即从文件系统中读取tablet数据和事务日志，并将工作承担过来。

绝大多数的关系数据库将数据存放在文件中，该文件会被不停地进行修改。相反，BigTable将文件存储在永远不变的（不能改变的）文件中。一旦数据写入到了BigTable的文件中，它就不再变化了。这样做也有助于故障的恢复。关系数据库中，故障恢复需要一系列复杂的操作来保证文件不会被破坏，因为在计算机出现故障之前只完成了部分写操作。BigTable中文件或者是不完整的（这时候文件可以被丢掉，可以从其他的BigTable文件和事务日志重新创建），或者是完整的，因此不会被破坏。最新的数据存储在RAM中以支持表的更新，而过时的数据存储在一系列的文件中。这些文件周期性地合并在一起，以减少磁盘文件的数目。

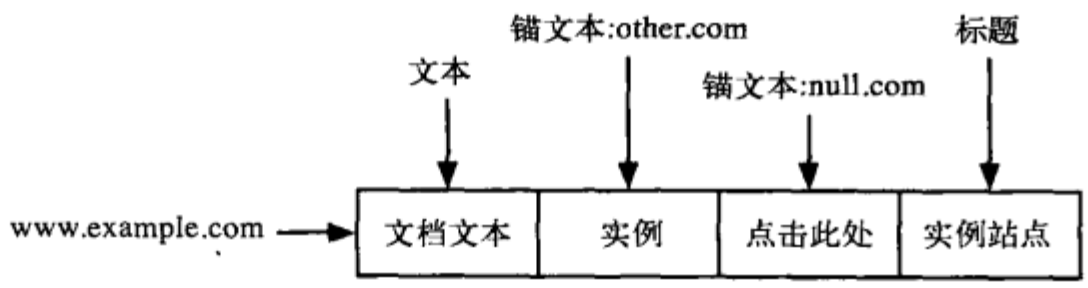


图3-13 BigTable中的一行

BigTable逻辑上是以行的方式组织的（见图3-13）。图中，该行存储了一个单独网页的数据。URL地址为www.example.com，是该行的主键，用于对该行的查找。行中有很多列，每个列有唯一的名字。每一列可以有一些时间标记，在图中没有给出来。行的主键、列的主键以及时间标记组合在一起，指向一行中的一个单元（cell）。

图中，对于文档的全文有一个文本列，还有一个标题列，这样，不需要对全部的文档文本进行分析，就可以快速地找到文档的标题。有两列用于锚文本。一个是 anchor : other.com，包含从站点 other.com 指向 example.com 的超链接的锚文本；如图中的单元所示，超链接上的文本是“example”。anchor : null.com 描述了从 null.com 指向 example.com 的一个超链接上的锚文本是“点击此处”。这些列都属于锚文本列组（column group）。可以向该列组中增加其他的列，以增加更多的链接信息。

BigTable中，每行可以有许多列，所有的行具有相同的列组，并不是所有的行都有相同的列。这是和关系数据库系统最主要的区别，但这种灵活性是很重要的，部分原因是由于没有那么多表。在关系数据库系统中，锚文本对应的列会存储在一个表中，而文档文本存储在另外一个表中。BigTable中只有一个表，所有的锚文本信息需要包装在一个记录中。将文档和所有的锚文本数据存储在一起，读取所有的文档数据只需要一次磁盘读取。在具有两个表的关系数据库中，要得到所有的文档数据，至少需要两次对磁盘的读取操作。

行根据它们的主键被分割为tablet。例如，所有以 a 开始的URL地址可以放在一个tablet中，而以 b 开始的URL地址可以存放在另外一个tablet中。这种基于范围的分割使得BigTable的客户端很容易就可以确定每一行是由哪个服务器提供服务的。要查找一个特定的行，客户端查阅行主键的范围列表来确定该行属于哪个tablet。接下来，客户端与这个tablet的服务器建立连接，来获取该行。行主键的范围在客户端进行缓存，这样可以保证大多数的网络通信都是在客户端和tablet服务器之间的。

BigTable的架构被设计用于提高速度和增大规模，而且出于经济上的考虑，使用廉价的、易于出故障的计算机。为了实现这些目标，BigTable牺牲了关系数据库中的一些主要特征，如复杂的查询语言和多表格的数据库。然而，这样的架构很适用于存储和找到网页的工作，该工作的主要任务是对各行进行高效的查找和更新。

3.7 重复检测

重复和近似重复（near-duplicate）的文档会在很多情况下出现。在办公室中，对文档进行备份、对文档创建新的版本是很常见的行为，对这些行为进行跟踪是信息管理中的重要内容。然而，在互联网上，情况会更加极端。除了一般的重复来源，抄袭（plagiarism）和垃圾（spam）是很普遍的，还有使用多个URL地址指向同一网页以及镜像站点（mirror site），都会使爬虫程序产生大量的重复页面。研究表明，在一个大型的信息采集系统中，30%的网页是

和另外70%的网页完全重复或近似重复的（如Fetterly et al.，2003）。

通常，非常相似的文档内容不能或者只能给用户提供少量的新信息，但在信息采集、索引和搜索过程中却消耗了大量的资源。为了解决这个问题，学者们提出了一些用于重复文档检测的算法，这样可以在索引和排序过程中将重复的文档删除，或者将重复的文档看作是一组文档。

对完全重复的文档的检测是相对简单的任务，可以使用检验和（checksumming）技术。检验和是根据文档内容计算的一个数值。最直接的检验和是文档文件中各字节的和。例如，对于一个含有文本“Tropical fish”的文件，检验和可以按如下的方式计算（以十六进制表示）：

T	r	o	p	i	c	a	l		f	i	s	h	和
54	72	6F	70	69	63	61	6C	20	66	69	73	68	508

含有相同文本的任意文档会含有相同的检验和。当然，任意的文档如果恰好具有相同的检验和，也将被认为是重复的文档。例如，文件中含有相同的字符但顺序不同，会具有相同的检验和。一些复杂的函数，如循环冗余校验（cyclic redundancy check，CRC）中，考虑到了字节出现的位置。

对于近似重复文档的检测比较困难，甚至如何定义近似重复也是不容易的。例如，对于网页来说，可能含有相同的文档内容，但广告、日期、格式不同。其他的页面可能由于修订或者更新在内容上略有不同。通常，使用两个文档之间相似度的域值来定义近似重复的文档。例如，如果文档D_1和文档D_2中有90%的词是重复的，那么它们就是近似重复的文档。

对于近似重复的检测，有两种应用场合。一个是在搜索场景（search scenario）下，这时它的目标是找到与给定文档D近似重复的文档。类似于所有的搜索问题，涉及查询文档和所有其他文档之间的比较。对于含有N个文档的集合，这样的比较需要的开销为$O(N)$。另外一个场景是发现（discovery），涉及在集合中找到所有近似重复的文档对。这个过程需要的开销为$O(N^2)$。尽管在信息检索技术中，在搜索场景中用基于词表达的文档来进行相似度的计算对于近似重复的文档检测是有效的。但在发现的场景中，相似度的计算需要一些新的技术，得出文档的简洁表示形式。这些简洁的表示形式也就是指纹（fingerprint）。

生成指纹的过程如下：

1）首先对文档进行分词处理。删除那些不是词的内容，如标点、HTML标签和空格。

2）对于给定的n，将这些词组合成n-gram。尽管在一些技术中使用非重叠的词序列，但n-gram通常是相互重叠的词序列（见4.3.5节）。

3）其中的一些n-gram被选择用于表示该文档。

4）对这些被选择的n-gram进行散列，以提高检索的效率，并进一步减少文档表示的大小。

5）将散列值进行存储，通常存储在倒排索引中。

很多的指纹算法都使用该方法实现，各算法之间主要的区别在于如何对n-gram的子集进行选择。对于近似重复文档的检测，随机选择固定数量的n-gram并不能改善性能。考虑两个近似相同的文档D_1和D_2。通过随机地从文档D_1中选择n-gram而生成的指纹，与随机地从文档D_2中选择n-gram而生成的指纹不可能有很多的重叠。更加有效的方法是使用事先制订的字符组合，选择以这些字符开始的n-gram。另外一个常见的方法称作 0 mod p，选择那些散列值模p为0的n-gram，这里p为参数。

图3-14中显示了使用重叠的3-gram生成指纹的过程，假定了散列值，以及 0 mod p 的选择

方法，其中 p=4。在选择过程之后，文档（在这里是句子）通过一些n-gram的指纹来表示，“fish include fish”、“found in tropical”、“the world including”以及“including both fresh water”。

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

a) 原始文本

tropical fish include, fish include fish, include fish found, fish found in, found in tropical, in tropical environments, tropical environments around, environments around the, around the world, the world including, world including both, including both freshwater, both freshwater and, freshwater and salt, and salt water, salt water species

b) 3-gram

938 664 463 822 492 798 78 969 143 236 913 908 694 553 870 779

c) 散列值

664 492 236 908

d) 使用 0 mod 4 选择的散列值

图3-14 指纹生成过程实例

在大规模的应用中，例如在网络上找到近似重复的文档，n-gram通常有5～10个词，而散列值为64位[⊖]。

通过对表示文本的指纹进行比较，可以找出近似重复的文档。近似重复的文档对是通过两个文档相同的指纹数量或者相同指纹占总指纹数量的比例来进行定义的。然而，指纹并不能代表所有的文档信息，这会导致在近似重复的检测中出现错误。恰当的选择方法可以减少这些错误，但不能消除这些错误。正如上面提到的，评价结果显示出，与指纹方法相比，基于词表的相似度计算如余弦相似度计算，通常更加有效。但这些方法的主要问题是效率不高。

最新开发的一种称为simhash（Charikar，2002）的指纹技术，吸取了基于词的相似度计算的优点，以及基于散列的指纹技术的高效性。该方法中散列函数具有特殊的性质，即相似的文档具有相似的散列值。更准确地说，通过余弦相似度计算的两个页面的相似度，正比于simhash方法生成的指纹中相同的位数。

simhash方法中，指纹计算的过程如下：

1）利用具有权值的特征集合来表示文档。简单的情况下，假定特征都是由词组成的，词的权值由它们出现的频率来确定。其他的权值计算方案在第7章中讨论。

2）对每一个词，生成b位的散列值（指纹的期望长度）。每个词都具有各自不同的散列值。

3）在b维的向量V中，分别对每维向量进行计算。如果词相应位的散列值为1，则进行对其特征权值的加法运算，否则进行减法运算，通过这种方式对向量进行更新。

4）当所有的词都处理完毕之后，如果向量V中第i维是正数，则将b位的指纹中第i位设置为1，否则为0。

⊖ 散列值通常使用拉宾指纹（Rabin fingerprinting，Broder等人，1997）来生成，该方法是以以色列计算机科学家 Michael Rabin 来命名的。

图3-15给出了一个8位指纹的处理过程。去除常用词（停用词）作为文本处理的一部分。实际中，b会取很大的值。Henzinger（2006）中叙述了一个基于大规模网页的评价方法，他使用的指纹有384位。如果一个网页和另外一个网页的simhash指纹中有多于372位是相同的，那么这两个网页就是近似重复的。研究表明，simhash指纹方法比n-gram的指纹方法有更好的效果。

3.8 去除噪声

一些网页中含有文本、链接和图片，但这些内容并不是和页面的主要内容直接相关的。例如，图3-16中，网页含有一个新的新闻。页面的主要内容用黑色方框括起来了。内容块（content block）在页面中只占20%的显示区域，余下的部分包括徽标、广告、图片、导航链接、服务（如搜索和警告）以及各种各样的信息，如版权。从搜索引擎的角度看，这些多余的内容绝大多数都是噪声（noise），对页面的排序会起负面的作用。搜索引擎中使用的页面表示的主要组件是基于词频统计的，页面中存在大量的与主要话题无关的词就成为一个问题。因此，出现了用于对网页中的内容块进行检测的技术，同时忽略页面上的其他内容或者在索引时降低其他内容的重要性。

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

a) 原始文本

tropical 2 fish 2 include 1 found 1 environments 1 around 1 world 1 including 1 both 1 freshwater 1 salt 1 water 1 species 1

b) 对词加权

tropical	01100001	fish	10101011	include	11100110
found	00011110	environments	00101101	around	10001011
world	00101010	including	11000000	both	10101110
freshwater	00111111	salt	10110101	water	00100101
species	11101110				

c) 8位的散列值

1 -5 9 -9 3 1 3 3

d) 对权值求和之后的向量V

1 0 1 0 1 1 1 1

e) 由向量V得到的8位的指纹

图3-15 simhash指纹处理过程

Finn等人（2001）提出了一个非常简单的方法，该方法是基于对HTML标签的观察得到的方法，在网页中主要内容部分的文本会比网页中附加内容的文本中含有更少量的HTML标签。图3-17（文档斜率曲线，document slope curve）中，给出了图3-16所示的网页中标签的累加式分布。页面的主要内容对应于该分布中间的“高地”部分。平坦的区域相对较小，这是由于页面的HTML源文件中含有大量的格式和外观描述的信息。

对该分布中最大的平坦区域进行检测的一种方法是，使用二进制位序列对网页进行表示，b_n=1表示第n个词素是一个标签，否则b_n=0。大多数用于文本格式的标签，如字体变化、标题和表格标签，在这里被忽略掉（可以用0来表示）。这样，主要内容块的检测可以看作是一个

优化问题，可以找到i和j两个值，用于最大化低于i和高于j的标签数量以及在i和j之间的非标签词素的数量，相应的就是最大化对应的目标函数：

$$\sum_{n=0}^{i-1} b_n + \sum_{n=i}^{j} (1-b_n) + \sum_{n=j+1}^{N-1} b_n$$

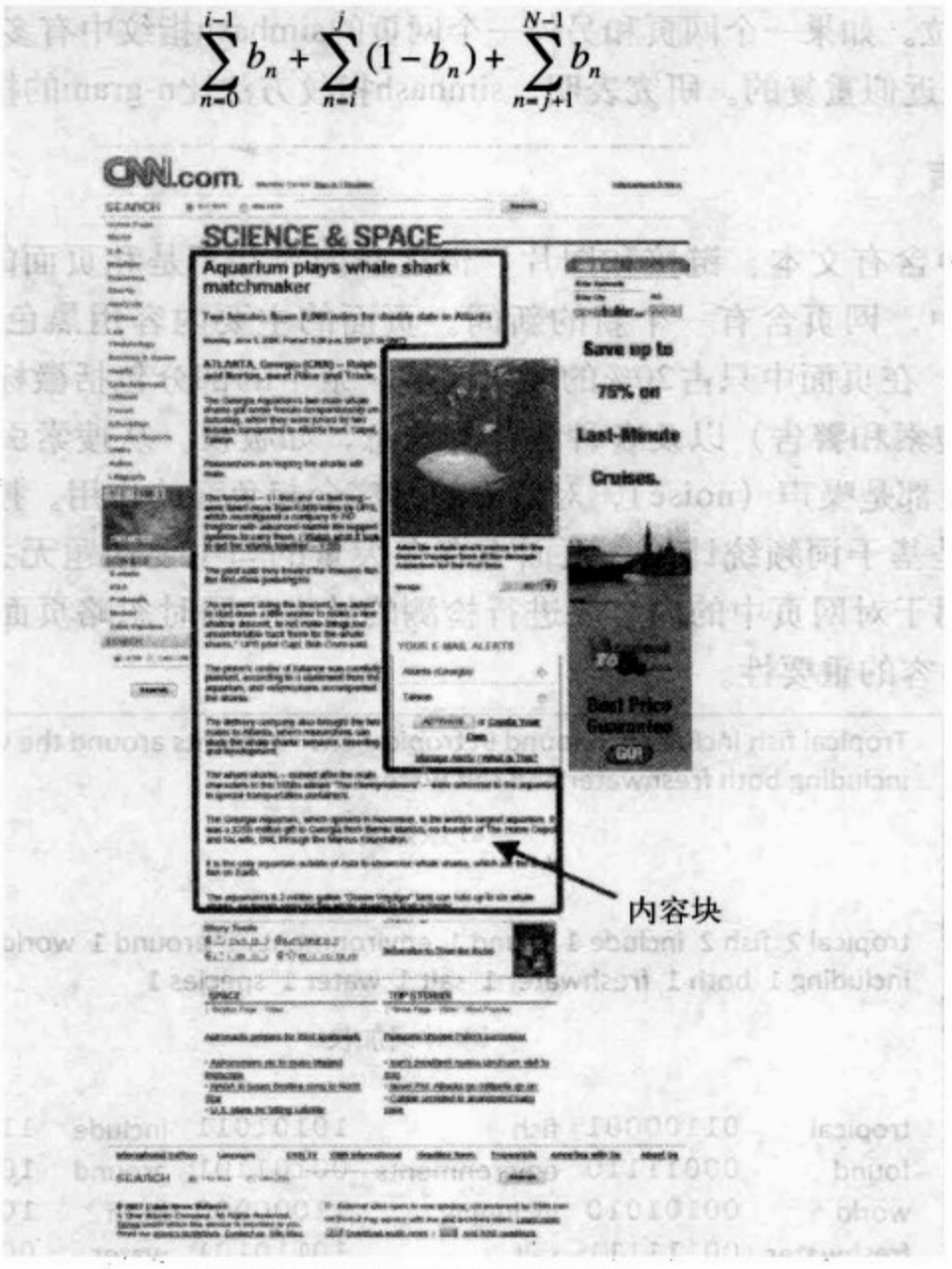


图3-16 网页中的主要内容块

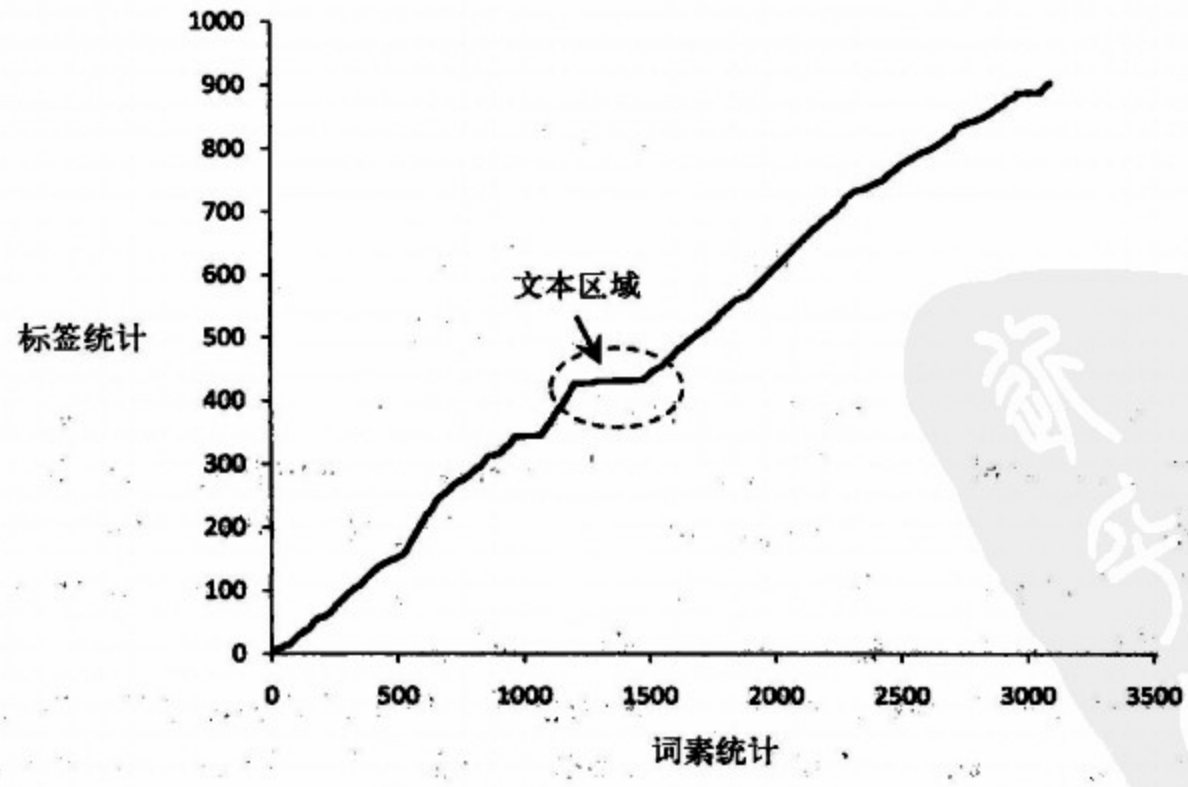


图3-17 标签统计用于区别网页中的文本块

N是页面中词素的数量。可以通过粗略地设定可能的i和j的值，计算这个目标函数。可以看到，只有非内容块中文本词素的比例小于标签的比例，这个过程才会起作用，而不适用于图3-17中的网页。Pinto等人（2002）对这个方法进行了修改，使用一个文本窗口对文档斜率曲线中的低斜率部分进行搜索。

网页的结构也可以直接用于识别页面的内容块。用浏览器显示一个网页，HTML解析器使用标签解释页面指定的结构，并创建文档对象模型（DOM）表示。DOM表示的类似于树的结构，可以用来识别网页中主要的部分。图3-18给出了图3-16中网页的部分DOM结构[⊖]。该结构中含有新闻文本的部分通过注释 cnnArticalContent 给出。

Gupta等人（2003）提出了一种方法，递归遍历DOM树，使用不同的过滤技术来删除和修改树中的节点，只留下内容部分。HTML中的图片、脚本等元素通过简单的过滤就可以去除。很多复杂的过滤技术可以用于去除广告、链接列表及没有内容的表格。

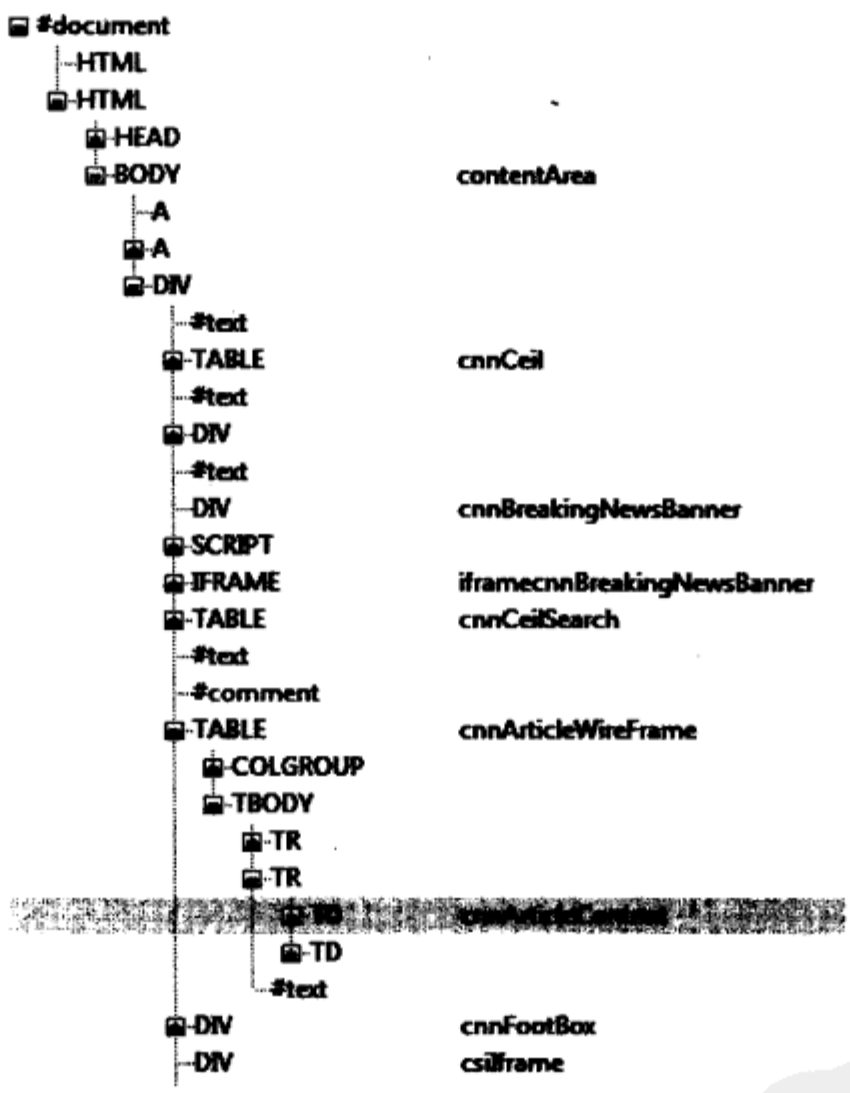


图3-18 示例网页的DOM结构中的一部分

DOM结构提供了网页中各组成部分的有用信息，但它很复杂，并且是逻辑和布局组成部分的混合。如图3-18中，文章的内容隐藏在HTML表格（TABLE标签）中，在一行（TR标签）的表格单元（TD标签）中。这时候使用表格是为了规定网页的布局，而不是在语义上对数据进行关联。识别页面中内容块的另外一种方法侧重于页面的布局和外观。换句话说，视觉特征，如块的位置、使用的字体大小、字体和背景颜色，以及分隔符（如横线和空格），这些都用于定义显示的网页中用户明显看到的信息块。Yu等人（2003）中提出了一个算法，从DOM

⊖ 使用Firefox中的DOM检查生成的。

树和视觉特征构建出可视块的层次结构。

上述讨论的第一个算法，即基于标签分布的方法，对于只有单一内容块的网页来说非常有效。使用DOM结构和视觉分析的算法，可以处理可能有多个内容块的网页。在有较多内容块的情况下，索引处理中可以利用每个块的相对重要性来生成更加有效的表示。判断网页中内容块重要性的一个方法是训练一个分类器（classifier），根据视觉和内容特征分配一个重要的类别（R. Song et al.，2004）。

参考文献和深入阅读

Cho和Garcia-Molina（2002，2003）撰写了一系列关于网络爬虫设计的很有影响力的文章。关于页面时新性策略的讨论，很多来自Cho和Grarcia-Molina（2003）的文章，3.2.7节则摘自Cho和Grarcia-Molina（2002）的文章。

现在有很多开源的网络爬虫。Heritrix爬虫[⊖]是为Internet Archive项目开发的，这是一个功能强大的、可升级的爬虫系统。系统采用模块化设计，在运行过程中可重新配置，这样的特点使得该系统特别适合在实验中使用。

在早期网络搜索中，主题搜索引起很大的关注。Menczer和Belew（1998）及Chakrabarti等人（1999）撰写了两篇很有影响力的文章。Menczer和Belew（1998）的文章中提出了一个主题爬虫想法，它是由多个自主的软件代理组成的，理论上每个用户使用一个代理。用户输入URL和关键词的列表，接下来代理找到那些对用户可能有用的网页，用户可以通过与系统之间的反馈来对这些网页做出评价。Chakrabarti等人（1999）的文章中侧重于对指定主题的索引进行采集，爬虫使用一个分类器来确定所爬取页面的主题，还有一个提取器，用于判断一个页面作为指向其他页面的链接来源的质量。他们利用传统的非主题爬虫作为参照来对系统进行评价，结果显示出，非主题爬虫使用一个主题相关的链接作为种子，这样做是不足以完成主题信息采集的。他们对于爬虫的评价是，网络中宽泛的链接结构导致非主题网络爬虫采集的网页很快就偏离了主题，而主题爬虫却可以成功地保持在一个主题的信息采集上。

Unicode规范无疑是一项详细的工作，覆盖了成千上万的字符（Unicode Consortium，2006）。考虑到一些非西方书写字母的特性，于是将一些Unicode的字符组合在一起形成一些符号。因此，这个规范必须很详细，不仅仅表示字符是什么，而且还有它们是如何关联到一起的。目前，仍然有很多字符阶段性地添加到Unicode中。

Bergman（2001）的文章对深层网络进行了广泛研究，尽管该研究已经比较过时了。他指出通过搜索引擎如何采样，才能有助于估计网络中有多少没有索引的内容。该项研究估计深层网络中存在5 500亿的网页，而网络中可访问的网页只有10亿。He等人（2007）最近的一篇综述文章指出，近些年来深层网络在不断地增加。Ipeirotis和Gravano（2004）的文章中提出了一个称为查询探测的技术，用于生成深层网络数据库的可搜索的表示。

网络地图、robots.txt文件、RSS源和原子信息源，它们每个都有自己的规范，可以在网络上找到[⊜]。这些格式说明成功的网络标准通常非常简单。

⊖ http://crawler.archive.org。

⊜ http://www.sitemaps.org。
http://www..robotstxt.org。
http://www.rssboard.org/rss-specification。
http://www.rfc-editor.org/rfc/rfc5023.txt。

正如上面所提到的，对于一些应用系统，数据库可以用来对网络采集的文档进行存储。然而，我们对数据库的讨论只大部分限定于和BigTable进行比较。有很多教科书，如Garcia-Molina等人（2008），提供了关于数据库如何工作的更多的信息，包括查询语言、锁定机制和恢复等重要特征的细节。经常引用的BigTable，在Chang等人（2006）的文章中有详细的叙述。其他大型的网络公司也建立了他们自己的数据库系统，这些系统都是出于相似的目的：为了实现大规模分布和较高的吞吐率，但都没有明确的查询语言或者详细的事务处理来支持。亚马逊公司的Dynamo系统具有较低的时间延迟（DeCandia et al.，2007），雅虎使用UDB系统来存储大规模的数据集（Baeza Yates & Ramakrishnan，2008）。

我们提到的DEFLATE（Deutsch，1996）和LZW（Welch，1984）是特定的文档文本压缩算法。DEFLATE是常见的Zip、gzip和zlib压缩工具的基础。LZW是Unix的compress命令的基础，也可以用于GIF、PostScript和PDF格式文件的压缩。Witten等人（1999）的文章中，提供了关于文本和图像压缩算法的详细讨论。

Hoad和Zobel（2003）对指纹技术进行了综述，对近似重复检测与基于词的相似度计算方法进行了比较。他们的评价侧重于查找文档的不同版本和抄袭的文档。Bernstein和Zobel（2006）提到了使用全指纹（不进行选择）查找相互衍生（co-derivative）的文档的方法，相互衍生是指从相同的来源衍生出来的文档。Bernstein和Zobel（2005）就重复文本的影响对检索效果的评估进行了实验。他们指出在一次TREC评测中，相关文档中有15%是冗余的，从用户的角度来看，这会对结果产生很大的影响。

Henzinger（2006）叙述了网络中近似重复检测的大规模评价。对Broder的“shingling”算法（Broder et al.，1997；Fetterly et al.，2003）和simhash（Charikar，2002）的两种方法进行了比较。Henzinger的研究中，使用了16亿的网页，表明在同一站点上没有一种方法能很好地对冗余文档进行检测，由于式样文本的不断重复使用，使得不同的页面看上去也是相似的。对于不同站点上的页面，simhash算法可以得到50%的准确率（这意味着，根据相似度域值确定的那些近似重复的页面中，有50%是正确的），而Broder算法可以达到38%的准确率。

关于从网页中抽取内容的方法，在很多文章中都有介绍。Yu等人（2003）和Gupta等人（2003）是关于这类方法的很好的文章。

练习

3.1 假设有两个文档集合，较小的集合中全部都是有用的、正确的、高质量的信息。较大的集合中含有一些高质量的文档，但还有一些低质量的、过时的、书写较差的文本。只对较小的集合建立搜索引擎的原因是什么？对两个集合都建立搜索引擎的原因是什么？

3.2 假设有一个网络连接，每秒可以传输10MB的数据。如果每个网页大小为10K，需要500毫秒进行传输，网络爬虫需要多少个线程才能完全利用网络连接？如果爬虫在对同一网络服务器发出的不同请求之间需要等待10秒钟，系统每秒钟最少需要连接多少台网络服务器，才能充分地利用网络连接？

3.3 在信息采集过程中，使用HEAD请求而不是GET请求的优点是什么？什么时候爬虫需要使用GET请求而不是HEAD请求？

3.4 爬虫为什么不使用POST请求？

3.5 给出本章中提到的组成深层网络的三种站点类型的名字。

3.6 为了实现对深层网络网页的信息采集，如何设计一个系统，能够自动地将数据填写到网页表单中？使用什么样的方法能保证爬虫的行为没有破坏性（例如，不会随机地增加博客评论）？

3.7 编写一段程序，根据计算机硬盘某个目录中的内容，创建一个有效的网站地图。假设来自网站 http://www.example.com的文件都是可存取的。例如，目录中有一个文件homework.pdf，该文件在 http://www.example.com/homework.pdf 上是可用的。在网站地图中，使用文件真实的修改日期作为最后修改的时间，并估计变化的频率。

3.8 假设为了使网页的采集更快，建立了两个信息采集机器，每个机器中的种子URL不同。这是一个分布式信息采集的有效策略吗？为什么？

3.9 编写一个简单的单线程网络爬虫程序。从单一的输入一个URL地址（可能是一个教授的个人主页）开始，爬虫下载一个网页，在下载下一个网页之前，至少等待5秒钟的时间。程序要能够根据上次爬取的文档中解析出来的链接标签，找到其他的页面。

3.10 UTF-16可以用于Java和Windows®。将它和UTF-8进行比较。

3.11 BigTable是如何处理硬件故障的？

3.12 设计一个用于压缩HTML标签的压缩算法。算法要能够检测HTML文件中的标签，并能够使用你设计的较短的编码对标签进行替换。编写一个编码器和解码器程序。

3.13 通过使用Unix的cksum命令对文档中的字节进行加法运算，可以对一个文档生成检验和。对文档进行编辑，看两种检验和是否发生了变化。对文档进行修改的时候，能够使检验和不发生变化吗？

3.14 编写一个程序，给文档生成simhash指纹。可以对词使用任意合理的散列函数。使用该程序对计算机上的重复文档进行检测，得出检测的准确率。检测的准确率随着指纹大小有什么变化？

3.15 对于一个给定的网页，绘出文档斜率曲线。给定的网页中至少包含一个有新闻文章的页面。测试一下主要内容块检测的优化算法的准确率。编写你自己的程序，或者使用http://www.aidanf.net/software/bte-body-text-extraction上的代码。对算法失效的情况进行说明。在这种情况下，对文档斜率曲线中低斜率区域，搜索的算法会有效吗？

3.16 对使用DOM结构来识别网页中内容信息的算法给出一个抽象的概括。特别地，对结构中内容和非内容元素进行识别的启发式方法进行描述。

第4章 文本处理

我一直在尝试理解这些词的意思。

——Spock，《星际迷航之最后的边境》

4.1 从词到词项

获得希望搜索的文本后，需要决定是否应该修改或者重构这些文本以简化搜索过程，这称为文本转换（text transformation），或者习惯地称为文本处理（text processing）。文本处理将词语可能出现的多种形式转化为更加一致的索引项（index term）。索引项在搜索中表示文档内容。

最简单的文本处理是什么也不做。一个典型的例子是文字处理器的“查找”功能。使用查找功能的时候，需要搜索的文本已经被收集起来并且呈现在屏幕上。当键入希望查找的词之后，文字处理器扫描文档并精确匹配键入的词。这个功能非常有用，因为用户有这个需求。几乎每一个文本编辑器都实现了这个功能。

精确搜索的问题在于它有很多局限性。其中最苦恼的是，它无法解决大小写敏感问题。假设要查找“computer hardware”，而文档中有一个句子以“Computer hardware”开始，那么就无法匹配这个句子，原因是查询的第一个字母是大写的。幸运的是，大多数文字处理器有搜索时忽略大小写的选项，可以认为这是在线文本处理非常基本的方式。像大多数文本处理技术那样，忽略大小写增加了在文档中搜索到查询的可能性。

大多数搜索引擎不区分大小写。不仅如此，它们还做了很多其他的事情。我们将在本章中看到，搜索引擎会忽略词中的标点符号，使得它们更容易找到。词素切分（tokenization）将词串切分成独立的词素。为了使查询处理更有效同时效率更高，有些词可能被完全忽略，这个过程称为停用词去除（stopping）。系统有可能使用词干提取（stemming），从而允许相似的词（如“run”和“running”）互相匹配。有些文档如网页中可能包含格式改变（如粗体或字体增大），或者明显的结构（structure，如标题、章节），这些也可以被系统利用。网页中包含的指向其他网页的链接，可以用来提高文档排序的性能。上述的所有方法都将在本章中讨论。

这些文本处理技术非常简单，但是它们对搜索结果影响很大。而且这些技术不涉及复杂的文本推理和理解。搜索引擎行之有效的原因，是文本的很多含义通过词语出现和共现[⊖]的次数获得，尤其当数据是从网络上的大量文本集合收集而来的。由于理解文本的统计性质是理解检索模型和排序算法的基础，因此本章首先讨论文本统计（text statistics）。自然语言处理（natural language processing）更复杂的技术涉及文本的句法和语义分析，以及它们在信息检索上的应用。虽然人们对此已经研究了数十年，但是目前它们对搜索引擎排序算法的影响很小。第11章会介绍这些技术如何应用于问答任务。另外，更复杂的文本处理技术也被用来

⊖ 词语共现衡量词组（通常为词对）在文档集合中共同出现的次数。如果一对、一组或者一个序列的词语共现的次数比预料的多，则称之为搭配（collocation）。第6章中讨论使用词项关联度量（term association measures）获取搭配。

为搜索过程识别额外的索引项和特征。本章也会讨论一些信息抽取（information extraction）技术，如识别人名、组织名、地址以及很多其他特殊类型的特征。分类（classification）可以用来识别语义类别，将在第9章中讨论。

最后，虽然本书关注英语文档的检索，但是信息检索技术本身可以应用于很多不同的语言。本章也会介绍不同语言需要哪些不同的文本表示和处理。

4.2 文本统计

尽管语言非常丰富并且变化繁多，但仍然是很容易预见的。描述一个特定话题或事件的方式很多，但是如果对描述一件事时使用的词进行计数，那么某些词出现的频率远高于其他词。有些高频词在描述任何事件时都是常见的，例如“and”和“the”。但是其他的高频词是特定事件所特有的。Luhn于1958年发现了这个现象，他提出一个词的重要性取决于它在文档中出现的频率。词语出现规律的统计模型在信息检索中很重要，应用于搜索引擎的许多核心组成部分中，如排序算法、查询转换和索引技术。这些模型将会在后续的章节讨论，在此只介绍一些统计词语出现的基本模型。

从统计角度看，文本最明显的特征之一是词语频率的分布非常倾斜（skewed）。一些词的出现频率很高，而很多词的出现次数很少。事实上，英语最高频的2个词（“the”和“of”）占了所有词出现次数的10%左右，最高频的6个词占了20%，最高频的50个词约占了40%。另一方面，在一个大规模文本集中，通常词表中约一半的词只出现一次。齐普夫法则（Zipf's Law）[⊖]描述了这种分布，它指出第r高频的词的出现次数与r成反比，或者说，一个词在词频统计表中的排名乘以它的词频（f）约等于一个常数（k）：

$$r \cdot f = k$$

本章经常提到一个词出现的概率正好等于这个词的出现次数除以所有词在文本中出现次数的总和。这样，齐普夫法则可以表示为：

$$r \cdot P_r = c$$

其中P_r表示第r高频词出现的概率，c是一个常数。对于英语而言，$c \approx 0.1$。图4-1给出了含有这个常数的齐普夫法则。图中很清楚地说明了在最高频的几个词之后，词的出现频率迅速下降。

以美联社1989年新闻报道集（称为AP89）为例，测试齐普夫法则在真实文本集合上预测词语出现次数的准确程度。AP89多年来一直被用于TREC评测。表4-1列出了AP89中词语出现次数的一些统计结果。尽管AP89的规模很小，它包含的词汇量却很大（近200 000个不同的词）。其中很大比例的词（70 000）只在语料中出现一次。长时间以来，人们一直认为在一个文本语料库或者书中，只出现一次的词语对于文本分析很重要，并且给它起了一个特殊的名字Hapax Legomena[⊖]。

表4-1 AP89中的统计信息

总文档	84 678
总的词出现次数	39 749 179
词汇量	198 763
词的出现次数>1000次	4 169
出现1次的词	70 064

⊖ 为纪念美国语言学家George Kingsley Zipf而命名。

⊖ 命名来自于研究圣经的学者。人们从13世纪开始研究圣经中词的出现，对词出现的位置建立索引（concordance）。索引是现代搜索引擎使用的倒排文件的祖先。据说第一部索引由500个僧侣完成。

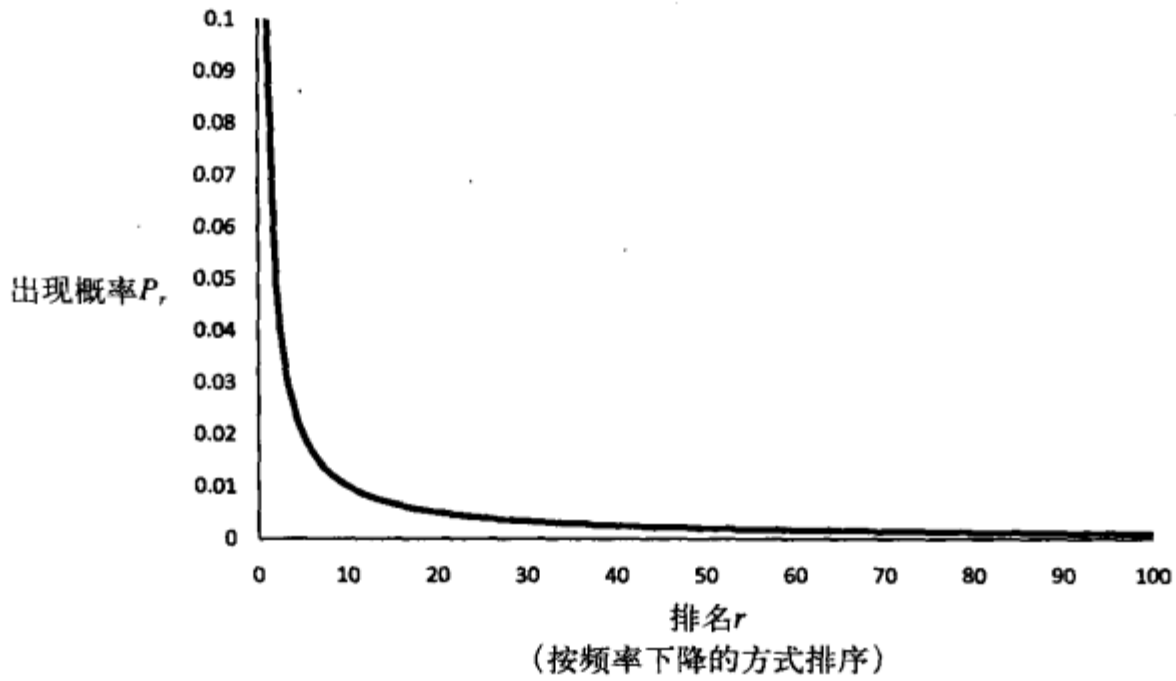


图4-1 假设齐普夫法则（次数排名×概率=0.1）时词出现的次数排名与概率的对比

表4-2罗列了AP89中前50个高频词，包括它们的频率、排名r、出现概率（转化为占总共出现次数的百分比P_r）以及$r \cdot P_r$值。从表中可以发现齐普夫法则是非常准确的，$r \cdot P_r$值近似等于一个接近0.1的常数。对于一些高频词，齐普夫法则出现了最大的差错。事实上，齐普夫法则通常对排名靠前和排名靠后（高频词和低频词）的预测不准确。表4-3给出一些AP89中低频词的例子。

表4-2 AP89中最高频的50个词

词	频率	r	P_r(%)	$r \cdot P_r$	词	频率	r	P_r(%)	$r \cdot P_r$
the	2 420 778	1	6.49	0.065	has	136 007	26	0.37	0.095
of	1 045 733	2	2.80	0.056	are	130 322	27	0.35	0.094
to	968 882	3	2.60	0.078	not	127 493	28	0.34	0.096
a	892 429	4	2.39	0.096	who	116 364	29	0.31	0.090
and	865 644	5	2.32	0.120	they	111 024	30	0.30	0.089
in	847 825	6	2.27	0.140	its	111 021	31	0.30	0.092
said	504 593	7	1.35	0.095	had	103 943	32	0.28	0.089
for	363 865	8	0.98	0.078	will	102 949	33	0.28	0.091
that	347 072	9	0.93	0.084	would	99 503	34	0.27	0.091
was	293 027	10	0.79	0.079	about	92 983	35	0.25	0.087
on	291 947	11	0.78	0.086	i	92 005	36	0.25	0.089
he	250 919	12	0.67	0.081	been	88 786	37	0.24	0.088
is	245 843	13	0.65	0.086	this	87 286	38	0.23	0.089
with	223 846	14	0.60	0.084	their	84 638	39	0.23	0.089
at	210 064	15	0.56	0.085	new	83 449	40	0.22	0.090
by	209 586	16	0.56	0.090	or	81 796	41	0.22	0.090
it	195 621	17	0.52	0.089	which	80 385	42	0.22	0.091
from	189 451	18	0.51	0.091	we	80 245	43	0.22	0.093
as	181 714	19	0.49	0.093	more	76 388	44	0.21	0.090
be	157 300	20	0.42	0.084	after	75 165	45	0.20	0.091
were	153 913	21	0.41	0.087	us	72 045	46	0.19	0.089
an	152 576	22	0.41	0.090	percent	71 956	47	0.19	0.091
have	149 749	23	0.40	0.092	up	71 082	48	0.19	0.092
his	142 285	24	0.38	0.092	one	70 266	49	0.19	0.092
but	140 880	25	0.38	0.094	people	68 988	50	0.19	0.093

表4-3 AP89中的一些低频词

词	频率	r	P_r(%)	$r \cdot P_r$
assistant	5 095	1 021	0.013	0.13
sewers	100	17 110	0.000256	0.04
toothbrush	10	51 555	0.000025	0.01
hazmat	1	166 945	0.000002	0.04

图4-2呈现了AP89中所有词的$r \cdot P_r$值的log-log图[⊖]。由于$P_r=\log(c \cdot r^{-1})=\log c-\log r$，图中齐普夫法则呈现为一条直线。图中清晰地显示了当排名靠后（大约排名10 000以后）时，齐普夫法则预测不准确。许多齐普夫法则的修正方法被提了出来[⊖]，其中一些和语言认知模型有有趣的联系。

使用齐普夫法则，给出词频可以推导出一个简单的公式来预测词的比例。出现n次的词的排名为$r_n=k/n$。通常来讲，一些词的词频有可能相同。假设词频为n的词集的最后一个的排名为r_n。这样，词频为n的词集的规模为r_n-r_{n+1}，即由词频为n的词集中最后一个的排名减去词频较高的词集中最后一个的排名（注意高频词的排名靠前）。例如，表4-4中的例子以词频下降的顺序给出词的排名。词频为5099的词的个数可以由词集最后一个“chemical”的排名减去词频稍高一些的词集中最后一个“summit”的排名得到1006−1002=4。

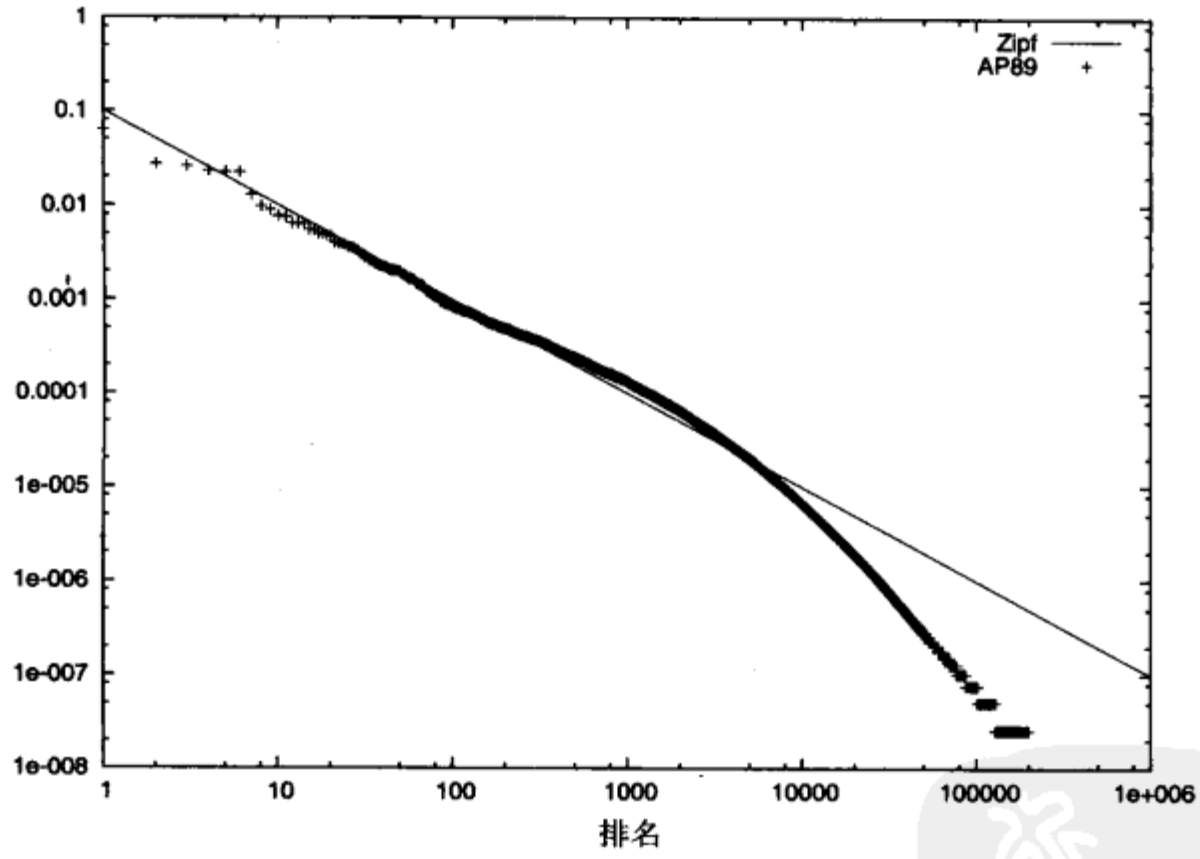


图4-2 齐普夫法则与AP89真实数据对比的log-log图。当排名较大时，齐普夫法则的预测很不准确

给出词频为n的词的个数为$r_n-r_{n+1}=k/n-k(n+1)=k/n(n+1)$，那么这些词所占的比例可以通过这个数除以所有不同词的个数得到。所有不同词的个数即词频为1的词集中最后一个的排名。词表中最后一个词的排名为$k/1=k$。因此词频为n的词占的比例为$1/n(n+1)$。例如，这个公式预

⊖ log-log图中，x轴和y轴的刻度表示x和y的对数，而非x和y本身。

⊖ 最有名的变形由数学家Benoit Mandelbrot（分形几何之父）提出。公式为$(r+\beta)^{\alpha} \cdot P_r=\gamma$，其中参数$\beta$、$\alpha$和$\gamma$可以针对特定文本进行调整。然而在AP89上，这个方法没有明显比齐普夫法则好。

测词表中一半的词只出现一次。表4-5将这个公式的预测结果与来自TREC另外一个语料集合的真实数据进行了比较。

表4-4 词频排名的例子

排名	词	词频
1000	concern	5 100
1001	spoke	5 100
1002	summit	5 100
1003	bring	5 099
1004	star	5 099
1005	immediate	5 099
1006	chemical	5 099
1007	african	5 098

表4-5 来自TREC第3卷语料库的336 310个文档中出现n次的词在文本中所占的比例。词汇量总数为508 209

出现次数(n)	预测比例($1/n(n+1)$)	真实比例	真实的词数
1	0.500	0.402	204 357
2	0.167	0.132	67 082
3	0.083	0.069	35 083
4	0.050	0.046	23 271
5	0.033	0.032	16 332
6	0.024	0.024	12 421
7	0.018	0.019	9 766
8	0.014	0.016	8 200
9	0.011	0.014	6 907
10	0.009	0.012	5 893

4.2.1 词表增长

与词的出现次数相关的另一种有用的预测是词表增长（vocabulary growth）。随着语料规模增大，新词不断出现。基于词的齐普夫法则分布假设可以预测，随着语料规模增大，一定规模新文本中含有的新词的数目减少。然而，不断出现的新词来源于新造的词（药名和刚起步的公司名）、拼写错误、产品号码、人名、邮件地址等等。Heaps（1978）观察发现，语料规模与词表大小的关系为：

$$v=k \cdot n^{\beta}$$

其中v为词汇量大小，语料中共有n个词，k和β是随不同语料变化的参数。这个公式有时称为Heaps法则。一般k和β满足和$10 \leqslant k \leqslant 100$和$\beta \approx 0.5$。Heaps法则预测当语料规模很小时，新词数量增长非常快。随着语料规模变大，新词数量无限增长，但是增长速度会变慢。图4-3比较了AP89中词汇的增长和Heaps法则当k=62.95和β=0.455时的结果。很明显，Heaps法则拟合效果很好。对于很多其他TREC新闻语料集合，参数值是类似的。举一个预测精确度的例子，分析完AP89的前10 879 522个词之后，Heaps法则预测的词表长度为100 151，而真实值为100 024。当只分析过很少的词的时候（<1000），Heaps法则对词表长度的预测精确度要差很多。

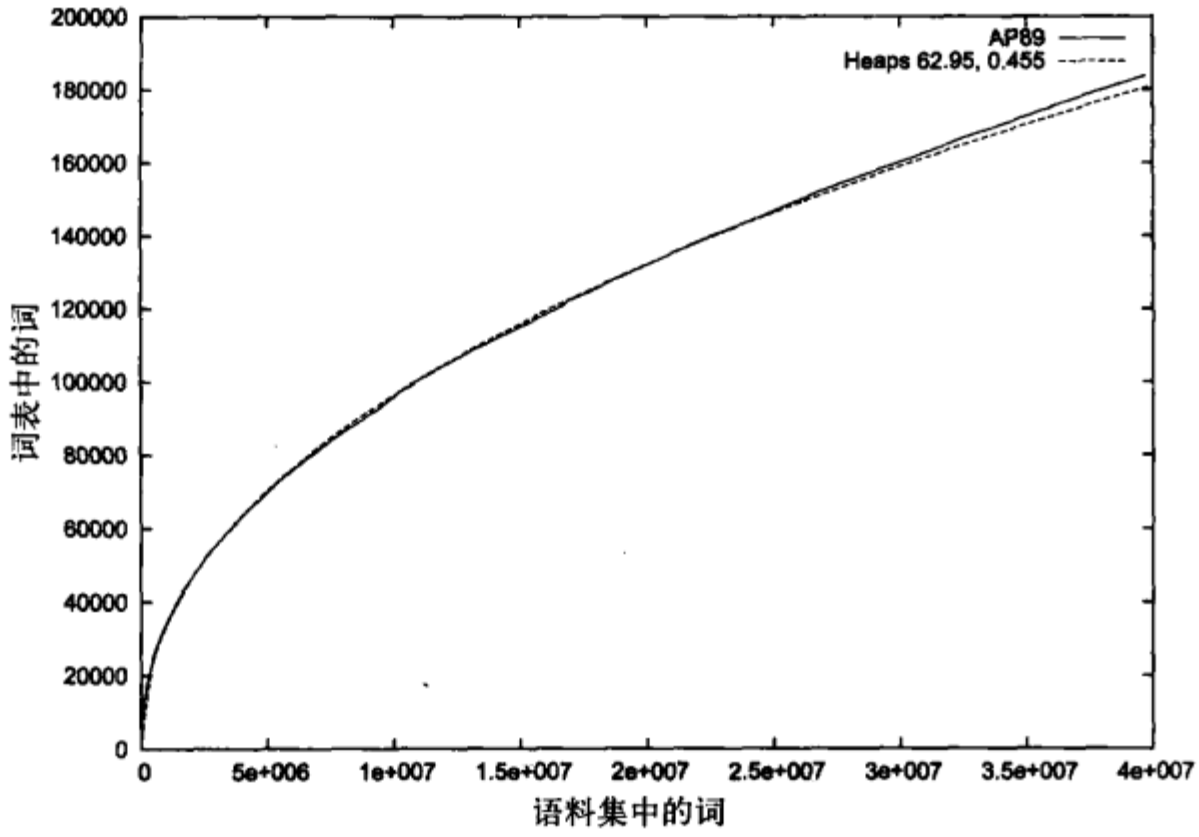


图4-3 TREC AP89词表增长与Heaps法则预测结果比较

网络量级的语料比AP89的规模大很多。AP89共包含约4000万词，而TREC网络语料GOV2[⊖]（相对较小规模）包含200亿以上词。当词的数目达到这个程度时，似乎新词的数目将最终跌到接近零，从而Heaps法则不再适用。结果表明情况不是这样的。图4-4给出了GOV2的词表增长和Heaps法则当k=7.34和β=0.648的预测结果。数据显示即使当词数超过3000万词表，仍然继续稳步增长。这对搜索引擎的设计有很大的影响，对此将会在第5章中讨论。Heaps法则在GOV2上拟合得很好，然而参数值与其他TREC语料使用的参数有很大的不同。

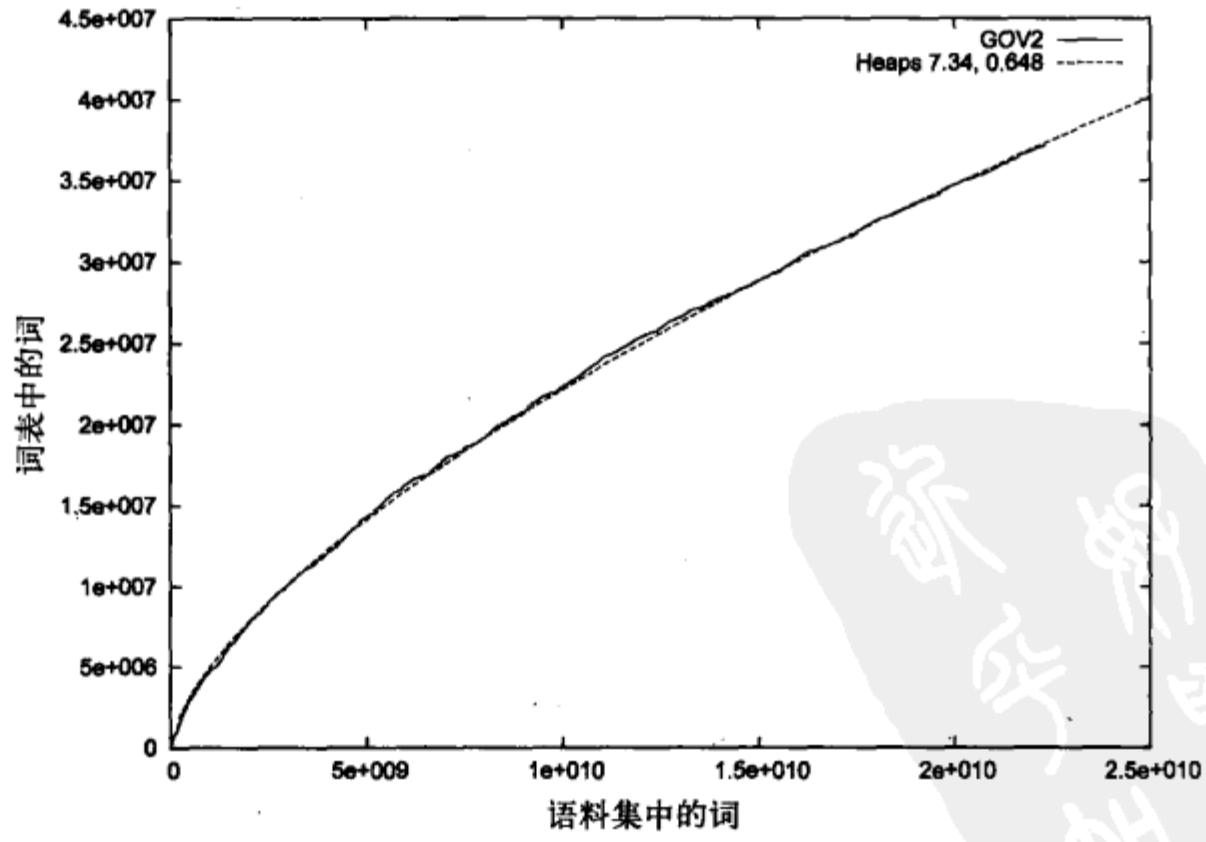


图4-4 TREC GOV2词表增长与Heaps法则结果比较

⊖ 2004年初从属于.gov域名的网页中爬取。详情请见8.2节。

4.2.2 估计数据集和结果集大小

统计词语出现也可以用于估计网络搜索结果的规模。所有网络搜索引擎都有类似于图4-5中的查询界面。给出查询内容（此例中为“tropical fish aquarium”）后，搜索引擎在输出排序后的结果之前，会立刻给出查询结果总数的估计值。通常这是一个很大的数目，搜索引擎系统一般都指出这只是估计值。

图4-5 网络搜索结果数量估计

为了估计结果数量，首先需要定义“结果”。在估计结果数量时，一个结果指包括所有查询词的任何文档（或者网页）。有些搜索应用系统对文档排序时，还包括了只含有一部分查询词的文档。但是考虑到网络的巨大规模，这样做通常是没有必要的。如果假设词的出现彼此独立，那么一个文档包括所有查询词的概率就简单地等于文档包含各个查询词的概率的乘积。例如，如果查询中包含三个词a、b和c，那么：

$$P(a \cap b \cap c) = P(a) \cdot P(b) \cdot P(c)$$

其中$P(a \cap b \cap c)$是联合概率，或者所有三个词在文档中出现的概率。而$P(a)$、$P(b)$和$P(c)$是每一个词在文档中出现的概率。搜索引擎总是可以得到一个词出现过的文档的数目（f_a、f_b和f_c）[⊖]，以及集合中包含的文档总数。因此这些概率可以很容易地估计：$P(a)=f_a/N$，$P(b)=f_b/N$，$P(c)=f_c/N$。这样可以得到：

$$f_{abc} = N \cdot f_a / N \cdot f_b / N \cdot f_c / N = (f_a \cdot f_b \cdot f_c) / N^2$$

其中f_{abc}是结果数量的估计值。

表4-6 GOV2网络语料中文档出现频率以及词语组合的估计文档出现频率（假设互相独立）。语料规模（N）为25 205 179

词	文档频率	估计频率	词	文档频率	估计频率
tropical	120 990		tropical breeding	5 510	393
fish	1 131 855		fish aquarium	9 722	1 189
aquarium	26 480		fish breeding	36 427	3 677
breeding	81 885		aquarium breeding	1 848	86
tropical fish	18 472	5 433	tropical fish aquarium	1 529	6
tropical aquarium	1 921	127	tropical fish breeding	3 629	18

表4-6给出了TREC GOV2网络语料中“tropical”、“fish”、“aquarium”和“breeding”的文档出现频率，以及这些词语组合的文档出现频率。同时，基于独立假设，表中还列出了这些组合的估计频率。很明显，这种独立假设导致对结果规模估计不准确，尤其是三个词的组合。问题在于组合中的词语在文档中的出现不互相独立。例如，一个文档包含词语“fish”，那么它比另一个不包含“fish”的文档更有可能包含“aquarium”。

如果搜索引擎含有词语的共现信息，那么对结果规模的估计可以更准确。显然对于含有两个词的查询，这样做可以给出真实的值。对于更长的查询，不假设独立性可以提高估计的

⊖ 注意这些是文档出现频率（document occurrence frequencies），而非词语出现的总数（一个文档中一个词可能出现多次）。

准确度。通常对于三个词：

$$P(a \cap b \cap c) = P(a \cap b) \cdot P(c \mid (a \cap b))$$

其中 $P(a \cap b)$ 是词语a和b是在文档中共现的概率，$P(c \mid (a \cap b))$ 是a和b都在一个文档中出现的情况下，c也在这个文档中出现的概率[⊖]。如果有共现信息，可以使用$P(c|a)$或$P(c|b)$较大的值估计这个概率。以表4-6中的查询“tropical fish aquarium”为例，这意味将“tropical”和“aquarium”同时出现的文档数乘以一个文档同时包含“aquarium”和“fish”的概率，从而获得结果集合的估计数量，或者表示为：

$$\begin{aligned} f_{tropical \cap fish \cap aquarium} &= f_{tropical \cap aquarium} \cdot f_{fish \cap aquarium} / f_{aquarium} \\ &= 1921 \cdot 9722 / 26480 = 705 \end{aligned}$$

类似地，对于查询“tropical fish breeding”：

$$\begin{aligned} f_{tropical \cap fish \cap breeding} &= f_{tropical \cap breeding} \cdot f_{fish \cap breeding} / f_{breeding} \\ &= 5510 \cdot 36427 / 81885 = 2451 \end{aligned}$$

这些估计比假设互相独立得到的估计值好得多，但是仍然太慢了。结果表明，无需存储更多的信息，比如词语三元组的出现次数，而只使用词频和当前（current）结果集合大小，便可以获得合理的结果集合大小估计。搜索引擎估计结果集合大小的原因是它们不对所有包含查询词的文档进行排序。相反，它们只对最相关的很小一部分文档进行排序。如果知道被排序的文档在所有文档所占的比例（s）和包含所有查询词的文档数量（C），这时假定包含所有查询词的文档分布均匀[⊜]，从而可以将结果集合大小简单估计为C/s。被处理文档所占的比例可以由包含查询中最低频词的文档被处理的比例来衡量，因为所有的结果必须包含那个词。

例如，如果在Galago搜索引擎中使用查询“tropical fish aquarium”对GOV2文档集进行排序。共有26 480个文档包含“aquarium”，处理其中的3000个文档之后，共有258个文档包含所有的查询词。那么结果数量估计为258/(3000/26 480)=2 277。处理完超过20%的文档时，估计值为1778（真实值为1529）。对于查询“tropical fish breeding”，处理完含有“breeding”的文档中的10%和20%后，估计值分别为4076和3762（真实值为3629）。这些估计值非常准确，并且不需要集合中文档的总数目。

事实上，估计搜索引擎中存储的文档总数目对于学术界（网络有多大）和企业界（哪个搜索引擎对网络覆盖面大）都意义重大。有很多研究这种技术的论文，其中一篇是基于以前用到的词语互相独立的概念。如果a和b是两个互相独立出现的词，那么

$$f_{ab} / N = f_a / N \cdot f_b / N$$

以及

$$N = (f_a \cdot f_b) / f_{ab}$$

为了得到N的合理估计值，这两个词应该互相独立。从表4-6中的例子观察得知，实际并非如此。可以更小心地选择查询词。例如使用“lincoln”（GOV2中的文档频率为771 326），可以预计与表4-6中的词对相比，查询“tropical lincoln”中的词更加互相独立（因为这个查询

⊖ 称为条件概率。

⊜ 在此假设同时处理文档（document-at-a-time），即同时处理所有查询词的倒排列表，从而给出完整的文档分值（参见第5章）。

中的词义上联系更少)。GOV2中“tropical lincoln”的文档频率为3 018，这意味着可以估计集合大小为$N = (120\ 990 \cdot 771\ 326)/3\ 018=30\ 992\ 045$。这与实际值25 205 179非常接近。

4.3 文档解析

4.3.1 概述

文档解析的目的是识别文本文档的内容和结构。大多数文档中主要的内容是词，上一节对词进行计数并且使用齐普夫分布进行建模。从文档的字符序列中识别每个词的过程称为词素切分或者词法分析（lexical analysis)。除了词，文档中还有很多其他类型的内容，如元数据、图片、图表、代码和表格。如第2章所述，元数据是文本内容之外的文档信息。元数据内容包括文档属性，如日期、作者，以及最重要的是，标记语言（markup language）中使用的用以识别文档组成部分的标签（tag)。最流行的标记语言是超文本标记语言(Hypertext Markup Language，HTML）和可扩展标记语言（Extensible Markup Language，XML)。

基于标记语言的语法（语法分析)，解析器使用标签和其他文档中的元数据来解析文档的结构，产生包含结构和内容的文档表示。例如，一个HTML解析器对一个使用HTML标签的网页进行结构解析，将网页表示成文档对象模型（Document Object Model, DOM)，提供给网络浏览器使用。在搜索引擎中，文档解析器的分析结果是文档的结构和内容表示，用以构建索引。由于索引需要能够表示集合中的每一个文档，因此文档解析器要比其他应用中使用的解析器具有更强的句法容错性。

讨论文档解析的第一部分主要关注识别构成文档内容的词素、词、短语。其他部分分别讨论与文档结构相关的重要话题，如标记、链接以及从文本内容中抽取结构。

4.3.2 词素切分

词素切分是指从文档中的字符序列中获取词的过程。这在英语中似乎很简单。在很多早期的系统中，一个词（word）定义为长度为3或者更多，并且以空格或其他特殊字符结束的字母数字串。所有的大写字母被转化为小写[⊖]。这意味着，以下面文本为例：

Bigcorp's 2007 bi-annual report showed profits rose 10%.

将会产生如下的标记序列：

bigcorp 2007 annual report showed profits rose

这种简单的词素切分过程足以应付小规模的测试集合。但是对于大多数搜索应用或者TREC集合上的实验，由于丢掉了太多的信息，这样做就不适合了。下面举一些例子说明词素切分对于搜索的有效性有很大影响。

- 短小的词（一个或两个字符）在一些查询中可能很重要。它们通常与其他词结合起来。比如，xp、ma、pm、ben e king、el paso、master p、gm、j lo、world war II[⊖]。
- 对于很多词，带连字符和不带连字符的形式都很普遍。有些情况下，连字符不需要。比如，e-bay、wal-mart、active-x、cd-rom、t-shirts。而其他情况下，连字符应该被认为是

⊖ 有时称之为大小写折叠（case folding)、大小写规范化（case normalization）或者小写转化（downcasing)。

⊖ 这些和其他例子从一个网络查询的小样本中得到。

词的一部分或者词的分隔符。比如，winston-salem、mazda rx-7、e-cards、pre-diabetes、t-mobile、spanish-speaking。

- 标签、URL、代码（code）及文档的其他重要部分必须正确解析。对于它们，特殊符号是很重要的组成部分。
- 大写的词可能与小写的词的意义不同，比如“Bush”和“Apple”。
- 撇号“’”可能是词的一部分、所有格的一部分，或者仅仅是一个错误。比如，rosie o’donnell、can’t、don’t、80’s、1890’s、men’s straw hats、master’s degree、england’s ten largest cities、shriner’s。
- 数字，包括带小数的数有可能很重要。比如，nokia 3250、top 10 courses、united 93、quicktime 6.5 pro、92.3 the beat、288358（这是一个真正的查询，它表示专利号）。
- 句点可能出现在数字、简称、URL、句子末尾和其他情形中。

从这些例子可以看出，词素切分比最先表现得更复杂。这些例子来自于查询的事实也强调了用于文档和用于查询的文本处理必须一致。如果不一致的话，很多文档的索引项不会匹配查询中对应的词项。检索失败很快会让词素切分不一致产生的错误变得很明显。

为了包含各种语言处理技术，以保证有效的匹配，词素切分过程应该简单而灵活。一种方法是利用第一遍词素切分先识别文档标记或标签。可以使用一个专门针对特定标记语言（如HTML）的解析器做这件事。如前面所提到的，解析器应该能够容忍句法错误，接下来便可以对文档结构中合适的部分进行第二遍词素切分。这一步将会忽略文档中与搜索无关的部分，如包括HTML代码的内容。

考虑到文档文本中几乎所有的内容都可能对查询意义重大，因此词素切分规则必须将绝大多数内容转化为可搜索的标记。一些更难的问题，如识别词语的变形、名字或日期，可以在其他独立的步骤中处理（词干提取、信息抽取和查询转换），而不是让词素切分器完成所有的事情。信息抽取通常将完整的文本作为输入，包括大写和标点符号，因此这些信息应该保留下来直到抽取工作完成。除了这一点限制，大写对于搜索来说非常的重要，而在索引过程中可以将文本转化为小写。这并不意味着查询中不会使用大写的词。事实上，在查询中大写经常用到。而在有些查询中，大写并不能减少歧义，从而不会影响检索的有效性。一些例子经常使用“Apple”（在实际中并不常见），这个问题可以通过查询重构（query reformulation）（见第6章）或者简单使用最普遍的网页（见4.5节）来解决。

如果决定在其他步骤中处理复杂的问题，那么最一般的做法是将连字符、撇号和句点看作词的结束标记（如空格）。对所有生成的标记建索引是很重要的，包括单字符的标记如“s”和“o”。举例来讲，这意味着查询[⊖]“o’connor”等价于“o connor”，“bob’s”等价于“bob s”，“rx-7”等价于“rx 7”。注意这意味着词“rx7”将是不用于“rx-7”的记号，因此会被单独建索引。将“rx 7”、“rx7”、“rx-7”联系起来的任务，将交给搜索引擎中查询转换模块来处理。

另一方面，如果完全依赖查询转换对不同的词语进行适当的联系或推理，可能会有降低有效性的风险，尤其对于那些没有足够数据做查询扩展的应用。这种情况下，可以在词素切分器中加入更多的规则，以保证从查询文本中产生的标记可以匹配从文档文本产生的标记。

⊖ 假设网络查询的普通语法是“<word>”表示严格匹配引号中的短语。

比如在TREC集合中，使用一个规则将所有包含撇号的词语解析成不包含撇号的串是非常有效的。这个规则将“O'Connor”解析成“oconnor”，“Bob's”变成“bobs”。针对TREC集合的另外一个有效规则是，将所有包含句点的串简称解析成不包含句点的串。这种情况下，一个简称就是指被句点分成单个字母的串。这条规则将“I.B.M.”解析为“ibm”，但是“Ph.D”仍然变成“ph d”。

总之，最通用的词素切分首先识别文档结构，然后将文本中任何以空格和特殊符号结束的字母数字序列识别为词语，并将大写字母转换为小写。这样做并不比本节开始提到的简单处理复杂多少，仍然将困难的问题交给信息抽取和查询转换来处理。在许多情况下，词素切分器中需要增加额外的规则来处理一些特殊的字符，从而保证查询和文档的词项可以匹配。

4.3.3 停用词去除

人类语言包含很多功能词。与其他词相比，功能词没有什么含义。最普遍的功能词是限定词（determiner）——“the”、“a”、“an”、“that”和“those”。这些词帮助在文本中描述名词和表达概念，如地点或数量。介词如“over”、“under”、“above”和“below”表示两个名词的相对位置。

这些功能词的两个特性促使在文本处理中对其特殊对待。第一，这些功能词极其普遍。表4-2显示AP89集合中几乎所有的高频词都属于这个类别。记录这些词在每一个文档中的数量需要很大的磁盘空间。第二，由于它们的普遍性和功能，这些词很少单独表达文档相关程度的信息。如果在检索过程中考虑每一个词而不是短语，这些功能词基本没有什么帮助。

在信息检索中，这些功能词的另一个名称是：停用词（stopword）。称它们为停用词是因为在文本处理过程中如果遇到它们，则立刻停止处理，将其直接扔掉。将这些词扔掉减少了索引量，增加了检索效率，并且通常都会提高检索的效果。

构建停用词词表需要谨慎行之。去除太多的词会损害检索效果，影响用户体验感受。例如，查询“to be or not to be”时所有词都是通常认为的停用词。虽然不去除停用词会让排序出现麻烦，但是去除停用词也会让非常合理的查询没有检索结果。

停用词词表可以简单由集合中前n（如50）个高频词构成。然而这样也会导致某些对查询很重要的词也被包含进来。更典型的做法是，或者使用一个标准的停用词词表[⊖]，或者手动维护一个高频词和标准停用词词表，将可能对特定应用有意义的词从词表中删除。也可以为文档结构中特定的部分（也称为域，field）定制停用词词表。例如，当处理锚文本时，“click”、“here”、“privacy”作为停用词可能便是合理的。

如果存储空间允许，最好索引文档中所有的词。如果需要去除停用词，那么可以去除查询中的停用词。在索引中保留停用词，便有多种方法来处理含有停用词的查询。例如，许多系统会去除查询中的停用词，除非停用词前面有加号标记。如果由于存储空间有限而无法在索引中保留停用词，那么也应该去除尽量少的词，以保持最大限度的灵活性。

4.3.4 词干提取

自然语言强大的表达能力在于，可以有大量不同的方式表达同一观点。这对依赖匹配词语找到相关文档的搜索引擎而言，便成为一个问题。很多技术允许搜索引擎匹配语义相关的

⊖ 例如，随Lemur工具包发布了一个停用词词表，被Galago采用。

词，而不是严格要求完全一致的词才能匹配。词干提取（stemming），也称为异文合并（conflation），是指获得一个词不同变形之间关系的文本处理过程。更准确地讲，词干提取将一个词由于变形（inflection）（如复数、时态）或者派生（derivation）（如在一个动词后加后缀-ation得到对应名词）产生的多种不同形式简化为一个共同的词干。

假设希望搜索与Mark Spitz的奥林匹克游泳运动事业相关的新闻报道，可能向一个搜索引擎输入“mark spitz swimming”。然而，许多新闻报道通常是对已发生事件的总结，因此它们有可能包含“swam”而非“swimming”。这时就需要词干提取器将“swimming”和“swam”归结为相同的词干（有可能是“swim”），从而允许搜索引擎匹配这两个词。

通常来讲，在面向英文文本搜索应用中，使用词干提取器会使搜索结果质量有小的但是明显的提高。在面向变形现象非常普遍的语种如阿拉伯语或俄语的应用中，词干提取是有效搜索的关键环节。

有两种基本类型的词干提取器：规则演算系统（algorithmic）和基于词典的系统（dictionary-based）。一个基于规则演算的词干提取器通常基于特定语言词缀知识，使用一个小的程序决定两个词是否相关。相反，基于词典的词干提取器没有本身的逻辑，而是依赖预先构建的相关词语的词典来存储词语的关系。

最简单的基于规则演算的英语词干提取器只处理后缀“s”。这种系统假设任何以“s”结束的词均是复数，因此cakes→cake，dogs→dog。当然，这个规则并不完美，它无法发现很多复数关系，如“century”和“centuries”。在非常特殊的情况下，它会发现一些不存在的关系，如“I”和“is”。第一类错误被称为错误否定（false negative），第二类错误称为错误肯定（false positive）[⊖]。

更复杂的基于规则演算的词干提取器考虑更多种类的后缀，如-ing或-ed，以减少错误否定。通过处理更多类型的后缀，系统可以找到更多的词语关系，换句话说，减少了错误否定率。然而，错误肯定率（找到不存在的关系）往往升高。

最流行的基于规则演算的词干提取器是Porter stemmer[⊖]。自从20世纪70年代以来，它被使用在很多信息检索实验和系统中，并且有很多不同的实现可用。这个词干提取器由很多步骤组成，每一步骤包含一套去除后缀的规则。每一步骤总是优先执行处理最长后缀的规则，有些规则的含义很明显，而其他规则需要深入理解才能明白它们的作用。这里将第一步（共五步）中的前两部分用来举例：

步骤1a：

- 用“ss”替换后缀“sses”（如stresses→stress）。
- 如果后缀“s”前面的部分包含一个元音，并且元音不紧邻这个“s”，那么删除这个“s”（如gaps→gap，而gas→gas）。
- 如果后缀“ied”或“ies”前面有超过一个字母，那么替换成“i”，否则替换成“ie”（如ties→tie，cries-cri）。
- 如果后缀是“us”或“ss”，就什么也不做（如stress→stress）。

步骤1b：

- 如果后缀“eed”或“eedly”位于一个元音后面第一个非元音之后，则替换为“ee”（如

⊖ 这些术语在任何二值决策过程中使用，以描述两种错误。这包括评价（见第8章）和分类（见第9章）。

⊖ http://tartarus.org/martin/PorterStemmer/。

agreed→agree，feed→fee）。

- 如果后缀“ed”、“edly”、“ing”或“ingly”前面存在元音，那么删除这个后缀。然后，如果词以“at”、“bl”或“iz”结束，那么在词尾增加“e”（如fished→fish，pirating→pirate）；或者如果词以非“ll”、“ss”或“zz”的双写字母结束，则删除最后一个字母（如falling→fall，dripping→drip）；或者如果词很短，则在词尾增加“e”（如hoping→hope）。
- 哇！

Porter stemmer在很多TREC评测中和搜索应用中证明是有效的。但是很难在一个相对简单的算法中包括一个语言所有的细节变化。最初版本的Porter stemmer犯了很多错误，错误肯定和错误否定都有。表4-7中给出一些错误的例子。很容易想像，如果混淆了“execute”和“executive”，或者“organization”和“organ”，就会使结果排序出现问题。一个最近的版本（称为Porter2）⊖解决了一些这样的问题，并且提供了一种机制允许指定例外。

表4-7 早期Porter stemmer错误示例。错误肯定指词干不同的词对。错误否定指词干相同的词对

错误肯定	错误否定	错误肯定	错误否定
organization/organ	european/europe	negligible/negligent	useful/usefully
generalization/generic	cylinder/cylindrical	execute/executive	noise/noisy
numerical/numerous	matrices/matrix	past/paste	decompose/decomposition
policy/police	urgency/urgent	ignore/ignorant	sparse/sparsity
university/universe	create/creation	special/specialized	resolve/resolution
addition/additive	analysis/analyses	head/heading	triangle/triangular

基于词典的词干提取器提供了解决错误的另外一种方法。可以在一个大的词典中存储相关词的列表，而不是尝试从字母模版来发现词的关系。这些词表可以由人来创建，可以期望错误肯定率会很低。相关的词甚至不必看起来相似，一个基于词典的词干提取器可以识别“is”、“be”和“was”，它们都是相同动词的不同形式。然而，一个词典不可能无限长，因此它无法自动融入新词。随着语言的不断发展，这是一个很重要的问题。通过对文本语料进行统计分析，自动建立词干字典是有可能的。当词干提取用于查询扩展时，这种方法非常有用，我们将在6.2.1节讨论它。

另外一种策略是结合基于规则和基于词典的词干提取器。一般地，一些不规则的词如动词“to be”在语言中是最古老的，而新词则遵循更规则的语法约定。这意味着新造的词很可能适合用基于规则的系统分析。因此，词典可以用来发现平常的词之间的关系，而规则可以用来处理陌生词。

这种混合方法的一个有名的例子是Krovetz stemmer（Krovetz，1993）。这个词干提取器使用词典来确定一个词是否正确。Krovetz stemmer中使用的词典包括英语的一个通用词典，同时使用一些手动生成的例外列表。词干提取之前，先确定一个词是否在词典中存在。如果存在，或者将这个词保留下来（如果它在通用词典中），或者基于例外列表对这个词进行词干提取。如果这个词没有包含在词典中，那么使用一个通用的变形和派生后缀列表，逐个检查这个词。如果找到一个匹配的后缀，那么从这个词删除这个后缀，然后再次检查这个词是否

⊖ http://snowball.tartarus.org。

包含在词典中。如果不包含在词典中，那么根据被删除的后缀修改这个词的结尾。例如，如果“-ies”匹配上的话，就使用“-ie”来替换，然后检查是否包含在词典中。如果词典中包括这个词，那么接受这个词干，否则词尾被替换为“y”。例如，这将导致calories→calorie。检查后缀的过程是一个序列（例如，先检查复数，然后检查是否以“-ion”结尾），因此可能会去除多个后缀。

Krovetz stemmer比Porter stemmer的错误肯定率低，但是取决于例外列表的大小。它的错误否定率较高。总之，用于搜索评价时，这两个词干提取器的效果是可以比较的。Krovetz stemmer的一个优势是，大多数情况下，它产生的词干是完整的词，而Porter stemmer通常产生词片段。如果词干被用于搜索界面，这将会带来一定的影响。

Original text:
Document will describe marketing strategies carried out by U.S. companies for their agricultural chemicals, report predictions for market share of such chemicals, or report market statistics for agrochemicals, pesticide, herbicide, fungicide, insecticide, fertilizer, predicted sales, market share, stimulate demand, price cut, volume of sales.

Porter stemmer:
document describ market strategi carri compani agricultur chemic report predict market share chemic report market statist agrochem pesticid herbicid fungicid insecticid fertil predict sale market share stimul demand price cut volum sale

Krovetz stemmer:
document describe marketing strategy carry company agriculture chemical report prediction market share chemical report market statistic agrochemic pesticide herbicide fungicide insecticide fertilizer predict sale stimulate demand price cut volume sale

图4-6 多个词干提取器对一个TREC查询的输出结果比较。停用词被去除

图4-6比较了Porter stemmer和Krovetz stemmer针对一个TREC查询的输出。虽然“marketing”由于在词典中存在而没有被简化为“market”，但从哪些词被简化为同一个词干的角度看，Krovetz stemmer的结果是相似的。Krovetz stemmer得到的词干大多是完整的词。一个例外是“agrochemic”，这是因为“agrochemical”不包含在词典中。注意这个例子中的文本处理，去除了停用词，包括单字符的词。这样便扔掉了“U.S.”，从而会对某些查询产生重大的后果。这种情况可以通过更好的词素切分和信息抽取来处理，这将在4.6节讨论。

像停用词的情况那样，如果不对文档中的词提取词干，而是以原来的形式建立索引的话，搜索引擎在回答各种各样的查询时会更加灵活。词干提取可以作为一种查询扩展，对此将在6.2.1节解释。在一些应用中，词和词干都建索引，从而提供灵活而有效率的查询处理。

前面提到词干提取对于一些语言特别重要，而对其他语言则基本没有影响。使用与语言有关的词干提取算法，是为多种语言定制或者实现一个搜索引擎的国际化（internationalizing）最重要的方面。我们将在4.7节讨论国际化的其他方面，在此只关注词干提取。

作为一个例子，表4-8包含一些从相同的词干派生出来的阿拉伯语词语。尝试将阿拉伯语词语简化为词干的词干提取算法显然无法工作（阿拉伯语有少于2 000个词干），但是需要考虑很多前缀和后缀。词形变化非常繁多的语言如阿拉伯语有很多词语变形，词干提取可以在很大程度上改变查询结果排序精度。一个包含高质量词干提取器的阿拉伯语搜索引擎，比不包含词干提取器的系统在获得相关文档上的效果平均好50%以上。相反，在大规模数据集合上，英语搜索引擎的效果提高少于5%，而在小规模、特定领域的集合上，约为10%。

幸运的是，许多语言的词干提取器已经被开发出来，并且作为开源软件自由使用。例如，Porter stemmer在法语、西班牙语、葡萄牙语、意大利语、罗马尼亚语、德语、丹麦语、瑞典语、挪威语、丹麦语、俄语、芬兰语、匈牙利语和土耳其语上都有可用版本[⊖]。另外，当只有一个文本语料可用的时候，才可以使用6.2.1节描述的统计方法建立词干提取器。

表4-8 阿拉伯语从词干ktb派生出的词的例子

kitab	a book
kitabi	my book
alkitab	the book
kitabuki	your book(f)
kitabuka	your book(m)
kitabuhu	his book
kataba	to write
maktaba	library, bookstore
maktab	office

4.3.5 短语和n元串

显然短语对信息检索很重要。很多提交给搜索引擎的两个词或三个词的查询都是短语，找到含有这些短语的文档将是有效排序算法的一部分。例如，给定查询“black sea”，与包含文本如“the sea turned black”的文档相比，含有这个短语的文档更有可能是相关的。描述一个话题的时候，短语比单个词更准确（如“tropical fish”相对“fish”），并且通常歧义更少（如“rotten apple”相对“apple”）。然而，短语对检索的影响可以很复杂。给定查询如“fishing supplies”，检索出的文档应该确切地包含这个短语，还是只要在同一段落甚至同一文档中包含“fish”、“fishing”和“supplies”这几个词即可？短语影响排序的细节，问题与搜索引擎包含的特定检索模型紧密联系，因此将这个讨论延到第7章。从文本处理的角度看，问题在于是否应该在词素切分和词干提取的同时识别短语，从而可以对短语建索引，进而加速查询处理。

短语有很多的定义，其中大多数多年来一直被人们在检索实验里研究。由于短语有一个语法定义，用句子的语法结构来识别短语似乎比较合理。

在信息检索研究中，使用最频繁的定义是短语等价于简单的名词短语（noun phrase）。短语经常被进一步限制为只包含名词或名词前的形容词的序列。这种标准定义的短语可以使用词性标注器（part-of-speech tagger， POS）识别。词性标注器根据上下文信息对文本中的每一个词赋予一个词性标记。词性标注器基于统计或规则的方法，并且使用人工标注的大规模语料训练。一般的词性标记包括NN（单数名词）、NNS（复数名词）、VB（动词）、VBD（动词，过去时）、VBN（动词，过去分词）、IN（介词）、JJ（形容词）、CC（连词，如“and”、“or”）、PRP（代词）和MD（情态助词，如“can”、“will”）。

图4-7给出一个词性标注器对图4-6中的TREC查询文本的处理结果。这个例子说明标注器可以识别名词序列构成的短语，如“marketing/NN strategies/NNS”，或者形容词序列后跟名词序列的短语，如“agricultural/JJ chemicals/NNS”。然而，标注器难免有错误。“predicted/VBN sales/NNS”就不能识别为一个短语，因为“predicted”被标为动词。

表4-9给出了一些高频的简单名词短语，它们来自一个主要包含新闻报道的TREC语料和一个相同规模的包含美国专利与商标部（PTO）1996年发布的所有专利的语料。这些短语通过词性标注被识别出来。例子中短语的频率大小显示PTO语料使用短语更频繁，因为专利措辞比较规范，会使用相当多的重复。在TREC语料中，有1 100 000个短语出现超过五次，而在PTO语料中有3 700 000个。很多TREC语料中的短语是专有名词，如“los angeles”或“european union”，或者是对检索很重要的话题，如“peace process”和“human rights”。有

⊖ http://snowball.tartarus.org/。

两个短语和文档的格式相关（“article type”、“end recording”）。另一方面，PTO语料中大多数短语则是用以描述专利的标准术语，如“present invention”和“preferred embodiment”，而相对少数的短语与专利的内容相关，如“carbon atoms”和“ethyl acetate”。有一个短语“group consisting”是频繁的词性标注错误造成的。

Original text:
Document will describe marketing strategies carried out by U.S. companies for their agricultural chemicals, report predictions for market share of such chemicals, or report market statistics for agrochemicals, pesticide, herbicide, fungicide, insecticide, fertilizer, predicted sales, market share, stimulate demand, price cut, volume of sales.

Brill tagger:
Document/NN will/MD describe/VB marketing/NN strategies/NNS carried/VBD out/IN by/IN U.S./NNP companies/NNS for/IN their/PRP agricultural/JJ chemicals/NNS ,/, report/NN predictions/NNS for/IN market/NN share/NN of/IN such/JJ chemicals/NNS ,/, or/CC report/NN market/NN statistics/NNS for/IN agrochemicals/NNS ,/, pesticide/NN ,/, herbicide/NN ,/, fungicide/NN ,/, insecticide/NN ,/, fertilizer/NN ,/, predicted/VBN sales/NNS ,/, market/NN share/NN ,/, stimulate/VB demand/NN ,/, price/NN cut/NN ,/, volume/NN of/IN sales/NNS ./.

图4-7 一个TREC查询的词性标注结果

虽然词性标注可以产生合理的短语，并且已经被用在很多应用中，但是总的来讲，对于大规模数据集合，这种方法作为短语建索引的初步工作，就太慢了。其他更简单、更快的方法可以产生同样的效果。一种方法是在索引中存储词的位置信息，只有在处理查询时才利用这个信息识别短语。这样提供了相当大的灵活性，因为既可以由用户，也可以通过对查询进行词性标注识别短语，并且短语不局限于紧邻的词语。句法短语的识别可以取代为测试词是否近邻，比如两个词是否在一个特定文本窗口内出现。第5章将讲述对位置建索引，第7章将讲述利用词近邻的检索模型。

表4-9 来自一个TREC语料和美国1996年专利语料的高频词

TREC语料		专利语料	
频率	短语	频率	短语
65824	united states	975362	present invention
61327	article type	191625	u.s. pat
33864	los angeles	147352	preferred embodiment
18062	hong kong	95097	carbon atoms
17788	north korea	87903	group consisting
17308	new york	81809	room temperature
15513	san diego	78458	seq id
15009	orange county	75850	brief description
12869	prime minister	66407	prior art
12799	first time	59828	perspective view
12067	soviet union	58724	first embodiment
10811	russian federation	56715	reaction mixture
9912	united nations	54619	detailed description
8127	southern california	54117	ethyl acetate
7640	south korea	52195	example 1
7620	end recording	52003	block diagram
7524	european union	46299	second embodiment

（续）

TREC语料		专利语料	
频率	短语	频率	短语
7436	south africa	41694	accompanying drawings
7362	san francisco	40554	output signal
7086	news conference	37911	first end
6792	city council	35827	second end
6348	middle east	34881	appended claims
6157	peace process	33947	distal end
5955	human rights	32338	cross-sectional view
5837	white house	30193	outer surface

在处理大规模数据集合并且要求快速响应时间的应用中，如网页搜索，在查询阶段测试词近邻可能太慢了。这样的话，就需要在文本处理阶段用一个更简单的定义识别短语：任何n个词的序列都构成一个短语。这就是所谓的n元串（n-gram）。两个词的序列称为二元串（bigram），三个词的序列称为三元串（trigram）。很多文本应用都是用到n-gram，本书也会频繁地提到这个术语，尤其是有关语言模型（language model）的部分（见7.3节）。在此关注词（word）n-gram。而字符（character）n-gram也会在一些应用中使用，如OCR，此时文本是含有噪声的，因此很难进行词匹配（见11.6节）。在对一些没有词分隔符的语言如汉语建索引时，也会使用字n-gram。产生n-gram（字符或词）的方法是，先选择一个n值，然后每次将窗口向前移动一个单元（字符或词）。换句话说，n-gram互相重叠。例如，词“tropical”包含如下字符的二元串：tr、ro、op、pi、ic、ca和al。基于n-gram的索引显然要比词索引大。

一个词n-gram出现越频繁，那么它越有可能对应一个有意义的短语。所有长度的n-gram构成一个齐普夫分布，一些常见短语出现非常频繁，大量的短语只出现一次。事实上，n-gram（包括单个词）的“排名-频率”数据比只有词本身更符合齐普夫分布。一些最常见的n-gram由停用词构成（如“and the”、“there is”），因此可以被忽略。但是和使用词时的情况一样，需要谨慎地丢弃信息。前面提到的查询例子“to be or not to be”无疑可以利用n-gram。可以对文档中一定长度内的所有n-gram建索引，然后在排序算法中使用它们。由于n-gram规模巨大，这似乎是对索引时间和磁盘空间的浪费。例如，一个1000个词构成的文档包含长度$2 \leqslant n \leqslant 5$的$n$-gram的数量为3 990。然而很多网络搜索引擎对n-gram建索引，原因是，它提供了一种在排序中快速融合短语特征的方法。

表4-10 Google n-gram样例统计信息

标记数：	1 024 908 267 229
句子数：	95 119 665 584
一元串数：	13 588 391
二元串数：	314 843 401
三元串数：	977 069 902
四元串数：	1 313 818 354
五元串数：	1 176 470 663

表4-10给出了一些样例的统计信息（Yang等，2007）。对网络上n-gram的分析发现，“all rights reserved”是英语中最频繁的三元串，而“有限责任 公司”则是汉语中最频繁的三元串。这两种现象都是由于网络上有大量公司站点，也说明了n-gram大多数并不是说话的通用模式，如“and will be”。

4.4 文档结构和标记

在数据库应用中，数据库记录的域或属性是搜索的关键部分。查询时提供这些域的取值范围。在某些文本应用如邮件或文献搜索中，作者（author）和时间（date）等域有相似的作用，因此成为查询的一部分。网络搜索中，查询通常与文档结构或域无关，但是这并不意味着这种结构不重要。从HTML标记得到的网页结构的有些部分，是排序算法用到的非常重要的特征。文档解析器必须识别这种结构并且在建索引时使用。

例如，图4-8显示了Wikipedia一条记录对应的网页的一部分[⊖]。这个网页有中含有排序算法可以使用的明显结构。页面的主标题是“tropical fish”，说明这个短语非常重要。这个短语在文本主体中使用粗体和斜体，进一步证明了它的重要性。其他被用作链接锚文本的词或者短语，则有可能是表示页面内容的好的词项。

Tropical fish

From Wikipedia, the free encyclopedia

Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species. Fishkeepers often use the term *tropical fish* to refer only those requiring fresh water, with saltwater tropical fish referred to as *marine fish*.

Tropical fish are popular aquarium fish , due to their often bright coloration. In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

图4-8 一个Wikipedia网页的一部分

这个网页的HTML源代码（见图4-9）表明应该表示更多的结构以帮助搜索。HTML中，每一个域或元素（element）都伴随着一个开始标记（如<hl>）和一个可选的结束标记（如</hl>）[⊖]。元素也可以有一些属性（包括属性值），以“属性名=属性值”给出。一个HTML文档的<head>元素包含的元数据不会被浏览器显示。关键词（<meta name="keywords"/>）对应的元数据元素给出一个词和短语列表，这个列表可以作为表达页面内容的额外的词项使用。Wikipedia中的这个列表是其他网页的标题。元数据元素<title>给出网页的标题（和主标题不同）。文档的<body>包含被浏览器显示的内容。<hl>指明主标题。其他不同大小、不同重要程度的标题由<h2>至<h6>指明。应该被显示成粗体或斜体的词项由<b>或<i>指出。不同于典型的数据库里的域，这些标签主要用于格式控制，并且可以在同一文档多次出现。如之前所述，这些标签也可以看作是一个词或短语重要性的标志。

像fish这样的链接很常见。它们是链接分析算法如PageRank（Brin & Page, 1998）的基础，同时也定义了锚文本。链接和锚文本对于网络搜索非常重要，将在下一节详细介绍。一个链接的title属性用来提供链接的额外信息，在这个例子中，它是关联的Wikipedia网页的URL末尾的词。网络搜索引擎也使用网页的URL作为一种额外的元数据。这个网页的URL是：

http://en.wikipedia.org/wiki/Tropical_fish

“tropical”和“fish”在URL中出现这个事实，增加了这两个词对网页的重要性。一个URL的

⊖ 网络百科全书：http://en.wikipedia.org/。

⊖ XML中，结束标记是必须的。

深度（网页在目录中的层数）也很重要。例如，URL为www.ibm.com的网页比对应如下URL的网页更有可能是IBM的主页：

www.pcworld.com/businesscenter/article/698/ibm_buys_apt!

```
<html>
<head>
<meta name="keywords" content="Tropical fish, Airstone, Albinism, Algae eater,
Aquarium, Aquarium fish feeder, Aquarium furniture, Aquascaping, Bath treatment
(fishkeeping),Berlin Method, Biotope" />
...
<title>Tropical fish - Wikipedia, the free encyclopedia</title>
</head>
<body>
 ...
<h1 class="firstHeading">Tropical fish</h1>
...
<p><b>Tropical fish</b> include <a href="/wiki/Fish" title="Fish">fish</a> found in <a
href="/wiki/Tropics" title="Tropics">tropical</a> environments around the world,
including both <a href="/wiki/Fresh_water" title="Fresh water">freshwater</a> and <a
href="/wiki/Sea_water" title="Sea water">salt water</a> species. <a
href="/wiki/Fishkeeping" title="Fishkeeping">Fishkeepers</a> often use the term
<i>tropical fish</i> to refer only those requiring fresh water, with saltwater tropical fish
referred to as <i><a href="/wiki/List_of_marine_aquarium_fish_species" title="List of
marine aquarium fish species">marine fish</a></i>.</p>
<p>Tropical fish are popular <a href="/wiki/Aquarium" title="Aquarium">aquarium</a>
fish , due to their often bright coloration. In freshwater fish, this coloration typically
derives from <a href="/wiki/Iridescence" title="Iridescence">iridescence</a>, while salt
water fish are generally <a href="/wiki/Pigment" title="Pigment">pigmented</a>.</p>
...
</body></html>
```

图4-9 Wikipedia示例网页的源代码

HTML预先定义了元素类型，对于所有文档都是一致的。相反，XML允许应用自定义元素类型，以及表示元素类型用到的标签。XML文档可以用schema来描述，类似于数据库模式。因此，XML的元素比HTML与数据的语义联系更紧密。搜索应用经常使用XML记录由信息抽取技术生成的文档的语义标注（semantic annotation），这将在4.6节中讨论。这些应用使用的文档解析器会记录这些标注以及其他文档结构，以便建索引时使用。

基于XML，数据库研究界定义了面向结构化数据搜索的查询语言XQuery[⊖]。XQuery允许查询时指定结构和内容限制，这就引起了争论：对于XML数据，数据库或信息检索方法比构建一个搜索引擎好？11.4节将仔细讨论这个问题。但是一般来讲，这与数据、应用和用户的需求有关。当XML数据中文本占很大部分的时候，信息检索方法更好些。第7章将会讨论针对包含结构化和元数据的文本文档设计的检索模型。

4.5 链接分析

将不同网页联系起来的链接是互联网的一个核心组成。在人们浏览网页的时候，链接提供了强大的导航作用，同时也帮助搜索引擎理解网页之间的关系。这种关系帮助搜索引擎更有效地对网页进行排序。但是需要了解的是，一些搜索应用如桌面搜索或企业搜索使用的很多数据库不包含或包含很少的链接结构。对于这类数据集合，链接分析对搜索效果没有影响。

⊖ http://www.w3.org/XML/Query/。

如上一节所见，一个网页链接在HTML被编码为：

For more information on this topic, please go to the somewhere page.

当在网页浏览器显示这个网页时，这些词“the somewhere page”会和一般的文本不同，比如带有下划线或者使用其他的颜色（或者两者都有）。点击那个链接，浏览器就会加载网页http://www.somewhere.com。在这个链接中，“the somewhere page”称为锚文本（anchor text），http://www.somewhere.com称为目标网页（destination）。这两部分在排序时都有用。

4.5.1 锚文本

锚文本的两个特性使其对网页排序非常有用。第一，它一般很短，可能有两个或三个词，并且这些词通常能够简洁地描述链出网页的主题。例如，指向www.ebay.com的链接很有可能在锚文本中包含“eBay”这个词。很多查询和锚文本很相似，它们也是对网页主题的简短描述。这就引出了一个非常简单的排序算法：搜索数据库中所有的链接，查找与用户查询完全匹配的锚文本。每匹配一次，就给目标网页的权重加1，网页排序时应该以权重下降的顺序。这个算法存在一些很严重的问题，其中之一是如何处理查询“click here”。总的来讲，链接的锚文本集合可以作为链出网页额外的文本属性，可以在排序算法中使用。

写锚文本的人一般不是目标网页的作者。这意味着锚文本可能是从另一个角度来描述目标网页，或者强调这个网页被某个群体视为最重要的特性。链接存在本身就说明了目标网页的重要性。虽然在网络搜索引擎讨论中，锚文本不像链接分析算法如PageRank那样经常被人提起，然而TREC评测结果显示，在一些类型的网页搜索中，它是网页表示中最重要的部分，特别当人们希望搜索一个特殊话题、人或机构的主页时，它必不可少。

4.5.2 PageRank

网页数量已达到数百亿，但其中大多数并不是很有趣。很多网页都是垃圾，不包含任何有用的内容。其他的网页则是个人日志、结婚公告或者家庭影集。这些网页对一小部分人来说是有趣的，但很可能并不普遍。另一方面，只有一些网页对很多人有用，受到人们的欢迎，包括新闻站点和流行公司的网站。

网络的巨大规模给搜索引擎造成了麻烦。假设一个朋友告诉你访问易趣的网站，而你不知道URL是www.ebay.com。你可能使用一个搜索引擎搜“eBay”，但是数百亿个网页中都包含“eBay”。搜索引擎如何选择最受欢迎（有可能就是正确的）的网页？一种非常有效的方法是，使用网页间的链接来衡量流行程度。最明显的衡量方法是统计每个网页的入链（inlink，指向这个网页的链接）数，作为排序算法的一个特征或证据。虽然这种方法证明很有效，但是却对垃圾信息很敏感。基于链接分析算法的衡量方法，需要保证网页排序更加可靠。在这些衡量方法中，PageRank最经常被提到。

PageRank基于随机冲浪（random surfer，像在网络中冲浪一样）的想法。想像一个名叫Alice的人正在使用她的网页浏览器。Alice很无聊，因此她没有目的地随便查看网页。她的浏览器顶部有一个特殊的“surprise me”按钮，当她点击这个按钮的时候，浏览器就会跳到一个随机的网页。每次加载完一个网页，她或者使用“surprise me”，或者点击这个网页中的某一个链接。如果她点击这个网页中的某一个链接，她不会优先选择任何一个链接，而是随机

选择一个。Alice非常无聊，以至于她会永远这样浏览网页[⊖]。

如果使用更有条理的方式解释，Alice使用如下算法浏览网络：

1）选择介于0和1之间的一个随机数r。

2）如果$r < \lambda$：点击“surprise me”。

3）如果$r \geqslant \lambda$：随机点击当前页面中的一个链接。

4）重新来一遍。

一般假定λ非常小，这样Alice点击链接的概率比点击“surprise me”的可能性大很多。虽然Alice浏览网页的路径是随机的，但她仍然更多地看到受欢迎的网页。原因是Alice通常受链接引导，而链接倾向于指向受欢迎的网页。因此，期望看到Alice访问大学站点的次数多于个人站点，但是少于CNN 站点。

假设CNN公布的一个故事中含有指向一个教授网页的链接。这时Alice也更有可能访问这个教授的网页，因为她频繁地访问CNN的网站。CNN中的一个链接比其他不那么受欢迎的站点中的数百个链接更能影响Alice的行为，因为Alice访问CNN的次数比那些其他站点多得多。

特殊的“surprise me”按钮可以保证Alice最终访问到互联网上所有的网页[⊖]。因为她计划长时间地浏览网页，并且网页数目是有限的，因此她会很多次地访问每个网页。然而有可能她访问一个受欢迎网站的次数比访问一个不受欢迎的网站多上千倍。注意，如果没有“surprise me”按钮，她有可能会停滞在一个没有链接的网页上，或者一个有链接但链接不指向其他网页的网页，又或者是一些构成循环链接的网页上。指向前两种网页的链接称为虚链(dangling link)。

假设当Alice正在浏览网页的时候，你恰好走进她的房间并且看到她屏幕上的网页。当你走进去时她正在看CNN网页的概率有多大，这个概率是CNN的PageRank。每一个互联网上的网页都有一个PageRank，这个值由网页的链接结构唯一决定。如这个例子所示，PageRank能够区分受欢迎的网页（有很多入链的网页，或者有来自受欢迎网页的入链的网页）和不受欢迎的网页。PageRank值可以帮助搜索引擎筛选包含“eBay”的数百万个网页，找到最受欢迎的那个（http://www.ebay.com）。

为了得到PageRank的合理估计，Alice需要点击成千上万亿次链接，因此不能依靠人得到PageRank。幸运的是，可以采用更有效率的方法计算PageRank。

假设目前网络上只有三个网页：A、B和C。又假设A链向B和C，B链向C，C链向A，如图4-10所示。

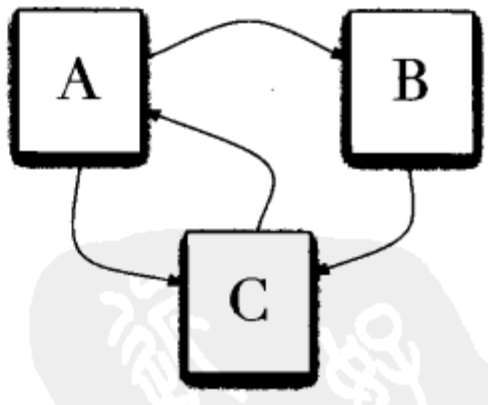


图4-10 一个只包含三个网页的“互联网”样例。箭头表示网页间的链接

页面C的PageRank，即Alice浏览这个网页的概率，依赖于A和B的PageRank。由于给定网页，Alice随机点击所有链接，如果她从A开始，那么有50%的可能她会进入C（由于A有2个链接）。换句话说，每个网页

⊖ PageRank计算过程对应于寻找在网络图中随机游动（random walk）的稳定概率分布。一个随机游动是马尔可夫链（Markov chain）的一个特例，下一个状态（下一个要访问的网页）只和当前状态（当前网页）相关。可能到达的网页由链接决定，并且概率相同。

⊖ “surprise me”按钮使这个随机游动模型称为一个遍历（ergodic）马尔可夫链，保证PageRank的迭代计算最终会收敛。

PageRank平均分配给所有的向外链接。如果忽略“surprise me”，意味着C的PageRank，表示为$PR(C)$，可以按下面公式计算：

$$PR(C)=\frac{PR(A)}{2}+\frac{PR(B)}{1}$$

更一般地，可以按下面公式计算任何网页u的PageRank：

$$PR(u)=\sum_{v\in B_u}\frac{PR(v)}{L_v}$$

其中B_u表示指向u的网页集合，L_v是网页v中包含的外向链接数（不考虑重复的链接）。

一个明显的问题是，不知道这些网页的PageRank，因为正在尝试得到它们。如果开始假设所有网页的PageRank是相同的（例子中的情况为1/3），那么很显然，可以通过多次迭代计算，比如，第一次迭代时，$PR(C)$=0.33/2+0.33=0.5，$PR(A)$=0.33，$PR(B)$=0.17。下一次迭代时，$PR(C)$=0.33/2+0.17=0.33，$PR(A)$=0.5，$PR(B)$=0.17。第三次迭代时，$PR(C)$=0.42，$PR(A)$=0.33，$PR(B)$=0.25。通过更多次迭代，PageRank值收敛到最终的$PR(C)$=0.4，$PR(A)$=0.4，$PR(B)$=0.2。

如果考虑“surprise me”按钮，页面C的PageRank的一部分来自于按这个按钮达到C的概率。假定当按这个按钮时，进入任何网页的概率为1/3，那么对C的PageRank的贡献为$\lambda/3$。这意味着现在C总共的PageRank为：

$$PR(C)=\frac{\lambda}{3}+(1-\lambda)\cdot\left(\frac{PR(A)}{2}+\frac{PR(B)}{1}\right)$$

类似地，一般的PageRank公式为：

$$PR(u)=\frac{\lambda}{N}+(1-\lambda)\cdot\sum_{v\in B_u}\frac{PR(v)}{L_v}$$

其中N是需要考虑的网页数。λ一般取0.15。

这个公式也可以表达为矩阵等式：

$$R=TR$$

其中R为PageRank向量，T为随机游走模型转移概率矩阵。元素T_{ij}表示从网页i进入网页j的概率，并且

$$T_{ij}=\frac{\lambda}{N}+(1-\lambda)\frac{1}{L_i}$$

如果熟悉线性代数的话，你可能知道R是矩阵T的特征向量（eigenvector）。

图4-11给出计算PageRank的一些伪代码。这个算法的输入是一个图G。图由顶点和边构成，因此$G=(V, E)$。在此，顶点表示网页，边表示链接，因此伪代码中使用字母P和L。使用（p, q）表示一个链接，其中p和q分别是源网页和目标网页。q不存在的链接为虚链，假设虚链都被删除。不含有出链的页面称为rank sinks，因为它们累积PageRank值却不分配出去。在这个算法中，假设这些页面存在指向集合中所有页面的链接。

第一步是猜测每个页面的PageRank。由于没有更好的信息，因此只是假设每个页面的PageRank相同。又由于所有网页的PageRank和为1，因此在输入向量I中每个页面的PageRank

赋为1/|P|。另一种方法使用一个和入链数有关的值，可能会使收敛更快。

```
1: procedure PageRank(G)
2:                ▷ G表示网络图，由顶点（网页）和边（链接）构成
3:     (P, L) ← G                          ▷ 将图分解为网页和链接
4:     I ←长度为|P|的向量                   ▷ 当前的PageRank估值
5:     R ←a vector of length |P|           ▷ 更好的PageRank估计结果
6:     for all 元素 I_i ∈ I do
7:         I_i ← 1/|P|                     ▷ 开始时每个网页的PageRank相同
8:     end for
9:     while R没有收敛do
10:        for all 元素 R_i ∈ R do
11:            R_i ← λ/|P|                 ▷ 随机选择每个网页的概率为λ/|P|
12:        end for
13:        for all 网页 p ∈ P do
14:            Q ←满足 (p, q) ∈ L并且 q ∈ P的网页p的集合
15:            if |Q| > 0 then
16:                for all 网页q ∈ Q do
17:                    R_q ← R_q +(1−λ)I_p/|Q|   ▷ 当前网页p的概率为 I_p
18:                end for
19:            else
20:                for all 网页 q ∈ P do
21:                    R_q ← R_q + (1 − λ)I_p/|P|
22:                end for
23:            end if
24:            I ← R                       ▷ 更新当前的PageRank估计值
25:        end for
26:    end while
27:    return R
28: end procedure
```

图4-11 PageRank迭代算法伪代码

每次迭代时，首先创建一个结果向量R，并且每个元素为$\lambda/|P|$。这是通过随机跳动方式进入每个网页的概率。下一步是计算通过点击链接进入一个网页的概率。对P中的每一个网页进行迭代。对每一个网页，获取正处在这个网页的估计概率I_p。从这个网页出发，用户随机跳动的可能性是λ，而点击某一个链接的概率为$1-\lambda$。共有$|Q|$个可选链接，因此跳到网页$q \in Q$的概率是$(1-\lambda)I_p/|Q|$。将元素Rq加上这个数。当没有向外链接时，假设用户随机跳动，所以，所有$|P|$个网页平均分配概率$(1-\lambda)I_p$。

简而言之，PageRank是使用查询无关的元数据提高网页搜索的排序效果的一个重要例子。网页的PageRank和具体的查询无关。搜索引擎使用PageRank优先选择PageRank较高的网页，而不是假设所有的网页满足查询的可能相同。然而在网页搜索中，PageRank并没有传统方法重要，它只是排序中使用的很多特征之一。但是，它往往对流行的查询影响最大，这是一个有用的性质。

与PageRank同一时期发展起来的链接分析的HITS[⊖]算法（Kleinberg，1999），同样很有

⊖ Hypertext Induced Topic Search.

影响。这个算法估计一个网页的内容的值（权威度，authority value）和链向其他网页的值(hub value)。和PageRank类似，算法只利用链接结构信息来迭代计算这两个值。与PageRank不同的是，HITS算法只对给定查询检索出来的网页子集合计算这两个值[⊖]。从对排序效果的影响来看，这是HITS算法的优势，但是如果搜索引擎的查询量很大时，计算量便太大了。

4.5.3 链接质量

众所周知，PageRank和锚文本抽取等技术已经在商业搜索引擎中使用，因此肆无忌惮的网页设计者便尝试创建无用的链接，以提高他们网页在搜索结果中的位置。这叫做链接垃圾(link spam)。然而甚至典型的用户也能无意识地愚弄简单的搜索引擎技术。对此有一个与博客有关的明显例子。

很多博客帖子都是评价其他博客的帖子。假设作者*A*读了作者*B*的博客上的一篇名叫*b*的帖子。作者*A*可能写了一个新的帖子，称为*a*，其中包含了指向帖子*b*的链接。在写帖子的过程中，作者*A*可能对作者*B*博客中的帖子*b*中发布了一个反向引用（trackback)。反向引用是一种特殊的评论，它可以通知作者*B*，有一个回复被发布在作者*A*的博客上。

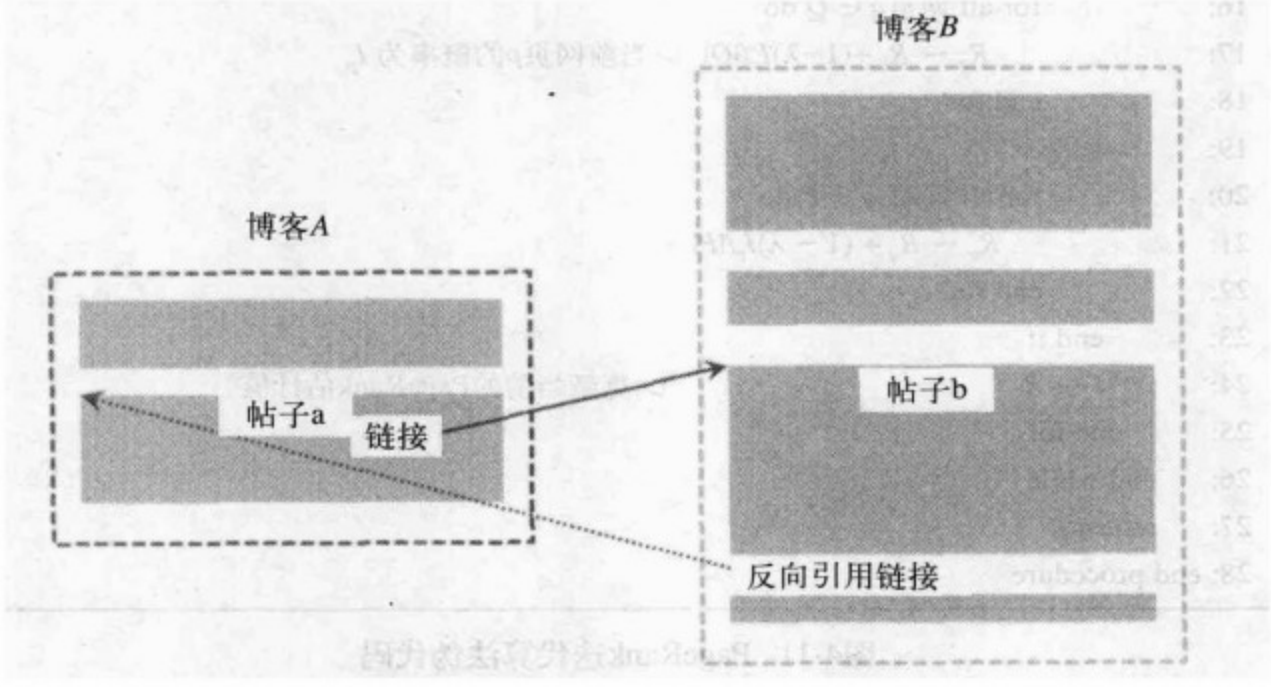


图4-12 博客发帖过程中的反向引用

如图4-12所示，帖子*a*和*b*产生了一个循环。帖子*a*链向帖子*b*，而帖子*b*中又包含一个指向帖子*a*的反向引用。直觉上会认为帖子*b*是有影响力的，因为作者*A*决定讨论它。但是从PageRank的角度，*a*和*b*彼此链接，因此两个有同样的影响力。这儿的问题是，反向引用和其他出现在帖子中的链接在本质上不相同。

博客的评论部分也可以是链接垃圾的来源。网页作者为了增强他们的网页，也许会在比较流行的博客的评论部分，发布指向他们网页的链接。基于对PageRank的讨论，可以获知一个流行网站包含的链接可以让另一个网站似乎更加重要。因此垃圾制造者们很喜欢这种评价部分。

面对这种情况，搜索引擎公司的一个解决方法是，自动地发现这些评价部分，并且在建索引时有效地忽略其中链接。另一种更简单的办法是，要求网站拥有者将不重要的链接标记

⊖ 与查询无关的HITS算法和与话题相关的PageRank也已经被定义了。

出来，以便被搜索引擎识别。“rel=nofollow”链接属性便来源于此。

目前大部分博客软件都被设计成让博客评论部分的链接包含"rel=nofollow"属性。因此，下面的帖子：

```
Come visit my <a href="http://www.page.com">web page</a>.
```

变成如下：

```
Come visit my <a rel=nofollow href="http://www.page.com">web page</a>.
```

这个链接会在博客中显示出来，但是搜索引擎会忽略所有标记了"rel=nofollow"的链接，这样可以帮助更好地计算PageRank和抽取锚文本。

4.6 信息抽取

信息抽取是一种从文本中抽取结构化信息的语言技术。很多应用都使用信息抽取，尤其是文本数据挖掘（text data mining）。对于搜索应用而言，信息抽取的最主要用途是，识别搜索引擎用来提高排序效果的特征。一些人推测信息抽取技术最终会做到从文本中抽取重要信息，存储到结构化的表格中，从而将文本搜索问题转变成数据库问题。但是目前的技术距离这个目标还很远。

已经讨论过的一些文本处理过程可以认为是信息抽取，比如识别名词短语、标题甚至变成粗体的文本。在这些例子中，一部分文本被识别出有某种特性，而这种特性可以用一种标记语言来描述，如XML。如果一个文档已经使用了HTML或XML来描述，那么识别一些结构特征（如标题）就很直接。但是其他的特征，如短语，在使用标记语言标注（annotate）之前需要额外的处理。在一些应用中，比如文档集合通过OCR（光字符阅读器）输入，这些文档不包含标记，因此最简单的结构如标题，也需要识别和标注。

这些类型的特征非常普遍，但是目前信息抽取大多数关注于研究具有特殊语义内容的特征，如命名实体（named entity）、关系（relationship）和事件（event）。虽然所有这些特征都包含重要的信息，但搜索应用最经常使用的是命名实体识别（named entity recognition）。一个命名实体是表达特定应用感兴趣的某一个事物的词或词序列。最一般的例子是人名、公司或机构名、地名、时间和日期表达式、数量和货币值。很容易想到其他对某些特定应用有意义的实体。例如，对于一个电子商务应用，在网页和评论中识别产品名和型号是很必要的。在一个医药应用中，识别药名、剂量和医疗条件可能很重要。识别和标注这些特殊的特征，有时称为语义标注（semantic annotation）。一些识别出的实体通过小平面技术（facet）直接应用于搜索中（见第6章），而其他的实体则作为浏览搜索结果的一部分使用。一个后者的例子是搜索引擎可以识别网页中的地址，并提供指向相应信息的链接。

```
Fred Smith, who lives at 10 Water Street, Springfield, MA, is a long-time
collector of tropical fish.

<p ><PersonName><GivenName>Fred</GivenName> <Sn>Smith</Sn>
</PersonName>, who lives at <address><Street >10 Water Street</Street>,
<City>Springfield</City>, <State>MA</State></address>, is a long-time
collector of <b>tropical fish.</b></p>
```

图4-13 信息抽取标记的文本

图4-13给出一句话以及它对应的信息抽取后的XML标注结果。这个例子中，抽取工作通过一个很有名的字处理程序完成[⊖]。除了通常的结构标记（<p>和<b>），很多其他的标记被增加进来，以说明哪些词是命名实体的一部分。例如在这个例子中，文本中识别出一个地址，包括一个街道（“10 Water Street”）、一个城市（“Springfield”）和一个州（“MA”）。

建立命名实体识别器主要有两种方法：基于规则的方法和基于统计的方法。一个基于规则的识别器使用一个或多个词典（lexicon，词和短语列表）对名字分类。例子如地名（如镇子、城市、州、国家、名胜古迹）、人名（名、姓）和组织名（如公司、政府机构、国际组织）。如果这些列表范围足够广泛，很多抽取可以简单地通过查找实现。然而在很多情况下，规则和模板被用来核对一个实体名，或者发现不在列表中的新实体。比如，一个模板如“<number> <word> street”可以用来识别街道地址。假如一个名字在地名词典中作为城市名出现，模板如“<street address>、<city>”或“in <city>”，可以用来确认它确实是一个城市名。类似地，一个模板如“<street address>、<city>、<state>”，也可以用来识别不在词典中出现的新城镇。新任命可以通过规则如“<title> <name>”识别，其中<title>可能包含如“President”、“Mr.”和“CEO”的词。在区分大小写的文本中，抽取名字一般简单一些，因为名字一般都大写。然而很多应用都会将文本转化为全部大写或全部小写。使用模板的规则通过人工得到，经常需要反复试验不断摸索，然而可以使用一个最初的规则集合作为种子（seed），然后使用一个自动的学习过程发现新的规则[⊜]。

一个统计的命名实体识别器对实体内部和周围的词建立统计模型，使用人工标注的文本语料训练这个模型。建立这种模型存在很多不同的方法，但是由于其重要性，在此简单介绍隐马尔可夫模型（Hidden Markov Model，HMM）方法。HMM被用在很多语音和语言处理应用中。比如，词性标注器就可以使用这个方法实现。

基于隐马尔可夫模型的信息抽取

信息抽取最困难的部分之一，是词语可以有很多不同的意思。比如，“Bush”可以表达一种植物或一个人。同样，“Marathon”可以是比赛的名字或位于希腊的一个地方名。基于这个词的上下文（即周围的词），人们可以分辨出这些不同的意思。比如，如果“Marathon”前面的词是“Boston”，它的意思几乎肯定是比赛名。可以使用一个具有马尔可夫性质（Markov property）的过程，来对文本中的词序列的生成（generation）[⊝]建模，从而形式化描述词的上下文。马尔可夫性质意味着，序列中下一个词的生成只依赖于前面较少的几个词。

更正式地说，一个马尔可夫模型（Markov Model）使用状态（state）集合和状态间的转移（transition）来描述一个过程。每一个转移都关联一个概率。过程中的下一个状态只取决于当前状态和转移概率。在隐马尔可夫模型中，每一个状态产生一个可能的输出集合，每一个输出也有一个与转移相关的概率。

图4-14给出一个状态图（state diagram），表示一个命名实体识别器可以使用的非常简单的句子生成模型。这个模型中，假定句子中的词要么是一个命名实体的一部分（这个例子中是人名、结构名或地名），要么不是。每一个实体类别使用一个状态表示，每个词之后，系统

⊖ Microsoft Word。

⊜ GATE (http://gate.ac.uk)是一个开源的工具包，它提供了一个信息抽取架构和一个可以针对特定应用定制抽取过程的环境。

⊝ 第7章将详细介绍生成模型（generative model）。

可能停留在当前状态（环状箭头表示），或者转移到另一个状态。有两个特殊的状态分别表示句子的开始和结束。每一个状态表示一个实体类别，与每一个状态相关联的是一个属于这个类别的词序列的概率分布。

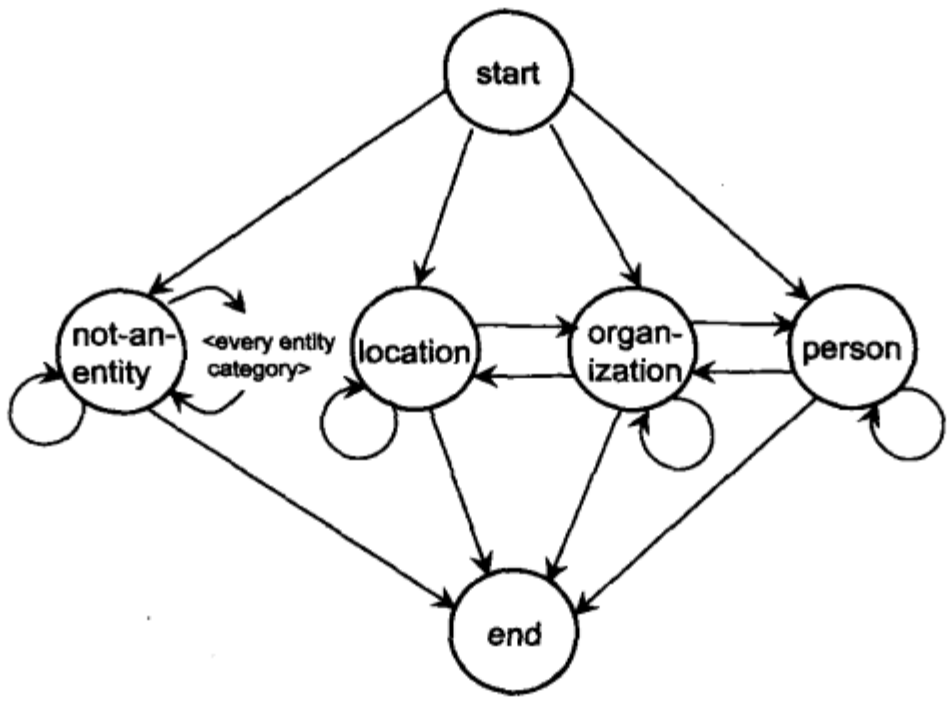


图4-14 统计实体抽取器的句子模型

这个模型的一个可能应用是构建新的句子。假设从开始状态出发，根据开始状态的转移概率表随机选择下一个状态。例如，可能转移到对应人名的状态。一旦进入对应人名的状态，则根据人名状态的输出概率分布选择一个输出。假设输出为“Thomas”，这个过程一直继续，每一步转移到一个新的状态，然后产生一个输出。最终的结果是一个状态集合和它们对应的输出。

虽然这样的模型可以用来产生新的句子，但它更经常地用来识别句子中的实体。方法是，给定一个句子，找到一个实体类别序列，使得产生这个句子的概率最大。只有状态转移时产生的输出是可见的（可以被观察到）；潜在的状态是隐藏的（“hidden”）。例如，对于图4-13这种的句子，识别器将找到当状态序列为

```
<start><name><not-an-entity><location><not-an-entity><end>
```

时，对于那个模型，其概率最高。与序列中的实体类别相关联的词将会被标记出来。Viterbi算法（Viterbi algorithm）[⊖]可以在HMM模型中找到最大概率的状态序列，它属于一种动态规划算法。

使用这种方法识别命名实体的核心问题是，句子模型中的概率必须从训练数据估计出来。为了估计转移和输出概率，使用人工标注了正确实体标签的文本作为训练数据。从这个训练数据，可以直接估计出某个类别产生词的概率（输出概率）和类别之间的转移概率。为了建立更准确的识别器，模型包含和命名实体紧密联系的特征，如大写的词和由数字组成的词。另外，转移概率同时依赖于前一个词和前一个类别[⊖]。例如，词“Mr.”的出现，增加了下一个类别是“人名（Person）”的概率。

虽然这样的训练数据在构建准确的HMM时很有用，但是构建这样的数据需要很大的人力。为了准确估计模型参数，需要标注大约一百万词的数据，相当于1 500多个新闻报道。比起为

⊖ 为了纪念电气工程师Andrew Viterbi而命名。

⊖ Bikel等（1997）描述了第一个基于HMM方法的命名实体识别器。

了抽取简单特征而构建规则集合而言，这将需要多很多的工作。基于规则和基于统计的方法，对人名、机构名和地名的识别准确率都可以达到约90%[㊀]，然而，基于统计的方法一般要更好。其他类别的实体，如产品名，识别起来要困难得多。选择哪种命名实体识别方法取决于实际应用、可用的软件和有多少标注者。

有趣的是，没有证据证明命名实体对通用搜索引擎有帮助。命名实体识别是问答系统中很关键的部分，对于特定领域或者垂直搜索引擎中的术语识别和索引，也可能很重要。在一些应用，如本地搜索引擎、分析查询时，或者作为一种理解和浏览搜索结果的工具时，命名实体识别也会很有用。

4.7 国际化

全世界都在使用互联网，并非只有英语使用者在使用。尽管目前有65%～70%的网页采用了英语，但这个比例一直在下降。超过一半的网络使用者，在搜索网页的时候不把英语作为主要的语言。其他的应用如桌面搜索和企业搜索，每天都在很多语言中使用。即使一个针对大多数使用者讲英语而设计的软件，在其数据库中也可能有很多非英语文档。尝试在你最喜爱的网络搜索引擎中使用“poissons tropicaux”（tropical fish）查询，看看检索出了多少法语网页[㊁]。

一个单语（monolingual）搜索引擎，如它的名字所隐含的那样，是指一个针对特定语言设计的搜索引擎[㊂]。本书中讨论的很多索引技术和检索模型，对于任何语言都适用，各种语言的差异对搜索引擎设计影响最大的，是为搜索建立索引项的文本处理过程。

如前面的章节所述，字符编码是搜索引擎处理非英语语言时的核心问题。Unicode成为软件国际化中一种主导的字符编码标准。

其他文本处理步骤也需要根据不同语言而改变。已经讨论过词干提取对于变形繁多的语言很重要，但是每种语言需要不同的词干提取器。词素切分对于很多语言也很重要，尤其是CJK家族。对于这些语言，核心问题是分词（word segmentation），需要在连续的字符串中识别出词或者索引项的分割点（一般没有空白符）。作为分词的另一种选择，有人对相互重叠的字二元组建索引。图4-15中给出了文本“旱灾在中国造成的影响”的分词和二元组结果。虽然基于二元组的搜索效果很不错，但很多应用还是优先采用分词，因为很多二元组并不对应真正的词。如果有足够的训练数据，可以使用统计方法实现分词技术，如隐马尔科夫模型。在其他语言中，分词也是一个问题。例如，德语中有很多复合词（如“fischzuchttechniken”表示“fish farming techniques”），需要被切分然后建索引。

1. 原始文本
旱灾在中国造成的影响
(the impact of droughts in China)
2. 分词
旱灾 在 中国 造成 的 影响
drought at china make impact
3. 二元组
旱灾 灾在 在中 中国 国造
造成 成的 的影 影响

图4-15 汉语分词和二元组

总的来讲，如果有了这些工具，对这些主要的语言建立一个搜索引擎并不困难。对于在

㊀ 意思是，识别出来的10个实体中有9个是对的，并且10个已知实体中有9个被识别出来。衡量指标的细节参见第8章。

㊁ 当然，如果使用一个法语版本的搜索引擎如http://fr.yahoo.com，找到的法语网页会多得多。

㊂ 将在6.4节讨论跨语言搜索引擎。

网络上有足够多在线文本的任何语言都是如此，因为这些文本可以用来建立和测试搜索引擎的组成部分。然而对于很多所谓“低密度”的语言而言，也许有很多人讲这个语言，但是在线的资源却很少，对这些语言建立有效的搜索引擎是一个挑战。

参考文献和深入阅读

文本和文档集合的性质和统计已经研究了很长时间，这种研究称为文献计量(bibliometrics)，作为图书馆和情报科学（library and information science）的一部分。情报科学杂志，如Journal of the American Society of Information Science and Technology（JASIST）或Information Processing and Management（IPM），包括很多这个领域的论文。信息检索从一开始就强调文本的统计特性，并且信息检索和情报科学的研究者们一直并肩工作。Belew（2000）从认知的角度，很好地讨论了齐普夫法则和文本的其他性质与IR的关系。随着1990年开始转移到统计方法，自然语言处理（natural language processing）领域的研究者也开始对研究文本的统计特性感兴趣。Manning和Sch ü tze（1999）总结了从这个角度观察得到的文本统计性质。Ha、Sicilia-Garcia、Ming和Smith（2002）给出一个有趣的结果，表明短语（或n-gram）一般也遵守齐普夫法则，并且将短语和词结合，可以更好地预测排名靠前时的频率。

Anagnostopoulos、Broder和Carmel（2005）的论文给出了估计查询结果集合大小的方法，也指出了很多这个领域的相关文献。类似地，Broder等人（2006）给出如何估计语料规模，并且跟以前的方法做了比较。

关于词素切分或者停用词去除的论文不多，它们都被默认为众所周知，因此论文中一般不提及。但是正如本章指出的那样，做好这些基本的步骤对于整个系统的效果很关键。很多年来，研究者都使用Van Rijsbergen（1979）公布的停用词词表。当它显然无法满足更大规模的TREC集合时，由马萨诸塞大学开发并随Lemur工具包一起发布的一个停用词词表被频繁使用。如前面提到的，这个词表包含400多个词。对于很多应用来讲，这个词表太长了。

最初描述Porter stemmer的论文写于1979年，再版于Porter（1997）。Krovetz（1993）的论文描述了他的词干提取算法，但是也采用了更深入的方法研究形态学对于词干提取器的作用[⊖]。Krovetz stemmer可以在Lemur网站上获取。其他语言的词干提取器可以从很多不同网站上获取（包括Lemur和Porter stemmer网站）。可以在Larkey、Ballesteros和Connell（2002）中找到阿拉伯语词干提取的方法描述。

在搜索中使用短语的研究有很久的历史。Croft、Turtle和Lewis（1991）描述了使用句法和统计方法处理查询，从而获取短语的检索实验，结果显示其效果和手动选择短语效果接近。很多参加TREC评测的小组使用短语作为搜索算法的一部分（Voorhees和Harman，2005）。

Church（1998）描述了一种建立统计（或随机）词性标注器的方法，成为目前很多标注器的基石。这个方法使用人工标注的数据训练出一个词性序列概率模型及一种词性对特定词语的概率。给定一个句子，采用使整个句子概率最高的词性标注序列作为结果。这个方法本质上和基于统计的实体抽取是相同的，只是将状态的含义从实体类别变成词性。Brill标注器（Brill，1994）是另一种比较流行的自动从标注数据学习规则的方法。Manning和Sch ü tze（1999）提供了关于词性标注方法的好的综述。

⊖ 形态学（Morphology）研究词的内部结构，而词干提取是一种词素处理（morphological processing）。

在文献中可以看到很多PageRank的变形。其中很多都是改进设计以提高计算效率或者使用在不同的应用中。与话题相关的PageRank版本在Haveliwala（2002）中有所介绍。PageRank和HITS都来源于文献计量学中提出的引用分析算法。

为了增强一个超文本文档（如网页）的表示，可以使用链接指向这个文档的其他文档的内容，这种思想已经出现一段时间了。例如，Croft和Turtle（1989）提出了一种基于合并相关超文本文档的文本的检索模型，而Dunlop和vanRijsbergen提出检索包含少量文本内容的文档（比如包含图像的文档），可以利用这个文档链接的所有文档中包含的文本。McBryan（1994）首先提出只利用与链接相关联的锚文本的方法。锚文本被证明对TREC评测中一些类型的网络搜索很关键，如Ogilvie和Callan（2003）。

一些在没有明显链接结构的数据库中应用链接分析的方法被提出（Kurland和Lee，2005）。在这种情况下，链接是基于文档内容的相似度定义的，通过一些相似度衡量方法如余弦相似度计算得到（见第7章）。

信息抽取技术的发展主要由一些研究项目如TIPSTER和MUC推动（Cowie和Lehnert，1996）。使用命名实体抽取为搜索提供额外特征的研究工作，早期主要在TREC评测中开展（Callan等，1992；Callan，Croft和Broglio，1995）。最有名的基于规则的信息抽取系统之一是FASTUS（Hobbs等，1997）。BBN的系统Identifier（Bikel、Schwartz和Weischedel，1999）基于统计模型，已经在很多项目中使用。McCallum（2005）给出了一个信息抽取研究的综述，参考了这个领域的最新进展。

可以在Wikipedia中找到所有主流编码体系的详细介绍。Fujii和Croft（1993）是最早讨论针对CJK语言搜索的文本处理问题的论文之一。ACM Transactions on Asian Language Information Processing[⊖]整个期刊一直都关注这个问题。Peng、Feng和McCallum（2004）描述了一个汉语分词的统计模型，并且给出了其他方法的参考文献。

练习

4.1 对Wikipedia数据集上的词和词二元串画出“排名–频率”曲线（使用log-log图）。画出两者结合的曲线。对于每条曲线，什么是参数c的最优值？

4.2 画出Wikipedia数据集的词汇增长曲线并估计Heap's Law的参数。处理文档的数据会让结果产生变化吗？

4.3 使用本章介绍的方法估计两个不同搜索引擎索引的网页数。使用多个查询进行多次估计，比较并讨论这些估计值的一致性（或不一致性）。

4.4 修改Galago词素切分器，使用另一种方式处理撇号和句点。描述你的词素切分器实现的新的规则。给出一些这个解析器效果好或者效果不好的例子（在你看来）。

4.5 检查Lemur中的停用词词表，找出你认为可能会产生问题的10个词，并且给出这些问题的例子。

4.6 使用Porter stemmer和Krovetz stemmer处理Wikipedia文档。比较分别产生的词干数量，找出它们结果中10个不同的地方，并且找出这种差异会对排序产生影响。

4.7 使用GATE词性标注器标注一个Wikipedia文档。定义一个或多个规则识别短语，并且列出10个最高频的短语，然后对Wikipedia的查询进行词性标注。这些识别出的短语有什么

⊖ http://talip.acm.org/。

问题么？

4.8 找出Wikipedia中拥有最多指向链接的10个文档。列出这些文档对应的锚文本集合。

4.9 计算Wikipedia文档的PageRank。列出PageRank最高的文档及其PageRank值。

4.10 图4-11给出了一个计算PageRank的算法。证明每次算法运行到第9行的循环时，向量I元素和为1。

4.11 实现一个基于规则的城市名识别器（可以选择城市名的一个子集来简化问题）。创建一个测试集合，通过人工浏览找到所有的城市，然后测试识别器。总结识别器的性能并且对错误识别的现象进行讨论。

4.12 利用网页创建一个非英语的小测试集合。使用网上可用的工具对它进行基本的文本处理：词素切分、词干提取和停用词去除。举例说明文档的索引项表示。

第5章　基于索引的相关排序

“必须再快点儿。”

——David Levinson，《独立日》

5.1　概述

由于这是一本技术书籍，所以如果你已经读到了这里，说明你可以理解一些关于数据结构（data structure）的事情，并且知道如何在程序中使用它们。如果你想存储一个项目的列表，那么链表或者数组是好的选择。如果你想根据一个属性快速地找到一个项目，那么哈希表是一个比较好的选择。更复杂的任务需要更复杂的结构，比如B树或者优先队列等。

为什么需要这些数据结构呢？严格来讲，并不需要它们。需要计算机做的大部分事情都可以只使用数组完成。然而，数组有一些缺点：没有排序的数组查找很慢，排序数组插入很慢。相反，哈希表和树对于查找和插入都很快。这些数据结构比数组要复杂，但是速度上的差别非常明显。

文本搜索与传统的计算任务有很大不同，所以它调用自己的数据结构：倒排索引(inverted index)。倒排索引是对许多不同种类数据结构的一个概括性词汇，它们具有一些相同的原理。我们很快将要看到，这个特殊的数据结构将用于相关排序函数。然而，将文档进行相关排序的相关排序函数都具有相似的形式，最有用的倒排索引几乎存在于所有搜索引擎中。

本章介绍计算机如何真正地处理搜索引擎的查询，所以本章也可以叫做“查询处理”(query processing)。本章的最后一节正是以此为题目，并且所给出的查询处理算法也是基于本章前面部分介绍的数据结构。

高效的查询处理对于网络搜索是特别重要的话题，它能处理的数据量是10年前很难想像的。每天，全世界的人们输入超过5亿的查询，在由几十亿的网页构成的索引上进行搜索，倒排索引是所有现代网络搜索引擎的核心。

搜索引擎各个独立的部分之间有很强的依赖关系。查询处理算法依赖于检索模型、指定索引的内容。反过来讲，不可能选择没有有效的查询处理算法的检索模型。我们要到第7章才详细讨论检索模型，本章首先描述一个抽象的相关排序模型，该模型引发了对索引的选择，然后是本章的四个主要部分。在第一部分，讨论不同类型的倒排索引以及各个不同索引能够获得关于文本的什么样的信息。第二部分给出压缩技术的概述，它在文本检索中是实现高效倒排索引的关键。本章的第三部分描述索引是如何构建的，包括对MapReduce框架的讨论，它能够用于非常大的文本集合。本章的最后一部分集中讨论索引如何被用来生成反映查询的文档相关排序。

5.2　抽象的相关排序模型

在介绍如何构建用于搜索系统的索引之前，我们首先考虑一个抽象的相关排序模型。本章要考虑的所有方法，都可以看作是对该模型的实现。

图5-1显示了模型的基本部分。图的左边是一个示例文档。文档由自然语言写成，很难由计算机直接分析。所以，例如在第4章看到文本被转换成索引项（index term）或者文档特征（document feature）。在本章，一个文档特征是被数值化的一些文档的属性。在图5-1中，展示了两种特征。在图的上面，有主题（topical）特征，它是对该文档属于某一主题的估计。在图的下面，有两个可能的文档质量（quality）特征。一个特征是链接到该文档的网页数量，另一个是这个页面上次更新至今的天数。这些特征不能指出该文档是否和一个查询相匹配，但是它们指出了它的质量：一个没有入链接且多年没有被编辑的页面，很可能和任何查询都不匹配。这些特征值都是用特征函数（feature function）产生的，特征函数就是一个数学表达式，它从文档文本中生成一些数值。在第4章讨论了一些重要的主题和质量特征，第7章将介绍生成较好的特征函数的方法。本章假设已经产生合理的特征值。

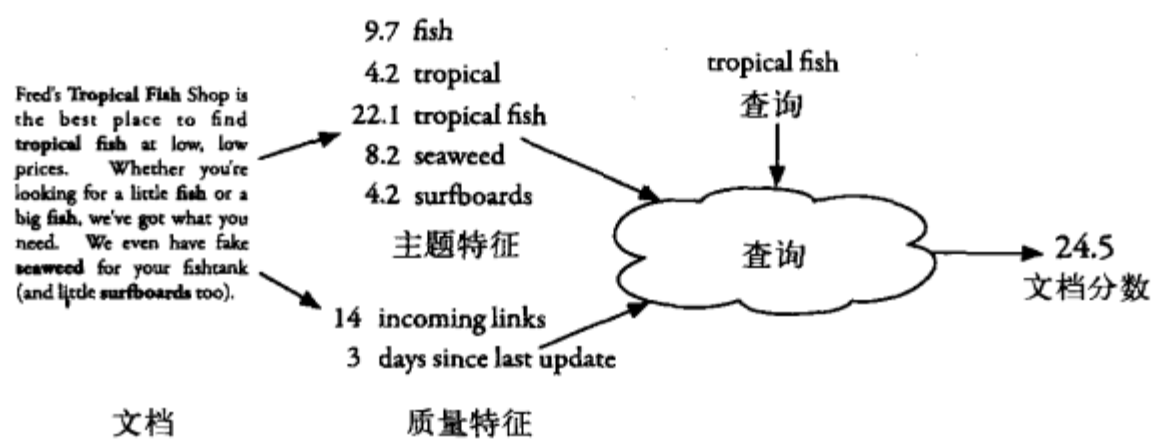


图5-1 抽象相关排序模型的组成：文档、特征、查询、检索函数及文档分数

图的右侧用云形来表示相关排序函数（ranking function），该函数通过合并查询和文档特征数据产生一个分数。目前，云的内容并不重要，只需知道大多数合理的相关排序函数忽略了许多文档特征，它们只关心一个与查询有关的较小的子集。该事实使得倒排索引成为搜索中一个引人注目的数据结构。

相关排序函数最终的输出是一个分数，我们假设是某个实数。如果一个文档获得较高的分数，这意味着系统认为该文档与查询匹配得很好，而一个较低的分数意味着系统认为该文档与查询匹配得不好。为了构造一个排好序的结果列表，文档需要根据它们的分数进行排序，以便让分数最高的文档最先出现。

假设有一个搜索引擎，正试图以适当的顺序对文档进行相关排序，以响应一个用户的查询，可能你会将文档分成堆，如："好"、"一般"和"坏"。计算机本质上正通过相关排序做着与此相同的事情。然而，你可能也会打破这个约束，仔细看看每篇文档，从而决定哪一个更相关。遗憾的是，获得文档的深层含义对计算机来说是非常难以做到的，因此搜索引擎致力于确定好的特征以及基于这些特征的相关排序。

一个更具体的相关排序模型

本章接下来的部分将介绍查询评价技术，该技术假设一些东西和相关排序特别相关。特别地，假设相关排序函数R是下面的形式：

$$R(Q, D) = \sum_i g_i(Q) f_i(D)$$

其中，f_i是某个特征函数，它从文档文本中获得一个数值。g_i是一个相似的特征函数，它从查

询中获得一个值。这两个函数形成一个特征函数对。每个函数对被乘在一起，所有对的乘积结果加起来，形成最终的文档分数。

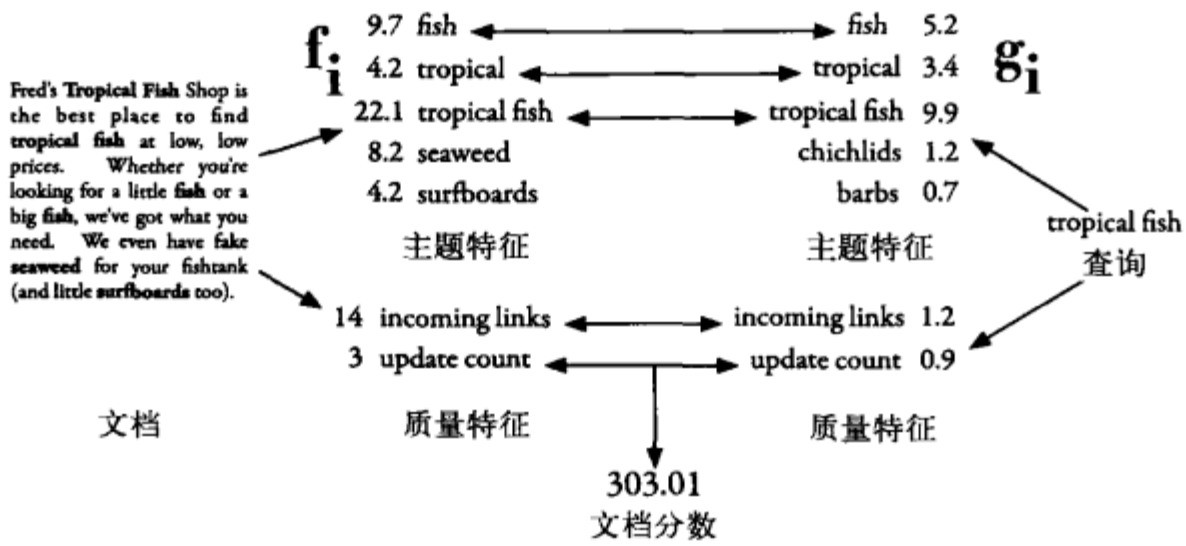


图5-2 更具体的相关排序模型。注意此模型中查询和文档都具有特征函数

图5-2展示了该模型的一个实例。与抽象相关排序模型一样，各种特征被从文档中提取出来。这个图只显示了一部分特征，但是实际中会有更多，这些都反映到公式中的$f_i(D)$函数上。可以简单地称它们为$f_{tropical}(D)$或者$f_{fish}(D)$。对于含有较多或者凸显“tropical”或者“fish”的文档，这些值将变大。

该文档有一些非主题特征。对于该示例文档，我们看到搜索引擎注意到它曾被更新3次，有14个入链接。虽然这些特征没有告诉我们任何关于该文档和一个查询的主题是否相匹配的信息，但是它们确实给了一些关于文档质量的建议。既然知道它偶尔被更新，就说明它不是刚刚被发布到网上的、而且也没有被废弃。我们也知道有14个其他的网页有指向它的链接，这可能意味着在它上面有一些有用的信息。

注意，也有一些应用于查询的特征函数。因为查询含有“tropical”，所以特征函数$g_{tropical}(Q)$获得一个较大的值。然而，$g_{barbs}(Q)$也有一个较小的非零值，因为它和查询中的其他词项相关。这些来自查询特征函数的值与文档特征函数的值相乘，然后求和，获得一个文档分数。

查询也有一些非主题的特征值，例如更新次数特征。当然，这并不意味着查询被更新了。这个特征值指出了文档更新对于与该查询相关有多重要。例如，如果一个查询是“today's weather in london”，我们可能更喜欢那些经常更新的文档，因为一个不是每天更新的文档，不可能对今天的天气信息有任何帮助。该查询应该对更新次数特征赋一个较高的值。相反，一个从不变化的文档，可能与查询“full text of moby dick”很相关。这个查询对于更新次数可以有一个较低的特征值。

如果一个检索系统每天不得不对超过成百上千万的特征进行求和，文本搜索系统可能不会很实用。实际上，查询特征（$g_i(Q)$）大多数为0。这意味着每个文档的求和运算只要在值非零的$g_i(Q)$上进行就可以了。

5.3 倒排索引

所有现代搜索引擎的索引都是基于倒排索引（inverted index）的。曾经也使用过其他的索引结构，最著名的是签名文件（signature file）⊖，但是倒排索引被认为是最有效、最灵活

⊖ 签名是一个位序列，它是一个文本（或者文档）块的简明表述，和在第3章中讨论的指纹类似。对文本块中的每个词，使用一个哈希函数以在签名中一些特殊的位置上置1。

的索引结构。

倒排索引和本书后面的索引相当。可以看一下本书的索引作为例子。书籍的索引将索引项（index term）按照字母表顺序排序。每一个索引项跟着一个和该项有关的页码列表。例如，如果想知道更多关于“stemming”的信息，可以在索引中查找以“s”开头的项，接着可以仔细查找每个词条，直到找到“stemming”为止。那里的页码列表可以把你带到第4章。

相似地，倒排索引通过索引项组织。通常认为单词是文档的一部分，但是如果转变这个想法，认为文档是附着在单词上的，这样的索引就被称作倒排（inverted）了。和传统书本类似，索引项经常也是按照字母顺序排列，但这不是必须的，因为经常可以使用哈希表来查找它们。每个索引项有其自身的倒排列表（inverted list），它们含有和该项相关的数据。在书本的索引中，相关数据是页码列表。在搜索引擎中，数据可能是文档列表或者单词所在位置的列表。列表的每个项被称作posting，在posting中，指向特定文档或位置的部分被称作指针(pointer)。文档集中每个文档被赋予一个唯一的编号，这样可以有效地存储文档指针。

书本中的索引可以存储除位置信息之外的更多信息。对于重要的单词，经常使用粗体标识其中一个页码，以指明此页包含关于该项的定义或者展开的讨论。倒排文件也能有扩展信息，此时posting能够包括除了位置之外的其他一系列信息。通过在每个posting中存储适当的信息，能够有效地计算出在上一节中介绍的特征函数。

最后，习惯上，书本索引的页码按照升序打印，以便最小的页码最先出现。传统地，倒排表也使用相同的方式存储。这些排序文档（document-orderd）列表以文档编号排列，这使得某些种类的查询处理更加有效，同时也提高了倒排表压缩率。然而，有些倒排文件也可以以其他顺序排列。

与倒排文件不同，其他索引通常有或多或少的缺点。例如，签名文件将文档集中的每个文档表示为一些比特位的集合。为了搜索一个签名文件，查询也被转化为签名的比特位的模式，然后进行比较。一般地，对于一个查询，所有签名必须都被扫描一遍，即使索引被紧密地编了码，这仍然是一个繁重的处理任务。倒排文档的优点是对大多数查询来说，只有一小部分索引需要被处理。另外，签名文件的匹配是有噪声的，所以签名的匹配不能保证文档的匹配。更重要的是，很难为相关排序搜索设计一种签名文件方法（Zobel et al., 1998）。

另一个方法是使用高维（spatial）数据结构，例如k-d tree。该方法中，每个文档使用一个高维空间中的指针来编码，查询也是如此。然后高维数据结构被用来寻找和查询最接近的文档。虽然许多相关排序方法本质上是高维的，但是大多数高维数据结构并不是为文本应用中的维度数设计的[⊖]。结果，使用倒排索引进行文档相关排序要比使用典型的高维数据结构快得多。

5.3.1 文档

最简单形式的倒排表只存储包含每个单词的文档编号，没有其他附加信息。这种倒排表与本书后面的索引形式类似。

图5-3展示了由表5-1（在本例中，文档指的是句子）中的4句话构建的这种类型的索引。索引包含4个句子中的每个单词。单词后面是一个列表框，每个框中都含有句子的编号，每个框都是一个posting。例如，可以快速地看到单词“fish”出现在所有的4个句子中，因为编号1、

⊖ 文档中的每一项都相当于一维，所以实际上存在成千上万维。相比之下，典型的数据库应用最多只有几十维。

2、3和4都出现了，也可以知道“fish”是唯一一个出现在4个句子中的单词。有两个单词关系很紧密“tropical”出现在除S_4之外的句子中，“water”没有出现在S_3中。

表5-1 来自Wikipedia的关于“tropical fish”条目的4个句子

S_1	Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.
S_2	Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.
S_3	Tropical fish are popular aquarium fish, due to their often bright coloration.
S_4	In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

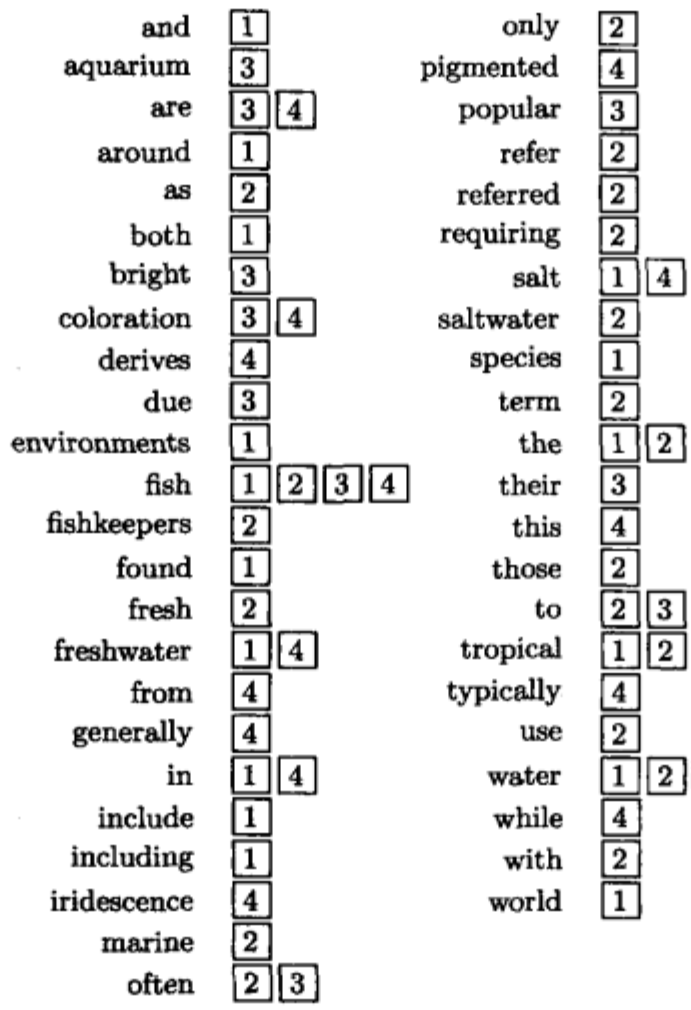


图5-3 表5-1中文档(句子)的倒排索引

注意，这种索引并不记录每个单词出现的次数，它只记录每个单词出现在其中的文档。例如，S_2包含单词“fish”两次，然而S_1只包含“fish”一次。“fish”的倒排表并没有区分句子1和2，两个都以相同的方式列出。后面几节中我们将看到包含单词出现频率的索引。

当考虑倒排表的交集时，它变得更有趣了。假设想找到含有“coloration”和“freshwater”两个单词的句子。倒排索引告诉我们，“coloration”出现在S_3和S_4中，同时“freshwater”出现在S_1和S_4中，因此可以很快地说出只有S_4既包含“coloration”又包含“freshwater”。既然每个列表都是按照句子编号排序的，因此，找到这些列表的交集需要$O(\max(m, n))$时间，其中m和n是两个列表的长度。算法和归并排序一样，使用本章后面将要介绍的倒排表跳转（skipping）

技术，开销降为$O(\min(m, n))$。

5.3.2 计数

我们知道，抽象相关排序模型认为文档是由特征组成的。使用倒排索引，索引中的每个单词反映了文档的特征。这一特征数据可以由相关排序函数进行处理，最终形成文档分数。在一个只含有文档编号信息的倒排索引中，特征是二元的，也就是说，如果文档包含一个词项，则值为1，否则为0。这是很重要的信息，但是当含有大量可能匹配的文档时，对于找到最好的几个文档来说，这太粗糙了。

例如，对于查询“tropical fish”，S_1、S_2和S_3三个句子都匹配。基于文档的索引（见图5-3）中的数据没有给出什么理由来选择其中的一个句子而不选其他的。

现在，看一下图5-4。这个索引看起来和前面的很像，仍然有相同数量的单词和posting，每个posting中的第一个数字和前面索引中的也一样。然而，每个posting现在有了第二个数字，它是每个单词在文档中出现的次数。这些不多的额外的数据，使得对于查询“tropical fish”更倾向于选择S_2，而不是选择S_1和S_3，因为S_2包含“tropical”两次，包括“fish”三次。

and	1:1			
aquarium	3:1			
are	3:1	4:1		
around	1:1			
as	2:1			
both	1:1			
bright	3:1			
coloration	3:1	4:1		
derives	4:1			
due	3:1			
environments	1:1			
fish	1:2	2:3	3:2	4:2
fishkeepers	2:1			
found	1:1			
fresh	2:1			
freshwater	1:1	4:1		
from	4:1			
generally	4:1			
in	1:1	4:1		
include	1:1			
including	1:1			
iridescence	4:1			
marine	2:1			
often	2:1	3:1		
only	2:1			
pigmented	4:1			
popular	3:1			
refer	2:1			
referred	2:1			
requiring	2:1			
salt	1:1	4:1		
saltwater	2:1			
species	1:1			
term	2:1			
the	1:1	2:1		
their	3:1			
this	4:1			
those	2:1			
to	2:2	3:1		
tropical	1:2	2:2	3:1	
typically	4:1			
use	2:1			
water	1:1	2:1	4:1	
while	4:1			
with	2:1			
world	1:1			

图5-4 具有单词计数信息的表5-1中文档的倒排索引

在这个例子中，S_2可能不是很明显的比S_1或S_3好，但是一般来讲，单词计数是预测文档相关性的强有力的手段，具体地，单词计数可以帮助区分那些关于某一特殊主题的文档和其他只是顺便讨论这一主题的文档。假设有两个文档，一个是关于tropical fish的，另一个是关于tropical island的。关于tropical island的文档可能包含单词“fish”，但是只有少数几次。另一方面，关于tropical fish的文档会包含单词“fish”多次。在这个例子中，使用单词出现的次数可以有助于我们将最相关的文档排到最高。

5.3.3 位置

当寻找与查询“tropical fish”相匹配的文档时，单词在文档中的位置是一个很重要的相关性预测。假设一个关于食物的文档，其中含有关于tropical fruits的章节，接着是关于saltwater fish的章节。到目前为止，我们还没有考虑过包含足够信息的索引，这些信息能够指出这个文档并不相关。虽然包含单词“tropical”和“fish”的文档看上去很相关，但是真正想要知道的是文档中是否准确地含有“tropical fish”这个短语。

and	1,15				
aquarium	3,5				
are	3,3	4,14			
around	1,9				
as	2,21				
both	1,13				
bright	3,11				
coloration	3,12	4,5			
derives	4,7				
due	3,7				
environments	1,8				
fish	1,2	1,4	2,7	2,18	2,23
			3,2	3,6	4,3
			4,13		
fishkeepers	2,1				
found	1,5				
fresh	2,13				
freshwater	1,14	4,2			
from	4,8				
generally	4,15				
in	1,6	4,1			
include	1,3				
including	1,12				
iridescence	4,9				
marine	2,22				
often	2,2	3,10			
only	2,10				
pigmented	4,16				
popular	3,4				
refer	2,9				
referred	2,19				
requiring	2,12				
salt	1,16	4,11			
saltwater	2,16				
species	1,18				
term	2,5				
the	1,10	2,4			
their	3,9				
this	4,4				
those	2,11				
to	2,8	2,20	3,8		
tropical	1,1	1,7	2,6	2,17	3,1
typically	4,6				
use	2,3				
water	1,17	2,14	4,12		
while	4,10				
with	2,15				
world	1,11				

图5-5 具有单词位置信息的表5-1中文档的倒排索引

为了判定这种情况，可以在索引中加入位置信息，如图5-5所示。这个索引和前面的索引有一些共同的结构特征，它有相同的索引项，每个列表包含一些posting。然而，这些posting不尽相同。每个posting包含两个数字：文档编号，接着是单词的位置。在前面的索引中，每个文档仅有一个posting。现在，每个单词出现一次就有一个posting。

看一下单词“fish”的列表。在其他的索引中，这一列表仅包含4个posting。现在它包含9个。前两个指出单词“fish”是S_1中第2个和第4个单词，接下来的三个posting指出“fish”是S_2的第7、18和23个单词。

这一信息在与其他的posting列表取交集时最有意思。在与“tropical”的倒排表取交集时，可以发现短语“tropical fish”出现的位置。在图5-6中，两个倒排表按顺序排好，单词“tropical”是S_1中的第1个单词，“fish”是S_1的第2个单词，这意味着S_1一定是以短语“tropical fish”开始的。单词“tropical”作为S_1的第7个词再次出现，但是“fish”没有出现在第8个单词上，因此这不是一个短语匹配。总之，短语“tropical fish”在四个句子中共出现4次。在图中，很容易看到短语匹配，它们出现在posting排成一列的位置上。

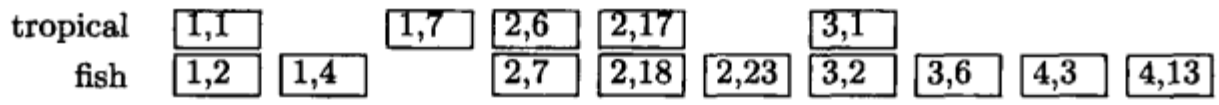


图5-6 对齐的“tropical”和“fish”posting表，以找到短语“tropical fish”

同一方法可以被扩展到用于发现较长的短语或一般的相近表示，如“找到tropical和fish在5个单词之内共现”。假设词“tropical”出现在位置p，可以查找“fish”的倒排表，如果它出现在$p-5$和$p+5$之间的任意位置，则构成一个匹配。

5.3.4 域与范围

真正的文档并非只是由单词构成，它们使用句子和段落将概念划分为逻辑单元。一些文档含有标题或小标题，它们可以作为文档内容的一个简短的摘要。一些特殊类型的文档有其独特的部分，例如每封电子邮件都包含发信人信息、主题等。所有这些都是文档域（document field）的实例，它们是文档的一部分，可以携带一些类型的语义。

在索引中包含一些域的信息是合理的。例如，有一位叫做Dr. Brown的教授曾经给你发了一封关于课程项目何时结束的电子邮件，但是找不到了。可以在电子邮件系统的搜索框里面输入“brown”，但是想要的结果可能和其他含有“brown”这个词的邮件混在了一起，如Brown University、brown socks等。如果将对“brown”的搜索限制在邮件中的“From:”这一行，将准确获得想要的结果。

即使不在查询里面显式地使用，域信息也是非常有用的。标题或小标题倾向于作为文档后面部分的很好的摘要。因此，如果一个用户想要找“tropical fish”，优先选择标题为“Tropical Fish”的文档是很有道理的，而不是标题为“Mauritius”的文档。在正文中多次提到“tropical”和“fish”的文档，这种对文档某些域的偏爱可以融合到相关排序函数中。

为了处理这种搜索，搜索引擎需要能够判别一个单词是否在某一特定的域中。一个办法是对每种文档域都单独建立索引。也就是说，可以为文档标题建一个索引，为小标题建立一个索引，为正文建立一个索引。搜索标题中的词的时候，可以简单地搜索标题索引。然而，在文档中任何一部分找到一个词都比较麻烦，因为需要从不同的索引中获取倒排表。

另一个办法是将一个词出现的域的信息也存储在该词的posting中。例如，我们可以指定数字0代表标题，1代表正文。每个倒排表的posting可以在结尾包括一个0或1。这个数据可以被用来快速地判定一个posting是否在标题中，而且每个posting仅需要增加1个比特位。然而，如果有除标题外更多的域，则需要更多的比特位来表示。

当面对更复杂的文档结构时，这两个办法都有问题。例如，假设想要索引一本书，和本书一样，一些书有不止一个作者。本书的XML描述可能为：

```
<author>W. Bruce Croft</author>,
<author>Donald Metzler</author>, and
<author>Trevor Strohman</author>
```

假设想找作者是Croft Donald的书，如果在一个搜索引擎中输入短语“croft donald”，本书会匹配吗？单词“croft”和“donald”在里面都出现了，事实上它们也是挨着出现的。然而，它们是在两个不同的作者域中。所以本书可能不能很好的匹配查询“croft donald”。但是前面两种处理域的方法（分别索引和posting表中加比特位），并不能区分这种情况。

接下来介绍范围表（extent list）。范围是文档中的邻近区域，可以使用单词的位置表示这

些范围。例如，如果一本书的标题起始于第5个词，结束于第9个词之前，可以将其编码为(5,9)。对于前面的作者，可以写作author:(1, 4),(4,6),(7,9)。(1,4)意味着前三个词（“W. Bruce Croft”）构成了第一位作者，接着是第二位作者（“Donald Metzler”），两个单词。单词“and”不属于作者域，但是接下来两个单词属于作者域，所以最后的posting是(7,9)。

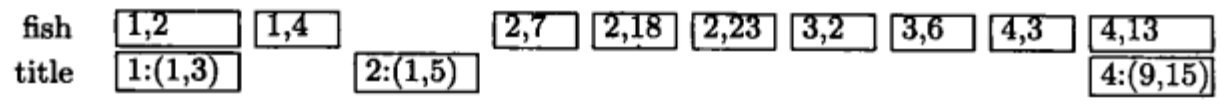


图5-7 对齐的“fish”和标题posting表，以找到单词“fish”在文档标题域的匹配词

图5-7展示了范围表实际是如何工作的。这里对于“fish”有与上一个例子相同的位置posting。还有一个标题域的范围表。为了清晰起见，在posting列表里有一些空隙，以便相应的posting排成一列。在两个列表的开始，看到文档1有一个包含两个单词（单词1和2，在第3个单词之前）的标题。可见这个标题包括单词“fish”，因为“fish”的倒排表指出，“fish”是文档1中的第2个单词。如果用户想要找到标题中含有“fish”的文档，文档1恰好相匹配，文档2不匹配，因为它的标题在第5个词之前结束，但是直到第7个单词“fish”才出现。文档3显然根本没有标题，所以不可能匹配。文档4有一个从第9个词开始的标题（可能这个文档是以日期或作者的描述信息开始的），它确实包含单词“fish”。最终，这个例子显示共有两个相匹配的文档：1和4。

这一概念可以扩展到各种域，例如小标题、段落、句子等。也可以用于有具体含义的更小的片段，例如地址、姓名，甚至仅仅记录动词。

5.3.5 分数

如果倒排表用于生成特征函数的值，那么为什么不将特征函数的值存储起来呢？这当然是可能的，而且一些非常高效的搜索引擎确实是这么做的。这个方法使得存储那些在查询处理阶段需要大计算量的特征函数值成为可能，它也将查询处理引擎的复杂性转化为索引的编码，这可能会更快。

下面具体介绍该方法。前一节有一个例子，说的是对于查询“tropical fish”，标题为“Tropical Fish”的文档应该比标题为“Mauritius”的文档更适合，即使“Mauritius”文档中“tropical”和“fish”多次出现。在进行查询评价的时候，计算能反映这一偏好的分数需要一些复杂的运算。“tropical fish”的posting必须被分成组，以便知道哪些是标题的posting，哪些不是。然后，必须为标题和非标题的posting定义一些分数，然后将这些数混在一起，对每个文档都需要进行这一操作。

一个候选的方法是在倒排表中存储最终的值。“fish”的倒排表可以类似于[(1:3.6),(3:2.2)]，这意味着对于“fish”，文档1的特征值是3.6，文档3的特征值是2.2。数3.6可能考虑了“fish”在标题中、在小标题中、在大号字体中、在粗体中以及在链到该文档的链接中出现的次数。也许文档中根本不存在单词“fish”，而是存在很多鱼的名字，如“carp”或“trout”等。3.6这个值就成了这篇文档和fish相关度的一个指示。

存储像这样的分数，可同时提升和降低系统的灵活性。提升灵活性是因为计算那些需要繁重计算的分数成为了可能，因为这些很难的打分工作放入了索引中。然而，灵活性也会降低，因为一旦索引建好，就再也不能改变评分的机制了。更重要的是，关于单词邻近的信息在这个模型中也不存在了，这意味着如果不对短语建立索引，在评分中就不能包含短语信息

了。这些事先计算的短语列表需要额外的空间。

5.3.6 排列

到目前为止，我们一直假设每个倒排表的posting应该按照文档的编号排列。虽然这是最通用的做法，但这并不是对倒排表排序的唯一方法。倒排表也可以根据分数进行排序，以便最高分数的文档最先出现。当倒排表已经存好分数，或者当只有一种分数需要通过倒排表计算时，这么做才有意义。通过存储文档的分数，查询处理引擎可以只专注于每个倒排表前面的部分，其中记录了最高分数的文档。对于仅由一个单词构成的查询来说，这么做特别有用。在以文档顺序排列的倒排表中，查询处理引擎需要扫描整个列表，才能找到k个分数最高的文档，而在以分数顺序排列的倒排表中，仅需要读取前k个posting即可。

5.4 压缩

有许多不同的途径可以存储数字信息，通常简单地区分为永久性和易失性存储。使用永久存储来保存那些想一直保留的文件和目录，除非我们自己删除它们。磁盘、CD、DVD、闪存及磁带等，都用于此目的。另一方面，动态随机访问存储器（Random Access Memory，RAM）被用来存储易失的信息，它们是那些只有在计算机运行时才需要的信息。当关闭计算机时，所有的信息将会消失。

根据速度和容量可以更好地区分存储的类型。磁带很慢，磁盘要快一些，但是动态RAM要快得多。现代的计算机如此之快，以至于动态RAM都跟不上它的速度了。所以微处理器包括至少两级的缓存，更快的存储器构成了处理器的寄存器。理想情况下，可以使用寄存器或者缓存作为短期存储器，但是这并不切实际，因为它们太昂贵了。

实际上，现代计算机包含一个存储器层（memory hierarchy），在最上层，容量很小，但是速度很快。基本的存储器容量更大，但是速度慢。搜索引擎的性能强烈依赖于它如何利用每种存储器的特性。

压缩技术是管理存储器层的最有力的工具。大规模数据集的倒排表自身也非常大。实际上，当包括词位置、文本范围等信息时，其容量基本上与文本集的大小相当[⊖]。压缩使得相同的倒排表数据占用更小的空间。显著的优点是，可以降低对磁盘和存储器的需求，从而节约开支。更重要的是，压缩使得更多数据存入存储器层。如果索引数据压缩为四分之一，则能够在处理器缓存中存储四倍的有用数据，同时能够以四倍快的速度向处理器提供数据。在磁盘上，压缩也将数据压得更紧密，这降低了寻道时间。在多核和多处理器系统中，多个处理器共享存储器，压缩的数据使得处理器能更有效地共享存储器带宽。

遗憾的是，没有免费的午餐。压缩在节省空间的同时，也带来一定的开销：处理器必须解压数据才能使用。因此，只选择那些压缩比高的压缩算法并不够。为了提高性能，需要选择既减小空间，又比较容易解压的压缩技术。

数学上，假设某个处理器每秒能够处理p个倒排表的posting。该处理器附在每秒能够提供m个posting的存储器上。每秒能够处理的posting数则为$\min(m, p)$。如果$p>m$，则处理器要花些时间等待posting从存储器中传过来。如果$m>p$，则存储器有些时候会闲置。

⊖ 例如，使用开源搜索引擎Indri构建的TREC文本集的索引，其大小是文本集的25%～50%。该文档集要比网页集合小得多。

假设在系统中引入压缩，压缩率为r，也就是说，现在能够在原本仅能存储一个posting的空间内存储r个posting。这使得处理器每秒能够读取mr个posting。然而，在处理它之前，处理器首先需要对每个posting进行解压，这使得处理速度为d的解压因子降低了，从而处理器每秒处理dp个posting。现在，每秒能够处理$\min(mr, dp)$个posting。

当不使用压缩技术时，$r=1$，$d=1$。任何可行的压缩技术使得$r>1$，$d<1$。可见，只有当$p>m$时，压缩才是对性能提高有帮助的技术，也就是说，处理器能够处理的倒排表数据比存储器提供的快。非常简单的压缩方案对r的提高和对d的降低都不大。复杂的压缩方案会大幅提高r和降低d。理想情况是选择一种使得$\min(mr, dp)$最大的压缩方法，这将发生在$mr = dp$时。

本节只考虑无损（lossless）压缩技术。无损技术使用较少的空间存储数据，但是不损失任何信息。也有有损（lossy）数据压缩技术，它们通常应用于视频、图像和音频的压缩。这些技术获得很高的压缩率（前面讨论的r），但是，是通过丢弃一些不重要的数据做到的。后面讨论的倒排表剪枝技术，可以认为是有损压缩技术，但是一般讨论压缩时，仅指无损方法。

特别地，使用压缩技术的目标是降低倒排表的大小。本节介绍的压缩技术特别适用于文本编号、词频以及文档位置等信息。

5.4.1 熵与歧义

到这里，大家已经看到许多概率分布的例子。压缩技术也是基于概率的。压缩背后基本的想法是，用较短的代码表示常见的数据元素，使用较长的代码表示不常见的数据元素。前面讨论的倒排表本质上是数字的列表，在不压缩情况下，每个数字占用相同的空间。有些数字比其他的数字更常见，如果将常见的数字用较短的代码编码，不常见的数字使用较长的代码，则最终可以节省空间。

例如，对于数字0、1、2、3，可以使用两个二进制比特位对这些数字编码。则数字序列：

0, 1, 0, 3, 0, 2, 0

可以编为一个二进制数字序列：

00 01 00 10 00 11 00

注意二进制序列中的空隙使得每个数的开始和结束位置更清晰，实际上这不是编码的一部分。

此序列实例中，数字0出现4次，其他的数字都只出现1次。可以使用一个0比特对数字0进行编码以节省空间。初步尝试的编码可能是：

0 01 0 10 0 11 0

这看起来很成功，因为该编码仅使用10个比特位，而不是原来的14个。然而，该编码存在歧义（ambiguous），也就是说，不清楚如何对其解码。记住，编码中的空格仅是为了方便表示，实际上并不存在。如果使用不同的空格，则获得一个该编码的完美的合法表示：

0 01 01 0 0 11 0

解码成：

0, 1, 1, 0, 0, 3, 0

很不幸，这并不是原来编码的数据。麻烦是，当在编码数据中看到010时，不能确定是由(0, 2)还是(1, 0)编码的。

未压缩编码没有歧义，在哪里加空格是确定的，因为每个数字都是2个比特。在压缩码中，编码数字占用1到2个比特，所以并不清楚在哪里加空格。为了解决这个问题，需要无歧义（unambiguous）码，也称作前缀码（prefix code）或者自由前缀码（prefix-free code）。无歧义码是那些在编码数据中仅有唯一合法途径加空格的编码。

将编码修正为无歧义的：

数字	编码
0	0
1	101
2	110
3	111

则有下面的编码：

0 101 0 110 0 111 0

这种编码需要13个比特位，而不是未经压缩版本的14个，所以还是节省了一些空间。然而，和上面的编码不同，该编码没有歧义。注意，如果编码以0开始，则占用1个比特位，如果编码以1开始，则占用3个比特位。这给出了一个确定性算法，以便在编码流中加入空格。

在练习题部分，你将证明没有任何一个无歧义编码能够对任意输入进行压缩，一些输入将产生更长的编码，这就是为什么获知需要编码的是何种数据是如此重要的原因了。在此例中，可以看到数字0经常出现，可以使用这个事实来降低被编码数据需要的空间。熵（entropy）用来衡量输入的可预见性（predictability）。在此例中，输入似乎可预见，因为数字0比其他数字更常见。利用熵可以产生一个对此有用的编码。

5.4.2 Delta编码

本章所要考虑的所有编码技术都假设小数字比大数字更常见。对于词频，这是一个很好的假设，文档中许多词仅出现一次，有些出现两次或者三次。只有很少词的出现超过10次。因此，使用小代码对小数字编码，大代码编码大数字是有道理的。

然而，文档编号性质则不同。一个典型的倒排表包含一些小的文档编号和一些非常大的文档编号。有些文档包含很多单词，因此会在倒排表中出现多次，但是倒排表中的文档编号分布的熵并不多。

如果考虑文档编号之间的不同而不是文档编号自身，情况就不一样了。倒排表一般是按照文档编号进行排列的。例如，无计数倒排表就是一些文档编号的列表，像这样：

1, 5, 9, 18, 23, 24, 30, 44, 45, 48

既然这些文档编号是有序的，那么序列中的每个文档编号都比它前面的大，比后面的小。这使得可以通过邻接文档编号之间的差额对列表进行编码：

1, 4, 4, 9, 5, 1, 6, 14, 1, 3

该编码列表以1开始，说明第一个文档编号是1。接下来是4，说明第二个文档编号比第一个大4：1 + 4 = 5。第三个数也是4，说明第三个文档的编号比第二个大4：5 + 4 = 9。

这一过程称作Delta编码（Delta encoding），差额经常被称作d-gaps。注意，Delta编码并没有定义存储数据的比特模式，所以它自身不节省任何空间。然而，Delta编码在将一个数字

列表变换成一个小数字列表时特别成功。由于将要讨论小数字列表的压缩方法，所以Delta编码是一个有用的性质。

在继续讨论之前，先考虑单词“entropy”和“who”的倒排表。单词“who”非常常见，所以大多数文档都含有它。对“who”的倒排表使用delta编码时，会有许多小d-gaps，例如：

$$1, 1, 2, 1, 5, 1, 4, 1, 1, 3, \cdots\cdots$$

相反，文本中单词“entropy”很少见，所以少数文档包含它。因此d-gaps会比较大，例如：

$$109, 3766, 453, 1867, 992, \cdots\cdots$$

然而，既然“entropy”很少见，所以这个大数字的列表不会很长。一般地，会发现常见项的倒排表压缩得非常好，而不常见的项压缩得不是很好。

5.4.3 位对齐码

5.4.1节介绍的编码是位对齐码，也就是说，代码区域之间的中断（空格）可以在任意比特位后面。在本节中，将描述一些常用的位对齐码。下一节将讨论只能以字节单位作为结尾的编码。这里所要讨论的所有技术，都是要寻找在倒排表中使用较小的空间存储小数字（例如词频、单词位置、文档编号的Delta编码等）的途径。

最简单的编码是一进制码（unary code）。你可能对二进制比较熟悉，它使用两个符号（通常是0和1）对数字进行编码。一进制数字系统基于1进行编码，也就是说，它使用单一的符号。这是一些例子：

数字	编码
0	0
1	10
2	110
3	1110
4	11110
5	111110

一般来讲，在一进制码中，使用k个1对数字k编码，接着是一个0。需要在结尾使用0使得编码无歧义。

此编码对形如0或1的小数字非常有效，但是很快就成为一种浪费了。例如数字1023可以表示为10个二进制比特位，但是在一进制码中需要1024个比特位。

现在，知道了两种数字编码。一进制很方便，因为它压缩了小数字而且天生是无歧义的。二进制对于大的数字是一个好的选择，但是它不是无歧义的。一种可行的压缩方法是，对经常出现的数字使用比那些不常出现的数字更少的比特位进行编码，这意味着二进制编码自身对于压缩并没有什么用。

1. Elias-γ码

Elias-γ码结合了一进制和二进制编码的长处。使用该码对数字k进行编码，需要计算两个量：

- $k_d = \lfloor \log_2 k \rfloor$

- $k_r = k - 2^{\lfloor \log_2 k \rfloor}$

假设k是二进制的形式，第一个值k_d是二进制数。假设$k > 0$，k最左面的二进制数是1。如果以一进制对k_d编码，并且以二进制对k_r编码，则获得Elias-γ码。表5-2给出了一些例子。

表5-2 Elias-γ码实例

数字(k)	k_d	k_r	编码
1	0	0	0
2	1	0	10 0
3	1	1	10 1
6	2	2	110 10
15	3	7	1110 111
16	4	0	11110 0000
255	7	127	11111110 1111111
1023	9	511	1111111110 111111111

该码中的技巧是，一进制的部分指出二进制部分有多少比特位。任何数字的编码结束于一进制码指出的比特位。对于大于2的数字，它使用更少的比特。对于大数字，节省的空间很可观。例如，现在可以使用19个比特对1023编码，而不是一进制中的1024个。

对于任意数字k，Elias-γ码需要为k_d以一进制形式提供 $\lfloor \log_2 k \rfloor + 1$ 个比特，为k_r以二进制形式提供 $k_d = \lfloor \log_2 k \rfloor$ 个比特。因此一共需要 $2\lfloor \log_2 k \rfloor + 1$ 个比特位。

2. Elias-δ码

虽然Elias-γ码是对一进制码的很大的改良，但是它对于可能包含大数字的输入并不理想。我们知道数字k可以表示成$\log_2 k$个二进制数，但是Elias-γ码需要两倍的比特位以使其编码无歧义。

Elias-δ码试图通过改变k_d的编码方式来解决这一问题。可以对k_d+1使用Elias-γ进行编码。也就是说，将k_d分解为：

- $k_{dd} = \lfloor \log_2 (k_d + 1) \rfloor$
- $k_{dr} = k_d - 2^{\lfloor \log_2 (k_d + 1) \rfloor}$

因为k_d可能是0，但是$\log_2 0$并没有定义，所以在此使用k_d+1。

然后使用一进制对k_{dd}编码，k_{dr}用二进制编码，k_r也用二进制编码。k_{dd}的值是k_{dr}的长度，k_{dr}是k_r的长度，这使得该码无歧义。表5-3给出了一些Elias-δ码的例子。

表5-3 Elias-δ码实例

数字(k)	k_d	k_r	k_{dd}	k_{dr}	编码
1	0	0	0	0	0
2	1	0	1	0	10 0 0
3	1	1	1	0	10 0 1
6	2	2	1	1	10 1 10
15	3	7	2	0	110 00 111
16	4	0	2	1	110 01 0000
255	7	127	3	0	1110 000 11111111
1023	9	511	3	2	1110 010 111111111

为了更有效地对大数字编码，Elias-δ码牺牲了小数字的功效。注意数字2的编码已经变为了4个比特位，而不是Elias-γ码的3个。然而，对于大于16的数字，Elias-δ码大小不会超过Elias-γ码，对于大于32的数字，Elias-δ码需要更少的空间。

Elias-δ码中，k_{dd}需要 $\lfloor \log_2(\lfloor \log_2 k \rfloor + 1) \rfloor + 1$ 位，接着是二进制的k_{dr}需要 $\lfloor \log_2(\lfloor \log_2 k \rfloor + 1) \rfloor$ 位，然后是二进制的k_r需要 $\lfloor \log_2 k \rfloor$ 位。总共需要大约$2\log_2\log_2 k + \log_2 k$位。

5.4.4 字节对齐码

虽然一些小技巧能有助于很快地解码位对齐码，但是在处理字节的处理器上，处理变长比特位码仍然比较麻烦。处理器能有效地处理字节，而不是比特位，所以实际应用中，字节对齐的编码会更快。

有许多字节对齐的压缩方法实例，但是这里只考虑一种比较流行的方法。该编码通常称为v-byte，是变长字节（variable byte length）的缩写。v-byte方法和UTF-8编码很像，是表示文本很常用的方法（见3.5.1节）。

和前面讨论的其他编码类似，v-byte方法使用短码表示小数字，用长码表示大数字。然而，每个码都是一系列字节，而不是比特位。所以，一个整数的最短v-byte码是一个字节。在某些条件下，这可能会浪费一些空间，对数字1，v-byte码需要比Elias-γ码多8倍的空间。通常，空间的差距并没有如此显著。

v-byte码真的相当简单。每个字节的低7位是二进制数，高位是一个决定位。每个编码的最后一个字节高位置1，否则为0。任何一个可以表示成7位二进制数的数，需要一个字节进行编码。更多关于空间占用的信息见表5-4。

表5-4 v-byte码中被编码数字空间的需求

k	字节数
$k<2^7$	1
$2^7 \leqslant k<2^{14}$	2
$2^{14} \leqslant k<2^{21}$	3
$2^{21} \leqslant k<2^{28}$	4

表5-5显示了一些编码的实例。小于128的数以原始的二进制形式保存在一个字节中，除非高位置1。对于较大的数，最小7位存储在第一个字节中，接下来的7位存储在后面的字节中，直到所有的非零位都被存储。

表5-5 v-byte码编码实例

k	二进制	十六进制
1	1 0000001	81
6	1 0000110	86
127	1 1111111	FF
128	0 0000001 1 0000000	01 80
130	0 0000001 1 0000010	01 82
20000	0 0000001 0 0011100 1 0100000	01 1C A0

使用字节对齐码存储压缩数据比位对齐码有很多优点。字节对齐码压缩和解压更快，因为处理器（编程语言）是按照处理字节而不是处理比特位设计的。因此，本书附带的Galago搜索引擎使用v-byte进行压缩。

5.4.5 实际应用中的压缩

压缩技术用来在真正的检索系统中对倒排表进行编码。本节将看到在Galago的PositionListWriter类中，如何使用压缩对倒排表进行编码。

图5-5阐明位置信息如何存储在倒排表中。下面仅考虑“tropical”的倒排表：

(1, 1) (1, 7) (2, 6) (2, 17) (3, 1)

每个数字对中，第一个数表示文档，第二个表示单词的位置。例如，列表第三项表示单词“tropical”出现在文档2的第6个单词位置。为了使此例更有说服力，又在列表中加入(2, 197)：

(1, 1) (1, 7) (2, 6) (2, 17) (2, 197) (3, 1)

每篇文档的位置可以合到一起，以便每个文档单独成一条（文档，词频，[位置]），其中词频是文档中该词出现的次数。该例中的数据现在变为：

(1, 2, [1, 7]) (2, 3, [6, 17, 197]) (3, 1, [1])

词频很重要，因为它使得列表即使没有圆括号和方括号也可读。词频指出方括号中位置信息出现了多少次，即使写成下面的形式，也可以无歧义地理解这些数字了：

1, 2, 1, 7 2, 3, 6, 17, 197 3, 1, 1

然而为了清晰，还是加上了括号。

虽然这些都是小数字，但是使用delta编码可以使它们变得更小。注意，文档编号是以升序排列的，所以可以放心地使用delta编码：

(1, 2, [1, 7]) (1, 3, [6, 17, 197]) (1, 1, [1])

第二项现在是以1而不是以2开始的了，但是这个1意味着“这篇文档编号比前一篇大1”。位置信息也是以升序排列，也可以使用delta编码：

(1, 2, [1, 6]) (1, 3, [6, 11, 180]) (1, 1, [1])

词频不能使用delta编码，因为不是以升序排列。如果还是要对其进行delta编码，则一些delta可能会是负数，如果没有额外的工作，则不能用压缩技术处理。

现在可以删除括号了，倒排表就是一些数字的列表：

1, 2, 1, 6 1, 2, 6, 11, 180 1, 1, 1

由于大多数数字都比较小，可以使用v-byte压缩以便节省空间：

81 82 81 86 81 82 86 8B 01 B4 81 81 81

01 B4是使用两个字节编码的180。其余的数字都使用一个字节编码，整个列表占13个字节。

5.4.6 展望

本节描述了三种倒排表压缩方法，实际应用中，还有许多其他的方法。即使压缩是一个古老的计算机科学研究领域，每年还是有新的压缩方法产生。

为什么需要这些新的方法呢？本节开始时曾谈到，压缩可以提高处理器处理数据的吞吐量。这意味着选择最好的压缩算法与CPU和存储系统的现状有紧密联系。长久以来，CPU处理速度的增长比存储器吞吐量的增长要快得多，因此高压缩比的压缩方法更为吸引人。然而，目前主流的硬件趋势是倾向于多个时钟速度较低的CPU内核。依靠这些系统存储器的吞吐量，低压缩比可能更有吸引力。

更重要的是，现代CPU的处理速度主要来自一些聪明的技巧，如分支预测，它帮助处理器猜测代码将如何执行。更容易预测的代码比不能预测的代码要运行得快得多。许多最新的压缩方法对于解压阶段更容易预测，因此也更快。

5.4.7 跳转和跳转指针

对于许多查询，不需要存储于倒排索引中的所有信息。相反，如果仅仅读取很小一部分和查询有关的数据，会更有效。跳转指针可以帮助达到这一目标。

考虑布尔查询“galago AND animal”。互联网上，单词“animal”大约出现在三亿文档中，而“galago”为一百万。如果假设“galago”和“animal”的倒排表以文档顺序排列的，有一个很简单的算法处理这个查询：

- 设d_g是“galago”倒排表的第一个文档编号。
- 设d_a是“animal”倒排表的第一个文档编号。
- 当“galago”和“animal”的倒排表中都有文档时，循环：
  - 如果$d_g < d_a$，移动d_g到“galago”倒排表中的下一个文档编号。
  - 如果$d_a < d_g$，移动d_a到“animal”倒排表中的下一个文档编号。
  - 如果$d_a = d_g$，文档d_a同时包含“galago”和“animal”。分别移动d_g和d_a到“galago”和“animal”的倒排表中的下一个文档编号

但是，这个算法花费太高。它几乎处理到了两个倒排表中的所有文档，所以得处理大约三亿次循环。超过99%的处理时间将花在处理包含“animal”但是不包含“galago”的两亿九千九百万的文本上。

可以通过在“animal”倒排表中使用向前跳转的方法来稍微改动一下该算法。每次当发现$d_g < d_a$后，向前跳过k个文档到达新的文档s_a。如果$s_a < d_g$，再向前跳k个文档。直到$s_a \geqslant d_g$。这时，已经将搜索范围缩小于往回的k个文档中，其中可能会包含d_g，这样就可以线性地查找了。

这一改良算法需要多少时间呢？既然单词“galago”出现一百万次，该算法将执行一百万次长度为k的线性搜索，期望的开销为500 000 × k步。一共跳过300 000 000/k次。那么该算法共花费500 000 × k + 300 000 000/k步。

表5-6给出了在一些k值下，所需要的处理步骤。当一次跳过25个文档时，可获得最好的期望性能。注意在这个k值下，对于“galago”的每次出现，在“animal”倒排表中不得不向前跳12次。这正是线性搜索的代价：较大的k值意味着在现行搜索中检查更多的元素。如果选择二元搜索，最好的k值变为了208，使用9.2百万步。

表5-6 跳转长度(k)以及期望的处理步骤

k	步骤
5	62.5 million
10	35 million
20	25 million
25	24.5 million
40	27.5 million
50	31 million
100	53 million

如果结合跳转的二元搜索如此有效，为什么会考虑线性搜索呢？这个问题就是压缩。对于二元搜索，需要在倒排表中能够直接跳到元素上，但是压缩后，每个元素占用不同大小的空间。另外，delta编码也可能用于文档编号，意味着即使可以在压缩序列中跳到某一位置上，也需要对前面的数据进行解压，才能获得文档的编号。这很令人沮丧，因为我们的目标是降低需要处理的列表数量，而压缩迫使我们不得不解压整个列表。

可以使用跳转指针表解决这一问题。跳转指针表是在索引中建立的一个额外的数据结构，可以更有效地在倒排表上跳转。

一个跳转指针(d, p)包括两部分，文档编号d和一个字节（或者比特位）位置p。这意味着有一个起始于位置p的倒排表posting，同时它前面是文档d的posting。注意跳转指针的定义解

决了压缩问题：可以从位置p开始解码，同时；我们知道d是p前面的文档，可以用它解码。

作为一个简单的例子，考虑下面的未经压缩的文档编号列表：

5, 11, 17, 21, 26, 34, 36, 37, 45, 48, 51, 52, 57, 80, 89, 91, 94, 101, 104, 119

使用delta编码压缩后，获得如下d-gaps：

5, 6, 6, 4, 5, 9, 2, 1, 8, 3, 3, 1, 5, 2, 23, 9, 2, 23, 7, 3, 15

接着在列表中加入一些跳转指针，使用基于0的位置（也就是，数字5在列表中的0号位置）：

(17, 3), (34, 6), (45, 9), (52, 12), (89, 15), (101, 18)

假设试图使用跳转指针(34, 6)进行解码。在d-gap列表的位置6上是数字2。把34加上2，获得文档编号为36。

更一般地，如果想要在列表中找到文档编号80，可以扫描跳转指针表，直到找到(52, 12)和(89, 15)。80比52大，但是比89小，所以在位置12处开始解码。可以发现：

- 52 + 5 = 57
- 57 + 23 = 80

至此，在列表中成功地找到80。如果要找的是85，再次从跳转指针(52, 12)处开始：

- 52 + 5 = 57
- 57 + 23 = 80
- 80 + 9 = 89

至此，既然85 < 89，可知85不在列表中。

通过对跳转指针用于“galago AND animal”的实例的分析，我们发现跳转指针的效率依赖于“animal”比“galago”更通用这一事实。给定此查询，25是一个很好的k值，但是为所有的查询仅选择了一个k值。对于大多数文档集和查询，优化的跳转距离大约是100字节。

5.5 辅助结构

倒排文件是搜索引擎中的主要数据结构，但是对于一个功能齐全的系统，通常也需要其他的结构。

1. 词表与统计

如前所述，倒排文件就是一些倒排表的集合。为了在索引中搜索，需要一些数据结构来查找某一项的倒排表。解决该问题的最简单的途径是，将每个倒排表分别存储在不同的文件中，每个文件都以相应项的名字命名。为了找到“dog”的倒排表，系统可以简单地打开以dog命名的文件，然后读取内容。然而，正如第4章所示，文档集可能有成百上千万的不同的单词，大多数词仅出现一两次。这意味着如果索引存储在文件中，会包括成百上千万的文件，大多数都非常小。

但是，现代的文件系统没有对这种存储方式进行优化。即使大多数的文件含有几个字节的数据，文件系统一般也要为每个文件预留几千字节空间，结果造成巨大的空间浪费。例如，在AP89数据集上，7万多的词仅出现一次（见表4-1）。每个倒排表需要20字节，总共需要大约2MB的空间。然而，如果每个文件需要1KB，结果是用70MB的空间存储2MB的数据。另外，许多文件系统还要在无序数组中存储目录信息，这意味着对于较大的文件目录，文件的查找

会非常慢。

为了解决这些问题，倒排表通常存储在单一文件中，这也说明了为什么它被称作倒排文件（inverted file）。在倒排文件中，还有一个被称作词汇（vocabulary或lexicon）的目录结构，它包含一个从索引项到倒排文件字节偏移量的查找表。

许多情况下，该词表应很小以便全部放入内存。这时，词表数据能在磁盘上以任意顺序排列。当搜索引擎启动的时候，它被载入哈希表中。如果搜索引擎需要处理非常大的词表，那么在搜索过程中，一些基于树的数据结构，如B树（B-tree），应该被用来最小化磁盘访问。

Galago对其词表结构使用的是混合策略。一个被称作词表的小文件存储的是倒排文件中从词表项到偏移量的查找表。对于倒排文件中每32KB的数据，这个文件仅包括一个词表条目。因此，一个32TB的倒排文件，需要少于1GB的词表空间，也就是说，对于合理大小的数据集，词表通常可以存储于内存中。倒排文件中的列表以字母顺序存储。为了找到一个倒排表，搜索引擎使用二元查找在词表中找到最近的条目，接着从该条开始读取偏移量，这个方法对于每个倒排表仅用一次磁盘寻道就能找到。

为了计算一些特征函数，索引需要存储特定的词汇统计信息，例如词频或者文档频率（见第4章）。当这些统计和某一特定项有关时，它们能容易地存储于倒排表的开始部分。一些统计和语料库相关，例如文档的总数。当这些统计不多时，可以忽略如何对其进行高效的存储。Galago在一个称作manifest的XML文件中存储这些和文档集有关的统计信息。

2. 文档、快照和扩展系统

迄今为止介绍的搜索引擎都是返回文档编号的列表及分数。然而，一个真正面向用户的搜索引擎，需要显示关于每个文档的文本信息，如文档的标题、URL或者文本摘要（第6章将详细介绍）等。为了获取此类信息，文档的文本需要被检索。

第3章中介绍了一些为快速访问而设计的文档存储方式。有许多方法可以解决这一问题，但是最后，需要一个将搜索引擎的结果从数字转化为人们可以读取的独立系统。

5.6 索引构建

在索引被用于查询处理之前，必须先从文本集中生成。构建一个小型的索引实际并不困难，但是随着输入规模的增大，一些索引构造技巧就变得很有用了。我们首先介绍简单的内存中索引构建，接着考虑输入数据不适合都在内存中的情况，最后考虑在多台机器上如何建立索引。

5.6.1 简单构建

图5-8给出了构建简单索引的伪码，处理仅调用很少的步骤。一系列文档传入BuildIndex函数，该函数将每个文档分析成标记，这在第4章讨论过。这些标记都是单词，可能使用一些额外的处理，例如转成小写字母或者提取词根。该函数使用诸如哈希表来删除重复的标记。接着，对于每一个标记，函数决定是否在索引I中生成一个新的倒排表，如果需要，则生成。最后，在倒排表中加入当前的文档编号n。

结果是一个标记的哈希表及倒排表。倒排表就是文档编号的列表，不包括特别的信息。如5.3.1节所示，对于非常简单的索引，这足够了。

该索引器能应用于许多小任务，例如索引几千个文档。然而，有两点限制。第一，它要

求所有的倒排表都存于内存中，对于较大的文档集，这可能不太可行。第二，算法是顺序的，不便于并行处理。该算法并行处理的主要障碍是哈希表，在内循环中需要经常访问它。对哈希表加锁会使分析并行，但是对于充分利用CPU的多个内核，这种自身提升并不足够，处理大数据集需要较少地依赖于内存，同时提高并行性。

```
procedure BUILDINDEX(D)                          ▷ D是一个文本文档集
    I ← HashTable()                              ▷ 倒排表存储
    n ← 0                                        ▷ 文档编号
    for all documents d ∈ D do
        n ← n + 1
        T ← Parse(d)                             ▷ 将文档分析成标记
        Remove duplicates from T
        for all tokens t ∈ T do
            if I_t ∉ I then
                I_t ← Array()
            end if
            I_t.append(n)
        end for
    end for
    return I
end procedure
```

图5-8 简单索引器伪代码

5.6.2 融合

解决前面例子中内存问题的典型做法是融合（merging）。我们可以构建倒排表结构I直到内存耗尽。这时，将部分索引I写到磁盘上，然后开始建造新的索引。最终，磁盘存放了许多部分索引，$I_1, I_2, I_3, \cdots, I_n$。然后系统将这些文件融合为一个。

根据定义，内存中不可能同时有两个部分索引，所以输入文件需要精细地设计以便它们能被融合。一个途径是按照字母顺序对部分索引排序，以便用非常小的内存融合部分索引。

图5-9给出了这种融合过程的实例。即使此图仅演示了两个索引，同时融合多个索引也是可能的。该算法本质上和标准的融合排序算法一致。既然I_1和I_2都是排好序的，它们中至少有一个指向下一块需要写入I的数据。来自两个域的数据交替产生排序的结果，I_1和I_2可能使用相同的文档编号，融合函数对I_2中的文档重新编号。

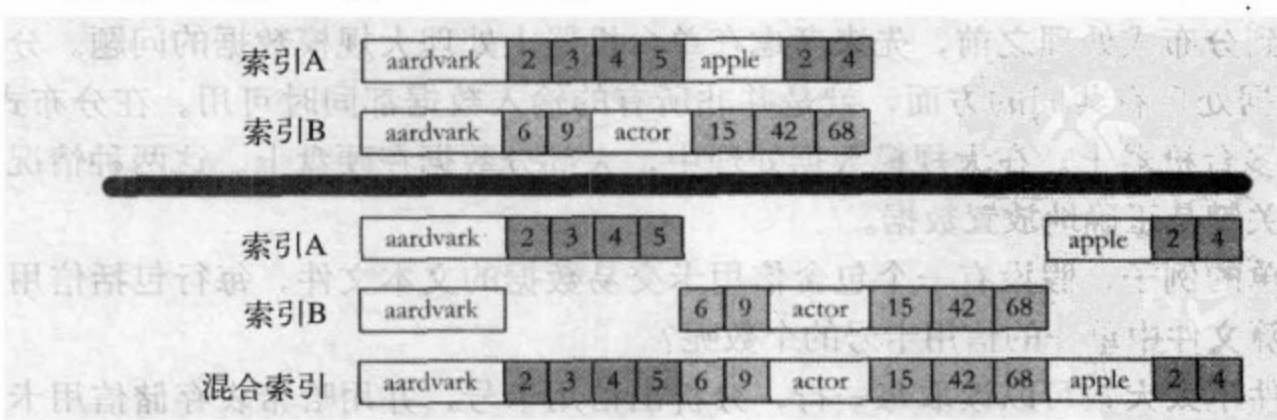


图5-9 索引合并实例。第一和第二个索引合并在一起以产生合并索引

即使只够存储两个单词（w_1和w_2）、一个倒排表posting，以及一些文件指针的内存，融合过程也能成功。实际上，为了更有效地利用磁盘，一个真正的融合函数可以读取很大的I_1和I_2，

并且将较大的*I*写出。

这一融合策略也显示了可行的并行索引策略。如果多台机器建造它们各自的部分索引，一台机器就可以合并所有的索引，形成最后的索引。然而，下一节将要探索更新的分布式索引框架，它正逐渐地流行起来。

5.6.3 并行与分布式

传统的搜索引擎模型一直使用一台快速的机器来生成索引和处理查询，这仍然适用于大量的应用，但不再是大系统的好选择了。对于这些大系统，同时使用众多廉价的服务器和分布式处理软件来协同它们的做法，正逐渐流行起来。MapReduce是使之可行的一种分布式处理工具。

两个因素促成了这一变化。第一，这些大的系统索引的数据量正爆炸式增长。现代的网络搜索引擎已经索引了几百亿的网页，但是更大的索引还在源源不断地出现。想一想如果地球上每个人每天写一篇博客，互联网每年会增加两万亿的页面。乐观的来看，一个典型的现代计算机能够处理几亿页面，尽管不具有多数用户期望的响应时间。这在互联网的规模和我们能够在目前单台机器上处理的规模之间，遗留了巨大的鸿沟。注意这一问题对几个主要的网络搜索引擎公司来说并不严重。更多的公司是想分析（analyze）互联网的内容，而不是要对公众服务。这些公司有相同的可扩展性问题。

第二是经济因素。个人电脑不可思议的普及，已使它们变得非常强大和廉价。相反，大型计算机服务于非常小的市场，因此没什么机会发展规模经济。随时间推移，规模的不同使得很难制造一台比个人电脑强大还便宜的计算机。许多大型的信息检索系统过去运行在大型主机上，但是，如今选择的平台包括了许多廉价通用的服务器。

廉价服务器和大型主机相比有一些缺点。首先，容易坏掉，而且随着服务器的增加，出错的可能性也会增加。其次，编程困难，大部分程序员很好地受训于单线程程序设计，很少会多线程或多进程编程，更没有多少懂协作式网络程序设计。已经开发了许多程序设计工具箱来帮助解决这类问题。RPC、CORBA、Java RMI和SOAP已经被开发出来完成跨机器的函数调用。MPI提供一种不同的抽象，称作消息传递（message passing），它流行于许多科学计算任务中。这些技术没有一个有好的容错性，而且编程模型很复杂。实际上，这些系统对于分布于多台机器上的数据没有什么帮助，还需要程序员的努力来完成。

1. 数据放置

在深入到分布式处理之前，先来考虑在单台机器上处理大规模数据的问题。分布式处理和大规模数据处理有共同的方面，就是并非所有的输入数据都同时可用。在分布式处理中，数据分散于多台机器上。在大规模数据处理中，大部分数据在硬盘上。这两种情况中，高效数据处理的关键是正确地放置数据。

举个简单的例子，假设有一个包含信用卡交易数据的文本文件，每行包括信用卡号和金额，如何计算文件中单一的信用卡号的个数呢？

如果文件不太大，可以读取每一行，分析出信用卡号，并用哈希表存储信用卡号。一旦整个文件都读完后，哈希表将为每个单一的信用卡号包含一个条目。计算哈希表条目的个数会给出想要的答案。但是对于大文件，哈希表可能太大，以至于不能存储于内存中。

现在，假设有相同的信用卡数据，但是文件中的交易根据信用卡号进行了排序。此时计

算单一信用卡号的个数变得非常简单了。读取文件中的每一行并分析出信用卡号。如果信用卡号和上一个不同，计数器加一。当读完整个文件时，计数器就是文件中唯一信用卡号的个数，此时不需要哈希表。

现在回到分布式计算。假设有不止一台计算机用于此项计数工作，可以将大的交易文件分割成小文件，每个计算机处理一小部分，然后结果可以被融合到一起，以产生最终结果。

开始是一个无序交易文件。接着将这个文件分成小的交易组，并且计算每组中单一的信用卡号个数。如何将结果合并呢？可以将每组中发现的信用卡号的数量加起来，但这并不正确，因为相同的卡号可能出现在不止一个组中，因此最终可能不止一次地被计数了。相反，需要保留每组中找到的单一卡号的列表，然后将这些列表合并在一起以产生最终的结果。最终列表的大小是整个集合中唯一信用卡号的数量。

相反，假设交易信息被更细心地分组，以至于所有相同卡号的交易被分到一组中。有了这个额外的限制，每组就可以分别计数了，然后每组的计数相加形成最终的结果。由于不可能重复计数，因此不需要合并，每个卡号恰好出现在一组中。

这些例子可能有些乏味，但是适当的数据分组能从根本上改变任务的性能特性。使用排好序的输入文件让计数任务变得简单，内存的需求几乎降为0，同时也简化了分布式计算。

2. MapReduce

MapReduce是一个分布式编程框架，它致力于数据放置和分布式。从上面几个例子可以看出，适当的数据放置可以简化计算问题。通过数据放置，MapReduce可以解决一些通常任务中的并行问题，同时更容易处理大量数据。

MapReduce的名字来自两块代码，它们是一个用户在使用该框架时需要编写的：Mapper和Reducer。MapReduce库在集群上自动启动大量Mapper和Reducer任务。MapReduce中最有意思的部分是数据在Mapper和Reducer之间传递的途径。

在介绍Mapper和Reducer如何工作之前，先来看看MapReduce的基本思想。map和reduce函数在函数式语言中很常见。简单地说，map函数把一个项列表变换为另一个相同长度的项列表。reduce函数把一个项列表变换为一个单一项。MapReduce框架不像它的定义一样严格：Mapper和Reducer都可以返回任意项数。然而，总体的想法是相同的。

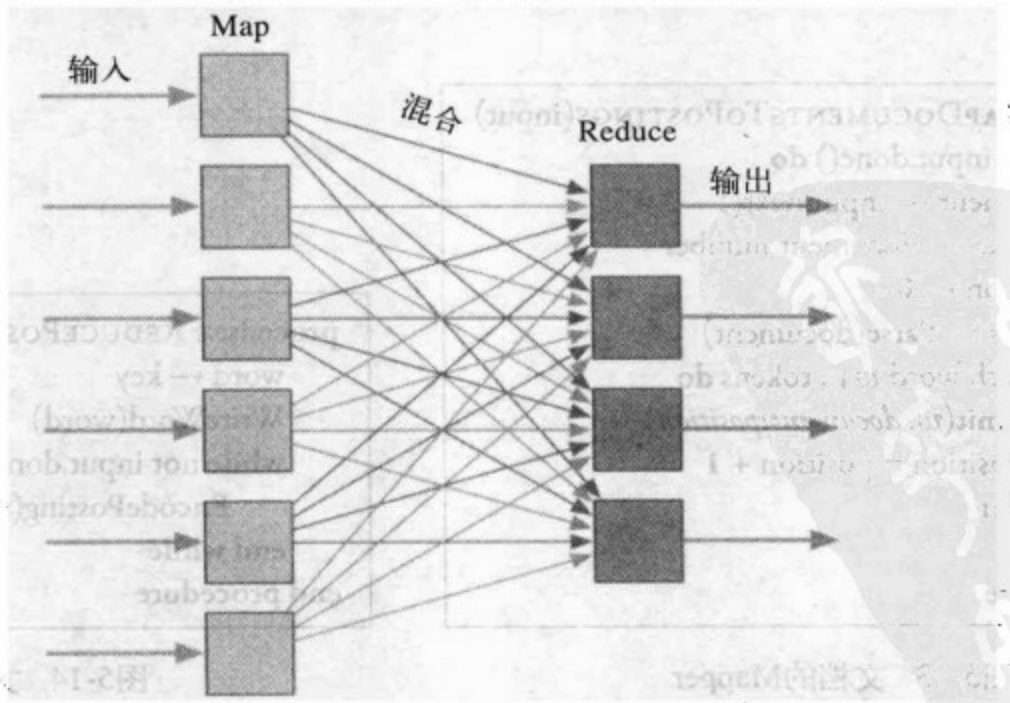


图5-10 MapReduce

假设数据来自一个记录集合，记录被送到Mapper，它把这些记录转换为一些键-值对。下一步是混合，是由库自己执行的。这一操作使用哈希函数，以便所有键相同的键-值对在同一台机器上都互相挨在一起。最后是reduce步骤，记录再次被处理，但是这次是在组中，也就是同时处理所有具有相同键的键-值对。图5-10简要总结了MapReduce的步骤。

```
procedure MapCreditCards(input)
    while not input.done() do
        record ← input.next()
        card ← record.card
        amount ← record.amount
        Emit(card, amount)
    end while
end procedure
```

图5-11 信用卡累加算法的Mapper

```
procedure ReduceCreditCards(key, values)
    total ← 0
    card ← key
    while not values.done() do
        amount ← values.next()
        total ← total + amount
    end while
    Emit(card, total)
end procedure
```

图5-12 信用卡累加算法的Reducer

作为MapReduce任务，前一节中的信用卡数据例子可以很好地运转。在Mapper（见图5-11）中，每个记录被分成键（信用卡号）和值（交易金额）。混合步骤对数据排序，以便有相同信用卡号的记录最终互相挨在一起。reduce步骤对每个唯一的信用卡号产生一个记录，以便使唯一的信用卡号的总数恰好是reducer产生的记录数。

一般地，假设Mapper和Reducer都是等幂的（idempotent）。所谓等幂，意味着如果针对相同的输入Mapper或者Reducer被调用多次，输出总是相同的。这种等幂使得MapReduce库是容错的。如果任何计算的部分出错，可能是由于硬件错误，MapReduce库就可以在别的机器上处理那部分输入。甚至当机器不出错时，有时机器因为配置错误或者比较慢的缺陷处理也会比较慢。这时，一台不正常的机器可能会比集群中其他机器更慢地返回结果。为了避免这种情况，当计算接近完成时，MapReduce库发布备份（backup）Mapper和Reducer，它们复制在最慢的机器上的处理。这保证慢机器不会成为计算的瓶颈。Mapper和Reducer的实现使得这些成为可能。如果Mapper或者Reducer直接修改文件，它们的多份拷贝就不能同时运行。

看一下使用MapReduce对语料库进行索引的问题。在索引器中，将存储具有单词位置的倒排表。

```
procedure MapDocumentsToPostings(input)
    while not input.done() do
        document ← input.next()
        number ← document.number
        position ← 0
        tokens ← Parse(document)
        for each word w in tokens do
            Emit(w, document:position)
            position = position + 1
        end for
    end while
end procedure
```

图5-13 文档的Mapper

```
procedure ReducePostingsToLists(key, values)
    word ← key
    WriteWord(word)
    while not input.done() do
        EncodePosting(values.next())
    end while
end procedure
```

图5-14 文档的Reducer

MapDocumentsToPostings（见图5-13）分析输入中的每篇文档。在每个单词位置，它产生键值对：键是单词自身，值是文档:位置（document:position），也就是文档编号和位置并列在一起。当ReducePostingsToLists（见图5-14）被调用时，产生的posting已经被混合在一起了，以便所有相同词的posting都在一起。Reducer调用WriteWord开始写倒排表，然后使用EncodePosting写每个posting。

5.6.4 更新

迄今为止，一直假设索引是批量处理的。这意味着文档集合作为索引器的输入，索引器构建索引，然后系统允许用户进行查询。实际上，大多数的文档集合是一直变化的。至少，文档集倾向于随时间变得越来越大，每天都有新的新闻和电子邮件。其他情况，例如网页搜索或者文件系统搜索，文档的内容也随着时间变化。有用的搜索引擎应该能够响应动态的文档集合。

可以用两种技术解决更新问题：索引合并及结果合并。如果索引是在内存中存储的，对于快速的更新有很多办法。然而，即使搜索引擎在内存中对查询进行评价，一般索引也是存在磁盘上的，因此直接基于磁盘的更新并不简单。5.6.2节介绍了索引是如何合并在一起的。这给出了一个简单的方法，用来在索引中增加数据：构建一个新的、较小的索引（I_2），然后与老的索引（I_1）合并，形成一个新的包含全部数据的索引（I）。在合并期间，I_1中被删除的文档的posting可以被忽略，以至于它们不出现在I中。

当索引更新大批量地进行时，可能有几千的文档同时更新，索引合并是可行的策略。对于单一文档的更新，这不是一个好的策略，既然将整个索引写入硬盘相当耗时，那么对于这些小的更新，最好对于新数据构建一个小的索引，但是不将它合并到大的索引中。查询分别在这个小的索引和大的索引中处理，结果被融合在一起以找到前k个。

结果合并解决如何处理新文档的问题：将它们加入新的索引中。但是如何从索引中删除文档呢？通常使用一个被删除文档列表。在查询处理过程中，系统检查被删除的文档列表，确定没有被删除文档进入到给用户显示的结果列表中。如果文档的内容修改了，可以使用被删除文档列表删除旧的版本，然后在最新的文档列表中加入新的版本。

结果合并假设有一个较小的、在内存中的索引结构来保存新的文档。这一内存中的结构可以是一个哈希表，如图5-8所示，因此即使对于一个文档，也可以被简单快速地更新。

为了获得更高的性能，与其只使用两个索引（内存中的索引和基于磁盘的索引），不如使用更多的索引。既然每个新的索引使查询处理变慢，那么使用太多的索引不是一个好主意。然而，使用太少的索引会由于过度的磁盘访问导致索引构建过程变慢。对这一问题特别理想的解决方案是几何分割（geometric partitioning）。在几何分割中，最小的索引I_0包含内存中能存下的数据，接下来的索引I_1包含大约是I_0的r倍的数据。如果m是机器中内存的字节数，则索引I_n包含mr^n和$(m+1)r^n$之间的字节的数据。如果索引I_n包含超过$(m+1)r^n$的数据，那么它被合并到索引I_{n+1}中。如果$r=2$，那么系统使用10个索引可以存储1000m字节的索引数据。

5.7 查询处理

一旦索引被建好，就需要处理其中的数据来产生查询结果。即使使用最简单的算法，基于索引的查询处理也要比不基于索引的快得多。然而，更聪明的算法可以加速查询处理的速

度几十甚至上百倍。我们首先介绍两个最简单的查询处理算法："document-at-a-time"和"term-at-a-time"，然后介绍更快更灵活的变种。

5.7.1 document-at-a-time评价

至少在概念上，document-at-a-time检索是最简单的使用倒排文件进行检索的方法。图5-15是对于查询"salt water tropical"的document-at-a-time检索的图。虽然posting已经对齐了，倒排表还是水平显示，以便每列表示一个不同的文档。此例中的倒排表包含词频，分数就是每个文档中的词频总和。垂直的灰色的线 v指出检索的不同步骤。第一步，第一篇文档的所有的词频被加起来生成文档的分数。一旦完成对第一篇文档计分，则开始第二篇文档，然后是第三篇、第四篇。

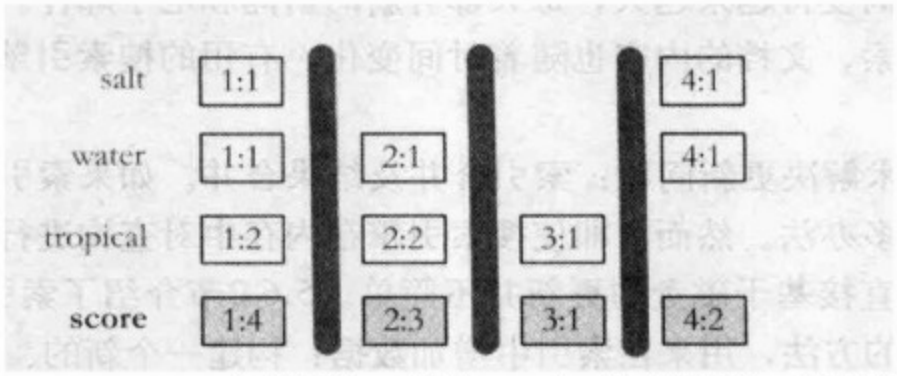


图5-15 document-at-a-time查询评价。数字(x:y)表示文档编号(x)和词计数(y)

图5-16显示了这一策略实现的伪代码。参数是查询Q，索引I，特征函数集合f和g，检索文档个数k。这个算法使用5.2节中的抽象相关排序模型对文档打分。然而，在这个简化的例子中，假设只有非零特征值$g(Q)$和查询中的词相关。这给出了一个倒排表和特征之间的简单的相关性：对于查询中的每一项有一个倒排表，以及为每个倒排表提供一个特征。本章后面将要探究有结构的查询，这是超越这个简单模型的标准途径。

```
procedure DocumentAtATimeRetrieval(Q, I, f, g, k)
    L ← Array()
    R ← PriorityQueue(k)
    for all terms w_i in Q do
        l_i ← InvertedList(w_i, I)
        L.add( l_i )
    end for
    for all documents d ∈ I do
        for all inverted lists l_i in L do
            if l_i points to d then
                s_D ← s_D + g_i(Q) f_i(l_i)          ▷ 更新文档分数
                l_i.movePastDocument( d )
            end if
        end for
        R.add( s_D, D )
    end for
    return the top k results from R
end procedure
```

图5-16 简单的document-at-a-time检索算法

对于查询中每个词w_i，从索引中找到一个倒排表，假设这些倒排表以文档编号排序。

InvertedList对象起始于每个倒排表的第一个posting，所有被找到的倒排表存储于数组L中。

在主循环中，函数为文档集中的每个文档循环一次，对每篇文档，所有倒排表都被检查，如果在一个倒排表中出现，特征函数f_i被评价，并且文档分数s_D通过加上权重函数的值来计算。然后，倒排表指针指向下一个posting。在每篇文档循环结束处，计算一个新的文档分数并加到优先队列R中。

为清晰起见，此伪代码可以进行简单的性能提升。然而，实际上，优先队列R每次仅需要包含前k个结果。如果查询包含多于k个结果，为了节省内存，分数最低的文档可以被删除，直到只留下k个。在文档集合的所有文档上进行循环是没必要的，可以只对那些最少出现在一个倒排表中的文档打分。

这个方法的主要好处是节省内存。唯一主要的内存使用来自于优先队列，一次只需要存储k个条目。然而，在实际实现中，大部分倒排表在评价期间会在内存中被缓冲起来。

5.7.2 term-at-a-time评价

图5-17显示了使用与term-at-a-time（见图5-15）相同的查询、评价函数以及倒排表数据的term-at-a-time检索。注意，虽然每个图的结构不同，但是两个图中的分数计算完全相同。

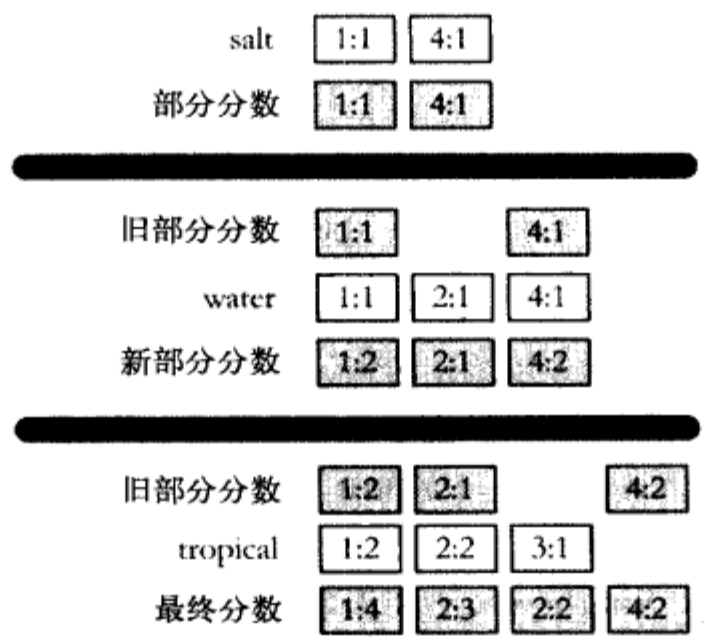


图5-17 term-at-a-time查询评价

和前面一样，灰色的线指明每步之间的界限。在第一步中，“salt”的倒排表被解码，并且部分分数（partial score）被存于累加器（accumulator）中。这些分数之所以被称作部分分数，是因为它们只是最终文档分数的一部分。累加器的命名来自于它们的任务，累加每篇文档的分数信息。第二步中，来自累加器的部分分数与来自“water”倒排表的数据合并，产生新的部分分数集合。来自“tropical”倒排表的数据在第三步被加入后，评分的过程就结束了。

此图隐含为每个倒排表生成一个新的累加器的集合。这是一个可能的实现技术，在实际中累加器是存储在哈希表中的。每篇文档的信息作为被处理的倒排表数据更新。所有的倒排表被处理后，哈希表包括最终的文档分数。

对于抽象相关排序模型的term-at-a-time检索算法（见图5-18）开始和document-at-a-time相似。它产生优先队列并且为查询中的每项找到一个倒排表，这和document-at-a-time算法很像。然而，接下来的一步则不一样。不是在索引中的每个文档上循环，外层循环是在每个倒排表上进行的。内层循环然后读取倒排表的每个posting，计算特征函数f_i和g_i并且将它的权重

加到累加器A_d上。主循环结束后，扫描累加器并加入优先队列，以决定前k个返回结果。

```
procedure TermAtATimeRetrieval(Q, I, f, g k)
    A ← HashTable()
    L ← Array()
    R ← PriorityQueue(k)
    for all terms w_i in Q do
        l_i ← InvertedList(w_i, I)
        L.add( l_i )
    end for
    for all lists l_i ∈ L do
        while l_i is not finished do
            d ← l_i.getCurrentDocument()
            A_d ← A_d + g_i(Q) f(l_i)
            l_i.moveToNextDocument()
        end while
    end for
    for all accumulators A_d in A do
        s_D ← A_d                          ▷ 累加器包含文档分数
        R.add( s_D, D )
    end for
    return the top k results from R
end procedure
```

图5-18 简单的term-at-a-time检索算法

term-at-a-time算法的主要缺点是累加器表A需要占用内存。而document-at-a-time策略仅需要很小的优先队列R，它包含有限数量的结果。然而，term-at-a-time算法通过更有效的磁盘访问弥补了此不足。它从头到尾读取每个倒排表，它需要最小的磁盘寻道并且需要非常小的缓冲表以获得较高的速度。相反，document-at-a-time算法在倒排表之间切换并且需要较大的缓冲表帮助降低寻道的代价。

实际上，document-at-a-time和term-at-a-time算法都没使用额外的优化。这些优化能急剧地提高算法的运行速度，并且对节省内存有很大的影响。

5.7.3 优化技术

对于查询处理，主要有两类优化方法。第一类是从索引中读取较少的数据，第二类是处理较少的文档。由于读取较少量数据的同时很难对相同数量的文档打分，因此这两类方法是相关的。当使用特别复杂的特征函数时，致力于对较少的文档评分是主要需要考虑的。对于简单的特征函数，最快的速度来自于忽略尽可能多的倒排表数据。

1. 倒排表跳转

在5.4.7节中，介绍了跳转指针。这类向前跳转是目前最流行的部分忽略倒排表的方法（见图5-19）。更复杂的方法（例如树结构）也是可能的，但是不经常使用。

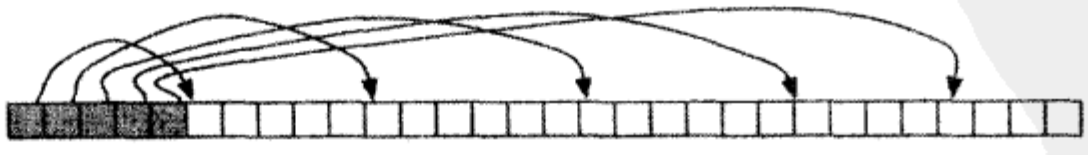


图5-19 倒排表中的跳转指针。灰格子是跳转指针，所指向的白色格子为倒排表posting

跳转指针不会提高读取倒排表的速度。假设有一个n字节长的倒排表，在每c个字符后加入跳转指针，指针为k字节长。读取整个倒排表需要读取$\Theta(n)$字节，但是使用跳转指针跳过倒排表需要$\Theta(kn/c)$时间，这和$\Theta(n)$相当。甚至没什么运行时间上的收获，c因子可以很大，典型的值是$c = 100$，$k = 4$，跳过一个倒排表仅需要读取全部数据的2.5%。

注意随着c变大，需要读取的数据减少了。所以，为什么不使c变得尽可能大呢？问题是如果c变得太大，平均性能会下降。下面详细看一下这个问题。

假设想要在倒排表中找到p个posting，该倒排表长n字节，每c字节有k字节的跳转指针。因此，倒排表中共有n/c个间隔。为了找到那p个posting，需要在跳转指针中读kn/c字节，但是也需要在p个间隔中读数据。平均来讲，假设需要的posting在两个跳转指针的中间位置，所以读取额外的$pc/2$字节来找到那些posting。则共读取的字节为：

$$\frac{kn}{c} + \frac{pc}{2}$$

注意，此分析假设p比n/c小得多。这就是假设每个posting位于其自身的间隔的原因。随着p增长接近n/c，一些我们需要的posting将位于相同的间隔。注意，一旦p接近n/c，就需要读取几乎所有的倒排表，所以跳转指针不是很有用了。

回到公式，可以看出当一个较大的c值使第一项变小时，它也使第二项变大。因此，为c选择一个完美的值依赖于p值，直到查询被执行时才知道p是什么。然而，使用先前的查询模拟跳转行为并且获得一个较好的估计c值是可能的。在练习题中，需要画一些这个公式的图，来获得平衡点。

虽然倒排表跳转似乎能够节省磁盘访问，但是实际上很少这样。现代的磁盘对于序列数据的读取要比随机跳转好得多。因此，在任何可见的加速之前，大多数磁盘需要跳过大约100 000个posting。即便如此，跳转仍然很有用，因为它降低了解码那些已经从磁盘上读取的压缩数据的时间，并且它极大地降低了处理缓存在内存中的倒排表的时间。

2. 联合处理

最简单的查询优化方式是联合处理（conjunctive processing）。所谓联合处理，就是每篇返回给用户的文档，需要包含所有查询项。联合处理是许多网络搜索引擎的默认模式，部分是速度原因，部分是因为用户的期望。对于短的查询，联合处理能同时提高效率和效果。相反，使用较长查询的搜索引擎，例如整个段落，联合处理就不是一个好的选项。

当有一个查询项较罕见时，联合处理做得最好。对于查询“fish locomotion”，单词“fish”的频率是“locomotion”的100倍。既然只对包括两个词的文档感兴趣，系统能跳过“fish”倒排表中的大部分，以便找到同时包括“locomotion”的文档的posting。

联合处理能用于term-at-a-time和document-at-a-time系统。图5-20为联合处理给出了改进的term-at-a-time算法。当处理第一个项时（$i = 0$），处理正常进行。然而，对于其余的项（$i > 0$），算法在第14行开始处理posting。如果有一个posting（17行），则为下一篇包括所有前面查询项的文档检查累加器表，并构造表l_i以向前跳到那篇包括所有前面查询项的文档。如果该posting不存在，则累加器被删除（21行）。

document-at-a-time版本（见图5-21）与老的document-at-a-time版本很类似，除了内部循环。它从找到当前被倒排表指向的最大的文档d（11行）开始，文档d不保证包含所有的查询

项，但仍是一个可行的候选。下一个循环试图跳过所有倒排表到达d点（14行）。如果不成功，循环结束，另一篇文档d被选择。如果成功，次文档被评分并且加入到优先队列中。

```
1: procedure TermAtATimeRetrieval(Q, I, f, g, k)
2:     A←LinkedList()
3:     L←Array()
4:     R←PriorityQueue(k)
5:     for all terms w_i in Q do
6:         l_i←InvertedList(w_i, I)
7:         L.add(l_i)
8:     end for
9:     for all lists l_i ∈ L do
10:        d_0 = −1
11:        while l_i is not finished do
12:            if i = 0 then
13:                d←l_i.getCurrentDocument()
14:                A_d←A_d + g_i(Q) f(l_i)
15:            else
16:                d←l_i.getCurrentDocument()
17:                d'←A.getNextAccumulatorAfter(d)
18:                A.removeAccumulatorsBetween(d_0, d')
19:                if l_i.getCurrentDocument() = d' then
20:                    A_d←A_d + g_i(Q)f(l_i)
21:                    l_i.moveNextDocument(d')
22:                else
23:                    l_i.skipForwardTo(d')
24:                end if
25:            d_0 = d
26:            end if
27:        end while
28:    end for
29:    for all accumulators A_d in A do
30:        s_D←A_d                    ▷累加器包含文档分数
31:        R.add (s_D, D)
32:    end for
33:    return the top k results from R
34: end procedure
```

图5-20 具有联合处理的term-at-a-time检索算法

在两个算法中，当第一个表(l_0)最短，最后一个表(l_n)最长时，系统运行得最快。这导致最后一个表中的最大可能跳转距离，这也是跳转最有帮助的地方。

3. 阈值方法

到目前为止，所有考虑过的算法都没有过多地探讨参数k。k是用户需要的结果的数量，对于许多搜索应用，该数字通常很小，如10或者20。因为这个小的k值，倒排表中的大部分文档将从来不会显示给用户。阈值方法专注于此k参数，目的是为了对更少的文档打分。

实际上，对于每个查询，有某一最小分数是每个被检索出的文档都需要达到的。该最小分数是第k个最高得分文档的分数。任何没有达到此分数的文档将不会显示给用户。在本节中，将使用希腊字母τ表示该值，称作阈值（threshold）。

如果能在处理查询之前知道合适的τ值，许多查询优化将变得可能。例如，如果一篇对用户有用的文档至少需要τ的分数，就可以避免将不会达到τ值的文档加入优先队列中

(document-at-a-time情况下)。

```
1: procedure DocumentAtATimeRetrieval(Q, I, f, g, k)
2:     L ← Array()
3:     R ← PriorityQueue(k)
4:     for all terms w_i in Q do
5:         l_i ← InvertedList(w_i, I)
6:         L.add( l_i )
7:     end for
8:     while all lists in L are not finished do
9:         for all inverted lists l_i in L do
10:            if l_i.getCurrentDocument() > d then
11:                d ← l_i.getCurrentDocument()
12:            end if
13:        end for
14:        for all inverted lists l_i in L do l_i.skipForwardToDocument(d)
15:            if l_i points to d then
16:                s_d ← s_d + g_i(Q) f_i(l_i)          ▷ 更新文档分数
17:                l_i.movePastDocument( d )
18:            else
19:                break
20:            end if
21:        end for
22:        R.add( s_d, d )
23:    end while
24:    return the top k results from R
25: end procedure
```

图5-21 具有联合处理的document-at-a-time检索算法

遗憾的是，不进行查询的评价就不知道如何计算真正的τ值，但是能估计它。这些估计被称作τ′。期望τ′≤τ，以便能够忽略任意分数小于τ′的文档。当然，τ′越接近τ，算法运行得越快，因为它忽略更多的文档。

使用document-at-a-time策略，给出估计的τ′很容易。R保留评价过程中截至当前的前k个分数最高文档的列表。假设R已经有k篇文档了，则可以设τ′为R中最低分数的文档的分数。使用term-at-a-time策略，直到查询评价快要结束时才有全部文档的分数。然而，仍然可以设置τ′为累加器表中第k大的分数。

4. MaxScore

τ′有了合理的估计值，在倒排表中就可以忽略一些数据。τ′的估值表示一篇进入最终相关排序表的文档分数的下限。因此，可以忽略倒排表中不会生成比τ′大的文档分数的那部分posting。

下面用一个简单的例子更仔细地看一下这是如何做到的。考虑查询“eucalyptus tree”，单词“tree”的频率大概为“eucalyptus”的100倍，所以对此查询评价的大多数的时间花费在对包含“tree”的文档的评分上，而不是“eucalyptus”。这没有充分利用好时间，因为要找到前k个包含两个单词的文档。

图5-22显示了实际操作上的影响。和本章前面的图一样，“eucalyptus”和“tree”倒排表的posting也是根据文档排成一行。此图显示有许多只包含单词“tree”但不包含“eucalyptus”

的文档。

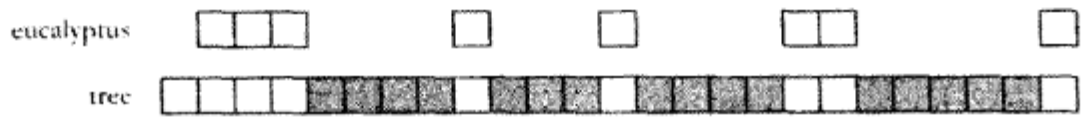


图5-22 对查询“eucalyptus tree”的MaxScore检索。灰色格子指出打分期间可放心忽略的posting

假设索引器在“tree”倒排表中计算最大的部分分数，该值为μ_{tree}。这是任何仅包含该单词的文档可能获得的最大分数（MaxScore）。

假设只对相关排序表中的前3个文档感兴趣（也就是k为3）。第一个被评分文档仅包括单词“tree”，接下来3个文档包括“eucalyptus”和“tree”。我们将使用τ'表示来自这3个文档的最低值。在此时，很可能$\tau' > \mu_{tree}$，因为τ'是两个查询项都包括的文档的分数，而μ_{tree}是仅包含一个查询项的文档的分数。在此介绍灰色方格的含义。一旦$\tau' > \mu_{tree}$，可以放心地跳过所有灰色的posting，因为已经证明这些文档将不会进入最终的相关排序表中。

虽然图中的posting数据是虚构的，但是对于真实的“eucalyptus”和“tree”倒排表，“tree”的posting有99%会是灰格子，因此可放心地忽略。这种跳转可以在不影响查询结果质量的前提下，急剧降低查询时间。

5. 提早终止

MaxScore方法保证查询处理的结果无论经过优化还是没有优化，都是相同的。然而在有些情况下，需要冒质量的风险，使得经过优化的结果和没有优化的结果不相同。

为什么选择这么做呢？一个原因是一些查询比其他的要费劲得多。看一下短语查询“to be or not to be”，如果使用非常一般的查询项，会产生非常长的倒排表。运行该查询会严重地降低系统资源，以至于不能为其他查询服务。为此，截取查询处理能够对其他使用该系统的人保证公平。

另一个原因是MaxScore太保守。它不会逃过任何可能会包括有用的候选文档的倒排表区域。因此，MaxScore会花许多时间查找可能根本不存在的文档。冒险忽略这些不重要的文档，对降低系统资源的消耗很有益。

提早终止是如何做到的呢？在term-at-a-time系统中，可以通过简单地忽略一些常见的查询项来做到。这和使用停用词没有太大区别。另外，可以在一些固定数量的posting被读取后，不再考虑其他项。此处的原因是，在处理大量的posting后，相关排序很好地完成了，读取更多的信息不会改变相关排序结果。这对于含有很多（几百个）项的查询来说特别适用，例如使用查询扩展技术时，很可能会是这样。

在document-at-a-time系统中，提早终止意味着忽略倒排表非常靠后的文档。如果文档是随机排序的话，这不是一个好主意，但这不是必须的。相反，文档能够以一些质量指标来排序，例如PageRank，在此条件下提早终止意味着忽略质量较低的文档。

6. 倒排表排列

目前为止，本章中所有的例子都假设倒排表以相同的顺序排列，即文档编号。如果文档编号是随机的，意味着文档是以随机的顺序排列的。这很容易导致与查询匹配最好的文档排在最后。由于好的文档在倒排表中都分散开了，所有可行的查询处理算法必须读取或者跳过整个倒排表，以保证不丢失好的文档。由于这些倒排表可能很长，需要考虑更好的排序方式。

一个提高文档排列的途径是基于文档质量对文档排列，这和前面章节讨论的一样。有大量可用的质量衡量指标，如PageRank或者用户点击的数量。如果许多好的文档已经被找到了，就可以考虑提早停止搜索。来自MaxScore节的阈值技术能在此使用。如果知道倒排表中的文档按照质量的降序排列，就可以在检索过程中，在倒排中剩余的文档上计算一个分数上限。当τ'超过剩余文档最高分数时，检索可以在不损失效能的情况下终止。

另一个选项是对每个倒排表根据部分分数排序。例如对于“food”倒排表，可以首先存储包括单词“food”许多实例的文档。在互联网应用中，这相当于将饭店页面提早加入倒排表。对于“dog”倒排表，可以首先存储关于dog的页面（也就是说包含许多“dog”的实例）。评价一个关于food或者dog的查询变得很简单。然而，其他的查询会很难。例如，如何评价查询“dog food”呢？最好的办法是使用如同term-at-a-time检索的累加器表。然而，不是一次读取整个倒排表，而是仅仅读取每个倒排表的一小部分。一旦累加器表显示已经找到许多好的文档了，则停止查找。可以想像，检索可能共现的查询项会变得更快，例如“tortilla guacamole”。当查询项不太可能共现时，例如“dirt cheese”，找到前面的文档将花费更长的时间。

5.7.4 结构化查询

在迄今为止已经看到的查询评价的例子中，假设每个倒排表相当于一个单一的特征，并且将那些特征加在一起形成最终的文档分数。虽然在简单的情况下能运行，但是可能希望有一个更有趣的评分函数。例如，在图5-2中，查询有大量有趣的特征，包括一个短语（tropical fish），一个同义词（chichlids），以及一些非主题特征（例如入链接）。

一种途径是在检索系统中写特别的相关排序代码，以检测这些额外的特征并且直接使用倒排表数据计算分数，但这是比仅使用特征的线性组合更复杂的途径。该方法极大地提高了可以使用的评分方式，并且非常有效。遗憾的是不够灵活。

另一个选择是构建一个系统支持结构化查询（structured query）。结构化查询是使用查询语言写的查询，容许改变查询中使用的特征以及那些特征的组合方式。查询语言不是普通用户使用的。相反，查询翻译器（query translator）将用户的输入转换为结构化查询表达。此翻译过程正是系统的智能所在，包括如何对词特征加权以及使用哪些同义词。一旦此结构化查询被生成，它就交给检索系统执行。

你可能已经对此类模型很熟悉了，因为数据库系统正是如此工作的。关系数据库使用结构化查询语言（Structured Query Language，SQL）。许多重要的应用包括一个用户界面和查询语言生成器，其他的逻辑都由数据库控制。将应用逻辑从数据库逻辑中分离出来，使得数据库变得既高度优化，又高度通用。

Galago包括一个结构化查询过程系统，将在第7章描述，此查询语言也在习题中使用。

图5-23给出了一个结构化查询的树形表示，写成Galago的结构化查询语言为：#combine(#od:1(tropical fish) #od:1(aquarium fish) fish)。

此查询指明文档分数应该是三个子查询的组合。第一个查询是#od:1(tropical fish)。在Galago查询语言中，#od:1操作符的意思是，其中的项在匹配的文档中需要依次出现，并且依照查询中的顺序。#od:1(aquarium fish)也是一样，最后的查询项是fish。每个子查询都类似一个文档特征，使用#combine操作符组合起来。

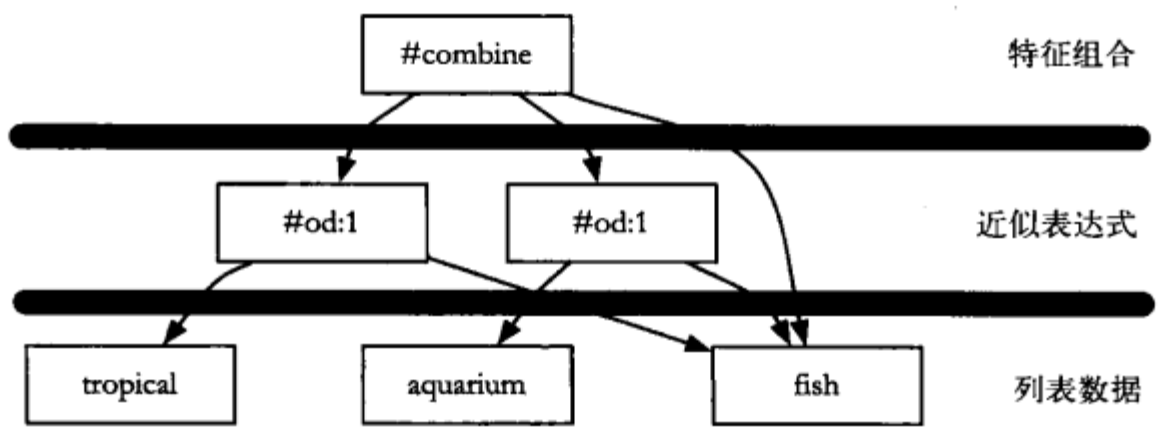


图5-23 结构化查询#combine(#od:1(tropical fish) #od:1(aquarium fish) fish)的评价树

该查询包括结构化查询表达式的主要类型，在树的最底层是索引项，这些项在索引中对应一些倒排表。再上一层是临近表达式，这些表达式组合倒排表以生成更复杂的特征，例如出现在文档标题中的“fish”的特征，或者“tropical fish”作为一个短语出现。在最顶层，从倒排表中计算的特征数据组成了最终文档的分数。在此层，来自倒排表的位置信息被忽略了。

Galago通过生成一个迭代对象树来评价结构化查询，和图5-23显示的树很像。例如，一个迭代器返回与#od:1(tropical fish)相匹配的文档。此迭代器使用“tropical”和“fish”的迭代器来发现这些匹配的文档。#combine操作符是文档分数迭代器，使用#od:1(tropical fish)、#od:1(aquarium fish)以及fish迭代器。一旦类似于此的迭代器被建成，对文档打分就仅仅成了对各个文档使用根迭代器了。

5.7.5 分布式的评价

一台现代计算机能处理令人惊奇的数据，也许对大多数任务是足够的。然而，处理大规模语料库或者大量用户，也许需要使用不止一台机器。

使用多台机器的一般方法是，将所有查询发给一台目录（director）计算机。然后该目录机将消息发给多台索引服务器（index server），它们各自做查询处理的一部分任务。目录机然后组织这一过程的结果，并将结果发送给用户。

最简单的分布式策略称作文档分布式（document distribution）。在此策略中，每台索引服务器的行为类似于整个文档集一小部分数据的搜索引擎。目录机发送查询的拷贝给每台索引服务器，每台机器返回前k个结果以及每个结果的分数。这些结果由目录机合并为一个相关排序表，然后返给用户。

一些相关排序算法依赖文档集的统计信息，例如一个查询项在文档集中出现的次数或者包括一个查询项的文档的个数。这些统计信息需要被各个索引服务器共享，以产生可比较的分数，方便高效的合并。在非常大的集群中，在索引服务器层的词项统计信息变化很大。如果每个索引服务器仅使用其自身的词项统计信息，相同的文档可能收到非常不同的分数，这有赖于哪台索引服务器被使用。

另一种分布式的方法称作词项分布式（term distribution）。在词项分布式中，为整个集群建立单一的索引。索引中的每个倒排表然后被分给一个索引服务器。例如，单词“dog”可能由第三台服务器处理，然而“cat”由第五台服务器处理。对于有n个索引、k个查询项的系统，所有查询项都在一台机器上的概率是$1/n^{k-1}$。对于有10台机器的集群，对3个查询项，概率仅为1%。因此，在大多数情况下，为处理一个查询的数据不会都存储于一台机器。

一台索引服务器，通常是存储最长倒排表的那台，被选中来处理查询。如果其他索引服

务器上有相关的数据，则通过网络传送给它们以处理查询。当查询处理完成后，结果被发送给目录机。

词项分布式方法比文档分布式方法复杂，因为需要在机器间传送倒排表数据。给出倒排表的大小，传输所涉及的消息能充满整个网络。另外，每个查询仅使用一个处理器，而不是多个处理，与文档分布式相比，这增加了整体查询时延。词项分布式的主要优点是寻道时间。如果有k个查询项以及n个索引服务器，对于文档分布式系统，处理一个查询的全部磁盘寻道数是$O(kn)$，但是在词项分布式系统中，仅需要$O(k)$。对于磁盘受限的系统，特别是寻道受限的系统，词项分布式可能更有吸引力。然而，最近的研究显示，词项分布式不太值得花费如此的努力。

5.7.6 缓存

第4章给出了文档中词频遵从Zipfian分布：一些词经常出现，但是大量的词出现的频率却很低。查询的分布也是类似。一些查询，例如关于名人或者当前的事件，在公众搜索引擎中会非常流行。然而，一个搜索引擎每天收到的大约一半的查询是唯一的。

这引出了对缓存（caching）的讨论。宽泛地讲，缓存的意思是，存储一些以后可能会用到的东西。在搜索引擎中，通常希望缓存对于查询的相关排序结果，但是系统也可以缓存来自磁盘的倒排表。

缓存对于搜索引擎十分适合。查询和相关排序表很小，也就是在缓存中存储它们不需要占用太多的空间。相反，在大规模语料库上处理查询是密集计算，这意味着一旦相关排序表被计算出来，保存它通常是很有意义的。

然而，缓存并不能解决所有性能问题，因为每天收到的大约有一半的查询是唯一的。因此，搜索引擎自身必须能非常快地处理查询。这导致搜索引擎和缓存系统之间的竞争。最近的研究建议，当内存空间吃紧时，缓存应该专注于最常见的查询，给缓存倒排表留下足够的空间。具有多个查询项的唯一查询，仍可能共享项词并且使用相同的倒排表。这解释了为什么倒排表缓存能够有比查询缓存高的命中率。

当使用缓存系统时，避免数据陈旧很重要。因为假设查询结果不会随时间改变，缓存才有意义，最终确实也是这样。缓存需要可被接受的失效时间，以容许新的结果。当处理5.6.4节讨论的分割的索引时，这很容易。每个缓存可以对应一个特定的索引分割，当那个分割被删除时，缓存也可以被删除。记住，一个使用缓存能处理某一峰值吞吐量的系统，在不使用缓存时，将处理小得多的吞吐量。这意味着，如果系统破坏了其缓存，在缓存变得相对广泛之前，系统会很慢。如果可能，缓存刷新应该在非负载峰值时进行。

参考文献和深入阅读

本章包括许多主题：索引、查询处理、压缩、索引更新、缓存及分布式。所有这些主题集中于一章，以便强调这些模块是如何一起工作的。

为了了解这些模块是如何相互联系的，研究真正的系统很有用。Brin & Page (1998) 写了一篇关于早期搜索引擎系统的文章，对如何构建一个完全的系统是很有益的概述。后续的文章展现出搜索引擎的体系结构已经变换了多次——例如Barroso等人（2003）。Dean and Ghemawar (2008)的MapReduce文章给出了关于MapReduce如何开发和实际工作的更详细的介绍。

商业搜索引擎的内部工作原理经常被认为是商业机密，因此它们确切的工作细节通常不会公开。一个重要的例外是TodoBR引擎——一个流行的巴西互联网搜索引擎，它的他们的引擎工作原理经常发表于文章中。例如他们关于两层缓存机制的文章（Saraiva et al., 2001），还有许多其他的文章。

Managing Gigabytes（Witten et al. 1999）这本书是索引构建的标准参考书，特别是它关于压缩技术的讨论。倒排表压缩仍然是一个活跃的研究领域。近来一个重要的研究是PFOR系列压缩（Zukowski et al. 2006），它利用现代处理器的特点所开发的技术，对于解压小整数特别快。Büttcher and Clarke（2007）在最新的硬件上比较了各种压缩技术。

Zobel and Moffat（2006）写了一篇关于近期倒排索引重要研究进展的综述文章，既包括索引的构建，也包括查询处理，该文章是关于此研究的最好介绍。

Turtle and Flood（1995）开发了MaxScore系列算法。Fagin等人（2003）使用相似的方法，虽然他们不是最先将其应用于信息检索的。Anh and Moffat（2006）改进了这些想法，建造了一个特别高效的检索系统。

Anh and Moffat（2005）和Metzler等人（2008）涵盖了计算能够被存于倒排表中的分数的方法。实际上，这些文章描述了如何计算既在检索中有用，又能在倒排表中紧密存储的分数。Strohman（2007）探索了建立打分索引以及基于它们处理查询的整个过程。

本章中的许多算法都是基于合并两个排序输入的，索引构建依赖于此，任何document-at-a-time检索过程也依赖于此。Knuth写了一整卷关于排序和搜索的书，包括大量关于合并以及关于基于磁盘合并的材料（Knuth, 1998）。如果Knuth的书太难懂，那么任何算法教科书都能给你关于合并的更细致的介绍。

Lester等人（2005）为索引更新开发了几何分割方法。Büttcher等人（2006）在此模型中增加了一些额外的内容，专注于在更新过程中如何处理非常通用的项。Strohman and Croft（2006）给出如何在不停止查询处理的情况下更新索引。

练习

5.1 5.2节介绍的抽象相关排序模型，其中文档和查询使用特征表示。使用特征表示文档和查询的优点是什么？缺点是什么？

5.2 本章的模型包括一个相关排序函数$R(Q, D)$，每篇文档和查询比较计算分数。这些分数然后用于确定最终的相关排序表。

某个相关排序模型可能包含不同的相关排序函数$f(A, B, Q)$，其中A和B是文档集中两个不同的文档，Q是查询。当A应该比B排得靠前时，$f(A, B, Q)$的值为1，否则为−1。

如果有一个相关排序函数$R(Q, D)$，请问如何在需要$f(A, B, Q)$形式的系统中使用它。为什么反之不可行呢？（在需要$R(Q, D)$的系统中，使用$f(A, B, Q)$。）

5.3 假设你建造了一个搜索引擎，使用100台计算机，每台存储100万篇文档，以便可以搜索1亿篇文档的文档集。你倾向于类似$R(Q, D)$的相关排序函数还是$f(A, B, Q)$呢？为什么？

5.4 假设你的搜索引擎利用基于相关排序函数$R(Q, D)$从文档集中检索出50篇文档。你的用户界面仅能显示10个结果，但是你可以从前50篇文档中选择任意文档来显示。为什么你可能会给用户显示除了前10篇文档外的其他文档呢？

5.5 文档很容易包括成千上万的非零特征。为什么查询只有很少的非零特征？

5.6 索引对于搜索文档并不是必须的。例如你的互联网浏览器具有查找功能，可以不使用索引查找文本。什么时候你应该使用倒排索引来搜索文本？使用倒排索引的优点是什么？缺点是什么？

5.7 5.3节介绍了许多在倒排表中不同的存储文档信息的方式。如果你需要一个非常小的索引，你可能需要哪种倒排表？如果你需要查找引用，例如Kansas City或者São Paulo，则需要哪种呢？

5.8 写一个程序，它能够建造一个简单的文本文档倒排索引，每个倒排表包括文档文件名。假设文件A包含文本“the quick brown fox”，文件B包含“the slow blue fox”。程序的输出应该为：

```
% ./your-program A B
blue B
brown A
fox A B
quick A
slow B
the A B
```

5.9 5.4.1节中，为2比特位二进制数字生成了一个无歧义的压缩方法。找到一个数字序列，当被“压缩”时，比不压缩占用更多的空间。

5.10 假设一个公司为2比特位二进制数开发了一种新的无歧义的无损压缩方法，称作SuperShrink。其开发者声称它将节省任意2比特位数字序列。证明该开发者在撒谎，更细致地，证明：

- SuperShrink从不会使用比未压缩编码更少的空间。
- 对于SuperShrink，有一个输入压缩版本比未压缩输入更长。

可以假设每个2比特位输入数字分别编码。

5.11 在选择压缩算法之前，为什么需要知道一些关于待被压缩数据类型的信息？具体关注习题5.10的结果。

5.12 开发一个Elias-γ编码器。验证该程序产生与表5-2一样的编码。

5.13 当执行两项布尔与查询时，其中一项出现一百万次，另一项出现一亿次，确定优化的跳转距离k。假设一旦集中于一个适当的区域时，使用线性查找。

5.14 5.7.3节中，看到优化的跳转距离c可以由$kn/c+pc/2$的最小值决定，其中k是跳转指针的长度，n是倒排表的大小，c是跳转间隔，p是找到的posting个数。

画出该函数在k=4，n=1 000 000，p=1 000时，随c变化的曲线。然后再画出当p=10 000时的曲线。注意优化的c值是如何变化的。

最后，将函数$kn/c+pc/2$对c求导，以对给定的参数集合（k，n，p）找到c的最优值。

5.15 第4章中，学到了Zipf定律以及文档集中大概50%的词仅出现一次。你的任务是使用MapReduce设计一个程序，验证Zipf定律。

该程序输出数字对列表，形如：

```
195840,1
70944,2
34039,3
```

```
...
1,333807
```

此输出示例指明195 840个单词在集合中仅出现一次，70 944个单词出现两次，34 039个单词出现三次，但是有一个词出现了33 3807次。你的程序为一个文档集合打印此类列表。

程序将使用MapReduce两次（两个Map步骤，两个Reduce步骤）以产生输出。

5.16 使用Galago搜索工具包写出习题5.15的程序。验证其通过索引本书网站上提供的Wikipedia文档集是正确的。

第6章　查询与界面

这是信息检索，不是信息分布。

——Jack Lint，《妙想天开》

6.1　信息需求与查询

尽管索引结构和相关排序算法是搜索引擎的核心部分，但是从用户的角度看，搜索引擎主要是用于提交查询和查看搜索结果的界面。用户无法改变排序算法的工作方式，但是他们能够通过构造查询、浏览检索结果以及重写查询的过程，与搜索引擎系统进行交互。这些交互过程是信息检索过程中的一个关键部分，决定了搜索引擎看起来是否提供了有效的服务。在本章中，要讨论查询转换和提炼（query transformation and refinement）、收集与显示搜索结果（search result）的技术。同时还要讨论跨语言搜索引擎（cross-language search engine），因为它们在很大程度上依靠查询和搜索结果的转换。

在第1章中，阐述了信息需求是用户使用搜索引擎的动机。信息需求有不同的类型，研究人员根据不同的维度对这些信息需求进行归类，例如需要查找的相关文档的数量，需求信息的类型，产生信息需求的任务等。并且指出了在有些情况下，人们很难明确地说明他们的信息需求是什么，这是因为这个信息在他们所拥有的知识中是一个“缺口”[⊖]。从搜索引擎设计者的角度上观察这些信息需求，可以得出两个方面的重要结论：

- 查询能够表达完全不同的信息需求，可能需要不同的搜索技术和排序算法来产生最好的排序结果。
- 查询仅仅是对信息需求的粗略表达。当用户发现难以表达出他的信息需求时，所提出的查询就会出现这种情况。然而，这种情况的产生通常是由于搜索引擎鼓励用户输入短查询导致的，这取决于搜索引擎的接口形式，而且过长的查询检索不到需要的结果。

第一个方面将在第7章中给予进一步介绍，第二个方面是本章讨论的主题。我们将给出提炼查询的一些技术，如拼写纠错（spelling correction）、查询扩展（query expansion）和相关反馈（relevance feedback），这些技术可以采用自动方式或与用户交互的方式实现。提炼查询是为了能够更好地表达信息需求和随后更好地检索文档。另外，检索结果显示的方式也是很重要的，可以帮助用户理解检索结果是否达到了他的信息需求。在这个方面，我们将要讨论的技术包括页面摘要生成（snippet generation）、检索结果聚类（result clustering）和文档高亮显示（document highlighting），这些技术用来帮助用户理解检索结果。

目前在搜索引擎上最常用的查询形式，是采用由几个关键词组成的短查询（在大量的网络检索研究中，查询长度平均为2到3个词），既然用短查询表达信息需求是模糊的、不准确的[⊖]，人们为什么不用更长的查询呢？这是有原因的。在过去，搜索引擎的查询语言（query

---

⊖ 这是Belkin著名的Anomalous State of Knowledge (ASK) 假设（Belkin et al., 1982/1997）。

⊖ 如果你想问某人哪种热带鱼容易饲养，你会只问他“热带鱼”或“鱼”吗？

language）是为专家或搜索中介（search intermediary）使用而设计的。他们之所以被称为搜索中介，是由于他们担当了需要查找信息的人和搜索引擎之间的媒介。这些查询语言是非常复杂的，下面的例子是搜索中介在提供法律信息的搜索引擎上构造的一个查询。

用户查询（User query）：Are there any cases that discuss negligent maintenance or failure to maintain aids to navigation such as lights, buoys, or channel markers?
搜索中介查询（Intermediaries query）：NEGLECT! FAIL! NEGLIG! /5 MAINT! REPAIR! /P NAVIGAT! /5 AID EQUIP! LIGHT BUOY "CHANNEL MARKER"

这个例子中的查询语言采用了一些通配符（wildcard）和各种形式的位置算符（proximity）来指定用户的信息需求。使用通配符可以限定与查询相匹配的单词中应包含的最小匹配长度的字符串。例如，“NEGLECT!”将匹配单词“neglected”、“neglects”或“neglect”。

位置算符用来限定与查询相匹配的词之间的距离限制条件。例如，词汇相邻的限制是一种类型位置算符，如上例中的双引号“CHANNEL MARKER”指定了这两个单词要相邻。而更常见的窗口操作符则限定了允许匹配的文本窗口的宽度（按词计算）。例如，“/5”指定了所限制的词汇必须出现在五个词宽度的文本窗口中。其他一些典型的位置算符还限定了句子和段落位置，例如“/P”限制了词汇要在同一段落中。在这个查询语言中，如果没有特别给出限制条件，就假定其中的词汇是“或”（OR）的布尔关系。

一些这样的查询语言操作符也可以用在搜索引擎界面上，例如用引号指定一个短语或用“+”指出必须包含的词汇，但搜索引擎通常更侧重于使用简单关键词查询（有时称为“自然语言”查询），以便大多数人都能够自己进行查询[⊖]。既然我们希望查询的构造能够更加自然，那么为什么不鼓励人们输入更详细的描述以表达他们想要查找的内容，而是仅仅采用几个关键词呢？事实上，在一些协作式问答（某些人提出问题，其他人给予回答）的应用领域，例如在10.3节描述的基于社区的问答系统上，查询的平均长度为30个词左右。问题在于当前的搜索技术无法很好地处理长查询。例如，大部分的互联网搜索引擎仅仅是将含有查询词的文档进行排序。如果一个人提交了一个包含30个词的查询，最可能的结果是什么都没找到。即使找到了包含所有词的文档，但在搜索结果中，往往也没有了那些符合语法结构的长查询中表达出的隐含语言含义。搜索引擎上采用的排序算法主要是基于将文本看作词集合的统计学方法，而不是基于句法或语义特征。

当人们了解到采用长查询会得到什么样的搜索结果后，很快就学会了如何得到更可靠查询结果的方法，就是采用与他们要查找的信息相关的几个关键词去构成查询。但是这显然增加了用户的负担，我们将要描述的查询提炼技术，就是要减轻用户的负担，并且改善那些粗略表达的查询。

6.2 查询转换与提炼

6.2.1 停用词去除和词干提取

正如上节所提到的，目前搜索引擎上最常用的查询方式是采用几个关键词的组合。在有些查询中，也采用引号来指定一个短语，或者用“+”限定必须包含某个词，但是在本章接下

⊖ 搜索引擎在内部可能仍会采用复杂的查询语言（如在7.4.2节中所述），但是不用在界面上。

来的部分，我们做了一个简化假设，认为查询只是简单的文本形式[⊖]。文本查询最初的处理过程应该对应于对文档的处理步骤，在查询文本中的词应转换成与文档文本处理时产生的词项相同的形式，否则在排序时会出现错误。尽管这样的处理似乎是显而易见的，但是却成为一些搜索方案中出现问题的根源。而且在查询转换和文档转换中，仍存在许多的不同之处，特别在停用词去除和词干提取处理上。对于其他一些处理步骤，如结构分析、词素切分，在查询中或者不需要这些处理（关键词查询没有语法结构），或者与对文档的处理方法基本上是相同的。

在4.3.3节中，提到了可以不在索引文档时去除停用词，而安排在处理查询的时候。在索引中保留停用词，可以增加系统处理含有停用词的查询的灵活性，对停用词的处理可以像普通词一样（留在查询中），或者去除它们，或者在某些条件下保留它们（例如，用引号或者“+”指定包含它们）。

查询的词干提取（query-based stemming）是增加搜索引擎灵活性的另一项技术。如果在建立索引时对文档中的词进行了词干提取处理，那么查询中的词也必须进行词干提取。可是在有些情况下，对查询进行词干提取会降低搜索结果的准确性。例如，对于查询“fish village”和“fishing village”，应该获得不同的搜索结果，但是有些词干提取算法将“fishing”处理为“fish”。如果索引文档的时候不进行词干提取，我们就能够在处理查询的时候决定是否对“fishing”提取词干。可以根据一些因素做这样的决定，例如，是否这个词是被引用的短语中的一部分。

为了使查询的词干提取能够获得更好的效果，一定要用恰当的词的变形来扩展查询，而不是将查询词减少到只剩下词干，因为并没有对文档进行词干提取。如果用词干“fish”代替了查询词“fishing”，查询就不再匹配包含“fishing”的文档。相反，查询应该扩展到包括词“fish”。查询扩展应该由系统（不是用户）根据某些形式的同义词操作去实现，如在7.4.2节中所述。或者也可以用词干和词来索引文档，这将使查询执行更有效，但是索引的规模也会增加。

每个词干提取算法都隐含着产生一些词干类别（stem class）。词干类别是指能够通过词干提取算法转换成相同词干的一组词。这些词干类别的获取方法，是在一个大规模的文本集合上运行词干提取算法，并记录哪些词能够映射到给定的词干上。词干类别的数目可以十分庞大。例如，这里展示了三个词干类别，它们是通过在TREC新闻集上采用Porter词干提取器获得的（每个词干类别中的第一个词是词干）。

```
/bank banked banking bankings banks
/ocean oceaneering oceanic oceanics oceanization oceans
/polic polical polically police policeable policed
-policement policer policers polices policial
-policically policier policiers policies policing
-policization policize policly policy policying policys
```

这些词干类别中，不仅包含较多的词（“polic”类包含22个词），而且也包含一些错误。与“police”和“policy”相关的词，不应包含在相同的词干类别中，而且这也会影响排序准确率。尽管其他的词没有错，但可能会用在不同的上下文中。例如，“banked”更常用在讨论

⊖ 根据对最近的网络查询的样本分析，用引号的查询约占1.5%左右，而用“+”的查询不到0.5%。

飞行或水池的时候，但是这个词干类别中的其他词，更常用在与金融相关的论述上。如果在扩展查询时直接使用词干类别，那么词干类别中词的数量也是一个问题。给一个简单的查询扩展了22个词，显然会影响系统的响应时间，并且如果没有正确地使用同义词操作符(Synonym operator)，那么可能引起检索错误。

这两个问题能够通过文本集合中词的共现分析来处理，进行这样的分析是基于这个想法：能够互相替换的词的变形，应该经常在文本中共现。我们采用如下具体处理步骤：

1）对于词干类别中的每对词，计算它们在W个词的文本窗口中共现的次数，W通常取50~100个词。

2）对于每对词计算共现或关联度（association）指标，这用来衡量词之间的关联程度。

3）构造一个图，其中顶点代表词，如果词共现的指标大于阈值T，则用边连接它们。

4）找到这个图的联通分支（connected component），它们构成一个新的词干类别。

在TREC实验中的词项关联度衡量方法（term association measure），是采用戴斯系数(Dice's coefficient）计算的。这个方法最早在20世纪60～70年代就被用在词项相似度计算和自动叙词表构建的研究中。如果n_a是包含词a的窗口（或文档）数量，n_b是包含词b的窗口数量，n_{ab}是包含词a和词b的窗口数量，N是集合中文本窗口的数量，那么戴斯系数为$2 \cdot n_{ab}/(n_a+n_b)$，这是一个简单的词共现比例值。还有一些其他的关联度衡量方法，将在6.2.3节中介绍。

一个图中的两个顶点间如果存在路径，它们就处于同一联通分支中。这个图可以表示词的关联，联通分支可以作为词的群集（cluster）或词的聚合（group），其中每个词都至少与在群集的簇中一个其他的词有高于阈值T的关联。参数T根据经验设定。在9.2节我们将讨论这种聚类以及其他的聚类方法。

把这个方法应用在上面的三个词干类别的例子中，使用TREC数据进行词共现分析，会获得下面的联通分支：

```
/policies policy
/police policed policing
/bank banking banks
```

在这个新的词干类别中，词的数量少了，也将不恰当的分组（例如“policy”和“police”）分离出去了。一些实验结果表明，这个方法通常在一般的查询扩展上能够获得较好的排序效果。

对于“fishing village”，这个查询怎么样呢？通过共现分析获得的词干类别是：

```
/fish fished fishing
```

这意味着并没有解决这个问题。正如前面提到的，查询上下文决定了词干提取是否恰当。用词“fish”和“fished”扩展“fishing in Alaska”，这个查询是合理的，但是不适合用在查询“fishing village”上。前面描述的共现分析，用到了一般形式的上下文，但没有达到对一些具体查询词的共现进行处理。

目前在一些应用中（如网络检索），大量的查询日志（query log）也成为可利用的资源，利用这些资源的统计分析，可以验证提取的词干，甚至获取词干类别。对含有相同词的查询，从中分析趋向于共现的词语变形，这可以作为“fish/fishing”问题的一种解决方法，因为“fish”与“village”共现在一个查询中的可能性很小。

与4.3.4节中所描述的方法相比，这个词干提取方法可以认为是基于词典的方法，其中词

典是基于规则演算式词干提取器的输入（即词干类别）自动生成的。这个方法也可以用在不具备规则演算式词干提取器的词干提取处理中。这时，词干类别采用非常简单的判别标准，例如将具有相似n元文法的所有词聚合在一起。一个简单的例子是，将前三个字符相同的词聚成一类。这些初始词干类别的规模要比规则演算式词干提取器获得的类别大很多，但是通过共现分析，使得最终得到的词干类别的规模，减小到与规则演算式词干提取器获得的类别相似的大小。检索实验结果证实了，采用规则演算式词干提取器和基于n元文法类别的词干提取器，在排序效果上只有很小的差别。

6.2.2 拼写检查和建议

拼写检查是查询处理过程中极为重要的部分。在网络搜索引擎中，大约有10%～15%的查询中包含拼写错误。人们通常采用“Did you mean：...”的形式来纠正这些错误。在查询日志中，包含大量的像下面这些简单错误的例子(从一个最近的网络查询样本中获得)：

poiner sisters
brimingham news
catamarn sailing
hair extenssions
marshmellow world
miniture golf courses
psyhics
home doceration

这些错误与在字处理文档中出现的错误很相似。但是在一些查询中，包含与网址、产品、公司和个人相关的词，这些词在标准拼写字典（standard spelling dictionary）中，几乎不可能被找到。下面是在同一查询日志中的一些例子：

realstateisting.bc.com
akia 1080i manunal
ultimatwarcade
mainscourcebank
dellottitouche

查询中拼写错误的严重程度以及种类的多样性，提出了一项重要的挑战。为了讨论在搜索引擎中，哪些拼写校正技术是最有效的，首先，我们回顾一下在普通文本中，是如何进行拼写校正的。

在一些拼写检查工具中，采用的基本方法是对于在拼写字典（spelling dictionary）中没有的词，就建议更正它们。将在字典中没有的词与字典中包含的那些词进行比较，并根据他们之间的一个相似度衡量标准，来提出更正建议。最常用的一个词之间比较的衡量标准是编辑距离（edit distance），编辑距离是将一个词通过编辑转换成另一个词所需要的操作数。Damerau-Levenshtein距离（Damerau-Levenshtein distance）是指通过计算这个转换过程中单个字符插入、删除、替换、交换的最少次数[⊖]。研究显示，80%或更多的拼写错误是由于这类单个字符错误引起的。

下面是Damerau-Levenshtein距离为1的一些词转换例子，它们只需要一个操作或编辑，就

⊖ Levenshtein与其相似，但没有将交换作为基本操作。

能够生成一个正确的词：

extenssions → extensions （插入型错误）
poiner → pointer （删除型错误）
marshmellow → marshmallow （替换型错误）
brimingham → birmingham （交互型错误）

另外，在doceration → decoration的转换过程中，编辑距离为2，因为有两步编辑操作：

doceration → deceration
deceration → decoration

提高错误拼写的词与字典中的词之间编辑距离的计算速度，可以采用各种方法和数据结构，包括只计算包含有相同的首字母的词（因为拼写错误很少出现在第一个字母上），有相同或相似长度的词（因为拼写错误很少改变词的长度），以及读音相同的词[⊖]。对于最后的这种情况，采用发音规则将词进行编码，有相同编码的词被作为可能的校正词。Soundex编码(soundex)是一种简单的语音编码（phonetic code），它最初被用来处理病历卡上的名字匹配问题。这个编码的规则如下：

1）保留第一个字母（用大写字母）
2）用连字符替换这些字母：a，e，i，o，u，y，h，w。
3）用数字替换下面这些字母：

1：b，f，p，v
2：c，g，j，k，q，s，x，z
3：d，t
4：l
5：m，n
6：r

4）删除相邻的重复的数字。
5）删除连字符
6）保留前三个数字或用0延长到三个字符。
这些编码的一些例子如下：

extenssions → E235; extensions → E235
marshmellow → M625; marshmallow → M625
brimingham → B655; birmingham → B655
poiner → P560; pointer → P536

最后的例子显示了，正确的词也不会一直有相同的Soundex码。还有一些为拼写校正而开发的更精细的语音编码方式（例如，GNU Aspell拼写检查器[⊜]采用的语言编码）。

设计这些编码是为了在这些编码（code）上计算编辑距离，可以减少寻找正确词汇的搜索过程。

一个拼写错误会有多个可能的更正形式，例如，拼写错误“lawers”可能会有下列编辑距离为1的更正形式：lawers→lowers、lawyers、layers、lasers、lagers。拼写校正程序要决定是否将所有这些词都展示给用户，并且以什么样的顺序展示。一个典型的策略是，按它们在语

⊖ 两个词发音相同但是意义不同，成为同音词（homophone）。
⊜ http://aspell.net/.

言中出现频率的递减顺序呈现。注意，这种处理方式没有使用拼写错误的上下文。例如，在查询“trial lawers”中出现的拼写错误，不会影响到建议更正词汇的展示顺序。在拼写校正过程中不考虑上下文，也使得在查询中的一些拼写错误被忽略了，因为这些拼写错误产生了另一个词。例如，在查询“miniature golf curses”时，显然是单个字符删除错误的例子，但是其中的拼写错误产生了词“curses”，这个词本身是正确的，所以就不能检测到这个查询中的拼写错误。

采用“Did you mean...”方式的典型界面中，要求拼写检查器生成一个简单的并且最优的建议。这意味着查询的拼写检查与字处理的拼写检查相比，利用上下文和频率信息去排序更正建议是更重要的，在字处理拼写检查中的更正建议，可以采用下拉列表的形式。另外，在查询中还包含着大量的语义连贯方面的错误（run-on error），其中词的边界被省略了或输入错了。下面这两个查询“ultimatwarcade”和“mainscourcebank”，是语义连贯错误的例子，其中也包含了单个字符的错误。用一些恰当的结构，将遗漏了一些像空格这样的分隔符可以看作是另一种单个字符的错误。

用于拼写校正的噪声通道模型（noisy channel model），是一种能够处理排序、上下文和语义连贯方面错误问题的通用结构。这个模型称为“噪声通道”，是因为它是基于香农的通信理论（Shannon & Weaver，1963），这个模型可以通俗地描述为：一个人在直觉上要输出（即写出）词w，是根据概率分布$P(w)$，然后这人要去写出词w，但是声音通道（大体上相当于人的大脑）使得这个人以概率$P(e|w)$写了词e。

概率$P(w)$称为语言模型（language model），用来获得在文本中词的出现频率信息（例如，在一个文档或查询中，词“lawyer”出现的概率是多少？），以及上下文信息，如在已知一个词出现的情况下，观察到另一词的概率（例如，词“lawyer”在词“trial”之后的概率）。我们将在第7章中更详细地介绍语言模型，目前我们可以将这个模型假定为描述了词出现的概率。

概率$P(e|w)$称为错误模型（error model），表示了不同类型拼写错误的频率信息。例如，与词w的编辑距离为1的词（或者字符串）概率将会非常高。词之间的编辑距离越大，概率值会越低，但是同音词也有很高的概率。需要注意，错误模型不仅对于拼写错误的词估计概率，而且对于拼写正确的词也有概率值（$P(w|w)$）。这使得拼写校正程序对于所有的词都提出更正建议，尽管最初的一些词是拼写正确的。这样，如果具有最高概率更正的词是相同的词，那么就不对用户提出更正建议。但是，如果上下文（即语言模型）提示了另一个词可能更恰当，那么就提出这个更正建议。概括地说，这就是拼写检查程序对于查询“golf curse”是怎么实现用“course”来代替“curse”的校正建议。

如何来估计拼写校正的概率呢？某个人写的词是e，所以需要计算$P(w|e)$，这是在看到这个人写的词是e的条件下，而正确的词是w的概率。如果只是想要找到具有最大概率值的拼写更正，或者想要排序这些拼写更正，就可以用$P(e|w)\ P(w)$来计算，这是错误模型概率和语言模型概率的乘积[⊖]。

为了能够处理语义连贯方面的错误和上下文，语言模型除了需要单个词的信息外，还需要获得词对的信息。一个词的语言模型概率可以采用这个词在文本中出现的概率和它接着前

⊖ 贝叶斯定理（Bayes's Rule）是用基于部分概率来表示$P(w|e)$，第7章中将讨论这个定理。

一个词出现的概率混合形式（mixture）来计算，或者按照下面的公式：

$$\lambda P(w)+(1-\lambda)P(w|w_p)$$

其中λ是描述两个概率相对重要程度的参数，$P(w|w_p)$是词w在词w_p之后出现的概率。例如对于“fish tink”的拼写错误，对于那些可能的更正，我们将错误模型概率乘以它们的语言模型概率。词“tank”和“think”具有很高的错误模型概率，因为它们只需要单个字符的纠正，并且它们都是很常用的词，这两个词也有相似的$P(w)$，但是概率$P(\text{tank}|\text{fish})$会比概率$P(\text{think}|\text{fish})$高很多，这样，与“think”相比，“tank”更可能是校正的词。

语言模型中需要的信息从什么地方获取呢？在一些应用中，对文本中词的统计的最好资源是采用搜索到的文档集合。在网络搜索中（以及其他的一些应用中）会有查询日志，其中包含了大量提交到搜索引擎的查询。既然我们的任务是纠正查询的拼写错误，查询日志可能是最好的信息资源。语言模型需要记录成对的词的信息，与分析一个大规模的文档集合中所有可能的词对相比，分析查询日志也会减少这些词对的数量。除了这些资源，对于这个应用，如果有一个可信的字典，也应该使用这样的字典。

在错误模型中$P(e|w)$概率的估计可以采用相对简单的方法，也可以用非常复杂的方法。简单的方法是假定所有具有相同编辑距离的错误有相等的概率，并且仅考虑在一个确定的编辑距离以内的字符串（通常取编辑距离为1或2）。更复杂的方法是对于一些确定类型的错误发生可能性进行概率估计，例如，当想要输入“e”时输入了“a”，这些概率估计是通过对大规模的文本集中查找一些正确拼写和不正确拼写的词对获得的。

Cucerzan和Brill（2004）描述了从查询日志和字典中的信息对查询进行拼写检查的迭代过程，下面是这些步骤的简要描述：

1）对查询进行词素切分（Tokenize）。

2）对于每个词素（token），通过编辑距离获得可替换的词和词对的集合。其中，如前所述对每种确定类型的错误，给予不同的权重来计算编辑距离。在查找可替换词的数据结构中，包含了查询日志和词典中的词与词对。

3）然后，使用噪声通道模型选择最优的更正。

4）寻找可替换词的过程与发现最优更正的过程不断重复进行，直到不能发现更好的纠正为止。

通过进行多次迭代，拼写检查程序可能会产生出与最初的查询相差很远（指编辑距离）的更正建议。例如，对于查询“miniture golfcurse”，拼写校正程序将经历下面的迭代过程：

miniture golfcurses
miniature golfcourses
miniature golf courses

这种拼写校正程序的相关实验结果显示，从查询日志中得到的语言模型对于校正准确率来说是最重要的部分，另外，在语言模型中，以词对形式使用的上下文信息也是很关键的。在校正过程中，即使仅有两次迭代，也会产生很大的不同。错误模型不是很重要的，使用一个对所有错误都具有相同概率的简单模型与更复杂的模型几乎同样有效。其他的一些研究表明，当不是根据查询日志，而是基于文档集合得到语言模型时，错误模型会更加重要。

实现查询拼写校正程序的最好方法，显然需要依靠一些可用的数据。如果有大量的查询

日志信息，就可以将它们使用到程序中。否则，在这些应用中的文档集合以及在某些情况下一个可信的字典，也都是可用的资源。在一些常规的拼写更正程序中，可以使用构建具体应用词典的方法，如Aspell。即使没有可用的查询日志数据，基于噪声通道模型的拼写校正程序也会非常有效并更具适应性。

6.2.3 查询扩展

在搜索引擎发展的早期阶段，从20世纪60年代开始，搜索引擎系统用户的一个基本工具是在线叙词表（thesaurus）。叙词表中描述了在文档集合中被索引的词汇，以及同义词、相关词或短语的信息，它是非常重要的，因为文档集通常根据叙词表中的词用人工进行了索引（标记，tagged）[⊖]。在叙词表中词项经过了仔细的选择并进行了质量控制，所以叙词表也称为受控词表（controlled vocabulary）。使用叙词表，用户可以决定要在查询中用到哪些词或短语，并且能够用同义词和相关词来扩展最初的查询。

表6-1展示了在医学主题词（MeSH）叙词表中的一个词条的相关部分。这个叙词表用在国家图书馆中医药搜索上[⊜]。表中“Tree Number”是用编号形式指出在广义和狭义的词项分类树上，什么地方能够找到这个词项。“Entry term”是一个同义词或相关的短语。

表6-1 医学主题词（MeSH）叙词表中“Neck Pain”的部分词条

MeSH Heading	Neck Pain
Tree Number	C10.597.617.576
Tree Number	C23.888.592.612.553
Tree Number	C23.888.646.501
Entry Term	Cervical Pain
Entry Term	Neckache
Entry Term	Antcrior Cervical Pain
Entry Term	Anterior Neck Pain
Entry Term	Cervicalgia
Entry Term	Cervicodynia
Entry Term	Neck Ache
Entry Term	Posterior Cervical Pain
Entry Term	Posterior Neck Pain

尽管在当前的搜索应用中，很少直接使用叙词表，但是在自动和半自动（semi-automatic）形式的查询扩展中，也用到了一些相关的技术。半自动技术需要用户与系统进行交互，例如，用户可以在系统提供的扩展词列表中选择扩展词项，网络搜索引擎将用户的初始查询用一个或多个词去扩展，或者替换查询中的一些词，然后将修改后的查询建议提交给用户。

查询扩展技术通常是基于对指定的文档集中词或词项共现的分析，文档集可以采用全部的文档集合、大规模的查询集合，或者在排序结果中最高排序的文档集。从这个角度上，对查询的词干提取也可以看作是一种查询扩展技术，只是扩展的词项限于对词的变形（word variants）。另外，采用常规叙词表（例如Wordnet[⊝]）的自动扩展技术，往往也不具备很好的效果。

有效进行查询扩展的关键，是要选择出适合查询上下文或主题的词汇。例如，“aquarium”可以是对查询“tropical fish tanks”中“tank”的一个很好的扩展，但是对于查询“armor for tanks”并不适合。一个常规叙词表中列出了不同上下文的相关词项，所以很难自动地使用它。我们将叙述处理这个问题的不同方法，如用查询中的全部词去找到相关的词，而不是分别扩

⊖ 在信息检索中，indexing 除了指创建用于搜索文档的索引的过程，而且还经常指用一个索引词项代表一个文档的过程。最近，人工创建索引的过程称为标记（tagging），特别在社会搜索应用的上下文中（见第10章）。

⊜ http://www.nlm.nih.gov/mesh/meshhome.html.

⊝ http://wordnet.princeton.edu/.

展每一个词。一个众所周知的扩展技术称为伪相关反馈（pseudo-relevance feedback），将在下一节与基于用户对检索结果中相关文档的反馈技术一起进行论述。

衡量词项相关性是一些查询扩展方法中的重要部分，目前已经提出了一些衡量方法，在6.2.1节中提到的戴斯系数（Dice's coefficient）是其中的一种方法。这个衡量方法的公式如下：

$$\frac{2\cdot n_{ab}}{n_a+n_b} \overset{rank}{=} \frac{n_{ab}}{n_a+n_b}$$

其中 $\overset{rank}{=}$ 指公式是排序相等的（产生了相同的词项排序）[⊖]。

另外一种度量方法是互信息（mutual information），这种方法被用在一些词搭配（collocation）的研究中。对于两个词（或词项）a和b，互信息用下面公式定义：

$$\log\frac{P(a,b)}{P(a)P(b)}$$

互信息用来衡量词之间能够相互独立出现的程度[⊖]。$P(a)$是在给定大小的文本窗口中词a出现的概率，$P(b)$是词b在文本窗口中出现的概率，$P(a, b)$是词a和词b出现在相同的文本窗口中的概率。如果两个词的出现是独立的，那么就有$P(a, b)=P(a)P(b)$，互信息为0，例如，我们可能希望两个词“fishing”和“automobile”将互相独立地出现。如果两个词趋向于共现，如“fishing”和“boat”，P(a,b)将大于$P(a)P(b)$，互信息将会增大。

为了计算互信息，我们采用下面简单的归一化频率形式来估计概率：$P(a)=n_a/N$，$P(b)=n_b/N$，$P(a, b)=n_{ab}/N$，其中n_a是包含词a的窗口（文档）数量，n_b是包含词b的窗口（文档）数量，n_{ab}是包含词a和词b的窗口数量，N是集合中的文本窗口数量，这样就有下面的计算公式：

$$\log\frac{P(a,b)}{P(a)P(b)} = \log N\cdot\frac{n_{ab}}{n_a\cdot n_b} \overset{rank}{=} \frac{n_{ab}}{n_a\cdot n_b}$$

这种方法的一个问题是，它更倾向于具有低频率的词。例如，两个词都具有频率10（即$n_a=n_b=10$），它们共现频率为各自出现频率的一半（$n_{ab}=5$），根据上面公式计算的对于这两个词的关联度为5×10^{-2}。但是对于两个词的频率为1 000，共现频率为半数时（$n_{ab}=500$），则这两个词的关联度为5×10^{-4}。期望互信息度量方法（expected mutual information measure）根据概率$P(a, b)$对互信息进行加权处理，以解决这个问题。尽管在通常情况下，期望互信息要基于所有共现和非共现的词进行计算，但是我们主要关注两个词共现的情况，如下面公式：

$$P(a,b)\cdot\log\frac{P(a,b)}{P(a)P(b)} = \frac{n_{ab}}{N}\log\left(N\cdot\frac{n_{ab}}{n_a\cdot n_b}\right) \overset{rank}{=} n_{ab}\cdot\log\left(N\cdot\frac{n_{ab}}{n_a\cdot n_b}\right)$$

我们还采用与上面相同的例子，假设$N=10^6$，这样对于低频词计算的关联度为23.5，而对于高频词计算的关联度为1350，明显更倾向于后者。事实上，偏向于高频词是这种方法的一个问题。

在一些应用中，另一种常用的关联度量方法是皮尔森χ^2检验方法（pearson's Chi-squared(χ^2) measure）。这个方法计算了两个词共现的观测值与这两个词在相互独立条件下共

⊖ 更规范地，当根据两个函数的函数值对词项进行排序后，获得了具有相同顺序的词项，则称这两个函数是排序相等的。排序保持操作包括单调变换（如log），缩放（乘以一个常数），以及平移（增加一个常数）。

⊖ 更准确地说，这是指点（pointwise）互信息。

现的期望值的比值，并根据期望值对这个比值进行归一化处理，得到：

$$\frac{\left(n_{ab}-N\cdot\frac{n_a}{N}\cdot\frac{n_b}{N}\right)^2}{N\cdot\frac{n_a}{N}\cdot\frac{n_b}{N}} \overset{rank}{=} \frac{\left(n_{ab}-\frac{1}{N}\cdot n_a\cdot n_b\right)^2}{n_a\cdot n_b}$$

$N\cdot P(a)\cdot P(b)=N\cdot\frac{n_a}{N}\cdot\frac{n_b}{N}$是两个词相互独立情况下共现的期望值。与期望互信息度量方法类似，χ^2检验通常根据所有词共现和非共现的情况进行计算，但是我们只着重考虑两个词共现的情况。事实上，当N值很大的时候，这种限制形式的χ^2检验与完全形式的χ^2检验产生了相同的词项排序结果。另外，可以注意到χ^2检验方法与互信息度量方法非常相似，并且都倾向于低频词。

表6-2概括了以上我们所讨论的衡量关联度的方法。为了展示它们实际的应用，我们在TREC新闻集上采用这些方法计算了与查询“tropical fish”相关的排序靠前的词项。

表6-2 词项关联度量方法

度量	公式
互信息(MIM)	$\frac{n_{ab}}{n_a\cdot n_b}$
期望互信息(EMIM)	$n_{ab}\cdot\log\left(N\cdot\frac{n_{ab}}{n_a\cdot n_b}\right)$
χ^2	$\frac{\left(n_{ab}-\frac{1}{N}\cdot n_a\cdot n_b\right)^2}{n_a\cdot n_b}$
戴斯系数（Dice）	$\frac{n_{ab}}{n_a+n_b}$

表6-3中是在不限制窗口大小情况下与词“tropical”有很高关联度的词（换言之，在文档中词共现的数量），其中有两个明显的特点需要注意，第一个是χ^2检验的排序与MIM方法的排序是相同的，第二个是MIM和χ^2检验如预期的那样，更倾向于低频词。有些词是不合理的（例如“itto”是指国际热带木材组织，“xishuangbanna”是中国热带植物公园），它们过于具体，不能用在查询扩展中。在EMIM和Dice方法中，排在前面的词是一些常规词，但在EMIM方法中，有时出现的词过于“常规”（例如，“most”）。

表6-4中是对于“fish”获得的排序靠前的一些词。因为这是一个高频词，尽管MIM和χ^2检验方法仍然都倾向于低频词，但获得的排序不再相同。由EMIM和Dice方法获得的排序在前的词非常相似，但排序不同。

为了显示改变窗口大小的影响，表6-5给出了用5个词大小的窗口获得的排序靠前的词。尽管MIM和χ^2检验方法仍能找到一些低频词，但是窗口尺寸变小影响了排序结果。EMIM方法获得的词有了些改善，变得更具体了。总体上，在一定的窗口大小范围内，简单的戴斯系数方法看来是最稳定和可靠的。

但是在这些表格中，即使最好的排序也几乎没有包含能够用作扩展“tropical fish”的词，表格中的这些词往往与一些具体的上下文相关，例如，热带雨林和水果或者禁渔。一种解决这个问题的方法是，找到与短语“tropical fish”密切相关的词汇。采用戴斯系数方法，使用

前面所述的TREC文档集，得到下面前10个排序的词：

表6-3 在TREC新闻集中与“tropical”最相关联的词（在文档上衡量词共现的数量）

MIM	EMIM	χ^2	Dice
trmm	forest	trmm	forest
itto	tree	itto	exotic
ortuno	rain	ortuno	timber
kuroshio	island	kuroshio	rain
ivirgarzama	like	ivirgarzama	banana
biofunction	fish	biofunction	deforestation
kapiolani	most	kapiolani	plantation
bstilla	water	bstilla	coconut
almagreb	fruit	almagreb	jungle
jackfruit	area	jackfruit	tree
adeo	world	adeo	rainforest
xishuangbanna	america	xishuangbanna	palm
frangipani	some	frangipani	hardwood
yuca	live	yuca	greenhouse
anthurium	plant	anthurium	logging

goldfish，reptile，aquarium，coral，frog，exotic，stripe，regent，pet，wet

显然，这个方法找到了更好地与上下文相关的词。可是为了使用这个方法，我们要对可用于查询中的每组词分析其关联性，这显然是不现实的，但是可以采用一些其他的方法来完成这件事情。

表6-4 在TREC新闻集中与“fish”最相关联的词（在文档上衡量共现的数量）

MIM	EMIM	χ^2	Dice
zoologico	water	arlsq	species
zapanta	species	happyman	wildlife
wrint	wildlife	outerlimit	fishery
wpfmc	fishery	sportk	water
weighout	sea	lingcod	fisherman
waterdog	fisherman	longfin	boat
longfin	boat	bontadelli	sea
veracruzana	area	sportfisher	habitat
ungutt	habitat	billfish	vessel
ulocentra	vessel	needlefish	marine
needlefish	marine	damaliscu	endanger
tunaboat	land	bontebok	conservation
tsolwana	rivet	taucher	river
olivacea	food	orangemouth	catch
motoroller	endanger	sheepshead	island

一种方法是对查询检索出的文档进行词共现的分析，这是伪相关反馈的基础，我们将在下一节论述这个方法。另一种方法是根据与指定词共现的其他词的分析，对文档集中的每个

词进行索引，创建出表示这个词的一个虚拟文档[⊖]。例如，下面的例子是35个与“aquarium”最密切相关的词（采用戴斯系数计算）：

zoology, cranmore, jouett, zoo, goldfish, fish, cannery, urchin, reptile, coral, animal, mollusk, marine, underwater, plankton, mussel, oceanography, mammal, species, exhibit, swim, biologist, cabrillo, saltwater, creature, reef, whale, oceanic, scuba, kelp, invertebrate, park, crustacean, wild, tropical.

表6-5 在TREC新闻集中与“fish”最相关联的词（在5个词的窗口上衡量共现的数量）

MIM	EMIM	χ^2	Dice
zapanta	wildlife	gefilte	wildlife
plar	vessel	mbmo	vessel
mbmo	boat	zapanta	boat
gefilte	fishery	plar	fishery
hapc	species	hapc	species
odfw	tuna	odfw	catch
southpoint	trout	southpoint	water
anadromous	fisherman	anadromous	sea
taiffe	salmon	taiffe	meat
mollie	catch	mollie	interior
frampton	nmf	frampton	fisherman
idfg	trawl	idfg	game
billingsgate	halibut	billingsgate	salmon
sealord	meat	sealord	tuna
longline	shellfish	longline	caught

这些词将构成表示“aquarium”文档的索引项。为了找到一个查询的扩展词，这些虚拟文档按照与正规文档相同的形式排序，这样得到对应于每个词的排序。在这个例子中，“aquarium”这个文档以很高的权重包含词“tropical”和“fish”，所以它可能对于查询“tropical fish”会有更高的排序，这意味着“aquarium”将是具有更高排序的扩展词项。另外，对于“jungle”这个词的文档，尽管以很高的权重包含“tropical”，但是不太可能包含“fish”，这个文档以及对应词的排序将远低于“aquarium”。

分析文档集的各种技术都要面对可计算性和准确性的问题，因为在搜索应用中的文档集往往是规模庞大并且文档质量参差不同。正如本节开始部分提到的，不是对整个文档集而是对搜索结果集合或大规模的查询日志进行分析。最近的一些研究经验指出，大规模的查询日志或许是最好的查询扩展资源。在这些查询日志中包含的是一些短文本，这比包含全部文本的文档更容易进行分析，其中也包含了一些其他数据，如在检索过程中被用户点击的文档的信息（即点击流数据（clickthrough））。

为了展示如何使用查询日志进行查询扩展，下面列出与包含“tropical fish”的查询相关的10个最高频率的词，这些查询是从一个流行的网络搜索引擎上最近的查询日志中获得的。

stores, pictures, live, sale, types, clipart, blue, freshwater, aquarium, supplies

⊖ 有时称为上下文向量（context vector）。

这些词指出了人们提出与热带鱼相关查询的类型（销售、供应、图片），其中大部分非常适合用作查询扩展。在当前的一些系统中，通常是以完整查询的形式给出查询扩展的建议，而不是以单个词的形式，在这种情况下，查询日志对于生成最好的提议将是非常有用的。例如，“tropical fish supplies”是比“supplies tropical fish”更常见的查询，也是对查询扩展更好的建议。

从这个角度上看，查询扩展能够作为发现相似查询的问题，而不是扩展词项的问题。相似的查询不总是包含相同的词。例如，查询“pet fish sales”可以作为对“tropical fish”的一个合理的候选扩展，尽管它没有包含词“tropical”。长期以来，人们已经认识到，语义相似的查询不仅可以根据词，而且可以根据它们具有相同内容的相关文档将其组合起来。点击流数据非常相似于这样的相关数据，最近的研究表明，能够根据点击流数据的相似程度将查询很好地进行组合或聚类。这意味着每个查询都可以用这个查询的点击网页集合来表示，并且查询之间的相似度可以采用一些方法来计算，如采用戴斯系数的方法，其中n_{ab}是两个查询具有的相同的点击网页数量，n_a、n_b是每个查询的点击网页数量。

总之，目前已经有很多自动或半自动的查询扩展方法，在一些搜索应用上，默认的方法是向用户提出可选择的查询建议。有些计算词项关联度的方法要优于其他方法，基于单个词的词项关联度方法没有产生很好的扩展词项，因为它没有使用查询的上下文信息。使用查询上下文最好的方法是，使用查询日志去分析词之间的关联以及根据点击流数据去找到相似的查询。如果没有可用的查询日志，最好的替代方法是采用伪相关反馈，这将在下一节中介绍。在基于文档集合构建自动叙词表的方法中，最优的方法是为每个词创建虚拟文档，并且在扩展每个查询时对它们进行排序。

6.2.4 相关反馈

相关反馈是具有很长历史的查询扩展和查询提炼技术。最初是在20世纪60年代提出的，它是依靠系统与用户的交互过程，识别出在用户初始查询的排序文档中的相关文档。尽管在上节中也讨论了一些半自动技术，但是在相关反馈中，不是让用户从词项列表中或可替换的查询中进行选择，而是让用户指出哪些文档是感兴趣的（即相关的），以及哪些是完全离题的（即不相关的）。根据这些信息，系统通过增加词项或对原始词项重新分配权重，自动地改写查询，并用改写的查询生成新的文档排序。

这个处理过程是在信息检索中应用机器学习（machine learning）方法的一个简单例子，其中通过训练数据（training data）（识别相关和不相关的文档）改善系统性能，修改查询事实上等同于学习一个分类器，通过它可以区分相关和不相关的文档。我们将在第7章和第9章进一步讨论分类和分类技术。可是与其他的机器学习应用相比，相关反馈产生的训练数据的数量是非常有限的，因为它只能依靠用户在当前查询期间的输入内容，而不能使用像点击流这样的历史数据。

调整查询的具体方法取决于所用的检索模型。在下一章中，我们将介绍在向量空间模型和概率模型中是如何使用相关反馈技术的。但通常的做法是，当一些词在相关文档中出现的频率比不相关文档或整个文档更高时，就将这些词增加到查询中，或者提高它们的权重。在伪相关反馈（pseudo-relevance feedback）中，也是采用了相同的思想，但不是让用户自己去识别相关的文档，而是系统将排序靠前的文档假设是相关的。在这些文档中，频繁出现的词

被用作扩展用户的初始查询，具体的做法也要取决于所采用的检索模型。在下一章中，我们将描述采用语言模型检索中的伪相关反馈。由伪相关反馈产生的扩展词项是根据整个查询的，因为它们是在这个查询的排序靠前的文档中抽取出来的，但是扩展的质量是由排序靠前的文档有多少是实际相关而决定的。

我们通过一个例子来说明这个方法是如何实现的，图6-1是一个通用的搜索引擎对查询“tropical fish”的搜索排序结果。

1. **Badmans Tropical Fish**

   A freshwater aquarium page covering all aspects of the **tropical fish** hobby. ... to Badman's **Tropical Fish**. ... world of aquariology with Badman's **Tropical Fish**. ...

2. **Tropical Fish**

   Notes on a few species and a gallery of photos of African cichlids.

3. **The Tropical Tank Homepage - Tropical Fish and Aquariums**

   Info on **tropical fish** and **tropical** aquariums, large **fish** species index with ... Here you will find lots of information on **Tropical Fish** and Aquariums. ...

4. **Tropical Fish Centre**

   Offers a range of aquarium products, advice on choosing species, feeding, and health care, and a discussion board.

5. **Tropical fish - Wikipedia, the free encyclopedia**

   Tropical fish are popular aquarium **fish** , due to their often bright coloration. ... Practical Fishkeeping • **Tropical Fish** Hobbyist • Koi. Aquarium related companies: ...

6. **Tropical Fish Find**

   Home page for **Tropical Fish** Internet Directory ... stores, forums, clubs, **fish** facts, ***tropical fish*** *compatibility and aquarium* ...

7. **Breeding tropical fish**

   ... intrested in keeping and/or breeding **Tropical**, Marine, Pond and Coldwater **fish**. ... Breeding **Tropical Fish** ... breeding **tropical**, marine, coldwater & pond **fish**. ...

8. ***FishLore***

   Includes **tropical** freshwater aquarium how-to guides, FAQs, **fish** profiles, articles, and forums.

9. **Cathy's Tropical Fish Keeping**

   Information on setting up and maintaining a successful freshwater aquarium.

10. **Tropical Fish Place**

    Tropical Fish information for your freshwater **fish** tank ... great amount of information about a great hobby, a freshwater **tropical fish** tank. ...

图6-1 查询“tropical fish”的前10个搜索结果

为了使用伪相关反馈扩展这个查询，我们可以假设前10个文档都是相关的。通过对这些文档进行全文分析，可以得到其中出现最频繁的词项以及相应的频率：

a (926), td (535), href (495), http (357), width (345), com (343), nbsp (316), www (260), tr (239), htm (233), class (225), jpg (221)

显然，这些词不适合用作扩展的词项，因为它们是整个文档集合中一些常见的停用词和HTML的字符串，也就是说，它们不能表示出这些文档所包含的主题。改进这个过程的一个简单方法是，在文档的页面摘要中对词计数，并且去除停用词。这样的分析方法得到了下面这些高频词：

tropical (26), fish (28), aquarium (8), freshwater (5), breeding (4), information (3), species (3), tank (2), Badman's (2), page (2), hobby (2), forums (2)

这些词是用作查询扩展的更好的候选词项，并且没有出现分别扩展“tropical”和“fish”时缺少上下文信息的问题。但是如果用户是对饲养热带鱼感兴趣，就可以用真正的相关反馈去改善扩展的词项，排序在第7位的文档被明确地标记为相关文档，这种情况下出现最频繁的词是：

breeding (4), fish (4), tropical (4), marine (2), pond (2), coldwater (2), keeping (1), interested (1)

用这些词的主要作用是增加扩展词项“breeding”的权重。如前所述，具体的加权方法取决于采用的检索模型。

在一些研究文献中，已经对相关反馈和伪相关反馈进行了广泛的探讨，并且显示了它们是改善排序结果的有效方法。但是在实际的检索应用中，却很少被使用它们，对于伪相关反馈，显然主要是由于无法预料自动处理过程的结果。如果最初的排序中没有包含相关的文档，通过伪相关反馈获得的扩展词项不可能有什么帮助，而且对于某些查询，可能会产生更严重的错误排序结果。为了避免这样的情况，候选的扩展词项可以展示给用户，但是一些研究显示了这种做法也不是特别有效。根据对查询日志的分析，提议可选择的查询是一种更为可靠的半自动查询扩展替代方法。

另外，相关反馈也可以用在其他应用中，如文档过滤。过滤涉及追溯一个人随着时间而改变的兴趣，并且在有些应用中，允许人们使用相关反馈去调整他们的个人描述文件。另一个相关反馈的简单应用是，在一些早期的搜索引擎中所采用的“more like this”功能，这个功能允许用户点击在搜索结果列表中与文档相关联的一个链接，这样可以产生与点击文档相似的另一个文档排序列表。这是一个相关反馈过程，但是对于训练数据仅限于一个相关文档。

尽管这些应用已经获得了一些成功，但是在目前更为常用的方法是，要求用户从一系列查询提议中去选择查询。当然不能保证这些提议的查询中就会包含用户想要查找的内容，并且在这个意义上，相关反馈支持了更精确的查询改写。但是使用相关反馈隐含着一个内在的假设：用户需要寻找的是一些相关文档，而不是在初始排序结果中的一个或两个文档。对于有些查询，例如寻找一个主题的背景信息，这可能是正确的，但是对于在互联网环境下的一些查询，用户将会更满意初始的排序结果，而不需要相关反馈。当初始查询失败时，给出的查询建议列表将会帮助用户重新提交查询，而对于这种情况，相关反馈不可能有什么帮助。

6.2.5 上下文和个性化

当前搜索引擎的一个特点是，对于相同的查询给出相同的检索结果，而不管是谁提交的查询，为什么提交查询，在什么地方提交的查询，在一个对话期间中还提交什么其他的查询。所有这些事项就是要选择可以用哪些词来描述当前的查询。另外还有一些统称为查询上下文的因素，也会影响检索出文档的相关性，并能够对排序算法产生很大影响。但是已经证实大部分的上下文信息是难以获取的，并且很难用对改善排序结果始终有效的方式进行表达。

有一些研究通过学习用户模型（users model）或者描述文件去表示用户的兴趣，使得搜索能够个性化（personalized）。例如，如果系统知道一个人对体育感兴趣，那么对他提出的查询“vikings”，检索出的文档可能会与另一个对历史感兴趣的人用同样的查询检索出的文档不同。尽管这个想法很吸引人，但是实现起来却存在很多问题。首先是用户模型的准确程度。最常见的建议是，根据用户查看的文档建立描述文件，例如访问的网页，邮件信息，或者桌

面上的字处理文档。这样的描述文件用不同权重的词来表示一个用户。与这个用户相关的文档中频繁出现的词，但又不是一些常见词，将被赋予很高的权重。但是在文档中会包含几百个甚至几千个词，并且用户访问的文档只能代表他们兴趣的一个方面，所以这个模型不是很明确。实验表明，采用这样的模型基本上并没有改善排序效果。

一个可替代的方法是让用户根据预定义的类别对自己进行描述。对大部分用户来说这除了增加一些想要避免的额外的（和可选择的）交互过程，还有一个基本的问题，就是通常对于体育感兴趣的人，也可能想要问关于历史的问题。对此，每个查询都应该具体到所属的兴趣类别，例如将查询“vikings”限制到“history”历史类别中，但是这与输入一个有更少歧义的查询并没有什么区别。用户多输入一两个词去澄清他的查询，要比试图将一个查询分类到某个指定的类别中会更有效，例如“vikings quarterbacks”或“vikings exploration”。

基于用户模型实现个性化的方法，涉及的另一个问题是隐私（privacy）。人们已经开始担忧存在公司或政府部门数据库中的个人详细信息。以匿名形式搜索和浏览网页已经逐渐成为研究和开发中的共识，所以在搜索引擎中，根据用户上网行为创建描述文件可能会不受欢迎，尤其目前，这种做法的益处还不是很明确。

用户模型和隐私上的问题，并不意味着上下文的信息没有用，而是对于任何使用上下文方法获得的益处，都要仔细地检查。在一些应用中，使用上下文信息明显是有效的，其中一种是使用检索记录和点击流数据去改善网络搜索性能，在这种应用中，上下文是指先前相同或相似的搜索历史，通常这个搜索历史是基于全部的用户群体。个别用户的检索历史也可以用作对整体检索查询的“缓存”结果，但是使用用户群体的大量查询显然会更加有效。

另一个有效使用上下文的应用是本地搜索（local search），它是从查询中或者从提交查询设备的位置信息中获得的地理信息，并根据这些信息调整排序结果。例如，查询“fishing supplies”会产生大量的来自于各个国家（或全球）供应信息的相关网页。但是查询“fishing supplies Cape Cod”应该能够利用“Cape Cod”提供的上下文信息，使这个区域的供应商获得更高的排序。与之相类似，如果查询“fishing supplies”是在科德角（cape cod）的一个城镇上通过移动设备提交的，那么根据获得的位置信息，也可以对临近这个城镇的供应商进行排序。

基于查询的本地搜索包含以下步骤：

1）识别与网页相关联的地理区域。这可以通过人工加入文档中的位置元数据进行识别，或者通过自动方式识别位置，例如，在文档中的地名、城市名称、国家名称等。

2）采用一些自动技术识别出与查询相关联的地理区域。根据对查询日志的分析显示，10%~15%的查询中包含位置相关信息。

3）将查询和文档中位置信息的对比结果结合，通常采用文本特征和链接特征，对网页进行排序。

自动识别文本中的位置信息是第4章中介绍的信息抽取技术的一个具体例子。位置名称通过采用地理学本体[⊖]（geographic ontology）以及地理信息系统（geographic information system）中的空间推理算法，映射到一个具体的区域和坐标上。例如，文档中出现的位置“Cape Cod”可以被映射到图6-2中所示的用经度和纬度限定的矩形框中，而城镇的位置可以

⊖ 本体与叙词表基本上相同，它是领域中概念以及概念之间关系的表达。叙词表描述了词、短语以及它们之间的关系，本体通常比叙词表有更丰富的关系集合。Taxonomy是本体中使用的一个术语，用于描述概念的类别。

对应于一个更具体的坐标（或者更小的矩形框）。尽管这听起来非常直接，但是也存在一些需要处理的问题，例如辨别地理名称的具体位置（例如，在美国有超过35个地方叫做斯普林菲尔德（Springfield）），决定位置信息是否有意义（如果网页内容是讨论“problems with Washington lobbyists”，那么“Washington”是否应该作为一个位置元数据呢？），以及如何结合文档中多个位置参照信息。

在排序中，地理信息对比涉及处理一些空间关系，如“包含”（例如，供应商网页上的位置元数据指出供应商是位于代表科德角的边界框中），“距离”（例如，供应商的位置距离查询中提到的城镇10英里以内），以及一些其他空间关系。怎样将地理信息恰当地纳入到排序中，以及如何在恰当的时候使用地理信息，将会对系统的效率和效能产生影响。

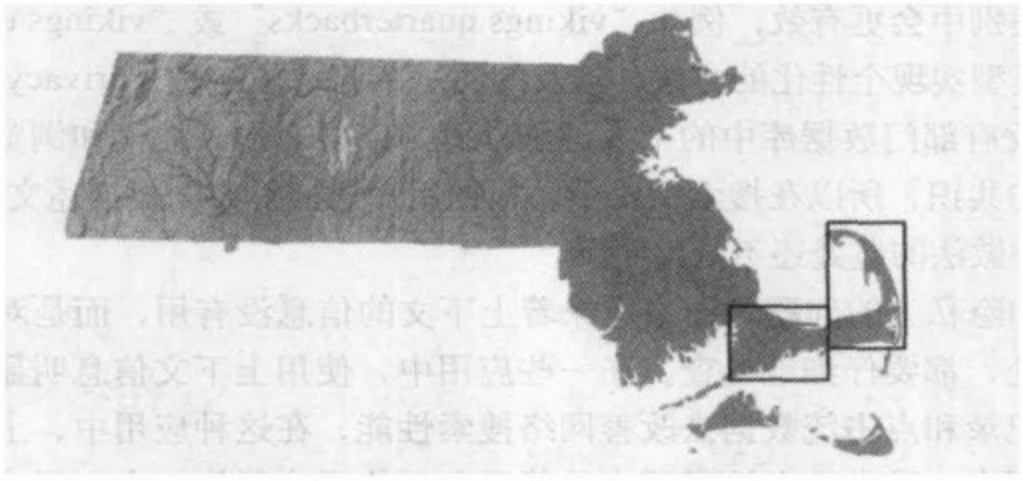


图6-2 矩形框表示的科德角地理位置

总之，对于改善搜索质量最有效的上下文信息是，在搜索引擎中已经发生的交互过程（即查询日志和搜索会话历史）。使用了地理上下文信息的本地搜索也能够对一部分查询产生实质性的改善。查询扩展通过增加词项丰富了查询，本地搜索提供了地理上距离的信息，这两种做法都使用上下文信息作为补充的特征改善了初始的查询。可是对于理解一个具体查询的上下文来说，没有什么方法可以替代用户提交一个更加具体的查询。事实上，本地搜索的大部分情况是依靠查询中叙述的位置信息，通常情况是用户检查搜索结果，然后改写查询，在仍没有得到满意的结果时，会提交更加具体的查询。我们将在下一节讨论搜索结果的显示，搜索结果必须要显示出与查询词项匹配的上下文，使得用户能够知道哪些文档应该仔细阅读以及如何重写查询。

6.3 搜索结果显示

6.3.1 搜索结果页面与页面摘要

用户与搜索引擎之间是否能够成功地进行交互，取决于用户对检索结果的理解。尽管已经提出了各种展示检索输出结果的可视化技术（Hearst 1999），但是大部分的搜索引擎采用排序的文档摘要列表构成检索结果页面，这些文档摘要（document summaries）与实际的文档或页面相链接。网络搜索的文档摘要通常包括页面的标题、URL、真实页面和页面快照的链接，还有更为重要的简短文本摘要或者称为页面摘要（snippets），用来传达对应页面中的内容。另外，大部分的结果页面中，还包含一些广告的简短描述和链接。在标题、URL、页面摘要和广告中的查询词被突出显示，使得能够更容易地辨别它们，这些词通常用黑体显示。

图6-3是网络搜索结果页面中的一个文档摘要的例子，其中页面摘要由两个句子中的一部

分组成。图6-1展示了更多的页面摘要的例子，其中有些页面摘要是一些完整的句子，但更多的是一些文本片段，它们是从网页上抽取出来的，有些页面摘要甚至不包含查询中的词汇。在本节中，我们将描述页面摘要生成算法的一些基本特点。

Tropical Fish
One of the U.K.s Leading suppliers of **Tropical**, Coldwater, Marine **Fish** and Invertebrates plus.. . next day **fish** delivery service ...
www.**tropicalfish**.org.uk/**tropical_fish**.htm Cached page

图6-3 网络搜索文档摘要示例

页面摘要生成是自动文摘的一个应用实例。尽管自动文摘技术是对于一些应用进行开发的，但主要还是用TREC文档集的新闻语料进行测试。自动文摘技术可以分为查询无关文摘（query-independent）和查询相关文摘（query-dependent）。搜索引擎结果页面中，页面摘要显然是查询相关文摘，因为用于生成页面摘要（snippet）的页面是根据查询检索得到的。但是在页面摘要生成过程中，也采用了一些查询无关文摘中的方法，例如考虑文本在页面中的位置以及文本是否在标题中。

在20世纪50年代，H. P. Luhn提出开展自动文摘技术的研究（Luhn，1958）。Luhn的方法是采用重要因素（significance factor）对文档中的每个句子进行排序，然后选择最靠前的一些句子作为文摘。句子的重要因素根据句子中出现的重要词进行计算，其中重要词定义为在文档中具有中等频率的词，“中等”的意思是这个词的频率介于预定义的高频和低频之间。根据文本中重要词的位置，可以得到被两个重要词“括起来”的文本片段，其中限制了在两个重要词之间非重要词的最大数量（通常为4）。计算这些“括起来”的文本片段的重要因素是，将其中重要词数量的平方除以片段中的总词数。如图6-4中的例子，其中文本片段的重要因素为$4^2/7=2.3$。一个句子的重要因素为句子中包含的所有文本片段重要因素的最大值。

w w w w w w w w w w w.
（初始句子）

w w s w s s w w s w w.
（识别重要词）

w w [s w s s w w s] w w.
（由重要词括起来的文本片段）

图6-4 采用Luhn算法获得重要词之间的文本片段示例

为了更具体地描述重要词的定义，下面的公式给出了一种基于频率的重要词限定标准，这个公式已经成功地应用在最近的一些研究中。$f_{d,w}$是词w在文档d中的频率，如果w不是停用词（去除高频词），那么w为重要词。

$$f_{d,w} \geqslant \begin{cases} 7-0.1\times(25-s_d), & \text{如果} s_d<25 \\ 7, & \text{如果} 25\leqslant s_d\leqslant 40 \\ 7+0.1\times(s_d-40), & \text{其他} \end{cases}$$

其中，s_d是文档d中的句子数量。例如，在本书的第1章第2页（这里是指原英文版书）中包含不到25个句子（略算为20个），所以重要词是频率$f_{d,w}$大于或等于6.5的非停用词，满足这个标准的词只有“information”（频率为9），“story”（频率为8）和“text”（频率为7）。

自从Luhn提出这个方法以后，在自动文摘研究中，有很多工作都致力于改善这个基本的方法，其中包括一些选择重要词、句子和句子片段的更好方法。页面摘要生成技术可以看作是Luhn方法的一种变形，其中查询词被作为重要词，并采用不同的句子选择标准。

从文档中选取句子生成页面摘要，可以利用一些典型的特征，例如对于新闻文档，这些特征包括句子是否是标题，句子是否是文档的第一或第二行，句子中包含查询词的总数，句子中每个查询词的数量，句子中出现的最长连续查询词，以及查询词密度指标，如Luhn重要因素。在这个方法中，采用这些特征的加权组合排序句子。但是互联网网页不具有新闻文档那样的结构性，其中经常包含大量不适合作为页面摘要的文本。针对这个问题，页面摘要的句子经常从互联网页面上附带的元数据中选取，例如通过HTML标签<meta name="description" content=...>识别“description”的内容，或者可以利用一些外部资源，如网络字典[⊖]。有些种类的网页具有更好的结构性，可以从直接文本中选取页面摘要的句子，如维基百科（Wikipedia）上的条目。

尽管在结果页面中生成页面摘要和文档摘要可以采用不同的方法，但是根据对点击流数据的分析，能够获得有效进行文摘的一些基本指导原则（Clarke等，2007）。最重要的是，所有的查询词要尽可能地出现在文摘中，以显示出它们和检索页面间的关系。但是当查询词已经出现在标题中时，就没有必要在页面摘要中再出现，这样可以使用不包含查询词的元数据或外部资源中的句子。另一个指导原则是选择和显示URL时，应强调它们与查询词的关系，例如可以突出显示在URL中的查询词。最后，搜索引擎用户显然会更喜欢页面摘要中是可读的句子（完整的句子或几乎完整的句子），而不是一些词或短语。在选择页面摘要的排序计算中，也应当包括衡量可读性这个特征。

页面摘要生成的效率也是在搜索引擎架构中要重点考虑的部分，在需要处理大量查询的情况下，显然采用查找、打开和搜索文档文件这样的处理过程，会产生无法接受的开销，所以文档要在查询时从本地的文档库或缓存中取出并解压缩。用于生成页面摘要的文档中，应去除所有的HTML标签和其他的“噪声”（如Javascript），但仍然需要将元数据与文本内容区分开来。另外，句子边界应该在建立索引的时候识别和标记出来，以避免在选择页面时进行这种消耗时间的操作。

6.3.2 广告与搜索

广告是搜索引擎中的关键部分，因为搜索引擎公司要用它去获取收益。对于在搜索结果中展示广告（竞价搜索(sponsored search)）来说，目的是要找到符合查询上下文信息的广告。当浏览网页时，应根据网页内容选择要展示的广告。内容关联广告（contextual advertising）的想法是让更多用户去点击广告（点选链接），这也决定了广告的费用。搜索引擎公司维护一个广告数据库，在其中搜索到与查询或网页最相关的广告。在这个数据库中的广告通常包括简短的文本描述和详细介绍产品或服务的页面链接，所以广告数据库的搜索也可以看作是常规文本搜索的一个特殊情况。

但事情也不是那么简单，不能仅根据简单文本的排序去选择广告。广告客户会出价竞买一些能够描述他们产品主题的关键词，这样，与查询匹配的关键词价格也是选择广告的重要因素。另外，有些广告对于用户群体更具吸引力，可以产生更多的点击流，这样，广告的受

⊖ 例如，Open Directory Project项目，http://www.dmoz.org。

欢迎程度也是选择广告的一个重要因素，广告的受欢迎程度可以通过一段时间内查询日志中的点击流数据来衡量，并可以根据全部查询或一些具体查询来衡量。具体查询的受欢迎程度只能用于一些定期出现的查询。对于大部分不常出现的查询（称为长尾查询[⊖]（long-tail queries)），可以采用广告的普遍受欢迎程度。通过考虑所有这些因素，即相关性、出价、受欢迎程度，搜索引擎公司可以制定一些策略来最大化他们的预期利润。

例如，一个专门经营热带鱼的宠物供应公司，可能会对关键词“aquarium”和“tropical fish”出很高的价格。如果查询是“tropical fish”，这个关键词当然是相关的，这个公司的广告内容中也应该包含了一些与查询匹配的词。这样，这个公司的广告将会在内容相关上获得很高的得分，并且在出价上也会获得很高的得分。尽管这个公司给出了最高的价格，但如果另一个广告是更受欢迎的，并且对于相同的关键词也给出了比较高的价格，那么它的广告仍有一些机会能被选择。

目前开展的一些研究致力于开发如何最大化广告客户利润的算法，他们借鉴了一些其他的领域，如经济学和博弈论。从信息检索的角度，关键问题是匹配短文本（查询和广告）以及选择代表网页内容的关键词。

在搜索网页的时候，在一些网页中通常包含了所有的查询词项。但是在查询与广告文本相比较时，情况并非如此。与常规的网页相比，广告中包含了较少的词和关键词，并且广告数据库规模与互联网相比要小几个数量级。能够匹配上查询中出现的广告关键词的变形，也是非常重要的，例如，如果一个宠物供应公司对“aquarium”出了高价，它当然希望能够接到来至查询“fish tank”的业务。当然这是经典的词表不匹配（vocabulary mismatch）问题，针对这个问题，已经提出了一些处理技术，例如词干提取和查询扩展。由于广告文本比较短，所以一些扩展技术不仅应用在查询上，也被应用于广告文档。

在一些实验中，有两种技术表现出很好的效果，一种是基于查询日志中用户会话的查询改写（Jones等，2006），另一种是用外部资源进行查询和文档扩展，如利用互联网资源(Metzler等，2007)。

研究结果表明，在一个会话期中有大约50%的查询被改写，用户通过对初始查询中词的替换、插入和删除，修改这些查询。在查询和查询中的短语之间，会有很多候选关联度，采用像6.2.3节中描述的统计检验的方法，可以决定哪个关联度是更显著的。例如，短语“fish tank”与“aquarium”之间的关联，可能会经常出现在用户想要找到更多的页面而重写他的初始查询的搜索会话期间。如果相对于这些短语的频率来说，这种关联更是经常出现的，它就是显著的。这些显著关联的词汇可以被用作候选的替换词汇，所以对于一个初始查询，可以生成一个排序的查询改写列表，这个列表更着重于生成匹配广告关键词的查询。

这样的扩展技术包括利用互联网去扩展查询，扩展广告文本，或者对两者都进行扩展。可以采用一种伪相关反馈的形式进行扩展，将广告文本或关键词作为查询进行互联网检索，从检索到的最高排序的网页中选择扩展词汇。一些实验结果表明，对于广告最有效的相关排序是，将完全匹配整个查询的广告排序在最前面，然后是能够完全匹配经过词干替换查询的广告，再后面是根据概率相似度匹配扩展后查询的扩展广告文本。相似度匹配将在7.3节中介绍。

⊖ 词“长尾（long-tail)”来自于第4章中描述的齐普夫分布中的长尾。假设一个查询是一个具体的词组合，大部分的查询都以较低的频率出现，搜索引擎处理的大部分查询实例的出现数量都相对较少。

例如，图6-5是一个搜索引擎对查询“fish tanks”生成的广告列表。其中两个广告显然是匹配的，“fish tanks”出现在搜索结果的标题中。还有两个（第二个与第四个）与查询天相同的词，但它们也是相关的。用6.2.4节中的伪相关反馈技术，根据前10个排序结果将产生“aquarium”（频率10）和“acrylic”（频率7）作为扩展词项。这将给包含“aquarium”的广告很高的相关程度得分，例如第二个。第四个广告被选取的原因大概是宠物供应商购买了关键词“aquarium”，或者是因为已经有一些用户点击了这个广告。第三个广告与查询是相似的，匹配了查询中的一个词。

对于网页的内容相关广告，关键词一般是从网页内容中选取，然后用这些关键词去搜索广告数据库，选择可以与网页内容一起展示的广告。关键词选取技术与上节中介绍的自动文摘技术很相似。一个简单的关键词选取方法是，根据当前页面的文档和文档集中词的相对频率，计算词的重要度，并根据这个重要度对词进行排序，选择排序在前的词作为关键词。

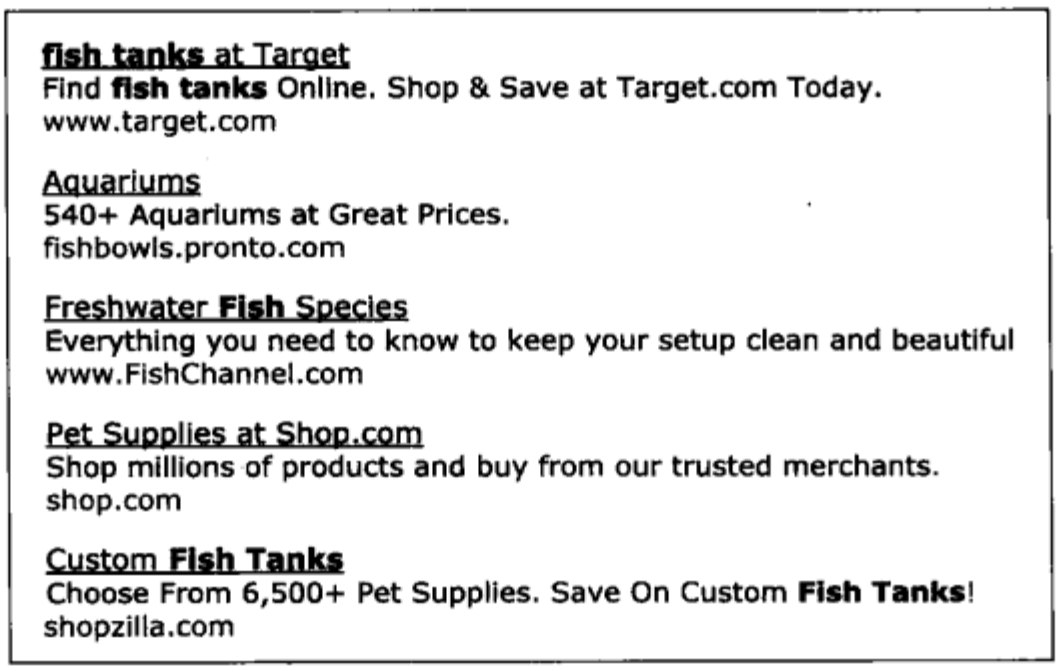


图6-5 搜索引擎对查询“fish tanks”展示的广告

一个更有效的方法是，采用将在第9章中介绍的基于机器学习技术的分类器（classifier）。分类器采用特征的加权组合，来决定哪一个词或短语是更重要的。典型的特征包括词或短语在文档中的频率，出现词或短语的文档数量，这些频率的函数值（例如，采用log值或归一化处理），在查询日志中出现的频率，词或短语在文档中的位置（例如标题、正文、锚文本、元数据、URL），以及是否词或短语是大写形式的或以某种形式突出显示。其中最有用的特征是文档和查询日志的频率信息（Yih等，2006）。

6.3.3 结果聚类

搜索引擎返回的结果中经常包含了与查询主题相关的不同方面。当查询比较模糊时，检索到的结果文档中，往往表达了对查询的不同解释。例如，我们已经看到了查询“tropical fish”是怎样检索到与养鱼缸、宠物供应商、图片以及其他的子主题（subtopic）相关的文档。

一个更简单的查询“fish”，甚至能够检索到混杂各种类型的文档集合，其中可能包含关于海洋、软件产品、摇滚歌手以及其他碰巧用到“fish”这个词的文档。如果用户只对查询主题的其中一个方面感兴趣，那么在浏览了一些其他方面的页面后，他可能会感到失望，这是要对检索结果进行聚类（clustering）的动机。聚类是将检索结果文档按照内容相似聚成一些

组并标记每个组，使得用户能够很快浏览到相关类别。

图6-6中是由一个网络搜索引擎对查询“tropical fish”检索出的排序靠前文档进行聚类的结果列表。这个列表中的每一簇用单个词或短语给予描述或标记，并指出每个簇中文档的数量，这个列表显示在常规搜索结果的旁边。用户如果对其中一个簇感兴趣，可以点击这个簇的标签查看簇中的文档，不需要再查看整个排序的文档去寻找符合兴趣的文档。在这个例子中，这些簇明显是与我们前面提到的子主题相关，如供应商和图片。

Pictures (38)
Aquarium Fish (28)
Tropical Fish Aquarium (26)
Exporter (31)
Supplies (32)
Plants, Aquatic (18)
Fish Tank (15)
Breeding (16)
Marine Fish (16)
Aquaria (9)

图6-6 搜索引擎对查询“tropical fish”的排序靠前文档的聚类结果（括号里的数字是每个簇中的文档数量）

聚类技术将在第9章中详细讨论。在本节中，我们主要介绍检索结果聚类的具体要求。第一个要求是效率（efficiency）。对于每个查询都要根据检索出的排序靠前的文档集合，生成一个特定的聚类结果。对于常见查询的聚类结果可以缓存起来，但是对于大部分的查询，聚类结果仍需要以在线方式生成，这个过程必须是高效的。通常的做法是根据文档页面摘要中的文本进行聚类，页面摘要比文档全文包含了更少的词，因为计算涉及词重叠比较，所以这将极大地提高计算速度。并且页面摘要文本更着重于查询的主题，而文档中通常要包含仅与查询部分相关的一些文本段落。

第二个重要的检索结果聚类的要求是，它们应该容易理解。例如，在图6-6中，每个簇用单个词或短语加上标签，用户能够猜想到在簇中是与标签的描述相关的文档。例如在标签为“Pictures”的簇中，可以合理地预料每个文档中都包含有鱼的图片。这是一个单因素（monothetic）分类的例子，其中一个类别中的每个成员都具有定义这个类别的属性[⊖]。这似乎是显而易见的，但事实上，大部分的聚类算法生成的类别形式都不是这样的。例如K均值算法（K-means）是根据词重叠生成一个簇或类别中的成员[⊖]。换言之，簇中的成员共有一些属性，但没有单独定义的属性，这称为多因素（polythetic）分类。对于检索结果聚类，首选产生单因素分类结果的技术（或者至少看起来是单因素分类的结果），因为产生这样的结果将更容易理解。

例如，在搜索结果中包含文档D_1、D_2、D_3和D_4，这些文档中包含的词项（即词或短语）属于集合{a，b，c，d，e，f，g}。用词项集合来表示每个文档：

D_1 = {a，b，c}

D_2 = {a，d，e}

D_3 = {d，e，f，g}

D_4 = {f，g}

一个单因素算法可能将a和e作为重要词项，生成两个簇分别是{D_1，D_2}（标签为a的簇）和{D_2，D_3}（标签为e）。这些簇是有重叠的，即一个文档可以属于多个簇。而基于词项重叠

⊖ 这也是在2400年前亚里士多德提出的类别定义。

⊖ K均值聚类将在第9章中介绍，其基本思想是，将一个文档与现有的群集中的代表进行比较，然后将其加入到最相似的簇中。

的多因素算法，可能仅生成一个簇{D_2，D_3，D_4}——其中D_2与D_3有两个词相同，D_3与D_4有两个词相同。这三个文档没有相同的单个词项，所以不能确定这个簇应该标记什么标签。

如果对于图6-1中的页面摘要，根据在多个文档中出现的非停用词进行简单的聚类，将得到下面结果：

aquarium（5）（文档1，3，4，5，8）
freshwater（4）（1，8，9，10）
species（3）（2，3，4）
hobby（3）（1，5，10）
forums（2）（6，8）

在这个技术的实现中，词和短语都会被用到，并且会采用更多的排序靠前的页面摘要（比如说200个）。在聚类中，还会用到一些其他的词或短语的特征，如它们是出现在标题中还是页面摘要中，短语的长度，短语在文档集中的频率以及聚类结果簇之间的相互重叠程度。

一种将检索结果组织到各个不同含义的类别中的方法，称为逐面分类法（faceted classification或简称facet）。逐面分类法是由一些类别构成，通常这些类别被组织成层次形式，每个类别用一组层面来描述与其相关的一些重要属性。例如，使用逐面分类法分类的产品，可以被标注为多个类别，并且每个层面都有值。层面的分类主要是通过人工去定义，但是也可以使用数据库模式以及正在研究的一些自动构建层面类别的技术，将数据以结构化形式组织起来，使得可以通过层面的形式浏览这些数据。人工定义层面的主要优点是，与自动生成簇标签相比，类别通常更容易被用户理解。其缺点是，对于每个新的应用和领域，都要重新定义类别，并且人工构建的类别常常是静态的，不会像动态构建簇那样对新的数据有相应的调整。

逐面分类法普遍应用于电子商务领域中，图6-7展示了在一个流行的零售商网站上，对于查询“tropical fish”返回的类别集合。图中数字是与查询匹配的每个分类中产品的数量，这些分类展示在检索结果的旁边，类似于前面讨论的聚类结果。如果选择了“Home & Garden”类别，图6-8就展示了一个子类的列表，如“pet supplies”，以及这个类别的层面，包括品牌名称、供应商或卖主名称、折扣程度以及价格。对于一个产品，例如鱼缸，能够包括在“pet supplies”类别中，并且也包含在适当的价格类别和折扣价格类别中。

这种组织方式在用户浏览检索结果时，具有很好的引导性和灵活性。

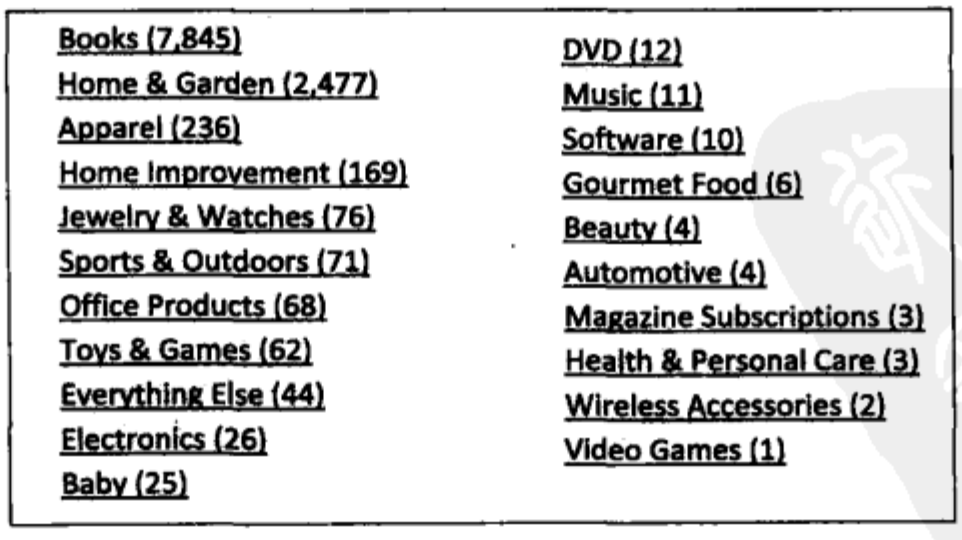


图6-7 一个流行的在线零售网站上对查询“Tropical fish”返回的类别结果

Home & Garden	**Discount**
Kitchen & Dining (149)	Up to 25% off (563)
Furniture & Décor (1,776)	25% - 50% off (472)
Pet Supplies (368)	50% - 70% off (46)
Bedding & Bath (51)	70% off or more (46)
Patio & Garden (22)	
Art & Craft Supplies (12)	**Price**
Home Appliances (2)	$0-$24 (1,032)
Vacuums, Cleaning & Storage (107)	$25-$49 (394)
	$50-$99 (797)
	$100-$199 (206)
Brand	$200-$499 (39)
<brand names>	$500-$999 (9)
Seller	$1000-$1999 (5)
<vendor names>	$5000-$9999 (7)

图6-8 在类别“Home & Garden”中的子类别和层面

6.4 跨语言搜索

通过对一个或多个采用不同语言的单语言搜索引擎中的查询进行翻译，可以实现跨语言搜索（cross-language search）[⊖]（参见图6-9）。跨语言搜索引擎中，采用一种语言的查询（例如英语）和其他各种语言的文档（例如法语和中文）。用户一般不会熟悉多种语言，所以跨语言搜索引擎必须能够自动翻译查询。由于系统使用多种语言的文档，所以一些系统为用户翻译这些文档。

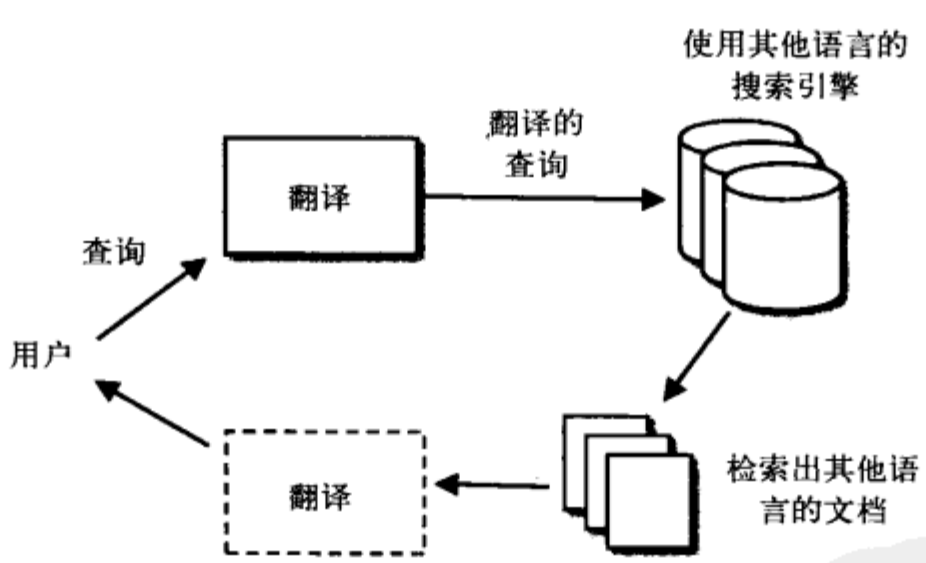


图6-9 跨语言搜索

自动翻译最显而易见的方法是，使用一个大型双语字典将源语言词汇（例如英语）翻译到目标语言（例如法语），通过在字典中查找句子中的每个词进行翻译。这个方法的主要问题是如何处理歧义，因为一些词会有多种意思。通常，简单地利用字典进行翻译效果很差，但是可以采用一些技术（例如查询扩展，见6.2.3节）去减少歧义，并将跨语言系统的排序效果提高到与单语言系统相当的程度。

基于统计机器翻译模型（statistical machine translation model）的方法，是最有效和最常

⊖ 也称为cross-language information retrieval（CLIR）、cross-lingual search和multilingual search。

用的自动翻译方法（manning & Schütze，1999）。与翻译查询不同，翻译文档或网页时，不仅歧义是一个问题，而且需要考虑翻译的句子是否符合语法。在翻译句子的时候，一些词的顺序可能会改变，可能会被略掉或者翻译成更多的词。统计翻译模型用概率来表示这些变化，这意味着这个模型描述了一个词被翻译成另一个词的概率、词改变顺序的概率、词被略掉或翻译成多个词的概率，用这些概率计算出句子最有可能的翻译结果⊖。

尽管基于词到词翻译概率的模型与基于字典的模型很相似，但是如果翻译概率很准确，那么它们也会有非常不同的翻译质量，并且能够将歧义词的不常见翻译意思从那些常见的翻译意思中识别出来。最近根据这个模型提出了基于短语的翻译模型（phrase-based translation model），它不是计算单个词的概率，而是计算词序列的概率，这种模型能够在翻译中更好地使用上下文。例如一个词“flight”能够根据短语“commercial flight”更准确地翻译它，而不会翻译为“bird flight”中的意思。

在统计机器翻译模型中，主要用平行语料库（parallel corpora）进行概率估计。平行语料库是由一种语言的文档和将它翻译成其他的一种或多种语言文档构成的文档集，这种语料主要是从政府组织、新闻组织获得，或者从互联网上采集，因为在互联网上有成千上万的翻译页面。在平行语料库中的句子采用人工或自动方式对齐（aligned），即每个句子要和它的翻译句进行配对，然后使用对齐后的句子训练翻译模型。

一些非常用词的翻译是要特别注意的，尤其像人名这样的专有名词。对于这些特别的词，互联网是一个丰富的资源。自动音译（transliteration）技术也用来处理人名的问题。人名通常不翻译成另外的一种语言，而是进行音译，即人名的翻译是根据一些规则或相似的发音用另一种语言的字符写出它，这样对同一个人名可能会有不同的翻译拼写方法。例如，利比亚领导人穆阿玛尔·卡扎菲（Muammar Qaddafi）的名字在互联网上有多种的音译形式，如Qathafi、Kaddafi、Qadafi、Gadafi、Gaddafi、Kathafi、Kadhafi、Qadhafi、Qazzafi、Kazafi、Qaddafy、Qadafy、Quadhaffi、Gadhdhafi、al-Qaddafi、Al-Qaddafi和Al Qaddafi。与之相类似，在阿拉伯语的网页上，“Bill Clinton”的名字也被翻译成很多种形式。

尽管一些网络搜索引擎并没有被视为是跨语言的搜索系统，但是它们也经常检索出各种语言的网页，由于这个原因，在有些搜索引擎的检索结果页面上，提供了一些翻译设置。图6-10中展示了根据查询“pecheur france”检索的一个网页结果示例，其中翻译选择用超链接显示，点击这个链接，就可以产生一个翻译的页面（不是页面摘要），这个页面清楚地显示包含了到体育杂志Le Pêcheur de France文档库的链接，但它被翻译成“The fisherman of France”。尽管这些翻译效果还不理想，但通常它也提供了足够的信息，使得用户能够理解网页的相关内容。这些翻译是采用机器翻译技术自动生成的，因为如果采用人工处理，将无法承受其高昂的费用。

Le pêcheur de France archives @ peche poissons - [Translate this page]

Le **pêcheur de France** Les média Revues de pêche Revue de presse Archives de la revue Le **pêcheur de France** janvier 2003 n°234 Le **pêcheur de France** mars 2003 ...

图6-10 查询“pecheur france”检索结果列表中的一个法语网页

⊖ 机器翻译模型的最简单形式实际上与7.3.1节中查询似然度模型非常相似。主要的不同是，将翻译概率$P(w_i|w_j)$结合到$P(Q|D)$的概率估计中，其中$P(w_i|w_j)$是词w_j被翻译成w_i的概率，$P(Q|D)$是从一个文档生成一个查询的概率，在翻译模型中，它成了一个文档翻译成一个查询的概率。

参考文献和深入阅读

本章包括了很多的主题，并且这些主题历经多年的研究，因此有许多相关的参考文献，与我们在此所叙述的内容相比，这些文献提供了更多的细节。下面的文章和书籍代表了一些重要的贡献，但是对于想要深入理解某一具体主题的人来说，这些文章或书籍都包含了一些其他的研究工作。

关于“natural language”的布尔查询或关键词查询的优点和缺点，已经历经了30多年的讨论。在法律检索领域，这个争论尤其活跃，可以参见在20世纪90年代早期第一个在大规模文档集上采用简单查询和排序方法的搜索引擎的介绍。Turtle（1994）描述了将专家布尔检索与基于简单查询的排序方法进行的数量化比较，他发现甚至在一些专业领域，简单查询也是非常有效的。下一章将更详细地讨论布尔检索模型。

J. Xu和Croft（1998）对于基于语料库分析的查询词干提取，进行了更为细致的叙述。要了解用在信息检索中的关联度衡量方法如戴斯系数的早期历史，可以参见Van Rijsbergen（1979）。Peng等（2007）提出了在网络搜索中基于语料库词干提取的一个新方法。

Kukich（1992）对一些拼写校正技术进行了综述。Jurafsky和Martin（2006）更详细地介绍了在拼写校正中采用的最小编辑距离和噪声通道模型。Guo等（2008）提出了将查询提炼的各个步骤（例如拼写校正、词干提取、短语识别）结合到一个简单模型中的方法。他们的结果显示，相对于分别进行这些步骤，采用一个融合的模型能够提高效果。

对于查询扩展已经开展了许多研究，Efthimiadis（1996）给出了查询扩展技术的历史和综述，其中包括基于叙词表的扩展。如前所述，Van Rijsbergen（1979）描述了信息检索中关联度衡量方法的进展，包括采用互信息的方法。在计算语言学中，在用到互信息构建词典时，经常会引用Church和Hanks的文章（1989），Manning和Schütze（1999）对这些方法以及本章中提到的其他相关度衡量方法进行了很好的综述。

Jing和Croft（1994）提出了根据共现的词组成的虚拟文档去构造“关联叙词表”的技术，Beeferman和Berger（2000）以及Cui等（2003）描述了使用查询日志的数据进行查询扩展。

Rocchio（1971）最先开展了相关反馈的研究工作，随后Salton和McGill（1983）及Van Rijsbergen（1979）也对其进行了大量的研究。J.Xu和Croft（2000）是在伪相关反馈研究方面经常被引用的文章，其中对基于排序靠前的文档的“local”技术和基于文档集中词相关的“global”技术进行了比较。根据对TREC评测实验的长达10年的研究，Voorhees和Harman（2005）这本书中包含了对相关反馈和伪相关反馈技术的一些描述。

上下文和个性化是一个流行的主题。在一些研讨会和会议上发表了许多相关的文章，例如Information Interaction in Context Symposium(IIiX)[⊖]。Wei和Croft（2007）通过实验研究提出了用户描述文件的潜在益处中存在的问题。Chen等（2006）和Zhou等（2005）讨论了能够有效地处理本地搜索查询的索引结构，并且给出了本地搜索的一般概述。V.Zhang等（2006）在本地搜索的研究中更注重于对查询日志的分析。

Luhn（1958）开始了自动文摘的最初研究工作，Goldstein等（1999）描述了较新的基于句子选择自动文摘的研究工作。与之相对应，Berger和Mittal（2000）的研究是基于文档的统计模型生成文摘，Sun等（2005）描述了基于点击流数据的技术。Clarker等的文章（2007）

⊖ 这个会议是从Information Retrieval in Context研讨会发展而来，在互联网上可以找到会议的议程。

和Turpin等的文章（2007）则集中研究页面摘要的自动生成。

Feng等（2007）对于竞价搜索的相关事项进行了总体综述。Metzler等（2007）和Jones等（2006）讨论了将查询匹配到较短广告文本的一些具体技术。对于内容关联广告（在浏览网页时提供广告）一些问题的讨论以及从网页中选择具体关键词的技术，可以参见Yih等（2006）。

正如前面提到的，用于检索结果的一些可视化技术已经提出多年了，我们在本书中略掉了这方面的大部分内容。Hearst（1999）对这些技术给予了细致的综述，Leouski和Croft（1996）率先提出了检索结果聚类的评测技术。Hearst和Pedersen（1996）显示了这种技术的潜在益处，Zamir和Etzioni（1999）强调了结果聚类的重要性，能够方便用户理解检索结果并且更容易对结果加以标记。Lawrie和Croft（2003）讨论了对于检索结果建立层次化摘要的技术。Zeng等（2004）着重于从检索结果中选择短语作为聚类的依据。在Hearst（2006）中，讨论了聚类方法和逐面分类法的相应优点和缺点。

更普遍的情况是，整个HCI（Human-Computer Interaction）[⊖]领域中的研究人员都在关注信息系统界面的设计和评价。Shneiderman等（1998）是这类研究的一个例子，Marchionini（2006）对于搜索界面的交互性（Interactive）或探索性（exploratory）方面的重要性进行了很好的综述。

跨语言搜索已经在TREC（Voorhees和Harman，2005）和欧洲评测会议CLEF[⊜]上展开了多年的研究。这个研究领域的第一个论文集是Grefenstette(1998)。在一些文章中，讨论了具体的CLIR系统上的相关研究问题，如音译（AbdulJaleel和Larkey，2003）。Manning和Schütze（1999）及Jurafsky和Martin（2006）对统计机器翻译模型进行了综述。

最后，在信息科学文献中，有大量的研究工作探讨了人们在搜索引擎上实际的交互过程和搜索行为，这些研究是对本章中介绍的面向系统方法的补充，也是理解搜索信息与相关过程的重要部分。美国信息科学与技术期刊（Journal of the American society of information science and technology，JASIST）是这类文章的一个很好资源。Ingwersen和Järvelin（2005）从计算机科学和信息科学的不同视角对于搜索领域进行了有趣的比较。

练习

6.1 用本书网址中提供的Wikipedia文档集，根据下列步骤创建一个词干群集（stem clusters）。

1）索引文档集（不进行词干提取）。

2）取出在索引中按字母顺序的前1000个词。

3）通过对这1000个词进行词干提取创建词干类别，并且记录哪些词具有相同的词干。

4）计算在每个词干类别中每对词之间的关联度（采用戴斯系数），根据在文档中的共现进行计算。

5）通过限定关联度阈值创建词干群集，根据计算后相互关联的所有词项构成这个词干群集。

比较本题中的词干群集和词干类别中词项数量和质量（根据你的观点）。

6.2 根据噪声通道模型编写一个拼写校正程序。采用单个词的语言模型，在错误模型中具有相同编辑距离的所有错误设定为相同的概率。仅考虑编辑距离为1或2的情况。要求你自

⊖ 有时也简称为CHI。

⊜ http://clef.isti.cnr.it/.

己实现计算编辑距离的程序（样例代码很容易在互联网上找到）。

6.3 为Galago搜索引擎实现一个简单的伪相关反馈算法。给出用你的算法进行查询扩展的样例，并简述你的方法的成功之处及存在的问题。

6.4 假设有一个包含地理名称的地名集，设计一个算法，检测出查询中的地名或位置名称。展示出你的算法对于哪些类型的查询是有效的，以及对于哪些查询你的算法将会失效。

6.5 描述在Galago中的页面摘要生产算法。这个算法对于只有很少文本内容的页面是否也有效？详细描述你将怎样改善这个算法。

6.6 选择一个商业搜索引擎，你认为在其竞价搜索中的广告是如何与查询相匹配的，举出例子证明你的想法。对于在网页中显示的广告，也进行同样的分析。

6.7 实现一个根据排序靠前的页面中选择短语进行检索结果聚类的简单算法。任何具有两个词的词序列都可以考虑作为短语。你的方法应该考虑在检索结果中的短语频率、文档集中的短语频率，以及与短语相关联的簇之间的重叠程度。

6.8 找到采用逐面分类法的4个不同类型的网站，并用具体样例描述它们。

6.9 给出5个你认为网页翻译效果很差的例子。你认为翻译效果不好的原因是什么？

第7章 检索模型

“没有一定会怎样，只有可能会怎样。”

——V（雨果·威明），《V字仇杀队》

7.1 检索模型概述

在过去45年的信息检索研究中，一个主要的目标就是要理解和形式化人们判定一段文本和所需信息是否相关的过程。为了形成完整的理解，可能需要理解语言是如何表示的以及是如何被人脑处理的。我们距离这种目标还有相当的距离。然而，我们能够提出关于相关性的数学检索模型，并且通过比较模型和人类行为来验证这些理论。好的模型应该在相关性上产生和人类决策非常相关的结果。换句话说，基于好的检索模型的排序算法，能够在排序结果的顶部返回相关的文档（具有较高的有效性）。

那么，这些模型取得了哪些成功呢？举个例子，在TREC测试数据集上针对通用搜索的排序算法，已经在上个世纪90年代提高了100%的效果。这些在效果上的改变相应地改进了关联检索模型。在过去的10年中，网络搜索的效果同时也得到了实质性的提高。在TREC数据集上的实验中，最有效的排序算法来自于被明确定义好的检索模型。在商用网络搜索引擎中，所使用的检索模型不是很清楚，但是毫无疑问的是，它们的排序算法都依赖于坚实的数学基础。

在没有检索模型的情况下，通过实验，也可以开发出排序算法。但是，直接使用检索模型被广泛证明是最好的方法。像所有的数学模型那样，检索模型提供了定义新的任务和解释假设的框架。当采用排序算法测试时，检索模型提供了比暴力方法（尝试所有可能）更有效的测试结构。

在本章的讨论中不能忽视一个客观事实，那就是相关性是一个复杂的概念。让人解释为什么一篇文档比另外一篇更相关，是非常困难的。同样，当要求人们判断对给定的检索词与一些文档的相关性时，他们往往都不会同意这样做。信息论方面的科学家已经撰写了很多关于相关性本质的专著，但是这里不会涉及这些材料。取而代之，会讨论相关性的两个核心方面，这两个方面对检索模型和评价机制都非常重要。

第一个方面就是在1.1节提到过的主题相关和用户相关的差异性。一篇文档如果被判定和一个查询是同一个主题，那么它们是主题相关的。换句话说，查询和文档是关于同一件事情的。一个包含亚伯拉罕·林肯传记的网页肯定和查询“亚伯拉罕·林肯”主题相关，同时也会和查询“美国总统”以及“美国内战”主题相关。用户相关性考虑用户在判定相关性时涉及的所有因素。这些因素可能包括文档的年代、语言、目标受众、新颖性等。例如，一篇包含所有美国总统的列表文档，对于检索词“亚伯拉罕·林肯”是主题相关的，但是不会被认为和想要检索林肯生平信息的用户相关。所有的检索模型都不能在用户相关性中融合全部的额外因素，但是有些模型会考虑这些因素。

关于相关性，考虑的第二个方面就是相关性是二元的还是多元的。二元相关性就是简单地判定一篇文档是相关的还是非相关的。显然，有些文档明显不如其他文档相关，但是与那

些和主题一点关系都没有的文档相比，具有更好的相关性。例如，还是考虑包含美国所有总统的列表文档，它和林肯传记不是那么主题相关的，但是肯定比林肯汽车的广告要更加相关一些。基于这种观察，一些检索模型和评价方法显式地确定相关性为多元变量。当人们被要求判断相关性时，多个层次的相关性对于评价肯定是重要的。三个层次的相关性（相关的、非相关的、不确定的）已经证明会使得判断更加容易。但是在各种检索模型中，多层次相关性的优势并不明显。因为，绝大多数的排序模型都将相关性按照概率进行计算，这也就涵盖了不确定相关的类型。

近年来，人们提出了很多检索模型。其中两种最早的模型分别是布尔模型和向量空间模型。虽然这些模型能在很大程度上由概率方法取代，但是还是经常在讨论信息检索时被提到，所以在深入介绍其他模型前，先简要介绍一下它们。

7.1.1 布尔检索

布尔检索模型被用在最早的搜索引擎中并沿用至今。它又称为精确匹配检索，因为被检索到的文档都能够精确匹配检索需求的，不满足的文档都不会被检索到。虽然这是一个排序中非常简单的形式，但是布尔检索并没有被一般地描述为一个排序算法。这是因为布尔检索模型假设在检索到的集合中，所有文档关于相关性都是等价的，同时也假设了相关性是二元的。布尔这个名称来自以下事实，在检索评价里只有两种输出结果（TRUE和FALSE），并且查询项往往被描述为布尔逻辑操作符（AND、OR、NOT）。正如第6章所述，邻近运算操作符和通配符也经常用在布尔查询项中。正则表达式命令中的搜索，例如grep，就是另外一种精确匹配的例子。

布尔检索有很多优点。这个模型的结果很容易推断并且容易向用户解释的。布尔查询项的运算域可以是任何文档特征，而不只是词语，所以可能直接在检索规范中融入元数据，例如文档日期或者文档类型。从实现的角度来看，布尔检索往往比排序检索更有效，因为文档可以在打分过程中快速地剔除。

除了这些正面因素，这种方法主要的缺点是效率完全依赖于用户。由于缺少复杂的排序算法，简单的查询项不能很好地工作。包含特定检索词的所有文档都会被检索到，并且这个检索到的集合会被按照某种和相关性几乎没有关系的顺序展现给用户，例如文档的发表日期。可以通过构建复杂的查询项来减少大部分相关文档的范围，但是这是一件很难的事情，需要专业的知识和经验。由于形式化查询项很难，一种名叫搜索中介（最后一章中有介绍）的用户随着布尔搜索系统而出现。中介的任务是将用户的信息需求转换为特定搜索引擎的复杂布尔查询项。这种中介至今仍然存在于一些特定领域，例如法律办公室。但是，现代搜索引擎的简单和有效性，使得绝大多数用户都能进行自己的搜索。

作为布尔检索形式的一个例子，针对索引了一组新闻故事数据集的搜索引擎，考虑下面的查询项。简单的查询项

```
林肯
```

会返回大量包含林肯汽车的文档以及关于林肯总统人名的故事。不管“林肯”这个词出现多少次，也不管这个词的上下文是什么，所有这些文档根据布尔检索模型的排序性质都是等价的。因此，用户可能尝试通过下面的查询项来缩小范围：

```
总统AND林肯
```

这个查询项会返回一组同时包含这两个查询词的文档，不管这些词语在文档中什么位置上出现。如果其中很多故事包含对福特汽车公司和林肯汽车的管理，这些文档也会作为关于林肯总统的故事出现在返回集合中。例如：

> 福特汽车公司今天宣布Darryl Hazel会接替Brian Kelley担任林肯水星品牌的总裁（译者注：总裁和总统英文都是President）。

如果有太多这种类型的文档被检索到，用户可以通过使用NOT操作符来剔除关于汽车的文档，例如：

> 总统AND林肯AND NOT（汽车OR轿车）

这会剔除掉那些不管在任何位置包含“汽车”或“轿车”的文档。一般说来，NOT操作符会在剔除非相关文档的同时剔除过多相关文档，这种方式不推荐使用。例如，在网页搜索“总统 林肯”时，返回的排序最靠前的文档是一个包含下面句子的传记：

> 林肯的遗体伴随着一趟九节车厢的丧葬火车离开华盛顿。

在查询项中使用NOT (汽车 OR 轿车)会剔除掉这个文档。如果返回的集合仍然很大，用户可以通过在查询项中增加会出现在传记中的词语来进一步缩小检索范围：

> 总统 AND 林肯 AND 传记 AND 生活 AND出生地 AND盖茨堡 AND NOT (汽车 OR 轿车)

遗憾的是，在布尔搜索引擎中，在查询项中通过AND操作符使用过多的搜索词项，往往会使检索结果为空。为了避免这种情况，用户可能尝试OR操作符作为替代：

> 总统 AND 林肯 AND (传记 OR 生活 OR出生地 OR盖茨堡) AND NOT (汽车 OR 轿车)

这会返回任何同时包含“总统”和“林肯”的文档，同时包含词汇“传记”、“生活”、“出生地”或者“盖茨堡”中的任何一个（还不能包含“汽车”或者“轿车”）。

经过这些步骤，拥有了一个合理的能够返回一组相关文档的查询项。但是仍然不能确定哪些词汇是更加重要的，也不能确定拥有更多的相关词汇就会更好。例如，包含下面文字的文档被一个网页搜索引擎（使用了词语重要性度量）排序为500：

> 总统的一天-假日活动-技工，迷宫，词语搜索，……“华盛顿的一生” 在线阅读整本书！亚伯拉罕·林肯研究网……

一个布尔检索系统不能对这个文档和另外一个被网页搜索引擎排序为499的文档进行任何区别。或者，这个文档会作为检索结果的第一个文档。

构建查询项的过程侧重于检索结果集合的大小，这个过程被称为数量检索，并导致了布尔检索模型的瓶颈。为了突破这些限制，研究人员开发了结合排序的模型，例如向量空间模型。

7.1.2 向量空间模型

向量空间模型是上世纪60～70年代绝大多数信息检索研究的基础，使用这个模型的论文也不断出现在各种会议中。这个模型由于简单、直观而很引人注目，实现的框架便于进行词项加权、排序和相关反馈等工作。从历史来看介绍这些概念非常重要，这些有效的技术已经经过多年的实验论证。但是，作为一个检索模型，它也有一些缺点。虽然提供了方便的计算框架，但是它对于加权和排序算法如何影响相关性，只提供了很少的详细说明。

模型中，文档和检索词都被假设是一个t维向量空间的一部分，其中t是索引词项（词语、

词干、短语等）的个数。一篇文档D_i表示为索引词项的一个向量：

$$D_i = (d_{i1}, d_{i2}, \cdots, d_{it})$$

其中d_{ij}表示第j个词项的权值。一个包含n个文档的数据集，可以表示为一个词项权值的矩阵（如下所示），其中每一行表示一篇文档，每一列表示对应文档在相关词项上的权值大小。

	词项$_1$	词项$_2$	$\cdots$	词项$_t$
文档$_1$	d_{11}	d_{12}	$\cdots$	d_{1t}
文档$_2$	d_{21}	d_{22}	$\cdots$	d_{2t}
$\vdots$	$\vdots$			
文档$_n$	d_{n1}	d_{n2}	$\cdots$	d_{nt}

图7-1给出了一个采用向量表示四个文档的简单例子。这个词项-文档矩阵已经被转置，所以现在词项对应行向量，文档对应列向量。词项的权值通过简单地计算在文档中的出现次数获得。在这个例子中，停用词没有被索引，所有词语都被词形还原了。例如，文档D_3表示为向量（1, 1, 0, 2, 0, 1, 0, 1, 0, 0, 1）。

D_1 Tropical Freshwater Aquarium Fish.
D_2 Tropical Fish, Aquarium Care, Tank Setup.
D_3 Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
D_4 The Tropical Tank Homepage - Tropical Fish and Aquariums.

Terms	Documents			
	D_1	D_2	D_3	D_4
aquarium	1	1	1	1
bowl	0	0	1	0
care	0	1	0	0
fish	1	1	2	1
freshwater	1	0	0	0
goldfish	0	0	1	0
homepage	0	0	0	1
keep	0	0	1	0
setup	0	1	0	0
tank	0	1	0	1
tropical	1	1	1	2

图7-1 四篇文档组成的数据集的词项-文档矩阵

查询项采用与文档相同的方式表示，即查询项Q表示为有t个权值的向量：

$$Q = (q_1, q_2, \cdots, q_t)$$

其中q_j是查询项中第j个词项的权值。例如，查询项是“tropical fish”，采用图7-1的向量表示，这个查询项的结果是（0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1）。向量空间模型的一个吸引人的方面是，可以采用简单的图形来对文档和查询项进行可视化。它们一般会在一个三维图片中表示为点和向量，如图7-2所示。虽然这样可视化有利于教学，但是会导致错误地认为三维空间能够应用到真实的高维文档空间中。请记住，这里的t个词项表示所有被索引的文档特征。在企业的网络应用中，这会对应成百上千或者上百万个维度。

基于这种表示，文档可以通过计算表示文档和查询的点之间的距离来进行排序。通常使用相似度度量（而不是距离度量或者不相似度度量）的方法，得分最高的文档被认为和查询具有最高的相似度。为此，已经有很多相似度函数先后被提出和测试。其中最成功的就是余弦相似度度量方法。余弦相关性度量是指查询项和文档分别对应的向量形成夹角的余弦值。当所有向量都被归一化后，所有文档和查询项都会被表示为长度相等的向量，两个完全相同的向量夹角的余弦值为1（角度为0），两个完全没有公共词项的向量的夹角的余弦值为0。余弦相关性的定义如下：

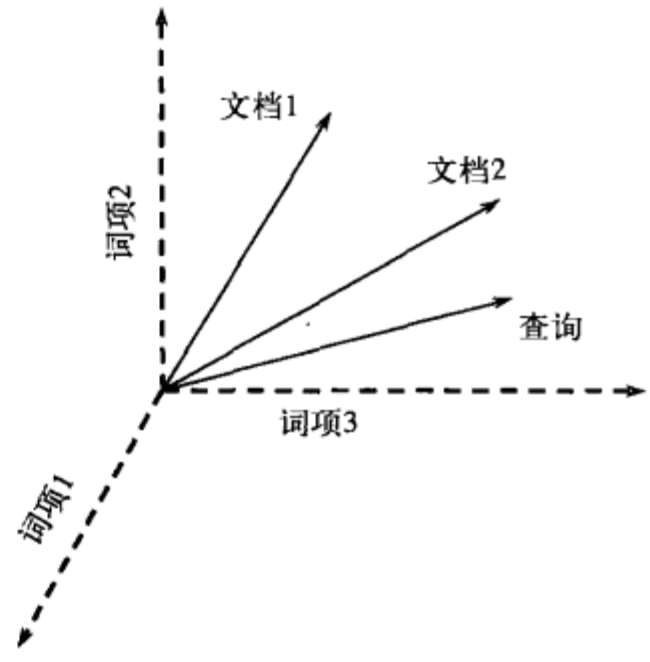


图7-2 文档和查询的向量表示

$$\cos(D_i,\ Q)=\frac{\sum_{j=1}^{t} d_{ij}\cdot q_j}{\sqrt{\sum_{j=1}^{t} d_{ij}^2\cdot\sum_{j=1}^{t} q_j^2}}$$

这个度量方法的分子是查询项和文档所有匹配词项对应权值的乘积之和（也叫点积或者内积）。分母通过除以两个向量长度的乘积来归一化分子。现在还不能从理论上解释为什么余弦相关性比其他相似度度量方法要好，但是在搜索质量的评价上，它就是要表现得好一些。

举一个例子，考虑被三个词项索引的两个文档D_1=(0.5, 0.8, 0.3)和D_2=(0.9, 0.4, 0.2)，其中的数字表示词项权重。假设查询项为Q=(1.5, 1.0, 0)已被同样的词项索引，那么这两个文档的余弦度量值为：

$$\cos(D_1,\ Q)=\frac{(0.5\times1.5)+(0.8\times1.0)}{\sqrt{(0.5^2+0.8^2+0.3^2)(1.5^2+1.0^2)}}$$

$$=\frac{1.55}{\sqrt{(0.98\times3.25)}}=0.87$$

$$\cos(D_2\ \ Q)=\frac{(0.9\times1.5)+(0.4\times1.0)}{\sqrt{(0.9^2+0.4^2+0.2^2)(1.5^2+1.0^2)}}$$

$$=\frac{1.75}{\sqrt{(1.01\times3.25)}}=0.97$$

第二篇文档得分较高，因为它的第一个词项权值较高，而且这个词项在查询项里面也有较高权值。这个简单的例子，说明基于向量空间模型的排序能够反映词项的重要性，以及匹配上的词项的个数。这一点在布尔检索中是不可能的。

在这里的讨论中，还没有介绍任何关于向量空间模型中词项权值的形式。事实上，这么多年来，许多不同的加权方法都已经被尝试过。其中绝大多数都是第2章介绍的tf.idf加权方法的变形。*tf*表示文档中词项的频率，反映了一个词项在文档D_i（或查询项）中的重要性。这个频率通常都是通过词项在文档中的出现次数归一化后得到，例如

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$$

其中tf_{ik}是文档D_i中词项k的词项频率，f_{ik}是词项k在文档中的出现次数。在向量空间模型中，归一化是余弦度量的一部分。一个文档数据集能够包含许多不同长度的文档。虽然归一化是为了减小这种情况的影响，但是长文档中很多词项都只出现一次，而其他词项都出现成百上千次。实验表明，为了减小这种高频词项的影响，通过对词项次数tf取对数，会比直接使用原始数值会更加有效。

倒置文档频率（idf）反映了文档数据集中词项的重要性。如果在其中出现过一个词项的文档越多，这个词项在文档之间就越没有区分性，也就对检索越没有用。这个权值的典型形式如下

$$idf_k = \log \frac{N}{n_k}$$

其中idf_k是词项k的倒置文档频率，N是文档数据集中文档的个数，n_k是词项k出现过的文档个数。虽然权值的这种形式是从直觉和经验演变出来的，但是在信息论（Robertson, 2004）中，有证据表明idf度量就是词项具有的信息量。

两种权值的影响通过相乘结合起来（即所谓的$tf.idf$）。之所以采用这种结合方式，更多的是根据经验。基于此，向量空间模型中，典型的文档词项权值的形式为：

$$d_{ik} = \frac{(\log(f_{ik})+1)\cdot\log(N/n_k)}{\sqrt{\sum_{k=1}^{t}[(\log(f_{ik})+1.0)\cdot\log(N/n_k)]^2}}$$

查询项中的词项权值的形式本质上是一样的。词项频率加1是为了保证频率为1的词项具有非零权值。在这个模型中需要注意的是，词项权值只针对出现在文档（或查询项）中的词项进行。考虑到余弦度量的归一化已经融合到权值计算中，一个文档的分值可以简单地通过将文档向量和查询向量求点积获得。

虽然在向量空间模型中没有关于相关性的显式定义，但是一个隐含的假设就是相关性是和查询项向量与文档向量的相似度有关联的。换句话说，和查询项“越接近”的文档就越相关。虽然与用户相关性的相关特征能够融合到向量表示中，但是这个模型主要是服务于主题相关性的。现在没有任何关于相关性是二值或者多值的假设。

在最后一章，会讨论相关反馈。这是一种根据用户判定的相关文档来修改查询项的技术。这种技术最早被应用到向量空间模型中。著名的Rocchio算法（Rocchio, 1971）是基于最佳查询的算法。这个算法会使得相关文档向量的平均向量和非相关文档向量的平均向量之间的差异最大化。在获得有限的相关信息的情况下，Rocchio算法通常（也是有效的）修改查询向量Q的初始权值，生成一个新的查询Q'：

$$q'_j = \alpha\cdot q_j = \beta\cdot\frac{1}{|Rel|}\sum_{D_i\in Rel} d_{ij} - \gamma\cdot\frac{1}{|Nonrel|}\sum_{D_i\in Nonrel} d_{ij}$$

其中q_j是查询词项j的初始权值，Rel是用户选定的相关文档集合，$Nonrel$是非相关文档集合，

|·|返回一个集合的大小，d_{ij}是文档i中第j个词项的权值，α、β和γ是控制每个部分影响的参数。前人的研究证明，非相关文档集合最好近似为所有没见过的文档（即所有不相关文档），并且对于参数α、β和γ，合理的数值分别为8、16、4。

这个公式通过加上一个基于相关文档的平均权值并减去一个基于非相关文档的平均权值来修改查询项。查询词项中，具有负数权值的都被剔除了。这种处理会产生一个更长的或者扩展的查询项。因为在相关文档中频繁出现但是没有在原始查询项中出现的词项，会被添加上（即它们会在修改后的查询项中具有大于0的权值）。为了限制扩展词项的数量，在相关文档对应的最高平均权值的词项中，一般只允许添加一个确定数目（例如50）的词项到查询项中。

7.2 概率模型

检索模型的特点之一，就是应该针对模型的假设提供一个基于假设基础的清晰说明。布尔模型和向量空间模型都对相关性和文本表示采用了隐含的假设，并影响到了排序算法的设计和效果。理想的情况应该在给定假设后，基于排序算法的检索模型能够超过其他任何方法的性能。这种证明对于信息检索事实上是非常困难的。因为尝试描述的是复杂的人类活动。检索模型的验证一般都是经验性的，不能是理论性的。

一个早期的关于有效性的理论说明，也就是人们所知道的概率排序原则（Robertson, 1977/1997）。这个原则推动了概率检索模型的发展，并使概率模型成为目前的主流。这些模型之所以取得这样的地位，是因为概率论为表示和操纵信息检索过程中固有部分的不确定性提供了坚实的基础。概率排序原则的原始表述如下：

> 如果一个参考检索系统[⊖]对每个查询的反馈都是数据集中所有文档根据和用户查询的相关性概率值降序排序的结果，并且其中的概率值都被尽可能精确地估计出来，那么该系统对于其用户的整体效果就是基于这些数据能够获得的最好结果。

基于一些假设，例如一篇文档对于一个查询的相关性独立于其他文档，就可能证明这段陈述是正确的。从这种意义上来说，根据相关性概率的排序会在任何给定排序上（例如取排序最靠前的10个文档）使精确率（即相关文档的比例）最大化。遗憾的是，概率排序原则并没有告诉如何计算或者估计相关性的概率值。现在有很多种概率检索模型，每种模型都提出了不同的方法来估计这种概率值。本章余下的章节基本上都在讨论一些最重要的概率模型。

本节从一个将信息检索看成分类问题的简单的概率模型开始，随后介绍一个基于这个模型的流行而且有效的排序算法。

7.2.1 将信息检索作为分类问题

任何将相关性假设为二元的检索模型中，对每个查询都会有两组文档：相关文档集合和非相关文档集合。给定一个新的文档，搜索引擎的任务可以描述为判断这个文档是否属于相关集合或者非相关[⊜]集合。也就是说，系统应该判断文档是相关的还是非相关的，如果相关就返回文档。

采用某种方法来计算相关文档的概率以及非相关文档的概率，随后就能看似合理地将具

⊖ “参考检索系统”现在应该叫做搜索引擎。

⊜ 注意，在信息检索中，我们绝没有谈论“无关紧要的”文档，而是谈论“非相关的”。

有最高概率的文档进行划分。换句话说，当$P(R|D) > P(NR|D)$时，判定文档D是相关的，其中$P(R|D)$是相关性的条件概率，$P(NR|D)$是非相关性的条件概率（如图7-3所示）。这就是著名的贝叶斯决策法则，能够采用这种方式进行分类的系统称为贝叶斯分类器。

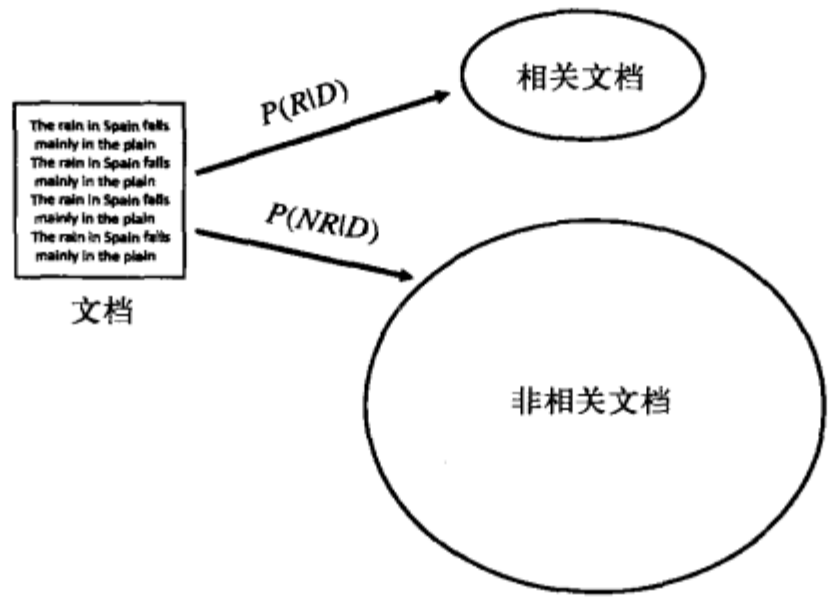


图7-3 判断一个文档是相关的还是非相关的

在第9章中会讨论分类的其他应用（例如垃圾过滤）以及其他的分类技术，但是这里专注于基于分类概率检索模型的排序算法。

当前面临的问题是如何计算这些概率。作为开始，先看$P(R|D)$。还不清楚如何能够计算它，但是给定相关集合的信息，应该能够计算$P(D|R)$。例如，如果知道在相关集合中一些特定词汇是如何出现的，那么给定一个新的文档，可以通过相对直接地计算这个文档中词汇的出现情况来进行计算。假设词语“总统”在相关文档中的概率为0.02，“林肯”的概率为0.03。如果新文档包括这两个词语，那么假设这两个词语的出现是相互独立的，并可以说观察到的相关文档中，词语整体出现的概率为$0.02 \times 0.03 = 0.0006$[⊖]。

那么，如何通过计算$P(D|R)$来获得相关性的概率呢？事实上，$P(R|D)$和$P(D|R)$之间的关系可以表示为贝叶斯法则[⊖]：

$$P(R \mid D) = \frac{P(D \mid R)P(R)}{P(D)}$$

其中$P(R)$是相关性的先验概率（换句话说，就是任何文档是相关的可能性），$P(D)$起到了归一化常数的作用。基于此，能够将决策规则采用如下方式表达：如果$P(D|R)P(R) > P(D|NR)P(NR)$，则判定文档是相关的。这和判定一篇文档相关的如下条件是一致的：

$$\frac{P(D \mid R)}{P(D \mid NR)} > \frac{P(NR)}{P(R)}$$

公式左边部分称为似然比率。在绝大多数分类应用中，例如垃圾过滤中，系统必须判定文档属于哪个类别，以便采取恰当的行动。对于信息检索，搜索引擎只需要排序文档，而不用做出分类判断（分类是困难的）。如果采用似然比率作为分值，排序较高的是那些对于属于相关

⊖ 给定两个事件A和B，联合概率$P(A \cap B)$是两个事件共同出现的概率。一般地，$P(A \cap B) = P(A|B)P(B)$。如果A和B相互独立的，那么有$P(A \cap B) = P(A)P(B)$。

⊖ 这条法则是根据英国数学家托马斯·贝叶斯命名的。

集合具有较高似然值的文档。

为了计算文档的得分，仍然需要判断如何利用$P(D|R)$和$P(D|NR)$的结果。最简单的方法就是采用前面例子中的假设，即将文档表示为词项的组合，相关集合和非相关集合表示为词项概率。在这个模型中，文档表示为一组二元向量特征，$D=(d_1, d_2, \cdots, d_t)$，其中$d_i=1$表示词项$i$出现在文档中，反之为0。其他主要的假设是词项独立性（也叫做朴素贝叶斯假设）。这意味着能够通过单独词项的概率乘积$\prod_{i=1}^{t} P(d_i|R)$来估计$P(D|R)$（类似地可以计算$P(D|NR)$）。由于假设词项独立以及文档中的二元特征，因此这个模型又称为二元独立模型。

文本中的词语显然不是独立出现的。如果词语“Microsoft”在文档中出现，那么“Windows”也很可能会出现。但是，词项独立是一个普通的假设，因为它往往能够简化模型中的数学计算。允许词项依赖的模型会在本章后续部分讨论。

回顾一下，这个模型中的文档是一组0、1值的向量，1表示出现，0表示不出现。例如，如果有五个索引词项，一篇文档可能表示为(1, 0, 0, 1, 1)，意味着文档包含词项1、4、5。为了计算这个文档在相关集合中出现的概率，需要词项在相关集合中是1或者0的概率。如果p_i是词项i在相关集合的某篇文档中出现（出现为1）的概率，那么样例文档在相关集合中出现的概率为$p_1 \times (1-p_2) \times (1-p_3) \times p_4 \times p_5$。概率$(1-p_2)$表示相关集合中词项2不出现的概率。对于非相关集合，使用s_i来表示词项i出现的概率[⊖]。

再回到似然比，使用p_i和s_i可以获得如下分值

$$\frac{P(D|R)}{P(D|NR)} = \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

其中$\prod_{i:di=1}$表示文档中值是1的词项概率的连乘。现在能够做如下一些数学推导：

$$\prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \left(\prod_{i:d_i=1} \frac{1-s_i}{1-p_i} \cdot \prod_{i:d_i=1} \frac{1-p_i}{1-s_i} \right) \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i}$$

$$= \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \prod_i \frac{1-p_i}{1-s_i}$$

第二个连乘覆盖所有词项，也能覆盖所有文档，所以对于排序能够忽略掉。由于连乘了很多较小的数值，会导致结果的精度问题，对乘积使用等价的取对数操作，即得分函数变为：

$$\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$$

你可能会想，这是一个根据特定查询的文档排序算法，但是查询项到哪里去了。很多时候，查询项只提供了相关集合的信息。可以假设，在其他信息存在的基础上，查询项中那个没有出现的词项对于相关文档和非相关文档具有相同的出现概率（即$p_i=s_i$）。也就是说，给定一个查询，文档的分值可以简单地将匹配到的词项概率求和即可。

如果没有相关集合的其他信息，可以额外假设p_i是一个常数，s_i可以被近似估计为整个文档数据集中的词项出现情况。做出第二个假设是基于这样的事实，即相关文档的数量远小于整体文档集合的大小。在前面介绍的分值方程中，设定p_i的值为0.5，给定词项i的权值为

⊖ 在许多关于这个模型的描述中，p_i和q_i用来表示这些概率。这里我们使用s_i来避免与表示查询词项的q_i混淆在一起。

$$\log \frac{0.5\left(1-\frac{n_i}{N}\right)}{\frac{n_i}{N}(1-0.5)} = \log \frac{N-n_i}{n_i}$$

其中n_i是包含词项i的文档数目，N是整个数据集中文档的数目。在存在相关文档信息的基础上，这说明从二元独立模型衍生出来的词项权值和idf权值非常相似。这里没有tf权值，因为文档已经被假设成具有二元特征了。

如果知道相关集合和非相关集合中词项的出现情况，就可以采用联立表来进行概括，如表7-1所示。这种信息可以通过用户在初始排序中识别相关文档得到的相关反馈来获得。在这个表中，r_i是包含词项i的相关文档数量，n_i是包含词项i的文档数量，N是整个文档数据集中所有文档的数量，R是和这个查询相关的文档数量。

表7-1 一个特定查询的词项出现情况的联立表

	相关	非相关	总数
d_i=1	r_i	n_i-r_i	n_i
d_i=0	$R-r_i$	$N-n_i-R+r_i$	$N-n_i$
Total	R	$N-R$	N

给定这个表格，明显地估计[⊖]p_i和s_i可以是$p_i=r_i/R$（包含词项的相关文档数量除以全部相关文档的数量），$s_i=(n_i-r_i)/(N-R)$（包含词项的非相关文档数量除以全部非相关文档的数量）。但是，采用这种估计会带来一种问题。那就是如果联立表中的一些数据为0，那么词项权值会是log0。为了避免这种情况，一个标准的解决方案是在每个数值上加0.5（整体数值上加1），这样使得新的估计为$p_i=(r_i+0.5)/(R+1)$和$s_i=(n_i-r_i+0.5)/(N-R+1.0)$。将这些估计放入整体的得分函数，得到：

$$\sum_{i:d_i=q_i=1} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)}$$

虽然这个文档得分只是将查询项中匹配上的词项权值累加起来，但是在相关反馈中，查询项可以扩展到包含相关集合的其他重要词项。注意，如果没有任何相关信息，可以设定r和R为0，这会使p_i为0.5，并得到前面讨论过的像idf那样的词项权值。

那么，这个文档函数在用于排序时有多好呢？事实证明，不是非常好。虽然它提供了融合相关信息的方法，但是在大多数情况下，并没有这些信息，只能使用类似于idf权值的词项权值。tf部分的存在，对排序的性能有显著的作用。在排序时如果没有这种信息，绝大多数的有效度量方法都会下降50%。也就是说，如果使用二元独立模型排序代替最好的tf.idf排序，将会在排序结果的最佳部分少看到50%的相关文档。

但是，事实上二元独立模型是最有效和流行的排序算法之一（称为BM2[⊖]）的基础模型。

7.2.2 BM25排序算法

BM25模型通过加入文档权值和查询项权值，拓展了二元独立模型的得分函数。这种拓展

⊖ 我们使用“估计”这个词语来表示使用例如联立表等数据来计算概率值，这是因为这个数值仅仅是对真实概率值的估计，如果有更多的数据，这个值会发生变化。

⊖ BM是最佳匹配（Best Match）的缩写，25只是Robertson和他的同事在保持权值变化时的一种数值规定(Robertson & Walker, 1994)。

是基于概率论和实验验证的，并不是一个正式的模型。

BM25在TREC检索实验上表现非常好，而且对包括网页搜索引擎在内的商业搜索引擎中的排序算法影响很大。现在BM25的得分函数有很多种变形，但是最普通的形式如下：

$$\sum_{i\in Q}\log\frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)}\cdot\frac{(k_1+1)f_i}{K+f_i}\cdot\frac{(k_2+1)qf_i}{k_2+qf_i}$$

其中求和公式覆盖查询项中的所有词项，N、R、n_i、r_i和上一节的描述一致，额外的条件是，如果没有相关信息，那么r和R都被置为0；f_i是词项i在文档中的频率；qf_i是词项i在查询项中的频率；k_1、k_2、K都是经验设定的参数。

常量k_1决定f_i增加时tf部分的词项权值如何变化。如果k_1=0，那么词项频率部分将被忽略，只有词项的存在与否会有所影响。如果k_1很大，那么词项权重部分会随着f_i线性增长。在TREC的实验中，k_1的典型值是1.2。这使得f_i的非线性影响是很大的。这和7.1.2节中讨论过的词项权重中的log$f$的作用类似。这意味着，一个词项在出现三到四次后，后继的出现只有很小的影响。常量k_2在查询项权重中具有类似的作用。这个参数典型的数值范围是0到1000，也就是说，系统性能对k_2的敏感性不如k_1。这是因为查询词项的频率非常低，并且比文档词项频率变化小。

K是一个更加复杂的参数，用来利用文档长度归一化tf因子。具体地

$$K=k_1\left((1-b)+b\cdot\frac{dl}{avdl}\right)$$

其中，b是一个参数，dl是文档长度，$avdl$是数据集中文档的平均长度。常量b控制长度归一化的影响，其中b=0对应于没有长度归一化，b=1表示完全的归一化。在TREC实验中，b=0.75被证明是有效的。

举一个计算的例子，考虑一个包含两个词项（“总统”和“林肯”）的查询，每个词项在查询中仅出现一次（qf=1）。考虑典型的情形，即没有相关信息（r和R都是0）。假设正在搜索一个含有500 000（N）个文档的数据集，在这个集合中“总统”出现在40000个文档中（n_1=40 000），“林肯”出现在300个文档中（n_2=300）。在评分的这个文档（关于林肯总统）中，“总统”出现15次（f_1=15），“林肯”出现25次（f_2=25）。文档长度只有平均长度的90%（$dl/avdl$=0.9）。相关参数使用k_1=1.2、b=0.75、k_2=100。基于这些数据，K=1.2 · (0.25+0.75 · 0.9)=1.11，最终的文档得分如下：

$$\begin{aligned}\mathrm{BM25}(Q,D)&=\log\frac{(0+0.5)/(0-0+0.5)}{(40\,000-0+0.5)/(500\,000-40\,000-0+0+0.5)}\\&\quad\times\frac{(1.2+1)15}{1.11+15}\times\frac{(100+1)1}{100+1}\\&\quad+\log\frac{(0+0.5)/(0-0+0.5)}{(300-0+0.5)/(500\,000-300-0+0+0.5)}\\&\quad\times\frac{(1.2+1)25}{1.11=25}\times\frac{(100+1)1}{100+1}\\&=\log 460\,000.5/40\,000.5\cdot 33/16.11\cdot 101/101\\&\quad+\log 499\,700.5/300.5\cdot 55/26.11\cdot 101/101\\&=2.44\cdot 2.05\cdot 1+7.42\cdot 2.11\cdot 1\\&=5.00+15.66=20.66\end{aligned}$$

注意，在没有相关信息时，权值的第一部分的影响几乎和*idf*权值一样（正如7.2.1节介绍的那样）。因为词项“林肯”在文档数据集中只有很小的频率，它具有较高的*idf*数值（7.42对比于2.44）。表7-2给出了不同词项出现次数的分值。这说明词项“林肯”的重要性，即使只出现一次也使得分值发生重大变化。将词项出现次数从25或者15减小到1，对最终结果影响并不大。这个例子同时也说明，一个包含很多次出现的单独词项的文档，会比同时包含两个查询词项的文档获得更高的分数。

表7-2 样例文档的BM25分值

“总统”的频率	“林肯”的频率	BM25分数
15	25	20.66
15	1	12.74
15	0	5.00
1	25	18.2
0	25	15.66

这个得分的计算看似繁琐，但是记住一点，在处理任何查询之前，词项权值的一些计算已经出现在索引阶段。如果没有相关信息，计算一个文档的得分只是简单地包括加入查询项中匹配上的词项的权重，如果查询词项多次出现（即$qf>1$），则会有一些较小的额外计算。另外一点是，BM25排序算法中的参数可以针对每种应用进行调节（例如调整使得达到最佳的性能）。调节参数的过程会在7.7节以及第8章中进一步描述。

小结一下，BM25模型是从将信息检索视为分类问题的模型中演化出来的一种有效的排序算法。这个模型关注于主题相关，并且显式地假设相关性是二元的。在下一节中，会讨论另外一种概率模型。这种模型能够直接融合词项频率，而不是通过扩展增加的方式来提高性能。

7.3 基于排序的语言模型

语言模型在很多语言技术中被用来表示文本，例如语音识别、机器翻译、手写体识别等。最简单的语言模型是一元语言模型，也就是语言中词汇的概率分布。这意味着语言模型是文本集合的索引词表中每个词语的出现概率。例如，如果文档数据集中只包含五个不同的词语，这个集合一个可能的语言模型是（0.2, 0.1, 0.35, 0.25, 0.1），其中每个数值表示词语出现的概率。如果将每个文档看成是词汇的一个序列，那么语言模型的概率就是预测序列中下一个词语的概率。例如，如果语言中的五个词汇分别是“猫”、“该”、“男孩”、“触摸”，那么可以预测的下一个词汇的概率是这些词语的概率之一。这些词语覆盖了所有的可能性，所以概率的总和为1。由于是一元模型，前一个词语对预测没有任何影响。例如，在这个模型下，可能会得到序列“女孩 猫”（概率0.2×0.1）和序列“女孩 触摸”（概率0.2×0.1）同样的可能性。

在语音识别等应用中，n元语言模型使用更长序列来预测词语。一个n元模型预测词语时考察前面的$n-1$个词语。最普通的n元模型是二元模型（使用前一个词语来预测）和三元模型（使用前面两个词语来预测）。虽然二元模型在信息检索中用来表示两个词语的短语（见4.3.5节），但会集中讨论一元模型，因为它更加简单而且被证明作为排序算法的基础是非常有效的。

对于检索的应用，使用语言模型来表示一篇文档的话题内容。话题这个概念经常被提到，但是却很少在信息检索讨论中被定义过。在这个方法中定义一个话题就是词汇上的一个概率

分布（换句话说，就是语言模型）。例如，如果是一篇关于在阿拉斯加钓鱼的文档，我们希望看到在语言模型中，关于钓鱼和阿拉斯加地点的词汇具有高的概率值。如果是关于在佛罗里达钓鱼的文档，一些高概率的词语会是一样的，但是会有一些关于佛罗里达地点的词汇具有高的概率值。换个情况，如果文档是关于钓鱼的电脑游戏，虽然会有很多关于钓鱼的词汇，但是绝大多数的高概率词汇会和游戏厂商以及计算机操作相关。注意，话题语言模型，简称为话题模型，包括了所有词汇的概率，而不仅仅是最重要的那些。绝大多数的词汇会具有默认的概率，这种概率对于任何文本都会是一样的，但是对话题很重要的词汇会具有异常高的概率值。

表示文档的语言模型可以被用来通过根据概率分布随机采样词汇来“生成”新的文档。如果假设语言模型是一个装满词语的桶，其中的概率值决定同种词语的实例个数，那么生成文档的方式可以通过伸手到桶中（不能看）拿出一个词语并写下来，然后把词语放回到桶中，然后再取。注意，并没有说能够通过这种过程生成原始文档。事实上，由于只用了一元模型，生成的文本将会因为没有任何句法结构而非常的糟糕。但是与文档主题相关的重要词汇经常出现。直觉上，使用语言模型可以很好地近似作者在撰写文档时构思的主题。

当文本被看成是一个词语的有穷序列模型时，序列中的每个点都有可能是t种不同的词语，这就能对应到词汇上的多项式分布。虽然有很多其他替代，但多项式语言模型在信息检索中是最普通的[⊖]。已经有人指出，多项式模型的一个不足是，不能很好地描述爆发性，即如果观察到一个词语“从桶中被取出”，那么它会倾向于被重复取出。

将文档表示为语言模型后，同样能够将查询的主题表示为语言模型。这种情况下，语言模型是主题的一种表示。这种主题是搜索信息的人在写下查询时头脑中所想的内容。这就导致了三种基于语言模型的检索概率值：一种是基于从文档语言模型生成查询文本的概率;一种是基于从查询项语言模型生成文档文本的概率;一种是基于对比查询项语言模型和文档主题语言模型的结果。在后续的两个小节中，会更加详细地描述这些检索模型。

7.3.1 查询项似然排序

在查询项似然检索模型中，根据文档语言模型生成查询文本的概率来对文档排序。换句话说，计算能够从表示文档的“桶”中取出查询项词语的概率。这是一个主题相关模型，因为查询项的生成概率是文档在同样话题上和查询项的接近程度的度量。

一般地，从一个查询项开始，可以通过计算$P(D|Q)$来对文档排序。根据贝叶斯法则，计算如下：

$$p(D|Q) \overset{rank}{=} P(Q|D)P(D)$$

其中，符号$\overset{rank}{=}$正如前面所述，表示右侧部分和左侧部分是排序等价的（因此，可以忽略掉归一化因子$P(Q)$），$P(D)$是文档的先验概率，$P(Q|D)$是给定文档后查询的似然函数。绝大多数情况下，$P(D)$都被假设是始终如一的（对所有文档都一样），所以不会影响到排序结果。被赋予非一致先验概率（例如文档日期或者文档长度）的模型在一些应用中是很有用的，但是这里只假设简单的一致先验概率。基于这种假设，检索模型通过$P(Q|D)$来排序文档，这个概率值可以采用文档一元语言模型来计算，如下所示

⊖ 在第9章的分类部分讨论多项式模型。

$$P(Q \mid D) = \prod_{i=1}^{n} P(q_i \mid D)$$

其中q_i是查询项中的词，查询项中有n个词语。

为了计算这个分值，需要估计语言模型概率值$P(q_i|D)$。明显的估计如下

$$P(q_i \mid D) = \frac{f_{q_i,D}}{|D|}$$

其中，$f_{qi,D}$是词语q_i在文档集合D中出现的次数，$|D|$表示D中词语的数量。对于多项式分布，这是一个极大似然估计，也就是说会将观察值$f_{qi,D}$最大化。这个估计的主要问题是，如果查询项中任何一个词语没有在文档中出现的话，那么查询似然模型给$P(Q|D)$的分值都会是0。显然这不适用于较长的查询。例如，查询中有6个词项，如果只有一个词项设在文档中出现的话，$P(Q|D)$以分值不应为0。这同样也不能区别那些缺少查询项不同数目词语的文档。另外，由于是为一篇文档建立一个主题模型，具体主题的相关词汇即使不会在文档中出现，也需要具有一定的出现概率。例如，一个关于电脑游戏文档的语言模型，即使“RPG”这个词语不会在文档中提到，也应该对该词语具有非零概率。这个词语的一个较小的概率值，可以使得文档对于查询“RPG 电脑 游戏”获得一个非零的分值，尽管会比某个包含全部三个词语的文档的概率值低一点。

平滑技术用于避免这种估计问题以及数据稀疏问题，这意味着我们不需要使用大量文本来估计语言模型的概率。平滑技术一般的方法是降低（或者打折）文档文本中出现词语的估计概率，并对文本中未出现的词语赋给估计的“剩余”概率。未出现词语的概率通常都是基于整个文档数据集中词语的出现频率来进行估计的。如果$P(q_i|C)$是文档集合C的数据集语言模型中词语i的出现概率，那么文档中未出现词语的估计概率为$\alpha_D P(q_i|C)$，其中α_D是控制赋予未见词语概率的系数[⊖]。一般来说，α_D依赖于文档。为了保证概率值和为1，文档中一个出现过的词语的概率被估计为$(1-\alpha_D)P(q_i|D)+\alpha_D P(q_i|C)$。

为了弄清楚这一点，考虑一个简单例子。这个例子只包含索引库中的三个词语w_1、w_2、w_3。如果这三个词语基于极大似然估计的数据集概率分别是0.3、0.5、0.2，基于极大似然估计的文档概率分别是0.5、0.5、0.0，那么对于文档语言模型，这个文档的平滑概率估计如下：

$$\begin{aligned} P(w_1 \mid D) &= (1-\alpha_D)P(w_1 \mid D) + \alpha_D P(w_1 \mid C) \\ &= (1-\alpha_D)\cdot 0.5 + \alpha_D \cdot 0.3 \\ P(w_2 \mid D) &= (1-\alpha_D)\cdot 0.5 + \alpha_D \cdot 0.5 \\ P(w_3 \mid D) &= (1-\alpha_D)\cdot 0.0 + \alpha_D \cdot 0.2 = \alpha_D \cdot 0.2 \end{aligned}$$

注意，虽然词项w_3没有出现在文档文本中，但是仍然具有非零概率估计。如果对这三个概率值相加，可以得到

$$\begin{aligned} P(w_1 \mid D) + P(w_2 \mid D) + P(w_3 \mid D) &= (1-\alpha_D)\cdot(0.5+0.5) \\ &\quad + \alpha_D \cdot (0.3+0.5+0.2) \\ &= 1-\alpha_D+\alpha_D \\ &= 1 \end{aligned}$$

⊖ 集合语言模型概率也叫做背景模型概率，或者是背景概率。

仍然保证概率是一致的。

α_D赋值的方式导致不同的估计形式。最简单的选择就是设定为常数，即$\alpha_D=\lambda$。集合语言模型概率估计中，估计词语q_i的概率为$C_{qi}/|C|$，其中C_{qi}是文档数据集中查询词出现的次数，$|C|$是集合中所有词语出现次数的总和。这样就可以估计$P(q_i|D)$如下：

$$p(q_i \mid D) = (1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}$$

这种形式的平滑称为Jelinek-Mercer方法。在查询似然模型的文档得分中替换掉这个估计，可以得到：

$$P(Q \mid D) = \prod_{i=1}^{n}\left((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{C_{q_i}}{|C|}\right)$$

正如前面所说，由于连乘很多很小的数值会导致精确率的问题，可以使用取对数的方法来将这个得分转换为等价排序求和，如下所示：

$$\log P(Q \mid D) = \sum_{i=1}^{n}\log(1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}$$

λ的较小值会导致较小的平滑，并且查询会倾向于变得更像布尔操作AND。因为任何查询词的缺失都会在本质上惩罚这个得分。更进一步地，极大似然估计得到的词语的相关性权值对确定分值是重要的。如果λ接近1，相对性权值会变得更不重要，此时查询会变得更像布尔操作的OR或者协调水平匹配[⊖]。TREC评测显示，对于短查询λ接近0.1时效果较好，对于长查询λ接近0.7时效果较好。短查询倾向于只包含显著性的词语，较低的λ值会偏爱包含所有查询词的文档。在较长的查询中，缺少一个词语的影响较小，较高的会更加强调包含许多高概率词语的文档。

在这一点上，你会发现查询似然检索模型一点都不像tf.idf权值。实验证明它只能达到BM25排序算法非常低的性能。但是，可以通过如下操作查询似然得分来展示它和tf.idf权值的关系：

$$\begin{aligned}
\log P(Q \mid D) &= \sum_{i=1}^{n}\log(1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{qi}}{|C|} \\
&= \sum_{i:f_{q_i,D}>0}\log\left((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}\right) + \sum_{i:f_{q_i,D}=0}\log\left(\lambda\frac{c_{q_i}}{|C|}\right) \\
&= \sum_{i:f_{q_i,D}>0}\log\frac{\left((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}\right)}{\lambda\frac{c_{q_i}}{|C|}} + \sum_{i=1}^{n}\log\left(\lambda\frac{c_{q_i}}{|C|}\right) \\
&\overset{rank}{=} \sum_{i:f_{i,D}>0}\log\left(\frac{(1-\lambda)\frac{f_{q_i,D}}{|D|}}{\lambda\frac{c_{q_i}}{|C|}} + 1\right)
\end{aligned}$$

⊖ 协调水平匹配简单地通过匹配上查询词项的数目来对文档排序。

在第二行中，将得分划分到出现在文档中的词语和不出现的词语（$f_{q_i,D}=0$）。在第三行，在最后一项增加

$$\sum_{i:f_{q_i,D}>0}\log\left(\lambda\frac{c_{q_i}}{|C|}\right)$$

并且在第一项减去它（它是分母的结尾），所以这里没有有效效应。对于所有文档，最后一项也是一样，可以在排序中忽略掉。最终的表达式通过对匹配上的查询词项加权获得文档的得分。虽然这个权值和*tf.idf* 不完全一样，但明显的关系是，这个权值正比于文档词项频率，并且正比于数据集频率。

一般来说，一种不同的更加有效的估计形式，来自于使用依赖于文档长度的变量α_D。这种方法称为狄利克雷平滑，这样命名的原因后续会阐述，变量定义如下

$$\alpha_D=\frac{\mu}{|D|+\mu}$$

其中μ是一个参数，其值根据经验设定。在$(1-\alpha_D)P(qi|D)+\alpha_D\ P(qi|C)$中替换$\alpha_D$，得到概率估计公式：

$$p(q_i|D)=\frac{f_{q_i,D}+\mu\frac{c_{q_i}}{|C|}}{|D|+\mu}$$

这个公式能够导致下面的文档得分函数：

$$\log P(Q|D)=\sum_{i=1}^{n}\log\frac{f_{q_i,D}+\mu\frac{c_{q_i}}{|C|}}{|D|+\mu}$$

和Jelinek-Mercer平滑方法类似，参数（这里对应μ）较小的数值对词语的相关性权值赋予了更大的重要性，较大的数值偏向于匹配上的词项数目。在TREC实验中，能够达到最佳效果的μ的数值范围在1000到2000之间（因为数据集概率都非常小）。一般的，狄利克雷平滑比Jelinek-Mercer平滑更有效，尤其是在绝大多数具有短查询的搜索应用中。

那么，狄利克雷平滑是从哪里来的呢？事实证明，狄利克雷分布[⊖]在多项式分布的概率估计中考虑先验知识的自然方式。贝叶斯估计确定估计概率的过程就是基于这种先验知识和观察文本的。最终的概率估计可以看成是结合了文本中的实际词项计数以及来自狄利克雷分布的伪计数。如果没有文本，词项q_i的概率估计可以是$\mu(C_{q_i}/|C|)/\mu$。这是基于数据集的一个合理猜测。拥有越多的文本（即更长的文档），先验知识就具有越小的影响。

可以通过使用7.2.2节中的样例来展示查询似然文档得分的计算过程。两个查询词项分别是“总统”和“林肯”。对于词项“总统”，$f_{q_i,D}=15$，并假设$C_{q_i}=160000$。对于词项“林肯”，$f_{q_i,D}=2.5$，并假设$C_{q_i}=2400$。文档中词语出现的次数$|d|$假设是1800，文档数据集中词语出现总的次数假设是10^9（500000个文档，平均每个文档2000个词）。μ的值假设是2000。给定这些数字，文档的得分是：

⊖ 根据德国数学家Johann Peter GustavLejeune Dirichlet（这个姓氏非常罕见）命名。

$$\begin{aligned}\mathrm{QL}(Q,\ D) &= \log\frac{15+2000\times(1.6\times10^5/10^9)}{1800+2000}\\ &\quad+\log\frac{25+2000\times(2400/10^9)}{1800-2000}\\ &= \log(15.32/3800)+\log(25.005/3800)\\ &= -5.51+-5.02=-10.53\end{aligned}$$

结果是负数？这里谈到在得分函数中对概率取对数的操作，同时词语出现的概率值很小。重要的是，利用这些得分得到排序结果的性能。表7-3展示了表7.2中用到的词项出现情况的查询似然得分。虽然BM25模型和查询似然的得分差异较大，但是排序结果却是相似的，仅有的一个例外，就是包含15次“总统”和1次“林肯”的文档排序，在查询似然上高于包含0次“总统”和25次“林肯”的文档，但是相反的情况在BM25上是正确的。

表7-3 一个样例文档的查询似然得分

“总统”的频率	“林肯”的频率	QL分数
15	25	−10.53
15	1	−13.75
15	0	−19.05
1	25	−12.99
0	25	−14.40

总结一下，查询似然是一种简单的概率检索模型，其中直接融合了词项频率。词项权值的有效性问题更容易理解而且由正规基础的概率估计取代。虽然BM25在绝大多数TREC数据集上表现更好，但是基本的查询似然得分结合狄利克雷平滑后，和BM25具有类似的性能。如果用上基于主题模型的更加复杂的平滑（在7.6节进一步描述），那么查询似然一定会超过BM25。这说明，对于使用词语的集合概率来替代平滑，我们使用相似文档中的词语概率来替代。

语言模型框架的简洁性，结合对多种检索应用的描述能力以及相关排序算法的有效性，使得这种方法对于基于主题相关性的检索模型是一个好的选择。

7.3.2 相关性模型和伪相关反馈

虽然基本的查询似然模型有很多优点，但由于描述信息需求和查询的方式使它还是受限的。例如，它很难将关于相关文档的信息融合到排序算法中，而且难以表达这样一个事实，即查询只是能够描述特殊信息需求的多种可能的查询之一。在本节中会展示如何通过扩展这个模型来解决这些问题。

在7.3节中提到可以将一个查询的主题表示为语言模型。我们不是将它称为查询语言模型，而是使用相关性模型这个名称，因为它表示相关文档覆盖的话题。查询可以看成是相关性模型生成文本的非常小的样本，相关文档可以看成是同一模型生成的文本的较大样本。给定关于查询的一些相关样例文档，能够估计相关性模型中的概率，然后使用这个模型来预测新文档的相关性。事实上，这个版本的分类模型在7.2.1节有所介绍，其中解释$P(D|R)$为给定相关性模型生成文档中的文本的概率。这个模型也叫做文档似然模型。不像二元独立模型那样，这个模型虽然直接融合了词项频率，但是事实上$P(D|R)$对比于跨文档是难以计算的。这是因为对比于查询，文档包含了非常大量的词语变量。例如，考虑两个文档D_a和D_b，分别包含5个

和500个词语。由于包含词语数量的巨大差异，对于排序而言，$P(D_a|R)$和$P(D_b|R)$的比较会比使用同样查询和文档平滑表示的$P(Q|D_a)$和$P(Q|D_b)$的比较困难得多。进一步地，仍然在获得相关文档的样例方面存在困难。

这里是另外一种方法。如果能够从一个查询估计一个相关性模型，那么就能将这个语言模型和文档模型直接进行对比。文档会根据文档模型和相关性模型的相似程度进行排序。一篇文档的模型如果和相关性模型非常类似，那么这篇文档可能是关于同一主题的。随后显然的问题就是如何比较两个语言模型。在概率论和信息论中，一个著名的度量名叫Kullback-Leibler分散度（本书中简称为KL-分散度[⊖]），能够度量两个概率分布的差异。给定真实的概率分布P以及另外一个用于估计P的概率分布Q，KL-分散度定义为：

$$\mathrm{KL}(P \| Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

由于KL-分散度总是正数而且对于比较稀疏的分布数值会很大，使用取负的KL-分散度来作为排序函数的基础（即较小的差异意味着较大的分值）。另外，KL-分散度是非对称的，这决定于选择哪个分布为真实分布。如果假设真实分布是查询（R）的相关性模型，近似分布为文档语言模型（D），那么负数的KL-分散度可以表示为：

$$\sum_{w \in V} P(w|R) \log P(w|D) - \sum_{w \in V} P(w|R) \log P(w|R)$$

其中求和公式是对整个词表V中的所有词语w进行的。公式右边第二项并不依赖于文档，所以可以在排序中忽略掉。给定一个简单的极大似然估计$P(w|R)$，基于查询文本的频率（$f_{w,Q}$）以及查询中词语的数目($|Q|$)，文档的得分计算如下：

$$\sum_{w \in V} \frac{f_{w,Q}}{|Q|} \log P(w|D)$$

虽然求和公式能够覆盖词表中的所有词语，但查询中没有出现的词语的极大似然估计值为0，而且不会对得分有任何贡献。另外，具有频率k的查询词对分值的贡献为$k \times \log P(w|D)$。这意味着这个分数和前面章节中描述的查询似然得分是排序等价的。换句话说，查询似然是检索模型的一种特殊情况，通过比较基于查询的相关性模型和文档语言模型来实现。

这个更加普通的模型的优点，是没有限制使用查询词项频率来估计相关性模型。如果将查询词看成是相关性模型的一种采样，那么基于见过的查询词的新采样的概率，就看起来是合理的。换句话说，从表示相关性模型的“桶”中取出词语的概率，必须依赖于刚取出的n个查询词语。更正式地，可以估计w的概率为给定已经观察到的查询词$q_1 \cdots q_n$下，观察到w的条件概率，近似估计如下：

$$P(w|R) \approx P(w|q_1 \cdots q_n)$$

根据定义，可以根据观察到的w和查询词的联合概率来表示条件概率：

$$P(w|R) \approx \frac{P(w, q_1 \cdots q_n)}{P(q_1 \cdots q_n)}$$

⊖ KL-也叫做信息分散度、信息增益、相对熵。

$P(q_1 \cdots q_n)$是一个归一化常量，计算如下：

$$P(q_1 \cdots q_n) = \sum_{w \in V} P(w, q_1 \cdots q_n)$$

现在的问题是，如何估计联合概率$P(w, q_1 \cdots q_n)$。给定一组表示为如下概率的文档集合C：

$$P(w, q_1 \cdots q_n) = \sum_{D \in C} p(D) P(w, q_1 \cdots q_n \mid D)$$

也可以做出如下假设：

$$P(w, q_1 \cdots q_n \mid D) = P(w \mid D) \prod_{i=1}^{n} P(q_i \mid D)$$

将这个表达式替换掉前面公式中的$P(w, q_1 \cdots q_n|D)$，可以得到下面的联合估计概率：

$$P(w, q_1 \cdots q_n) = \sum_{D \in C} P(D) P(w \mid D) \prod_{i=1}^{n} P(q_i \mid D)$$

如何解释这个公式呢？先验概率$P(D)$经常被假设为统一数值，可以忽略。表达式$\prod_{i=1}^{n} P(q_i|D)$事实上是对文档D的查询似然得分。这意味着，对$P(w, q_1 \cdots q_n)$的估计是在一组文档中对w的语言模型概率的权值的平均结果，这些权值是对那些文档的查询似然得分。

基于相关性模型的排序，实际上需要两轮操作。第一轮是利用查询似然来排序，进而得到相关性模型估计中需要的权值。在第二轮中，使用KL-分散度，以通过对比相关性模型和文档模型来对文档进行排序。注意，实际上是通过在查询中添加词语来实现对查询相似文档的相关性模型的平滑。在基于查询频率估计的相关性模型中，很多具有0概率的词语现在都会具有非零概率值。这里，将明确介绍在6.2.4节说明过的伪相关反馈。换句话说，相关性模型为伪相关反馈和查询扩展提供了一个正式的检索模型。下面是利用相关性模型进行排序的概括步骤：

1）根据对查询Q的查询似然得分对文档排序；

2）选择排序靠前的某个数目的文档构成集合C；

3）利用估计概率$P(w, q_1 \cdots q_n)$来计算相关性模型概率$P(w|R)$；

4）利用下面的KL-分散度来对文档进行再次排序[⊖]：

$$\sum_{w} P(w \mid R) \log P(w \mid D)$$

这些步骤中的有些需要进一步的解释。在第1步和第4步中，文档语言模型概率($P(w|D)$)需要采用狄利克雷平滑方法来进行估计。在第2步中，模型允许集合C作为整个数据集，但是由于对估计$P(w|R)$排序较低的文档作用很小，通常只适用排序在10～50的文档。这也使得计算$P(w|R)$快了很多。

由于类似的原因，第4步的求和没有在词表中的所有词语上进行。一般只有较小数量（10～25）的高概率词语才会被用到。另外，原始查询词语的重要性通过在相关性模型中结合原始查询频率来估计。这个估计使用与Jelinek-Mercer相似的方法，即$\lambda P(w|Q)+(1-\lambda)P(w|R)$，其中$\lambda$是混合参数，其值一般都是经验确定的（在TREC实验中典型的值是0.5）。这种结合使

⊖ 更准确地说，这个得分是负的交叉熵，因为删除了项$\Sigma_{w \in V} P(w|R) \log P(w|R)$。

得估计相关性模型对于查询扩展和平滑成为一个清晰的过程。

正如所有的检索模型一样，另外一个重要的问题就是这个模型结果如何。基于TREC实验，使用相关性模型的排序是最好的伪相关反馈技术之一。另外，相关性模型对比与查询似然排序在一组查询上获得了显著的性能改进。但是，像所有当前的伪相关反馈技术一样，这种改进是不一致的，一些查询上会产生较差的排序或者奇怪的结果。

表7-4和7-5展示了使用这种技术在TREC 20世纪90年代的新闻语料的一大组数据集上进行的一些样例查询时，采用相关性模型估计得到的概率最高的16个词语[⊖]。

表7-4 四个样例查询的相关性模型产生的高概率词项（采用最靠前的10个文档进行估计）

president lincoln	*abraham lincoln*	*fishing*	*tropical fish*
lincoln	lincoln	fish	fish
president	america	farm	tropic
room	president	salmon	japan
bedroom	faith	new	aquarium
house	guest	wild	water
white	abraham	water	species
america	new	caught	aquatic
guest	room	catch	fair
serve	christian	tag	china
bed	history	time	coral
washington	public	eat	source
old	bedroom	raise	tank
office	war	city	reef
war	politics	people	animal
long	old	fishermen	tarpon
abraham	national	boat	fishery

表7-5 四个样例查询的相关性模型产生的高概率词项（采用最靠前的50个文档进行估计）

president lincoln	*abraham lincoln*	*fishing*	*tropical fish*
lincoln	lincoln	fish	fish
president	president	water	tropic
america	america	catch	water
new	abraham	reef	storm
national	war	fishermen	species
great	man	river	boat
white	civil	new	sea
war	new	year	river
washington	history	time	country
clinton	two	bass	tuna
house	room	boat	world
history	booth	world	million
time	time	farm	state
center	politics	angle	time
kennedy	public	fly	japan
room	guest	trout	mile

⊖ 这是一个非常大的文档集合，规模超过第6章中用于生成词项关联表的语料。第6章中的那些表都是基于鲁棒跟踪数据获得的，这些数据由超过50万个文档组成。这里的表是在超过6百万个文档组成的TREC数据集上生成的。

需要注意的是，虽然这些词语都是合理的，但是它们都非常依赖于数据集中使用的文档。例如，在TREC新闻语料中，许多故事提到亚伯拉罕林肯都是关于林肯在白宫卧室的主题。这个卧室被克林顿总统用于招待客人，而林肯总统在美国内战中作为办公室。在查询“总统林肯”和“亚伯拉罕林肯”时，这种类型的故事会反映在概率排序靠前的词语中。使用这些词语来扩展查询，显然会使得检索偏爱于这种类型的故事，而不是关于林肯的一般传记。另外一个需要注意的就是，基于10篇文档的词语和基于50篇文档的词语没有很大的差异。但是，基于50篇文档的词语在一定程度上更加普通，因为较大规模的文档集合包含了更加多样的主题。在查询实例“热带鱼”的结果中，基于10篇文档的相关性模型词语会明显更加接近于主题。

小结一下，通过根据KL-分散度来比较查询模型和文档模型得到排序结果，是查询似然得分的一种推广。这种推广允许更加精确的查询，以反映话题词语的相对重要性。这些话题是检索者在写下查询时头脑中的想法。相关性模型是一种基于正式语言模型框架的有效伪相关反馈技术。但是就像这类技术一样，在具体检索应用中，需要谨慎使用基于相关性模型的查询扩展。

语言模型提供了正式但直接的方法来描述基于主题相关性的检索模型。即使更加复杂的模型可以基于融合词项依赖度和短语来开发，主题相关性也只是有效搜索需求的一个部分而已。在下一节中，会专注于融合所有有利于用户相关性的各种证据的检索模型。这种模型是用户使用搜索引擎时真正关心的。

7.4 复杂查询和证据整合

高效检索需要将与文档可能相关的许多片段式的证据组合起来。在前面章节中介绍的检索模型中，这种证据都由反应主题内容的词语的出现次数组成。但是，一般来说，许多其他类型的证据可以考虑进来。对于词语，可以想到考虑特定词语是否出现在彼此靠近的位置，或者词语是否出现在特定的文档结构中，例如章节标题或者题目，或者词语是否彼此相关。另外，诸如发布日期、文档类型或者检索中的PageRank值都非常重要。虽然例如查询似然或者BM25等检索算法能够扩展后包含一些类型的证据，但是启发式地依赖于通过调整参数和适应新的检索应用来“解决”检索算法问题是困难的。相反，真正需要的是一个能够藐视不同类型证据的框架，例如相对重要性以及它们是如何结合的。应用于商业应用和开源搜索引擎中（同时被融合到Galago中）的推理网络检索模型，是能够做这种事情的一种方法。

推理网络模型是一种基于贝叶斯网络的形式化方法的概率模型。该模型提供了一种在查询语言中定义和评价操作运算符的机制。这些操作符中一些被用于确定证据的类型，其他用于描述如何被融合。这里将推理网络的版本使用语言模型来估计用于评价查询的概率值。

在这一节中，首先给出一个推理网络模型的概述，然后展示这个模型如何用来作为搜索引擎应用中强大的查询语言的基础。下一节中，将介绍网络搜索并解释推理网络模型如何用来结合多种用于高效排序的证据来源。

采用推理网络查询语言的查询比使用两三个词语的简单文本查询要复杂得多。绝大多数用户不能理解这门语言，正如绝大多数的关系数据库用户不能理解结构化查询语言（SQL）一样。相反，有些应用可以翻译简单的用户查询到更加复杂的查询网络版本。越复杂的融合额外特征和权值查询，就越能反映高效排序中更好的正确融合。这个观点会在下两节中讨论的例子而变得更加明确。

7.4.1 推理网络模型

贝叶斯网络是一种用于区分一组事件和事件之间依赖关系的概率模型。这种网络是有向无环图（DAG），图中的节点表示带有一组可能输出的事件，弧表示事件之间的依存概率。特定事件输出的概率或者置信度[⊖]可以通过给定父节点事件的概率来确定（或者根节点部分给定一个先验概率）。用于检索模型时，节点表示事件，例如观察到的一个特定文档、一种特定的证据片段或者一些证据片段的组合。这些事件都是二元的，可能的输出值只有真（TRUE）或假（FALSE）。

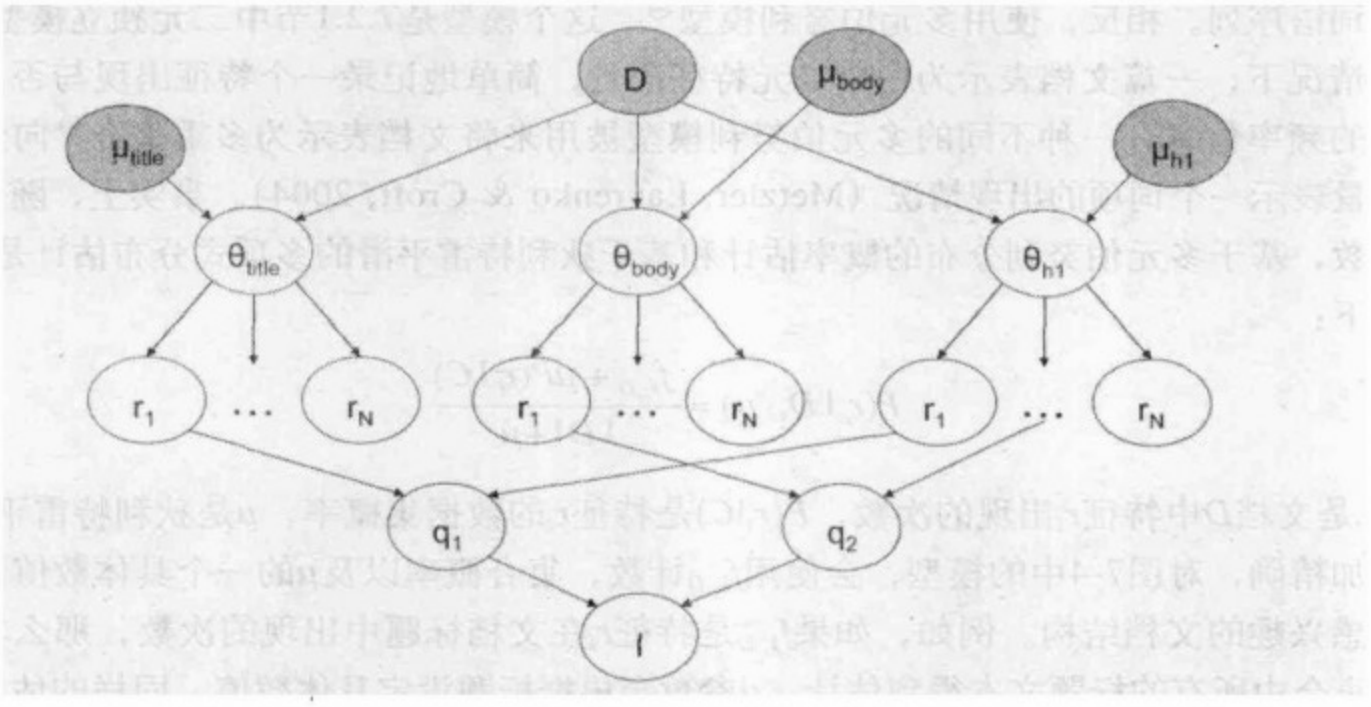


图7-4 推理网络模型样例

图7-4展示了一个推理网络，其中被融合的证据有网页的标题（title）、正文（body）和所有<h1>标题（heading）。在这个图中，D是文档节点，这种节点对应于观察到一个文档（网页）的事件。文档数据集中，每篇文档对应一个文档节点，假设每次只考察一个文档。r_i表示的节点是文档特征（证据）。这些特征相关的概率都基于使用参数μ估计得到的语言模型θ。每种文档结构（title、body或者headings）对应一个语言模型。除了基于词语出现的特征外，r_i节点也表示邻近特征。邻近特征有很多种不同的形式，例如在一个特定（长度）“窗口”的文本中的词语共现情况。这种窗口会在下一节中详细介绍。还有基于语言模型的特征也是允许的，例如文档日期，只是没有出现在这个例子中。

查询节点q_i用于将表示型节点的证据和其他查询节点结合起来。这些节点表示更加明确的证据和文档特征的共现情况。采用最简单的布尔逻辑的AND和OR操作，可以实现多种结合方式。整个网络用于计算$P(I|D, \mu)$，这个概率表示给定文档和参数μ的是一种信息需求。节点I的信息需求是一个特定查询节点，用于结合所有其他查询节点的信息到一个单独的概率值或者置信分数上。这个得分用来对文档进行排序。从概念上讲，这意味着必须对数据集中的每个文档评价一个推理网络，但是由于其他排序算法，索引被用来加速计算。一般来说，表示型节点都被索引了，其中查询节点是针对用户或者搜索应用对每个查询设定的。这意味着需要对大规模的邻近特征进行索引。除了词语外，这样做会显著增加创建索引的规模（正如第5章所述）。在一些应用中，和邻近特征相关的概率都是在查询时计算的，目的是为了提供更加灵

⊖ 置信网络是一类用于模型不确定性的技术的名称。一个贝叶斯网络就是一个概率置信网络。

活的具体查询。

在推理网络图中的连接，定义为查询和数据集中表示每个文档的连接到的表示型节点。表示型节点的概率通过每个文档的语言模型进行估计。注意，这些节点不表示一篇文档中特定特征的出现情况，而是捕捉文档特性概率。因为语言模型可以生成这个概率。例如，词语“林肯”对应的节点表示一个文档和主题相关与否的二元事件。文档的语言模型被用来计算事件是真（TRUE）的概率。

推理网络中的所有事件都是二元的，我们不能真实地使用一个多项式模型来将文档表示为一个词语序列。相反，使用多元伯努利模型[⊖]，这个模型是7.2.1节中二元独立模型的基础。在这种情况下，一篇文档表示为一个二元特征向量，简单地记录一个特征出现与否。为了捕捉词项的频率信息，一种不同的多元伯努利模型被用来将文档表示为多重集合[⊖]向量，其中每个向量表示一个词项的出现情况（Metzler, Lavrenko & Croft, 2004）。事实上，随着恰当的选择参数，基于多元伯努利分布的概率估计和基于狄利特雷平滑的多项式分布估计是一样的，计算如下：

$$P(r_i \mid D,\ \mu) = \frac{f_{r_i,D} + \mu P(r_i \mid C)}{|D| + \mu}$$

其中$f_{i,D}$是文档D中特征r_i出现的次数，$P(r_i|C)$是特征r_i的数据集概率，μ是狄利特雷平滑参数。为了更加精确，对图7-4中的模型，会使用$f_{i,D}$计数，集合概率以及μ的一个具体数值用来针对特定的感兴趣的文档结构。例如，如果$f_{i,D}$是特征r_i在文档标题中出现的次数，那么集合概率可以从集合中所有的标题文本得到估计，μ参数会根据标题设定具体数值。同样的估计公式也被用于词语的邻近特征。例如，对于“纽约”这个要求词语必须相互紧邻的特征，$f_{i,D}$就是“纽约”在文本中出现的次数。

确定如何融合证据的查询节点是查询语言中操作的基础。虽然贝叶斯网络允许任意形式的组合（受到概率法则的约束），但是推理网络检索模型是基于那些能够被高效计算的操作的。对于网络中的每个节点，都需要指明对于父节点的所有可能状态的每个输出的概率。如果父节点的个数很多，这显然会变得非常繁琐。幸运的是，许多有趣的组合能够表示成简单的公式。

作为结合过程的例子以及说明如何有效地完成，考虑布尔操作AND。图7-5中给定一个简单的样例网络，包含一个查询节点q和两个父节点a和b，可以描述条件概率如表7-6所示。

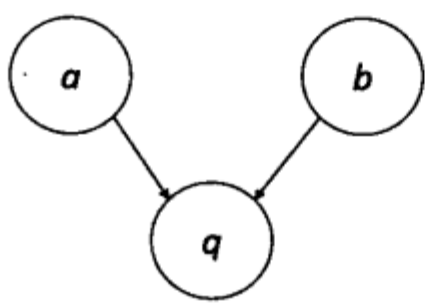


图7-5 三个节点的推理网络

表7-6 示例网络的条件概率

$P(q$=TRUE$\mid a, b)$	a	b
0	错误	错误
0	错误	正确
0	正确	错误
1	正确	正确

⊖ 根据瑞士数学家雅加达·伯努利（Jakob Bernoulli）命名（也叫詹姆斯，或者杰奎琳，是同一个家庭中八个著名的数学家之一）。多元伯努利模型会在第9章中进一步介绍。

⊖ 多重集合（也叫做袋（bag））是一种集合每个成员都有一个关联数据记录成员出现的次数。

这里用p_{ij}来表示表7.6中第一列的数值，其中i和j对应父节点。例如，p_{10}表示给定a为真、b为假时，q是真的概率。为了计算q的概率，使用这个表及其父节点的概率（来自表示型节点）如下：

$$\begin{aligned} bel_{and}(q) &= p_{00}P(a=\text{FALSE})P(b=\text{FALSE}) \\ &\quad + p_{01}P(a=\text{FALSE})P(b=\text{TRUE}) \\ &\quad + p_{10}P(a=\text{TRUE})P(b=\text{FALSE}) \\ &\quad + p_{11}P(a=\text{TRUE})P(b=\text{TRUE}) \\ &= 0\cdot(1-p_a)(1-p_b)+0\cdot(1-p_a)p_b+0\cdot p_a(1-p_b)+1\cdot p_{aPb} \\ &= p_a p_b \end{aligned}$$

其中p_a是a为真的概率，p_b是b为真的概率。使用$bel_{and}(q)$表示这是一个来自于AND组合的置信值（概率）。

这意味着AND的证据组合的概率，可以通过简单地将父节点的概率相乘起来得到。如果其中一个父节点的概率很低（或者没有使用平滑技术时的0），那么组合也会得到一个很低的概率值。看起来对这种组合是合理的。可以采用同样的方式来定义许多其他结合操作。如果一个q节点有n个概率为p_i的父节点，那么下面的一组公式定义了一些普通的操作符：

$$bel_{not}(q) = 1 - p_1$$

$$bel_{or}(q) = 1 - \prod_i^n (1 - p_i)$$

$$bel_{and}(q) = \prod_i^n p_i$$

$$bel_{wand}(q) = \prod_i^n p_i$$

$$bel_{\max}(q) = \max\{p_1, p_2, \cdots, p_n\}$$

$$bel_{sum}(q) = \frac{\sum_i^n p_i}{n}$$

$$bel_{wsum}(q) = \frac{\sum_i^n wt_i p_i}{\sum_i^n wt_i}$$

其中wt_i是第i个父节点的关联权值，显示其证据的相对重要程度。注意，NOT是一元操作（即只有一个父节点）。

加权AND操作是非常重要的，也是下一节中即将介绍的查询语言中最普通的操作之一。使用这种形式的结合并限制证据（表示型节点）到单独的词语上，可以得到像查询似然那样的排序。

基于这里描述的基础模型和结合操作，可以定义一种查询语言，用来在搜索引擎中得到基于复杂证据组合的排序结果。

7.4.2 Galago查询语言

这里展示的Galago查询语言类似于基于查询网络检索模型的开源搜索引擎中的查询语言[⊖]。

⊖ 例如Inquery和Indri。

这个版本的查询语言关注于这些语言对大量搜索应用最有用的方面，并且增加了使用任意特征的能力。注意，Galago搜索引擎并不是基于任何特定检索模型的，而是为实现检索模型提供了一个有效的框架。

虽然查询语言可以容易地处理简单的结构化文本文档，但是更有趣的特征使用了基于文档结构的证据。假设结构是具体使用的标签对，例如在HTML或者XML中的。考虑下面的文档：

```
<html>
<head>
<title>院系描述</title>
</head>
<body>
下面列表描述…
<h1>农业</h1> …
<h1>化学</h1>…
<h1>计算机科学</h1> …
<h1>电子工程</h1> …
</body>
</html>
```

在Galago查询语言中，一个文档被看成是可以包含任何标签的文本序列。在上面的例子中，这个文档由HTML标签文本构成。

对于文档中的每个标签类型T（例如title、body、h1等），定义T的上下文[⊖]为T类型的所有标签中出现的所有文本和标签。在例子中，所有出现在<body>和</body>之间的文本和标签都定义为body的上下文。单独的上下文对每个单独的标签名称生成。因此，一个上下文定义了一个子文档。注意，由于标签嵌套，特定词语会在很多上下文中出现。这里也会出现嵌套上下文。例如，在<body>上下文中嵌套了<h1>上下文，<h1>上下文由所有出现在body上下文同时在<h1>与</h1>之间的所有文本和标签构成。下面是这个样例文档中的title标签、h1标签以及body上下文：

```
title上下文:
<title>院系描述</title>

h1上下文:
<h1>农业</h1> …
<h1>化学</h1>…
<h1>计算机科学</h1> …
<h1>电子工程</h1> …

body上下文:
<body>
下面列表描述…
<h1>农业</h1> …
<h1>化学</h1>…
<h1>计算机科学</h1> …
<h1>电子工程</h1> …
</body>
```

⊖ 上下文有时也称为域（field）。

每个上下文都由一个或者多个扩展构成。一个扩展是出现在同样类型标签上下文中的一个单独开始/结束标签对之间的文本序列。对于这个例子，在<h1>上下文中，有下面这些扩展，<h1>农业</h1>、<h1>化学</h1>等。title和body上下文只有一个扩展，因为分别只有一个<title>标签和<body>标签。给定标签类型的扩展数目，由文档中出现的那种标签类型标签对的个数决定。

除了文档创建时定义的结构外，还有由特征抽取工具获得的用于表示结构的上下文。例如，日期、人名和地址都能在文本中被特征抽取工具识别和标注。只要这种信息通过标签对表示，就能够被查询语言像其他文档结构那样被表示。

词项是查询语言中的基本构建模块，对应于推理网络模型中的表示型节点。许多类型的词项都能被定义，例如简单的词项、有序或者无序的短语、同义词和其他。另外，许多选项可以被用来确定一个词项必须出现在一个特定上下文中，或者必须使用一个利用给定上下文估计得到的语言模型来打分。

简单词项：
词项 - 会被归一化和词形还原的词项。
“词项” - 不会被归一化和词形还原的词项。
举例：
总统
"NASA"

邻近词项：
#od:N(...) - 顺序窗口 - 词项必须顺序出现，最多有*N*−1个。
#od(...) - 没有限制的顺序窗口 - 所有词项必须顺序出现在当前上下文中的任何位置。
#uw:N(...) - 无序窗口 - 所有词项必须出现在长度为*N*的窗口中，可以是任何顺序。
#uw(,..) - 没有限制的无序窗口 - 所有词项必须出现在当前上下文中，可以是任何顺序。
举例：
#od:1(白 宫) - 精确短语匹配“白 宫”。
#od:2(白 宫) - 匹配“白 * 宫”（其中*是任何词语或者为空）。
#uw:2(白 宫) - 匹配“白 宫”和“宫 白”。

同义词：
#syn(...)
#wsyn(...)
这两个表达式是等价的。它们都是将列举的所有词项看成是同义词。#wsyn操作符将词项视为同义词，并且允许对每个词项加权。这些操作的对象只能是简单词项或者邻近词项。
举例：
#syn(狗 犬) - 基于两个词项的简单同义词。
#syn(#od:1(联合 众国) #od:1(美利坚 联合 众国) - 构造了两个邻近词项的同义词。
#wsyn(1.0唐纳德0.8 唐 0.5犬) - 加权同义词，表示词项的相对重要性。

无名词项：
#any:.() - 用来匹配扩展类型。
举例：
#any:person() - 匹配任何人的扩展。
#od:1 (林肯 死 于 #any:date()) - 精确匹配短语的形式：“林肯 死 于 <date>...</date>”。

上下文限制和评测：

expression.C1,...,CN - 匹配出现在上下文C1到CN之间的任意表达式。

expression.(C1,...,CN) - 采用文档中C1到CN的连续上下文生成的语言模型来评价表达式。

举例：

dog.tittle - 在title扩展中匹配词项“dog”。

#uw(smith jones).author - 在author扩展中匹配两个人名“smith”和“jones”。

dog.(title) - 在文档的标题语言模型中评价词项。这意味着估计dog在给定文档中的出现概率，这个概率通过文档中的title域中这个词项出现的次数除以文档的title域中所有词项的个数得到。类似地，平滑采用title域中的出现情况相对于整个数据集的概率来实现。

#od:1(亚伯拉罕 林肯).person.(header) - 对文档中所有的“header”文本构建语言模型，并评价那个上下文中的#od:l(亚伯拉罕 林肯)(即在header上下中的person扩展中精确匹配这个短语)。

置信操作被用来融合词项、短语等特征。置信操作有不加权和加权两种。在加权操作中，列出了证据的相对重要性。这可以用来控制查询中的每个表达式对最终得分的影响。过滤(filter)操作用来排除那些不包含特定证据的文档。所有的置信操作都可以嵌套。

置信操作：

#combine(...) - 这个操作是推理网络模型中$bel_{and}(q)$操作的归一化版本。详见后续的讨论。

#weight(...) - 这是$bel_{wand}(q)$操作符的归一化版本。

#filter(...) - 这个操作符类似于#combine，区别是文档必须包含所有词项的至少一个（简单、邻近、同义词等）。嵌套置信操作的评价不变。

举例：

#combine(#syn(狗 犬) 训练) - 对两个词项排序，其中一个是一组同义词。

#combine(biography #syn(#od:l总统 林肯) #od:l(亚伯拉罕 林肯)) - 对两个词项排序，其中一个是一组同义词“总统 林肯”和“亚伯拉罕 林肯”。

#weight(1.0 #od:l(国内 战争) 3.0 林肯 2.0 演讲) - 对三个词项进行排序，并且加权使得词项“林肯”最重要，然后是“演讲”，最后是“国内 战争”。

#filter(水族馆 #combine(热带 鱼)) - 只考虑包含“水族馆”和“热带”或“鱼”的文档，并且根据查询#combine (水族馆 #combine(热带 鱼))来对这些文档排序。

#filter(#od:1(约翰 史密斯).author) #weight(2.0 欧洲 1.0 旅行) - 对“约翰 史密斯”出现在author上下文中的关于“欧洲”或者“旅行”的文档进行排序。

正如刚才描述的那样，#combine和#weight操作分别是bel_{and}和bel_{wand}操作符归一化的版本。这两个操作符的置信度计算如下：

$$bel_{combine} = \prod_{i}^{n} p_i^{1/n}$$

$$bel_{weight}' = \prod_{i}^{n} p_i^{wt_i / \sum_{i'}^{n} wt_{i'}}$$

完成这种归一化是为了使操作符更像原始的都被归一化了的bel_{sum}和bel_{wsum}操作符。归一化的一个优点是，可以根据各种类型的均值（平均）来描述这些操作的置信度计算。例如，bel_{sum}计算所有父节点置信度的算术平均值，相应的bel_{wsum}计算加权算数平均值。类似地，$bel_{combine}$和bel_{wand}分别计算几何平均值和加权几何平均值。

filter操作符还可以用于数值（numeric）和日期（date）域操作符，从而使非文本的证据可以融合到得分数。例如，下面的查询

#filter(新闻.doctype #dateafter(12/31/1999).docdate
#uw.20(布朗.person #any:company() #syn(钱 现金 支付))

对1999年后出现的新闻文档进行排序。这些文档同时至少包含一个提到人名“布朗”的长度为20的文本片段、一个公司名称、至少三个讨论金钱词语中的一个。推理网络模型可以非常容易地处理这种类型的证据融合，但是为了简单起见，没有在Galago中实现这些操作。

推理网络模型中的另外一个支持Galago查询语言的部分就是文档的优先性。文档优先特征允许对数据集中的文档指定一个先验概率。这些先验概率会影响对某种特点的文档优先排序，例如最近撰写或者简短的文档。

优先性：

#prior:name() - 使用给定名称指定的文档。优先性是文件或者对每个文档提供先验概率的函数。

举例：

#combine(#prior:recent() 全球 变暖) - 使用prior指定越是最近的文档具有越高的权值。

作为使用这种查询语言更加详细的例子，在下一节中，会讨论网络搜索和融合多种类型证据后的有效排序。#feature操作可以定义任意特征（新证据），这会在第11章中讨论。

7.5 网络搜索

根据流行程度来判断，网络搜索显然是最重要的搜索应用。数百万人每天使用网络搜索来完成从购物到研究的涉及范围极大的任务。考虑到其重要性，网络搜索是用于解释已经讨论的检索模型如何具体应用的明显例子。

网络搜索与对一组新闻文档提供搜索的应用有一些显著区别。主要的区别包括数据集的规模（数十亿计的文档）、文档之间的连接（即联系）、文档类型的范围、查询的规模（每天数千万个）、查询的类型等。这些话题中有些已经在前面的章节中讨论过一些，其他的会在后面讨论，例如垃圾的影响。在这一节中会关注查询的特征以及对排序算法最重要的那些文档。

在网络环境中有些不同类型的搜索。Broder (2002)提出了一种流行的描述搜索的方式。在这个分类体系中，搜索被分为三类：信息式、导航式、交互式。信息式搜索的目的是查找关于一些可能包含一个或多个网页的主题相关的信息。由于每次搜索都是寻找一些类型的信息，本书称之为主题搜索。导航式搜索的目的是，找到用户曾经见过或者假设存在的特定网页[⊖]。交互式搜索是为了查找可以执行购物或者下载音乐任务的网站。每种类型的搜索都有一种关联的信息需求，但却是不同类型的信息需求。基于主题相关性的检索模型主要关注于第一类信息需求（和搜索）。为了获得其他类型搜索的有效排序，需要有一种能够融合用户相关性证据的模型。

商业网络搜索引擎在它们的排序算法中融合了成百上千的特征。许多特征都来自查询日志中海量的用户交互数据。这些特征可以被广泛地分成关于网页内容的特征、网页元数据、锚文本、链接（例如PageRank）、用户行为等。虽然锚文本来源于一个网页中的链接，但是它和其他来自于网页链接结构分析的特征的使用方法不一样，所以被分成单独的一类。页面元数据是关于页面的信息。这些信息不是页面内容的组成部分，例如页面的“年龄”、更新的频率、页面的URL、所在网站的域名、网页相关材料中文本内容的数量等，例如图片和广告。

有趣的是，理解这些特征的相对重要性以及如何利用它们来对一个网页获得更好的搜索排序，是搜索引擎优化（SEO）的基础。例如，搜索引擎优化人员会改进用于网页title标签的

⊖ 在TREC评测中，导航式搜索称为主页搜索和命名网页搜索。主题搜索称为特定搜索。导航式搜索类似于已知项搜索，这种搜索已经在信息检索领域讨论很多年了。

文本，改进heading标签中的文本，确保域名和URL包含重要的关键词，尽量改进网页相关的锚文本和链接结构。这些技术中有些并不被网络搜索引擎公司视为是恰当的，这会在9.1.5节中进一步讨论。

在TREC环境中，检索模型已经在测试网页集合和混合的查询类型上比较过。关于用户行为的特征以及一些诸如更新频率的网页元数据特征，在TREC数据集上都是无法使用的。其他特征中，对导航式搜索最重要的是文档的title、body、heading(h1、h2、h3、h4)中的文本，所有指向文档的链接上的锚文本，PageRank数值和入链（inlink）数目（指向这个网页的链接数目）等。注意，不是说其他特征不会影响网络搜索引擎中的排序，只是刚才的这些特征是TREC实验中最具显著影响的。

给定整个网络规模的大小，许多网页能够包含所有的查询词项。一些排序算法只对这些网页排序，通过布尔运算AND来实现过滤。这会导致问题，如果只有网络的一个子集被使用（例如一个站点的搜索应用），同时也会对主题搜索形成特殊的风险。例如，TREC主题网络搜索中包含全部查询项的网页中，只有50%的页面被判断为是相关的。与过滤不同的是，排序算法会非常倾向于包含所有查询词项的网页。另外，词项邻近性也非常重要。词项相互邻近出现的额外证据会显著提高排序的性能。许多结合词项邻近性的检索模型已经被开发出来了。下面的方法被用于推理网络模型中，并且得到了好的结果[⊖]。

依赖模型假设在相关文档中查询词项易于相邻的出现。例如，给定查询“绿色 政党 政治观点”的相关文档中，容易包含短语“绿色 政党”和“政治 观点”而且相对位置很近。如果将查询看成是一组词项Q，可以定义S_Q为Q的所有非空子集构成的集合。Galago查询试图通过下面的方式捕捉词项之间的依赖关系：

1）每个由邻近查询词项组成的$s \in S_Q$，容易作为完整短语出现在相关文档中（例如，使用#od:1操作符来表示）。

2）每个$|s|>1$的$s \in S_Q$，容易（有序或者无序）出现在相关文档的一个大小合理的窗口文本中（例如，在窗口中表示$|s| = 2$为#uw:8，表示$|s| = 3$为#uw:12）。

作为例子，这个模型对查询“embryonic stem cells”生成如图7-6所示的Galago查询语言表示，其中权值根据产生最佳结果的经验设定的。

```
#weight(
        0.8 #combine(embryonic stem cells)
        0.1 #combine( #od:1(stem cells) #od:1(embryonic stem)
                      #od:1(embryonic stem cells))
        0.1 #combine( #uw:8(stem cells) #uw:8(embryonic cells)
                      #uw:8(embryonic stem) #uw:12(embryonic stem cells)))
```

图7-6 Galago查询的依赖模型

给定网络搜索排序中重要的证据片段，现在能够给出Galago查询整合证据到有效排序中的例子。对TREC查询“pet therapy”，可以生成如图7-7所示的Galago查询。关于这个查询，首先需要注意的是，它清楚地展示了一个复杂查询可以从简单查询中生成。许多邻近词项都被添加到里面，并且所有词项都被基于锚文本、title文本、body文本的上下文所评价。从效果来看，即使这会导致索引规模增加很大，邻近词项也可以被索引。这些相对较大的查询表达

⊖ Metzler and Croft （2005b）给出了正式的模型描述。

式带来的好处，是评价可以在查询时非常高效地进行。

```
#weight(
        0.1 #weight( 0.6 #prior(pagerank) 0.4 #prior(inlinks))
        1.0 #weight(
            0.9 #combine(
                #weight( 1.0 pet.(anchor) 1.0 pet.(title)
                         3.0 pet.(body) 1.0 pet.(heading))
                #weight( 1.0 therapy.(anchor) 1.0 therapy.(title)
                         3.0 therapy.(body) 1.0 therapy.(heading)))
            0.1 #weight(
                1.0 #od:1(pet therapy).(anchor) 1.0 #od:1(pet therapy).(title)
                3.0 #od:1(pet therapy).(body) 1.0 #od:1(pet therapy).(heading))
            0.1 #weight(
                1.0 #uw:8(pet therapy).(anchor) 1.0 #uw:8(pet therapy).(title)
                3.0 #uw:8(pet therapy).(body) 1.0 #uw:8(pet therapy).(heading)))
        )
```

图7-7 对于网络数据的Galago查询

PageRank和入链证据被作为先验概率融合到这个查询中。换句话说，这个证据和特定查询是相互独立的，能够在索引时计算。查询中的权值可以通过TREC网页数据集上的实验确定，该数据集基于.gov域的数据抓取。对于完整的网络或者其他数据集，证据的相对重要性可以不同。事实证明，网页主体部分的文本比文档中的其他部分以及锚文本都更加重要，这一点反映在它们的权值上。

在TREC数据集上的实验也显示，许多对高效导航式搜索具有决定性的证据，对主题搜索并不重要。事实上，主题搜索需要的特征只有简单的词项和文档中body部分的邻近性词项，其他特征不会改进性能，但是也不会降低性能。主题搜索和导航式搜索的另外一个区别是，使用伪相关反馈对主题搜索有用，但是会使得导航式搜索变差。导航式搜索是为了查找特定网页，所以通过增加一些额外的词项的平滑查询，会增加结果的“噪声”。这一点并不奇怪。如果一个搜索被明确是主题类的，那么查询扩展会有用。但是这很难准确地确定。因为通过扩展的潜在效用是变化的，而且一定程度上是不可预测的。这种技术一般来说是没有用的。对于交互式搜索，需要识别好的网站证据。这一点导航式搜索看起来也需要。这意味着同样的排序算法能够被用于不同类型的网络搜索。

其他研究表明，用户行为信息能够显著影响排序的性能，例如点击流数据（即过去哪个文档被点击过，排序中哪个位置被点击过）和浏览数据（网页的驻留时间，跟随的链接）。这种类型的证据可以在推理网络框架中使用额外操作符加入，但是随着证据片段数目的增加，如何确定最有效的结合方式以及证据的权值就变得更加重要。在下一节中会讨论使用用户的显式和隐式反馈数据来学习加权和排序算法的技术。

7.6 机器学习和信息检索

信息检索领域和机器学习领域有很多重要的重叠。在上个世纪60年代，相关反馈被提出作为基于用户关于初始排序中文档重要性的反馈技术。这是一个简单的机器学习算法的例子，需要构建一个分类器来区分相关文档和非相关文档。在80年代和90年代，信息检索研究人员使用机器学习方法来基于用户反馈学习排序算法。在最近10年，有许多人研究使用机器学习

方法进行文本分类。但是，许多机器学习对信息检索的应用受到可用训练数据规模的限制。如果系统需要尝试构建一个对每个查询的单独的分类器，就会有非常少的关于相关文档的数据。然而，其他机器学习应用可能有成百上千甚至更多的训练样例。尝试通过所有查询上的训练数据来学习排序算法的方法，也会受到较小规模的查询数据和典型信息检索测试数据集上的相关性判断的限制。

随着网络搜索引擎的出现，从用户交互中积累了海量的查询日志，潜在的训练数据的规模非常庞大。这使新技术的开发可以在信息检索和搜索引擎设计领域中产生显著的影响。在下一节中会描述能够融合对网络搜索重要的许多证据片段并加权的学习排序算法技术。

机器学习中另一个非常活跃的领域就是开发出文本的复杂的统计模型。在7.6.2节会介绍这些模型是如何被用于改进基于语言模型的排序的。

7.6.1 排序学习

到目前为止，所有提到的概率检索模型都可以归为生成式模型。对于文本分类，生成式模型假设文档都是通过一些潜在的模型（在文本分类中，常常是多项式分布）生成的，而且使用训练数据来估计模型的参数。然后属于某个类别（例如查询的相关文档）的概率使用贝叶斯法则和文档模型来进行估计。相对而言，判别式模型估计属于某个类别的概率，是直接通过基于训练数据的观察到的文档特征来计算的[⊖]。一般的分类问题中，生成式模型会在训练样本较少时表现更好，但是判别式模型常常在数据充足时表现更好。给定网络搜索引擎中可用的潜在训练数据的规模，判别式模型被期望在这个应用中具有优势。正如我们提到的那样，判别式模型更容易融合新的特征，对于网络搜索可以考虑成百上千的特征。

早期应用在信息检索中学习判别式模型（判别式学习），使用的是logistic回归来预测一个文档是否属于相关类别。存在的问题是训练数据的规模，以及随之而来的依赖于人工显式判断相关性技术的有效性。即使得到一个商业网络搜索公司的资源，显式的相关性判断仍然需要花费很多才能获得。另一方面，查询日志包含了大量的隐式相关信息，其形式是点击流和其他用户交互。对应于此，基于这种形式数据的判别式学习技术被开发了出来。

在搜索中，用于学习排序函数最著名的方法就是基于支持向量机（SVM）分类器的。这种技术会在第9章中更加详细地讨论，这里只给出一个简单的描述，说明Ranking SVM如何被用来学习排序[⊜]。

Ranking SVM的输入是针对一组查询的偏序排序信息的一组训练集合

$$(q_1, r_1), (q_2, r_2), \cdots, (q_n, r_n)$$

其中q_i是一个查询，r_i是所需排序的文档关于查询的部分排序信息或者相关性级别。这意味着，如果文档d_a应该比d_b排序更高，那么$(d_a, d_b) \in r_i$，否则$(d_a, d_b) \notin r_i$。这些排序是从哪里来的呢？如果相关性判断可以获得，所需排序就会将所有判断为高相关性级别的文档高于那些低相关性的文档。注意，这里使用了相关性的多个级别。这种多级别经常被用于网络搜索引擎的评价中。

但是，如果不能使用相关性判断，那么排序可以基于点击流和其他用户数据。例如，如果用户在一个查询的排序中点击了第三个文档而不是前面两个，那么可以假设第三个文档应该在r中被排序较高。如果d_1、d_2、d_3分别是搜索输出排序中的第一个文档、第二个文档和第

⊖ 我们会在第9章再次讨论生成式分类器和判别式分类器。

⊜ 这部分描述基于Joachim的关于点击流数据用于排序学习的论文(Joachims, 2002b)。

三个文档，那么对这个查询在所需查询的点击流数据可以成对出现为(d_3, d_1) 和(d_3, d_2)。这种排序数据是充满噪声的（因为点击不是相关性判断）和不完整的，但是这种数据会有很多，并且实验显示这种类型的训练数据可以被有效使用。

假设正在学习一个线性排序函数 $\vec{w}.\vec{d}_a$，其中 $\vec{w}$是一个用于通过学习调整的权值向量，$\vec{d}_a$是文档d_a的特征表示向量。正如上一节介绍的那样，这些特征都是基于网页内容、网页元数据、锚文本、链接和用户行为的。但是，不同于语言模型概率，这个模型中的特征都是依赖于查询和文档内容较简单而且不够正式的匹配结果的。例如，其中会有查询和文档body共有词语数目的特征，和title、header、锚文本的类似特征。 $\vec{w}$向量中的权值决定这些特征的相对重要性，类似于推理网络操作中的权值。如果一个文档表示为三个整数值的特征 $\vec{d}=(2, 4, 1)$，权值向量是 $\vec{w}=(2, 1, 2)$，那么计算排序函数得到的得分就是：

$$\vec{w}.\vec{d}=(2,\ 1,\ 2).(2,\ 4,\ 1)=2.2+1.4+2.1=10$$

给定查询的训练数据和排序信息，希望找到一个权值向量 $\vec{w}$来尽可能多地满足下面的条件：

$$\forall(d_i,\ d_j)\in r_1:\vec{w}.\vec{d}_i>\vec{w}.\vec{d}_j$$

$$\forall(d_i,\ d_j)\in r_n:\vec{w}.\vec{d}_i>\vec{w}.\vec{d}_j$$

简单来说，这意味着对排序数据中所有的文档对，希望对相关性得分较高的文档在排名（或排序）上超过相关性得分较低的文档。遗憾的是，没有有效的算法来 $\vec{w}$对找到精确的解决方案。但是，能够重新形式化这个问题为一个标准的SVM优化问题，如下所示：

$$\mathrm{min}i:\frac{1}{2}\vec{w}.\vec{w}+C\sum\xi_{i,j,k}$$

$$subject\ \ to:$$

$$\forall(d_i,\ d_j)\in r_1:\vec{w}.\vec{d}_i>\vec{w}.\vec{d}_j+1-\xi_{i,j,1}$$

$$\forall(d_i,\ d_j)\in r_n:\vec{w}.\vec{d}_i>\vec{w}.\vec{d}_j+1-\xi_{i,j,n}$$

$$\forall_i\forall_j\forall_k:\xi_i,\ j,\ k\geqslant 0$$

其中ξ称为松弛变量，用于允许对困难或者噪声训练样本的错误分类，C是用于阻止过拟合的参数。过拟合是指学习算法在训练数据上的排序函数表现很好，但是对新查询排序文档表现不好。能够实现这种优化和产生分类器的软件包，可以从网上获得[⊖]。

这种优化从哪里得到呢？心急的读者可以直接跳到第9章中解释一般SVM分类器的部分。目前，可以说SVM算法会找到一个具有下述性质的分类器（即向量 $\vec{w}$）。在训练数据中的每对文档可以表示为向量$(\vec{d}_i-\vec{d}_j)$。如果对这对文本计算得分为 $\vec{w}.(\vec{d}_i-\vec{d}_j)$，SVM分类器会找到一个 $\vec{w}$使得最小的得分尽可能大。对于负的训练实例（不在排序数据中的一对文档），同样的事情也是真实的，这意味着分类器会使得对最难排序的文档对的得分差别尽可能大。

注意，这个模型并没有指出需要使用的特征。SVM甚至也用于从不同的检索模型得分学习对应的特征权值，例如BM25模型和语言模型。融合多个搜索引擎，对一个给定查询的结果，被证明在许多实验中都是有效的。这部分内容会在10.5.1节进一步讨论。需要注意的还有

⊖ 例如*SVM^{light}*，见http://svmlight.joachims.org。

Ranking SVM（或者其他一些判别式技术）学习得到的权值也可以被直接用到推理网络查询语言中。

虽然诸如Ranking SVM的线性判别式分类器也许对网络搜索具有优势，但是还有一些只有较少训练数据和较少可用特征的其他搜索应用。对于这些应用，主题相关性的生成式模型也许更加有效，特别是那些能够持续从更好的估计技术中提高的模型。下一节会讨论将文档建模成一个混合主题模型后，如何改进估计。

7.6.2 主题模型和词汇不匹配

在一般的信息检索中，一个重要的话题就是词汇不匹配。这是指那种相关文档不能匹配查询的情况。因为它们使用了不同的词语来描述同一主题。在网络环境中，许多文档包含全部的查询词项，所以这可能不会成为一个话题。但是，在小数据集上的搜索应用中，甚至是网络搜索，这一点都是重要的。TREC上的实验已经证明，使用查询扩展技术时主题查询会产生更好的结果。查询扩展（例如使用伪相关反馈）是用于降低词汇不匹配的标准技术，尽管词形还原在一定程度上也能克服这个问题。另外一种方法是可以通过增加相关词项扩展文档。对于采用语言模型表示的文档，这等价于在语言模型上平滑概率，从而文本中没有出现的词语具有非零概率。注意，这不同于使用对所有文档都一样的数据集概率来进行平滑。相反，需要通过一些方法来增加文档中和主题相关的词语的概率。

为了达到这个目的，涌现了许多技术。如果一个文档属于某个类别或者文档集簇，那么文档集簇中的词语概率可以被用来平滑文档语言模型。在第9章中会详细介绍这种方法。一种被称为隐含语义索引或者LSI[⊖]的技术，将文档和词项映射到一个精简的维数空间中，从而先前使用成千上万词汇索引的文档，现在可以使用仅仅几百个特征就能表示。在这个新空间中，每个特征都是许多词语的混合或者集簇，这种混合式实际上是文档表示的平滑。

隐含狄利特雷分配（LDA）模型来源于机器学习领域。它将文档表示成许多主题的混合。一个主题就是前面定义过的一个语言模型。在诸如查询似然的检索模型中，每个文档都被假设成一个单独主题。实际上，主题和数据集中的文档是一样多的。相比之下，在LDA方法中，假设有一个固定数目的潜在（隐含）主题能够被用来描述多个文档的内容。每个文档都表示为这些主题的混合，都能达到类似于LSI的平滑效果。在LDA模型中，一个文档首先选择主题上的一个分布来生成得到，然后文档中的词语是选择一个主题并从这个主题生成的。

使用在语言模型中的“桶”模拟，需要多个桶来描述这个过程。对每个文档，需要一个装主题的桶。主题的数目是每个主题依赖的已经选出的主题分布数量。对每个主题，都有另外一个桶，用来装词语。词语的数目是主题语言模型中依赖于概率的词语数量。然后，为了生成一个文档，首先从主题桶中选出一个主题（仍然不能看到），然后到这个选中主题对应的词语桶中取出一个词语。这个过程重复进行，取出第二个词语。

概括地说，生成一篇文档的LDA过程如下：

1）对每个文档D，根据参数α在一个狄利特雷分布中选择一个多项式分布θ_D。

2）对文档D中的每个词语位置：

a）从多项式分布θ_D中选择一个主题z。

b）从$P(w|z,\beta)$中选择一个词语w，$P(w|z,\beta)$是给定主题z和参数β时的多项式条件概率。

⊖ 这种技术也叫做隐含语义分析或者LSA(Deerwester er al., 1990)。注意，“隐含”被用来作为“隐藏”的意思。

有许多技术可以用于将文档数据集作为训练数据来学习主体模型和θ分布，但是所有这些方法一般都非常慢。一旦拥有这些分布，就能生成文档中词语的语言模型概率：

$$P_{lda} = (w \mid D) = P(w \mid \theta_D, \beta) = \sum_z P(w \mid z, \beta) P(z \mid \theta_D)$$

这些概率值随后可以用于平滑和查询似然概率混合表示的文档，如下所示：

$$P(w \mid D) = \lambda \left(\frac{f_{w,D} + \mu \frac{c_w}{|C|}}{|D| + \mu} \right) + (1 - \lambda) P_{lda}(w \mid D)$$

因此，实际上最终的语言模型概率是极大似然概率、数据集概率以及LDA概率的混合。

如果LDA概率被直接用于文档表示，排序的性能会显著削减。因为这些特征都太平滑了。在TREC实验中，K（主题的数目）的数值在400左右。这意味着数据集中的所有文档都会被表示为400个主题的混合。考虑到在数据集词表中可能有数百万的词语，仅仅匹配主题会导致匹配单独词语时一些精确率的丢失。但是，当用于平滑文档语言模型时，LDA概率可以显著地提高查询似然排序的性能。表7-7展示了从TREC新闻故事样本中生成的LDA主题（总共100个）中高概率的词语[⊖]。注意，主题的名称不会自动生成。

表7-7 LDA模型从四个主题生成的高概率词项

Arts	*Budgets*	*Children*	*Education*
new	million	children	school
film	tax	women	students
show	program	people	schools
music	budget	child	education
movie	billion	years	teachers
play	federal	families	high
musical	year	work	public
best	spending	parents	teacher
actor	new	says	bennett
first	state	family	manigat
york	plan	welfare	namphy
opera	money	men	state
theater	programs	percent	president
actress	government	care	elementary
love	congress	life	haiti

搜索应用中采用LDA的主要问题，就是估计模型中的概率比较费事。除非有快速方法，否则这种技术只能限制在小规模数据集上（数十万篇文档，不超过百万）。

7.7 基于应用的模型

本章中，介绍了各种各样的检索模型和排序算法。从一些关注设计和实现搜索应用的人士的角度来看，问题在于哪些技术应该在何时使用？答案是依赖于应用和可用的工具。绝大多数的搜索应用包含相对较少的互联网数据集，一些根据链接和锚文本的较少链接。网络搜

⊖ 这个表来源于Blei等人（2003）。

索引擎中表现很好的排序算法往往不能在其他应用中得到最好的排序。根据应用需求定制的排序算法，几乎总是能得到最好结果。

这样做的第一步是，构建一组测试数据集、文档和相关性判断。这可以将不同版本的排序算法在结果上进行比较。在第8章中详细讨论评价方法，对于高效搜索引擎，这是一个重点问题。

第二步是识别哪些证据或者特征能用于表示文档。简单的词项和邻近词项几乎都是有用的。明显的文档结构——例如标题、作者和日期域——也都几乎总是对搜索很重要。在一些应用中，数值域可能也是重要的。文本处理技术例如所考虑到的词形还原和停用词，也都是必须考虑的。

可以用于查询扩展的另外一个重要的信息来源是具体应用的知识宝库。在以往的信息系统中，这些都是令人非常普通的。因为都会尝试通过手动或者自动的方式来建造它们。虽然同义词和它们包含的相关词经常都是非常不全的，但是都能在排序性能上得到显著不同。

识别出各种文档特征和其他证据后，下一步就是确定如何融合它们来计算文档的得分。诸如Galago的开源搜索引擎使得这相对容易实现。因为融合和对证据加权都能够表示为查询语言。许多变种都能进行快速的测试。其他搜索引擎则没有这种程度的灵活性。

基于简单检索模型的搜索引擎正被用于搜索应用中。对如何在BM25模型或者查询似然模型中计算出得分以及如何融合到推理网络模型的描述，能够被用来指导通过恰当的查询交互和额外的得分代码来获得类似的结果。例如，在知识库中，同义词和相关词信息不应该简单地用来在查询中增加词语。除非使用#syn操作的一些变种，否则排序的性能会降低。Galago中的#syn的实现可以用来作为如何在一个搜索引擎中增加这种操作的例子。

开发搜索引擎应用时许多时间都会用来调节排序算法的排序性能上。如果没有一些潜在的检索模型的概念，那么做这种事情是吃力不讨好的。本章介绍的检索模型（BM25、查询似然、相关性模型、推理网络和Ranking SVM）可能对于排序算法的成功提供了蓝图。对于这些模型，好的参数值和权值都已经能从广泛发表的论文中获得。这些数值可以用来作为确定是否需要针对具体应用进行修改的起点。如果有足够可用的训练数据，那么诸如Ranking SVM的判别式技术可以用来直接学习最佳的权值。

参考文献和深入阅读

在信息检索中，由于检索模型是最重要的话题之一，从上世纪50年代起就有很多论文都描述这方面的研究。van Rijsbergen的书（van Rijsbergen, 1979）最有价值的贡献之一，就是概述了这个领域早期的研究。在本书中会集中介绍一些主要的论文，而不是尝试全面的阐述。这些参考文献会按照本章中展示的话题顺序来进行讨论。

可以理解的是，关于相关性本质的讨论已经在信息检索领域进行了很长时间。早先的论文中，应用最多的就是Saracevic（1975）。有一篇更现代的文章对这个领域的工作进行了概述（Mizzaro, 1997）。

关于布尔和排序的对比研究的话题，Turtle（1994）完成了一个性能对比试验，对比了专业搜索人员使用他们能够生成的最好的布尔查询和使用排序的关键词搜索，结果发现布尔搜索没有任何优势。正如Turtle和Croft（1991）介绍那样，当使用简单的布尔查询和排序进行对比时，排序的性能会高很多。

向量空间模型最早在Salton等（1975）中提出，随后在Salton和McGill（1983）中详细介绍。基于这个模型最全面介绍加权实验的论文是Salton和Buckley（1988）。在7.1.2节中描述的词项加权技术在这篇论文工作的基础上有一点提高。

将信息检索看成是分类问题的描述出现在论文van Rijsbergen（1979）中。关于二元独立性模型及其在BM25排序方程中应用的最佳论文是Sparck Jones 等（2000）。

最早在信息检索中使用语言模型的工作是Ponte和Croft （1998）。这篇文章描述了一个基于多元伯努利语言模型的检索模型。这个工作随后迅速被许多论文跟进，实现了多项式版的检索模型（Hiemstra, 1998; F. Song & Croft, 1999）。Miller等（1999）使用隐马尔可夫模型描述了同样的方法。Berger和Lafferty（1999）说明了如何将词语的翻译概率融合到语言模型方法中。在10.3节会再次提到这种翻译模型。Kraaij等（2002）讨论了关于非一致先验概率的研究工作。一些关于语言模型和信息检索的论文，出现在Croft和Lafferty（2003）中。

Zhai和Lafferty（2004）给出了信息检索语言模型中关于平滑技术的最精彩的描述。采用集簇和最近邻来平滑的方法，在Liu和Croft（2004）和Kurland和Lee（2004）中有介绍。

早期关于词项独立模型的描述，出现在van Rijsbergen（1979）中。F. Song和Croft（1999）介绍了用于信息检索的二元语言模型，但是在Gao等（2004）和Metzler和Croft（2005b）中，更加一般的模型获得了明显更好的检索结果，特别是在较大规模数据集上。

关于查询扩展的相关性模型出现在Lavrenko和Croft（2001）中。Lafferty和Zhai（2001）提出了一种相关方法来构建查询模型并和文档模型进行对比。

在信息检索领域已经有很多实验证明了证据融合可以显著提高排序的性能。Croft（2000）回顾了这些结果，证明信息检索在被看成有大量特征可以使用的分类问题时，这种改进并不奇怪。Turtle和Croft （1991）描述了推理网络模型。这个模型用来作为Inquery搜索引擎（Callan et al., 1992）以及商业搜索引擎WESTLAW（Pritchard-Schoch, 1993）的WIN版本的基础。这个模型包括语言模型概率的扩展版本在Metzler和Croft（2004）中有所介绍。这个扩展被实现为Indri搜索引擎（Strohman et al., 2005; Merzler, Strohman, et al., 2004）。Galago查询语言就是基于Indri查询语言的。

7.5节中介绍的网络搜索方法在Ogilvie和Callan（2003）的基础上，是基于表示不同文档结构部分的语言模型的混合来对文档进行打分的方法。BM25F排序函数（Robertson et al., 2004）是BM25的拓展，同样被设计以高效的融合来自于不同文档域。

网络搜索中的垃圾信息，也叫做对抗性信息检索，是很重要的，以至于可以作为一个完整的子领域。它被开发出来用于解决那些被不同兴趣团体（例如垃圾制造者和搜索引擎优化人员）处理过的文档数据集上的搜索技术。在第9章中会讨论关于垃圾的话题。

关于排序学习方程，早期的工作包括logistic回归技术的使用（Cooper et al., 1992）。Fuhr和Buckley（1991）最早清楚地介绍了如何使用和实际查询词语无关的特征（例如使用查询词项中匹配个数作为特征，而不用匹配上的词项）来增强跨查询的学习排序方程。在信息检索中使用Ranking SVM是Joachims（2002b）介绍的。Cao等（2006）介绍了对这种方法的提高排序性能的改进。RankNet（C. Burges et al., 2005）是一种用于学习排序方程的神经网络方法，这种方法被用于微软网络搜索引擎中。Agichtein, Brill, and Dumais（2006）介绍了如何将用户特征有效地融合到基于RankNet的排序中。Ranking SVM和RankNet都使用了偏序排序信息（例如二元优选性）。另外一类学习模型称为列表模型。它使用整个排序列表来学习。相关的

例子包括Gao等（2005）提出的线性判别式模型。该模型基于语言模型学习特征的权值，方法类似于用于融合语言模型和其他特征的推理网络模型。另外一种列表方法是Metzler和Croft（2005b）提出的词项独立模型，也是基于特征线性组合的。Gao和Metzler的模型都提供了直接最大化精确率（一种重要的信息检索指标）的学习技术。关于列表学习模型的更多信息，可以在Xia等人（2008）中找到。

Hofmann（1999）描述了一种将文档表示成主题混合的概率版本的LSI（pLSI）。LDA模型由Blei等（2003）提出。这个模型的许多扩展随后被提出，但是都还没有被应用到信息检索中。Wei和Croft（2006）描述了LDA应用到信息检索中的情况。

练习

7.1 使用某个网络搜索引擎的“高级搜索”方法，想出三个使用布尔操作AND、OR和NOT比使用在常规搜索框中使用同样查询更好的例子。你认为搜索引擎是使用严格的布尔模型来实现高级搜索的吗？

7.2 你能想出另外一种能够用于向量空间模型的相似度度量么？使用一些样例文档和带有编造权值的查询，来对比你的度量和预先的相似度。浏览关于网络的信息检索领域，看看你的度量是否已经被人研究过了（从van Rijsbergen的书开始）。

7.3 如果每个词项在一个t-维空间中都被表示为一个维度，那么向量空间模型可以假设为词项都是相互垂直的。解释这个假设并讨论你是否认为它是合理的。

7.4 从条件概率中衍生出贝叶斯法则：

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

给出一个例子，包含一个条件概率和一个联合概率的例子，将文档中词语的出现作为事件。

7.5 基于Galago实现一个BM25模型。证明它能工作，然后用文字描述它。

7.6 展示你的BM25实现中修改参数值的效果。

7.7 什么是二元语言模型的“桶”模拟？给出一个例子。

7.8 使用Galago实现查询似然，研究长查询和短查询在性能上的影响。参数设置是否产生差别？

7.9 使用Galago实现一个关于伪相关反馈的相关性模型方法。证明通过对查询生成一些扩展词项后，你的方法可以工作，并用文字描述它。

7.10 证明bel_{wand}操作通过简单词项计算查询似然得分。bel_{wand}操作计算的是什么？

7.11 使用Galago为推理网络查询语言实现一个#not操作。列举一些例子来说明它是如何工作的。

7.12 使用Galago完成一个针对推理网络查询语言的数值操作。

7.13 编写一个界面程序，实现将一个用户的查询作为文本并转换为推理网络查询。确保你使用了邻近操作。对比简单查询和转换后查询的性能。

第8章　搜索引擎评价

“评价一下Mr. Spock。”

——Captain Kirk，《星际迷航之无限太空》

8.1　搜索引擎评价的意义

搜索引擎的评价是衡量一个搜索引擎系统是否取得进步的关键。在一个搜索引擎系统上线之前，人们非常有必要去了解它能否在一个特定的应用环境下有效地工作。然而，对于一个新版本的搜索引擎系统，无法单凭感觉来判断它是否好于其他版本。而且以往的经验证实，人们直觉上的一些猜测，如提出一些改进搜索质量的策略或是有吸引力的搜索模型，往往效果并不明显。因此，需要实施一些模拟实验来评估新版本的搜索引擎的设计，当新产品上线之后，还需要继续监测和适时调整其性能。

搜索引擎的主要评价指标是效果（effectiveness）和效率（efficiency）。简单来讲，效果衡量的是搜索引擎返回正确的搜索答案的能力，而效率衡量的是搜索引擎的搜索速度。对于一个给定的查询及相关性的定义可以更准确地定义这两个评价指标，即“效果”为搜索引擎的实际排序结果与人工排序结果的拟合度，“效率”为搜索引擎的排序算法所用的时间和空间复杂度。搜索引擎是一个大众使用的工具，是一个互动的过程，不同类型的用户拥有各自不同的查询习惯以及对查询结果的要求。在这种情况下，“效果”和“效率”也会被很多其他的因素所左右，例如，显示查询结果的用户界面、查询优化和查询扩展等技术的使用。对使用这些因素的搜索结果的“效果”和“效率”评价虽然非常重要，但由于这些因素很难进行控制，使得对它们的评价变得非常困难。鉴于此，本文的评价是在严格定义的实验环境下进行的，以确保评价结果的公平公正性。

搜索引擎的“效果”和“效率”是息息相关、相互制衡的。例如，即使我们采用了某种策略使得搜索引擎“效果”得到了一些提高，但却明显影响了系统的“效率”（如查询的吞吐率），那么这种策略也不会被搜索引擎系统所采纳。一般来说，搜索引擎更重视“效果”的提升，这是由于搜索引擎最大的用处就是能给用户反馈出相关的查询结果。因此，搜索引擎的“效果”是研究的重点。当某项技术被证实对提升“效果”确实是非常有效的时候，工作的重点才转移为如何找到一种高效的实现方式。但这并不意味着关于系统构架和系统效率并不重要。第5章中提及的如何建设一种可扩展和高效的搜索引擎技术，也被很多研究机构所研究。但众所周知，如果一个搜索引擎系统仅仅是速度快，却没有提供良好的查询结果，是没有丝毫意义的。

那么，可否在搜索引擎的两个指标“效果”和“效率”之间进行适当的折中呢？设计者们试图考察是否可以设置一些参数来调节搜索引擎系统，使得系统既能返回较高质量的查询结果，同时又不会影响系统的效率。然而，就目前的研究现状而言，还没有可靠的显著提高搜索质量的技术可以被融入搜索引擎系统中，使搜索速度得到同步的提升。我们期待着这一目标在未来的研究工作中得以实现。

除了搜索的“效果”和“效率”，另一个值得考虑的因素是搜索引擎设计的造价。我们或许知道如何高效地使用搜索引擎技术，然而要想达到这个目的，或许会在处理器、内存、硬盘和网络方面需要大量的投资。一般情况下，如果我们的目的是这三个因素中的任意两个，那么第三个因素基本上也就满足了。例如，如果我们希望搜索引擎系统具有较高水平的效果和效率，那么顺理成章地，需要高额的系统配置。另一方面，假设我们追求系统的效率且想减少系统配置的成本，那么势必影响系统的效果。对于这三个因素而言，两个极端的选择情况是，可以简单地使用字符串匹配方法（grep）进行搜索或者直接使用某些机构的检索系统如美国国会图书馆检索系统。在一个大的文本集中使用字符串匹配算法（grep）会导致较差的搜索“效果”和“效率”，但是花费很低。在人工整理的美国国会图书馆系统中进行检索，可以获得非常高质量的检索结果，同时也可以获得较快的检索速度（虽然有时候有些时间拖延，等待系统回应），然而花费很高。因此，使用一个高效的搜索引擎进行直接检索，是解决这两个极端情况的一个较好的折中。

在对搜索引擎进行评价时，使用最频繁的一个术语就是“优化”。搜索引擎中的检索和索引技术在“效果”和“效率”方面，都有很多参数需要进行调整以优化系统性能。通常情况下，这些优化参数是使用训练语料和代价函数获取的。训练语料是真实数据的一个抽样，而代价函数一般是一个取最大值或最小值的函数，也是衡量系统优化的一种方式。例如，训练语料可以是一个含有若干查询及其对应的标准查询结果的数据集，而排序算法的代价函数可以表示为对排序效果的一种衡量。系统优化的过程也因此可以描述为，在训练语料上调节排序算法的参数，使得代价函数达到最大或最小。这种优化的过程不同于“搜索引擎优化”，后者主要是指通过对网页顺序的调整来确保某些网页有较高的排名。

在本章后续的部分中，主要针对搜索引擎的两个重要的评价指标“效果”和“效率”，着重讨论几种较为重要的评价策略。同时也将介绍如何在严格控制的实验环境中实施一些实验步骤，以确保结果是有意义的。

8.2 评价语料

评测最基本的要求是可以将不同的技术方案在同一个平台上进行比较。为了使得这种对比公平公正，且可以重现，实验设置和使用的评测数据必须是固定的。最早的大规模搜索性能评价开始于20世纪60年代，一般称为克兰费尔德[⊖]实验（Cleverdon，1970年）。研究者们通过收集一些测试集（test collections），包括文档集、查询以及文档与查询的相关性判断等，来完成这次评测。在其他与语言相关的研究领域，如计算语言学、机器翻译或是语音识别领域，也需要大量的测试语料（text corpus多以文档的形式表示）来做各种统计分析。这些语料，或者称为评测集（evaluation corpus），在信息检索领域是非常特殊的，这是因为该评测集中除了文档信息之外，还需要包括查询与文档的相关性判断。

评测集多年的变更，反映出检索数据的变化以及典型搜索应用的变化。在此给出如下的例子来展示这些变化。从20世纪80年代起，每隔10年左右构建一个评测集，最终有如下三个评测集：

- CACM：标题和摘要来自于1958～1979年的ACM通信，其中的查询和相关性判断由人工制定。

⊖ 克兰费尔德是英国的一个城市，也是评测实验完成的地点。

- AP：文档集来自于1988～1990年的美联社新闻报道（来自TREC的1～3光盘）。查询来自于TREC的话题51到话题150的标题。话题和相关性判断由政府信息分析员进行制定。
- GOV2：文档集来自于2004年初的从.gov结尾的网站上抓取的网页。查询来自于TREC的话题701到话题850的标题。话题和相关性判断由政府信息分析员进行制定。

CACM数据集的建立，是由于当时大多数的搜索应用集中在参考书目（bibliographic records仅包含标题和摘要）的搜索任务上，而并不需要全文搜索。表8-1显示了该数据集中文档的数目（3204篇），同时可以看出，每篇文档中词语的平均数量也非常的小（64个）。整个文档集的大小仅有2.2兆字节，甚至小于现在的一个音乐或MP3文件的大小。该数据集由计算机方面的论文摘要构成，其中的“查询”由计算机学院的师生共同构建完成，这些查询都来自实际的信息需求。CACM查询的示例如下所示：

局域网络的安全因素、网络操作系统、分布式系统

其中，每个查询返回文档的相关性判断由同一个人来决定。这在当时是可行的，因为查询的数据集规模很小，并且构建查询的相关师生对这些数据集都非常熟悉。表8-2中列出了CACM数据集中的查询都比较长（平均13个词），而评测集的规模却非常小，平均每个查询约有16个相关的文档。

表8-1 三种数据集的统计信息。其中，统计每篇文档的平均词数时没有进行词形转换

数据集	所包含的篇章数量	大小	平均每篇含有的词的数量
CACM	3 204	2.2 MB	64
AP	242 918	0.7 GB	474
GOV2	25 205 179	426 GB	1073

AP和GOV2数据集是由NIST支持的TREC评测系列数据集的一部分。AP数据集是一个典型的全文数据集，并在20世纪90年代首次使用。廉价的磁盘存储技术以及大量的在线文字录入，促进了大量的搜索应用的产生，如新闻事件、法律文件以及百科文摘等数据库的搜索等。AP数据集，无论是文档的数量还是文档集的字节大小，都要比CACM数据集大得多（大约大出两个数量级）。由于AP数据集的文档是由整篇的新闻事件构成，导致文档的平均长度也变长了。相比AP而言，GOV2数据集又增加了两个数量级的规模。它被设计为检测网络搜索应用技术的平台，由以.gov结尾的网站提供。由于这种网站提供较长的政府政策描述或是表格，因此，文档的平均长度是这三种数据集中最长的。

表8-2 三种数据集的查询统计信息

数据集	查询的数量	每个查询的平均词数	每个查询的平均相关文档数
CACM	64	13.0	16
AP	100	4.3	220
GOV2	150	3.1	180

AP和GOV2数据集的查询都是与TREC话题相关的。这些话题由NIST聘用的政府信息分析员所制定。早期的TREC话题一般反映了政府或工业生产中的专业分析员的需求，并且相当的复杂。后期的TREC话题设计则更多反映了普遍的信息需求，但它们依然保留了TREC的话题设计模式。图8-1是其中的一个示例。TREC的话题模式共有三个部分，由不同的标签表示。

其中，<title>标签表示一个较短的查询，类似于网页搜索中的典型查询；<description>标签是对该查询的详细描述，是查询的扩展版本，如图8-1中所示，<description>要比<title>的解释更为详尽；而<narrative>标签则描述了检索返回的文档与查询的相关性的标准。这些标准一般由人工分析员所使用，以便对最终查询检索后返回文档的相关性做出判断，提高一致性，而并非是查询的一部分。近期，在大部分的TREC评测中，人们一般使用话题的<title>部分作为查询，关于查询的各种统计数据如表8-2所示。

```
<top>
<num> Number: 794

<title> pet therapy

<desc> Description:
How are pets or animals used in therapy for humans and what are the
benefits?

<narr> Narrative:
Relevant documents must include details of how pet- or animal-assisted
therapy is or has been used. Relevant details include information
about pet therapy programs, descriptions of the circumstances in which
pet therapy is used, the benefits of this type of therapy, the degree
of success of this therapy, and any laws or regulations governing it.

</top>
```

图8-1 TREC话题举例

TREC中的相关性判断依赖于待评价的任务。对于这些表格中的查询，如果待评价的任务重视较高的召回率，则意味着不能丢失太多的相关反馈文档。给定某项任务的上下文语境，如果某篇文档包含的信息可以用来帮助写出一篇查询相关的报道，则TREC分析员认为该文档与查询是相关的。在第7章中讨论过用户相关（user relevance）和话题相关（topical relevance）的不同之处。尽管TREC的相关性定义确实需要顾虑所发现信息的有用性，但是分析员需要进一步判断同样含有有用信息的所有文档是否都是相关的。但这并不是真实用户愿意做的事情，TREC更侧重于“话题相关”。对于CACM数据集而言，所有的相关性评价均是二值的，即判断一篇文档是查询相关还是不相关。对于大部分的TREC数据集而言，同样是这样的。然而对于某些任务而言，多值的相关性评测或许是适合的，我们将在8.4节讨论关于二值和多值的效果（effectiveness）相关性评价方法。此外，不同的检索任务还将影响人工分析员的数量、分析的方式以及效果评价的具体方法。例如，在第7章中描述了导航式（navigational）检索，用户使用这种检索时，只需要查看一个相关网页。即对于某一查询，只需要反馈一个相关文档。

创建一个新的测试集是一件费时费力的任务，特别是为了获取较高的召回率，相关性判断工作需要相当大的人力投入。当测试集非常小的时候，可以对测试集中的大部分文档进行相关性检测。然而，对于类似于GOV2这样大规模的测试集，这显然是不可能完成的任务。在这种情况下，使用一种称为检索池（pooling）的技术。该技术将不同的搜索引擎（或不同的检索方法）的检索结果分别进行排序，并选取前k个结果（对于TREC评测而言，k的范围一般在50~200之间）汇成一个检索池。该检索池中的文档都可看作查询相关的文档，这些文档经过去重后以随机的顺序存在，并协助人们完成相关性判断。检索池为每个查询准备了大量的相关反馈文档，如表8-2所示。然而，检索池中的相关文档数量是不完整的，一个新的检索算

法检索出的所有相关文档并不一定都在检索池中，这显然是存在问题的。也就是说，如果一个新的检索算法找到很多查询相关的文档，然而这些文档若不在检索池中，则将被视为查询无关的文档，也因此相应地，低估了这种新的检索算法的性能。但是通过对TREC数据的详细考察发现，该数据足以对新的检索算法进行相关性判断方面的比较。

TREC数据对于评测新的搜索技术是非常有帮助的，但是它们仍然存在一些缺陷。例如，对于评价一个商业网站的产品搜索技术而言，追求高召回率以及使用新闻文档作为评测集显然是不合适的。当然，可以建立新的TREC任务评估这些新的应用，但是这个过程要花费几个月甚至几年的时间。另一方面，新的搜索任务和新的数据类型，例如博客、论坛以及人工标注的视频，也在持续不断地发展。但幸运的是，使用下面提到的这几点基本的准则，为任意的应用任务开发一套新的评价语料会变得不那么困难。

1）考察一个文档集对于某一应用任务而言是否有代表性，要从数量、大小以及文档的类型等几个方面来考察。比如，对于某些任务而言，可能要列出所有的真实的数据集，而对于某些任务，只需要给出真实数据集的示例或者很少的一部分数据集即可。此外，如果目标应用任务具有普适性，那么需要使用多个不同类型的数据集来评测。例如，在TREC的高召回率任务中，一些不同类型的新闻数据集和政府数据集都作为评测集使用。

2）对于评测集所使用的查询，应该选择在目标应用中用户提交的有代表性的查询。这些查询一般来自相似的应用系统的查询日志，或者通过询问潜在的用户获取查询的样例。虽然通过上述方法可以获取大量的查询，但是相关性判断却成为一个最主要的瓶颈。我们必须获取足够数量的查询，以验证一项新的技术是否具有显著性。TREC实验的一项分析显示，若使用25个查询且使用MAP（8.4.2节将介绍）效果评价方法对两个搜索引擎系统进行评价，MAP0.05的区别将导致13%的结论错误。若使用50个查询，错误率将降低为4%以下。在MAP评价方法中，随着查询数量的改变，0.05的差别的含义是不同的。如果使用如8.6.1节中所提及的显著性检验（significance test）用于评价两个系统不同的MAP，当使用50个查询时，10%的相对差别足以保证一个较低的错误率。如果有资源或应用系统能够提供相关性判断，则能够通过一些可信赖的文档结果来判别是否与查询相关，而并非通过现有的查询来对文档做出判断。有一些策略，例如，针对每个查询（大量的查询），分别从排序好的检索文档中选取一小部分（如前10个）；或者选出哪些能够很好地区分几个对比系统的文档等，这些策略都已经证实了是有效的。然而，如果只使用一小部分的查询，检索的结果只能说是象征性的，而并非令人信服的。在这种情况下，查询至少要很有代表性。此外，由于要以应用为目标，因此查询还需要具有很好的覆盖性。例如，如果对“地域检索”算法进行测试，相关的查询应该包含很多不同种类的地域信息。

3）查询与文档的相关性应该由提出问题的人们判断，或者由受过专业培训的人员来判断（这些人员知道对于一个应用怎样决定文档的相关性）。相关性看起来像是个非常主观的概念，不同的人对相关性的看法不同，有时候同一个人在不同的时间段的判断结果也是不同的。虽然存在这些不同，但TREC实验分析显示，通过相关性判断，系统的相对性能还是非常稳定的。换句话说，相关性判断的不同并没有给系统对比实验的错误率带来很显著的影响。查询相关性判断的文档的数量，以及相关性判断的方法，依赖于效果评价方法的选择。对于大部分应用而言，人们更容易在至少三个层次的相关性上进行选择，即直接相关（definitely relevant）、直接不相关（definitely not relevant）和可能相关（possibly relevant）。当然，如果某种效果评

价方法需要的话，也可以将这三个层次转化为二元判定，即将“可能相关”分为“直接相关”或“直接不相关”。尽管如此，有些应用和效果评价方法，可以支持多于3个层次的相关性。

最后，非常有必要强调一点，很多用户行为可以被看作模糊的相关性假设，如果这些可以被挖掘，就可以很大程度上降低创建测试集的代价。例如，类似于在搜索结果列表中点击某一篇文档的用户行为，以及将该文档移动到文件夹的行为，或者发送到打印机的行为等都将显示，这个文档是查询相关的。在前面的章节中已经描述了查询日志和点击链接（clickthrough）可以用来支持一些操作，如查询扩展和拼写纠错等。接下来，将讨论查询日志在搜索引擎评价中的重要角色。

8.3 日志

在网页搜索引擎的发展中，捕获用户和搜索引擎交互的查询日志成为一种极其重要的资源。从评价的观点来看，这些日志提供了大量的数据，来显示用户如何浏览搜索引擎对于某一查询所提供的结果。在通用的网页搜索应用中，用户和查询的数量可能有几千万。和典型的TREC数据集中几百个查询相比，查询日志数据能够潜在地支持更大规模的和更真实的评价。然而，这些数据的主要缺点是，不如显示的相关判断精确。

一个额外的顾虑是需要维护用户的隐私（privacy）。当查询日志被共享、为研究目的发布，或者用来构建用户profile（见6.2.5节）时，这成为一个尤为重要的问题。可以采用多种技术使日志数据匿名化，例如删除标识信息或者可能包含个人数据的查询，虽然这对于某些应用而言，会降低日志的可用性。

对于每个查询而言，典型的查询日志包括下面的数据：

1）用户标识或者用户会话标识。这些可以通过多种方式获得。如果用户登录一个服务，使用搜索工具条，甚至通过cookie，这些信息使得搜索引擎能识别用户。会话是指在一段时间内，提交给搜索引擎的一系列查询。在某些条件下，仅可能在会话环境中识别出用户。

2）查询项。查询以用户输入的形式存储。

3）结果URL列表，包括它们的排序信息以及是否被点击[⊖]。

4）时间戳。时间戳记录查询提交的时间。额外的时间戳也可能记录某一具体的检索结果被点击的时间。

经过适当的转换，日志中的点击数据（第3条）已被证实类似于显式相关性判定结果，并且已经被应用于训练和评测搜索引擎。有关用户交互的更细致的信息，能够通过客户端应用程序获得，例如浏览器上的搜索工具条等。虽然这些信息并不总是可用，但是除了点击数据外的其他一些用户行为，也被证实可以很好地对相关性判断作出预测。其中两个最好的预测是页面驻留时间（dwell time）和搜索退出行为。页面驻留时间是指用户在一个点击结果上花费的时间，从开始点击到用户结束页面或者退出搜索应用程序。搜索退出行为是用户退出搜索应用程序的方式，例如进入另一个URL，关闭浏览器窗口，或者超时等行为。其他行为，如打印页面，也是非常好的预测，但是不经常发生。

虽然点击结果页面和相关性有很大关系，但是它们不能直接取代显式相关性判定的地位，因为它们高度依赖于排位靠前的或者有其他特征的页面，例如受欢迎的或者有好的摘要的结

⊖ 在一些日志中，只记录被点击的URL。记录所有的结果能产生用户偏好，并提供“反例”用于多种任务。

果页面。这意味着，即使与查询相关的页面排序靠后，排在前面的页面还是会被更频繁地点击。消除这种偏见的一种方法是记录用户点击数据，来预测用户对返回文档集的偏好(preference)。用户偏好在7.6节中首次提到，用来训练相关排序函数。比如，和文档d_2相比，用户偏好文档d_1则意味着d_1更相关，或者等价地，它应该排序更靠前。偏好最适合于文档可以有多个相关级别的搜索任务，并且将更多的注意力集中于用户相关性，而不是话题相关性。相关性判定（多级或者二级）可以用来生成偏好，但是偏好并不暗示特定的相关级别。

通过挖掘一些策略，可以对用户点击数据进行分析，来发现用户的使用偏好。这些策略通过对用户行为进行观察，并通过实验进行验证。和7.6节介绍的相似，采用了一个被称作Skip Above and Skip Next的策略（Agichtein, Brill, Dumais, & Ragno, 2006）。给定一个查询的结果集合以及相关排序位置p的点击结果，该策略假设排序在p之前的所有没有被点击的结果不如p的结果。另外，紧挨着点击结果后的未被点击的结果，比点击结果更不相关。例如，下面给定相关排序结果列表和点击数据：

d_1

d_2

d_3（点击）

d_4

该策略将生产下面的偏好：

$d_1 > d_2$

$d_3 > d_1$

$d_3 > d_4$

既然只有当排序较高的文档被忽略时才能生成偏好，那么前面提及的一个主要的偏见也被消除了。

该“跳转”策略利用用户的点击模式产生偏好。然而，用户行为的变化，将导致获取的数据存在噪声和不一致的现象。查询日志对于不同用户提交的相同的查询，保存了许多组实例，这些点击数据可以被汇总起来，以便消除来自每个用户之间的不同的潜在噪声。具体地，点击分布（click distribution）信息可用来识别哪些是理想的具有更高点击频率的点击数据。这些点击已被证实是查询相关的文档。对于给定的查询，可以对排在p位置上的结果d，使用日志中所有的查询实例，计算出实际的点击频率$O(d, p)$。也可以通过对所有的查询结果进行平均，计算排在位置p的期望点击频率$E(p)$。对排在p位置上的结果d的点击误差（click deviation）$CD(d, p)$的计算方法为：

$$CD(d, p)=O(d, p)-E(p)$$

继而使用$CD(d, p)$的值“过滤”点击数据，并为跳转策略提供更可靠的点击信息。

一个典型的评价场景是，使用两个或者更多个系统对一个给定的查询集合的检索结果进行比较。给定一个查询，偏好是决定哪些文档与之相关的另一种评价方法（相关性判断是典型的方法）。对于每个系统的结果列表的质量而言，或者使用基于偏好，或者使用基于相关性判定的方法进行效果评价。下一节描述在研究和系统开发中，最经使用的效果评价方法。

8.4 效果评价

8.4.1 召回率和准确率

两个最常用的评价方式是召回率（recall）和准确率（precision），在Cranfield的研究中被提出，用来总结和比较搜索结果。对于一个查询，直觉上讲，召回率衡量搜索引擎找到所有相关文档的能力，准确率衡量它排除不相关文档的能力。

这些度量方法事先假定，对于一个给定的查询，将有一个被检索（retrieved）的文档集，以及一个未被检索（not retrieved，其余的文档）的文档集。显然，这可以应用到布尔搜索的结果中，但是我们将要看到，相同的定义也能用于相关排序搜索中。另外，如果相关性被假设为二元的，那么对于一个查询的结果，可以如表8-3所示。在此表中，A是相关文档集合，$\overline{A}$是不相关集合，B是被检索到的文档的集合，$\overline{B}$是未被检索到的文档集合。运算符$\cap$给出两个集合的交集。例如，$A \cap B$是相关并且被检索到的文档集合。

表8-3 基于二元相关性的简单检索定义的文档集

	相关的	不相关的
被检索	$A \cap B$	$\overline{A} \cap B$
未被检索	$A \cap \overline{B}$	$\overline{A} \cap B$

大量的效果评价方法可以使用上面的统计表定义。如下的这两个公式是经常使用的：

$$\text{召回率} = \frac{|A \cap B|}{|A|}$$

$$\text{准确率} = \frac{|A \cap B|}{|B|}$$

其中$|\cdot|$表示集合的大小。换句话说，召回率是相关文档被检出的比率，准确率是计算检出的文档中相关文档的比率。使用这些度量方式的一个潜在假设是，该任务致力于检索尽可能多的相关文档，并使检索到的不相关文档数量达到最小。也就是说，即使对某个查询而言有500篇相关文档，用户也倾向于将它们全部找到。

表8-3中的搜索结果类似于7.2.1节提到的二元分类器的输出。当一个文档被检索到时，即预测该文档是否是相关文档。从这个观点来看，预测（或者检索）中有两类错误。这些错误被称作假阳性（false positive，非相关文档被检出）和假阴性（false negative，相关文档未被检出）。召回率和假阴性错误类型相关，但是准确率并不直接和假阳性类型相关。相反，一种被称作虚报率[⊖]（fallout）的评价方式，定义为不相关文档被检出的比率，和假阳性相关：

$$\text{虚报率} = \frac{|\overline{A} \cap B|}{|\overline{A}|}$$

既然给出的虚报率和召回率可以共同评价搜索引擎的性能，为什么还是要使用准确率来替代虚报率呢？很简单，这是因为准确率对于使用搜索引擎的用户更有意义。对于一个查询，如果有20篇文档被检出，0.7的准确率值表示被检出的20篇中有14篇是相关的。另一方面，虚

⊖ 在分类和信号检测文献中，错误通常包括类型I和类型II错误。召回率经常被称作真阳性率，或者敏感度。虚报率被称作假阳性率，或者虚率。还有一个特殊的虚报率是1−虚报率。准确率被称作正预测值，并且经常用于医疗诊断测试中，其中正测试是正确的概率值，非常重要。真阳性率和假阳性率被用于刻画ROC（receiver operating characteristic）曲线，随着鉴别度的变化，该曲线显示了这两个量之间的折中。该阈值表现为分类器作出正预测时的值。在搜索中，该阈值反映文档排序中的某个位置。在信息检索中，召回率-准确率折线图一般取代了ROC曲线。

报率总是很小，因为有太多非相关文档。假设有1 000 000篇非相关文档，虚报率将是6/1000000 = 0.000006。如果系统的准确率降为0.5，这会很容易被用户发现性能下降；但若使用虚报率，则为0.00001，如此小的数字则不易发现性能的改变。由于语料库中大多数文档都和任意给定的查询不相关，因此如果从分类器的角度来对这一搜索任务进行评价，将会导致与直觉相反的结果。这是因为将一个文档定义为不相关文档通常是正确的决策，这将导致被训练用来寻找最小化分类错误的搜索引擎倾向于什么都不检索！

F值度量（F measure）是综合召回率和准确率的效果评价方法，它用来评价分类性能以及一些搜索引擎应用的性能。它的好处是使用单一的数字总结系统性能。它定义为召回率和准确率的调和平均数（harmonic mean），即

$$F = \frac{1}{\frac{1}{2}\left(\frac{1}{R} + \frac{1}{P}\right)} = \frac{2RP}{(R+P)}$$

为什么使用调和平均数而不是通常的数字平均数呢？调和平均数强调较小的数值的重要性，而数字平均数受异常值影响较大。例如，如果返回几乎整个文档数据库的检索结果，那么召回率是1.0，而准确率趋向于0。算术平均数是0.5，但是调和平均数接近0。调和平均数显然更好地总结了检索结果的效果[⊖]。

这里讨论的大多数检索模型的输出都是经过排序的文档。召回率和准确率这两种度量方法，必须基于检出的排好序的文档集合。一种可能是在排序的每个文档位置上，计算召回率和准确率。图8-2显示了使用两种排序算法进行检索的前10篇文档，以及对于一个有6篇相关文档的查询，在每个排序位置所计算的召回率和准确率的值。这些排序反映了不同的检索算法或者搜索引擎的输出。

召回率	0.17	0.17	0.33	0.5	0.67	0.83	0.83	0.83	0.83	1.0
准确率	1.0	0.5	0.67	0.75	0.8	0.83	0.71	0.63	0.56	0.6

召回率	0.0	0.17	0.17	0.17	0.33	0.5	0.67	0.67	0.83	1.0
准确率	0.0	0.5	0.33	0.25	0.4	0.5	0.57	0.5	0.56	0.6

图8-2 针对6个相关文档的两种排序方案的召回率和准确率

在排序位置10（也就是说，当有10篇文档被检出时），这两个排序有相同的召回率和准确率。召回率是1.0，因为所有相关文档都被检出了；准确率是0.6，因为两个排序算法在检出的

⊖ F值度量的更通用形式是加权调和平均数（weighted harmonic mean），它使用权重反映召回率和准确率的相对重要性。$F = PR/(\alpha R + (1-\alpha)P)$，其中$\alpha$是权重。它经常被变换为$\alpha = 1/(\beta^2 + 1)$，其给出$F_\beta = (\beta^2 + 1)PR/(R + \beta^2 P)$。实际上，通常的$F$度量是$F_1$，也就表示其中的召回率和准确率同样重要。在一些评测中，准确率和召回率通过变换β的值进行强调。例如$\beta > 1$强调召回率。

10篇文档构成的集合中，都包含6篇相关文档。然而，考察较靠前的排序位置，第一个排序算法明显要好。例如，在排序位置4（4篇文档被检出），第一个排序算法的召回率是0.5（6篇相关文档有3篇被检出），准确率为0.75（4篇被检出的文档中，有3篇是相关的）。第二个排序的召回率是0.17（1/6），准确率为0.25（1/4）。

如果一个查询有更多的相关文档，或者相关文档零散地分布在排序中，则针对每个排序位置的召回率-准确率值的列表将会很长，而且不容易进行其他操作。因此，研究者探索出许多技术，用以概述排序的效果。第一种技术是简单地在一些预定义的排序位置上计算召回率-准确率值。事实上，对于一个查询，为了比较两个或者更多的排序算法，只要在预定义排序位置上计算出准确率即可。如果一个排序位置p的准确率比另一个排序高，召回率也会较高。这通过比较图8-2相应的召回率-准确率值可以看出。这一效果评价方式被称作位置p的准确率（precision at rank p）。对于排序位置p，可以选取许多可能的值，但由于用户比较关心排序靠前的输出结果，因此，最常使用的是P@10和P@20的准确率度量。注意，如果使用这些度量，则暗示搜索任务已经变为在给定排序中找到尽可能相关的文档，而不是找到尽可能多的相关文档。因此，对于各种不同的搜索算法而言，位置在20之后的输出结果则不考虑了，此外，也不需要区分位置1到p的搜索对比结果的位置的不同。然而对于某些任务，它们也可能比较重要。例如，图8-2中的两个排序算法，使用P@10的度量结果是相同的，就有必要考虑位置10之后的排序结果。

另一个评价排序算法效果的方法是，当召回率从0.0到1.0每次增加0.1这一标准值时，计算准确率的相应变化。每个排序算法使用11个数字表示。该方法的优点在于，可以用于评价所有排序结果中的相关文档，而不仅仅是那些排序靠前的文档。然而，以图8-2中的召回率-准确率值为例，很明显，在这些标准召回率值上，通常并不能获得上述方法中所描述的准确率的值。在此例中，只有在0.5和1.0的召回率处计算准确率。为了获取在所有标准召回率处的准确率的值，采用了插值（interpolation）的方法[⊖]。由于该值被用做对多个查询以及生成召回率-准确率图求平均效果的基础，因此在下一节中将详细讨论插值。

第三种评价排序算法的度量，也是最常用的一种方法，是计算当一个额外的相关文档被检出（也就是当召回率增加）时，准确率的平均值。如果一个相关文档由于某种原因[⊜]没有被检索到，则该文档对于平均值而言是毫无贡献的。对于图8-2中的第一个排序算法，平均准确率（average precision）为：

$$(1.0 + 0.67 + 0.75 + 0.8 + 0.83 + 0.6)/6 = 0.78$$

对于第二个排序为：

$$(0.5 + 0.4 + 0.5 + 0.57 + 0.56 + 0.6)/6 = 0.52$$

平均准确率这种度量方法有许多优点，它使用一个单一值来衡量所有相关文档的排序结果，但是这个值严格依赖于排序位置较靠前的相关文档。对于那些目的是找到尽量多相关文档的任务进行评价，这是一种合适的度量方法，然而，该方法仍然反映了“排序越靠前，越重要”这一直觉。

对于某一查询，这三种评价方法可以对排序算法的效果进行总结。为了对检索算法进行

⊖ 插值指的是任何在两个已知点之间计算新点的技术。

⊜ 一个最普遍的原因在于，我们仅考察那些排序位置靠前的文档（如前1000篇），而这些文档的数量非常有限。

真实的评定，必须使用多个查询进行测试。给定查询集合以及潜在的大量的检索结果，需要一种总结检索算法性能的方法，即针对整个查询集合评价排序算法的性能。在下一节中，将讨论在很多评价实验中使用到的平均化技术。

8.4.2 平均化和插值

在下面对平均化技术的讨论中，都以图8-3所示的两个排序为例。这些排序结果分别是使用相同的排序算法对两个不同的查询进行的检索排序。平均化技术的目标是，通过某个查询集来总结某个排序算法的排序性能。不同的查询经常会产生不同数量的相关文档，这个例子也是如此。图8-3给出了这两个查询的前10个相关排序位置的召回率-准确率值。

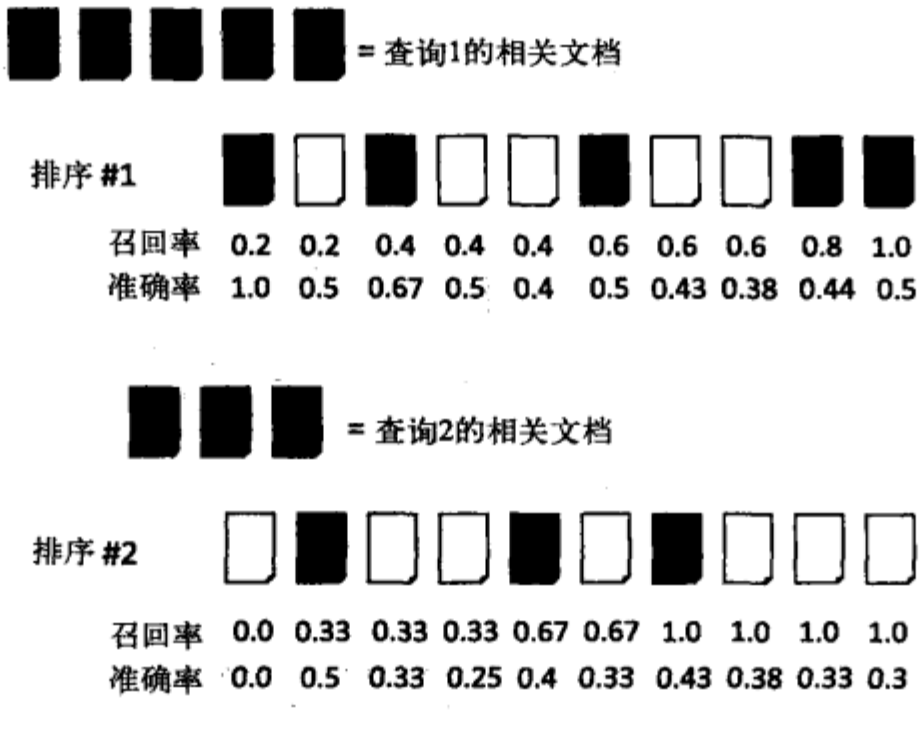


图8-3 两个不同查询在同一种排序算法上的召回率和准确率值

平均准确率为每个查询的相关排序结果赋予一个评价数字，从多个查询中总结该排序算法性能的最简单的途径，就是对这些数字进行平均。这种效果评价方法，称为平均准确率（Mean Average Precision, MAP[⊖]），目前大部分研究文章以及系统评测[⊜]都使用这种评价方法。既然它是基于平均准确率的，也就意味着它假设对于每个查询，用户期望找到更多的相关文档。因此，虽然已经提出许多减少相关性评定工作的方法（比如Carterette等，2006），但是使用MAP来比较检索算法或者系统，仍需要大量的文档相关性判定工作。

对于图8-3中的例子，MAP计算如下：

查询1的平均准确率 = (1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62

查询2的平均准确率 = (0.5 + 0.4 + 0.43)/3 = 0.44

MAP = (0.62 + 0.44)/2 = 0.53

⊖ 这听起来比average average precision要好。

⊜ 在一些评测中，平均准确率的几何平均值（Geometric Mean of the Average Precision，GMAP）经常用来代替算术平均值。由于它对平均准确率的值求积，因此可以加强对较低性能的查询的重视。它定义为：

$$GMAP = \exp\frac{1}{n}\sum_{i=1}^{n}\log AP_i$$

其中，n是查询的数量，AP_i是查询i的平均准确率。

MAP评价方法为排序算法的性能提供了一个非常简洁的总结。然而，在通常情况下，虽然这种度量方法非常有效，但是在这一过程中，有时会丢失太多的文档信息。召回率-准确率图（Recall-precision graph），以及标出详细数值的召回率-准确率表格，在不同的召回率水平上，能够给出更详细的排序算法的性能。图8-4是图8-3中的两个查询的排序结果的召回率-准确率图。每个查询所对应的图呈现出非常不同的形状，很难比较。为了生成一个能够总结所有查询效果的召回率-准确率图，图8-3中的召回率-准确率的值应该进行平均化。为了简化平均化过程，每个查询的召回率-准确率值被转化为在标准召回率等级上的准确率值，如上一节所提及的。基于此，将标准召回率等级上的所有查询的准确率值进行平均，来对排序算法进行评价[⊖]。

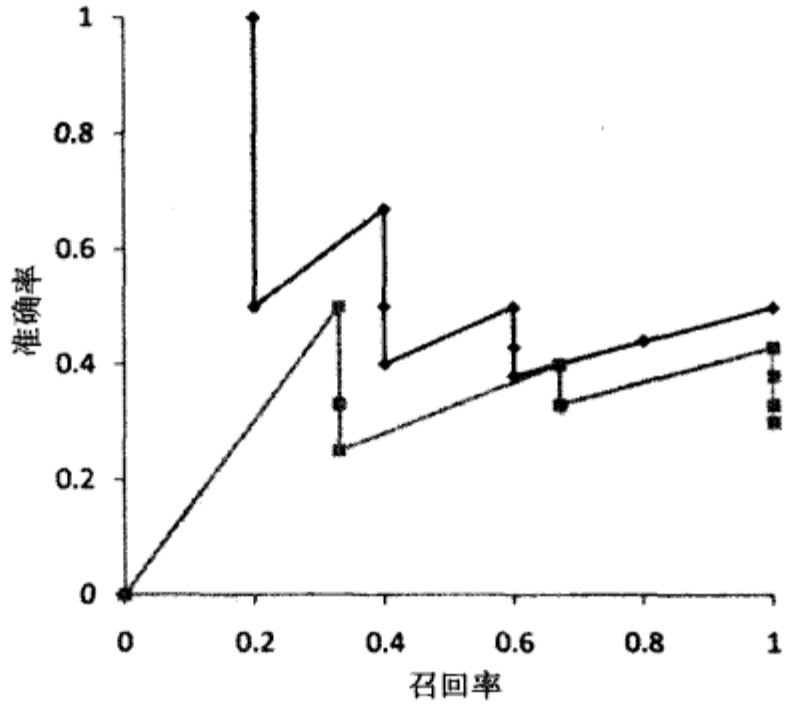


图8-4 两个查询的召回率—准确率图

标准召回率等级是0.0到1.0，增量为0.1。为了获得每个查询在这些召回率等级下的准确率，例如图8-3中所示的召回率-准确率数据点，必须进行插值。也就是说，需要基于这些数据点定义一个函数，让其在每个召回率等级处都有值。目前有许多种插值的方法，但是自从1970年以来，只有一个方法被用于信息检索评价中，该方法在任何标准召回率等级R处，定义准确率P为：

$$P(R) = \max\{P';\ R' \geqslant R \wedge (R', P') \in S\}$$

其中，S是观测点（R, P）的集合。在任意的召回率等级处，该插值定义准确率为在较高召回率等级处，召回率-准确率点中观测到的最大准确率，产生如图8-5所示的阶梯函数。

因为搜索引擎并不完美，并且几乎总是能够检索到一些不相关的文档，所以随着召回率的提高，准确率趋于降低（虽然这不总是正确的，如图8-4所示）。这种插值方法和这一观察一致，它产生一个单调下降的函数，这意味着准确率的值总是随着召回率的升高而下降（或

⊖ 在文献中被称为宏平均（macroaverage）。宏平均首先为每个查询计算兴趣的度量，然后将这些度量进行平均。微平均（microaverage）合并了来自查询的所有适当的数据点，并且基于合并的数据点计算度量。例如，在相关排序位置5处的微平均准确率为 $\sum_{n=1}^{n} r_n / 5n$，其中r_i是查询i的前5个文档中相关文档的数量，n是查询的数量。宏平均被用于多数检索评价中。

者不变)。该插值也为0.0召回率等级定义了一个准确率的值，否则不容易理解。该插值方法背后通常的直觉是，召回率-准确率值由具有最好的准确率值的相关文档集决定。例如在查询1中，用户最有可能去查看前3个文档。

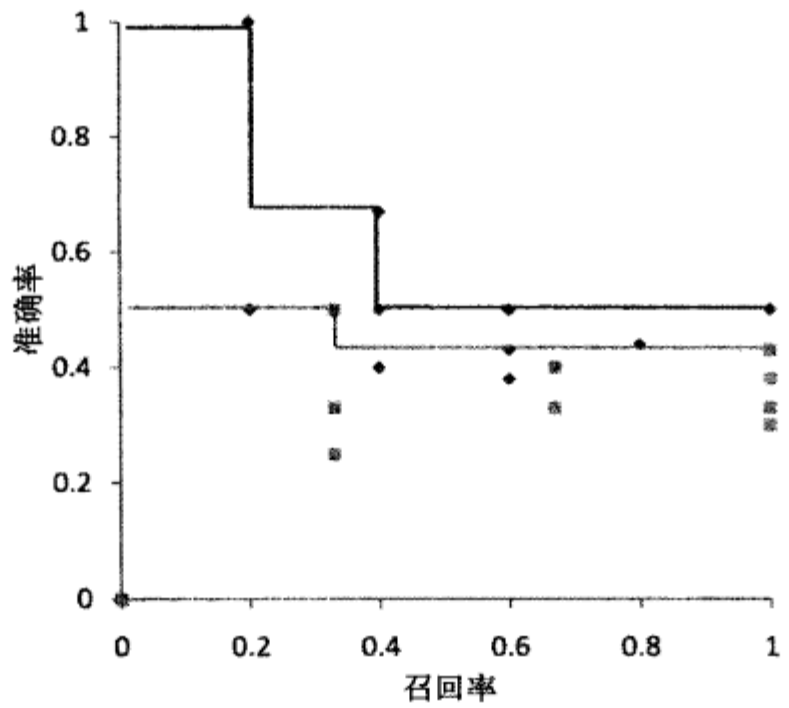


图8-5 两个查询的插值召回率—准确率折线图

表8-4 在标准召回率等级上使用插值法的准确率值

召回率	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
排序1	1.0	1.0	1.0	0.67	0.67	0.5	0.5	0.5	0.5	0.5	0.5
排序2	0.5	0.5	0.5	0.5	0.43	0.43	0.43	0.43	0.43	0.43	0.43
平均	0.75	0.75	0.75	0.59	0.47	0.47	0.47	0.47	0.47	0.47	0.47

标准召回率等级处的平均准确率，通过简单地对每个查询的准确率值进行平均而获得。表8-4给出了两个查询的排序结果经插值后的准确率值，以及平均准确值。最终的平均召回率—准确率折线图如图8-6所示。

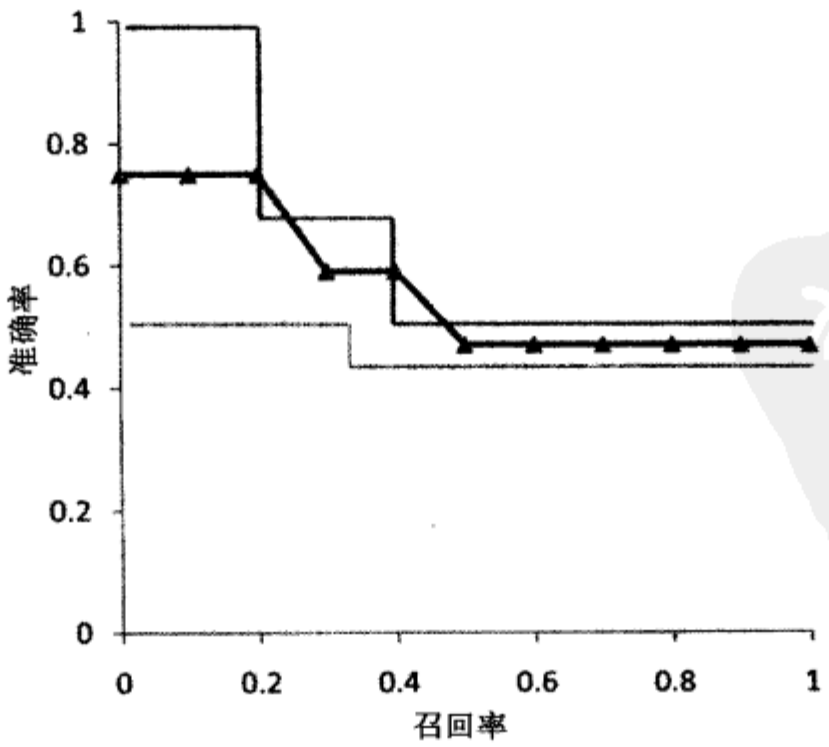


图8-6 基于标准召回率的平均召回率—准确率折线图

平均召回率-准确率折线图通过简单地连接标准召回率等级处的平均准确率点画出，而不是使用另一个阶梯函数。当对多个查询进行平均时，平均召回率-准确率图趋于逐渐平滑。图8-7给出了一个使用50个查询的TREC评测中的典型的召回率-准确率折线图。

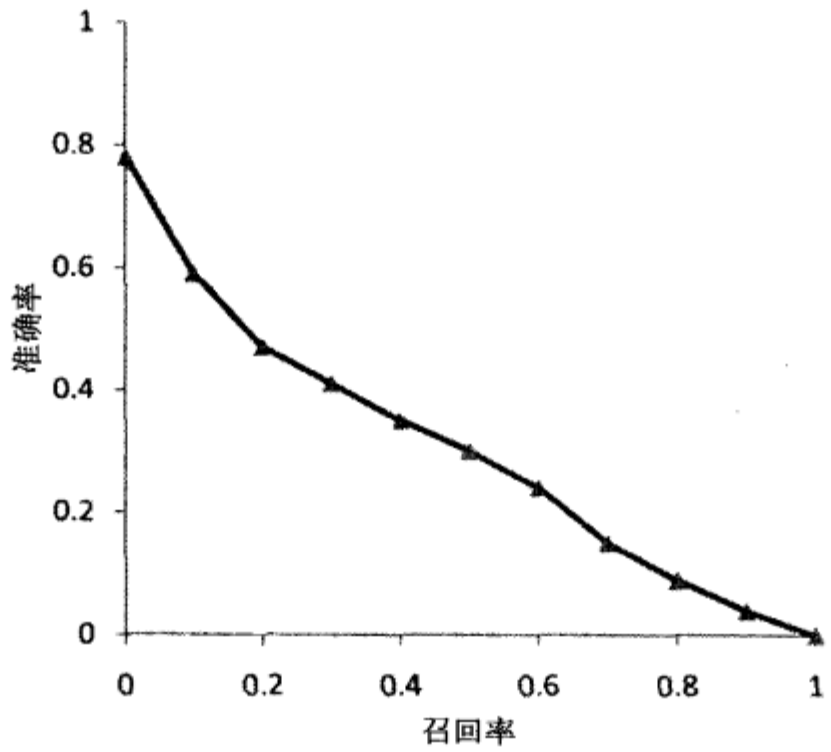


图8-7 TREC中50个相关查询的典型召回率-准确率折线图

8.4.3 关注排序靠前的文档

在许多检索应用系统中，用户倾向于查看排序靠前的相关文档。在网络检索的情况下，这意味着很多用户仅查看检索结果的第一页或前两页。除此之外，一些检索任务如导航式检索（见第7章）或是问答（question answering）（见第1章），仅需要一个单独的相关文档。在这种状况下，召回率并不是一个合适的评价指标。相反，我们更关注搜索引擎检索到的相关文档是否排在结果列表的前面。

针对上面所述的情况，计算*p*位置的准确率的评价方法（precision at rank p）已经被提出，其中*p*一般设为10。这种评价方法易于计算，通过平均多个查询的评价结果来对搜索引擎进行评价，同样，这种评价方法也非常易于理解。这种方法最主要的缺点在于，对于给定数量的相关文档，不能很好地区分不同的排序情况。例如，如果在检索返回的前10篇文档中，只有一个相关文档，根据P@10的计算方法，这篇文档位于排序中的第1位和第10位，准确率结果是一样的。其他对排序位置较为敏感的评价方法随后将介绍。

排序倒数（reciprocal rank）评价方法主要用于上面所述的只需要返回一个单独的相关文档的情况。它定义为返回第一个相关文档位置的倒数。平均排序倒数MRR（mean reciprocal rank）是针对一组查询的排序倒数平均值。例如，对于一个查询而言，假设返回的前五个文档的情况分别为dn、dr、dn、dn、dn，其中dn是不相关的文档，而dr是相关文档，那么排序倒数值为1/2=0.5。即使有更多的相关文档被检索出来，像这样的一个排序，dn、dr、dn、dr、dn，排序倒数的值依然是0.5。排序倒数评价方法对排序的位置非常敏感。排序倒数从1.0降到0.5，即从排序第一的位置降到第二的位置，而针对排序dn、dn、dn、dn、dr，将得到1/5=0.2的排序倒数值。对于这两个排序结果的MRR值为(0.5+0.2)/2=0.35。

对于网络搜索评价及其相关的应用（Järvelin & Kekäläinen, 2002）而言，DCG

(discounted cumulative gain) 是一种较为普遍的评价方法。这种方法基于两点假设：

1）高相关性的文档比边缘相关的文档要有用得多。

2）一个相关文档的排序位置越靠后（例如，在排序结果列表的最后），对于用户的价值就越低，因为它们很少会被用户查看。

这两点假设产生了一种新的评价方法，这种方法为相关性设定等级，作为衡量一篇文档的有用性或者叫增益（gain）的标准。这种增益从排序靠前的结果开始计算，在靠后的排序位置上，或许会减少或大打折扣（discounted）。DCG方法是在一个特定的排序p的前提下，计算总的增益。具体地，它定义为：

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

其中，rel_i指的是检索回的文档中排序为i的文档的相关性等级。例如，已有报告指出网络检索评价往往使用一种六分等级制（从“Bad”到“Perfect”，(i.e.$0 \leqslant rel_i \leqslant 5$)）的人工相关性评价方法。如果使用二元等级制的相关性评价，则reli被指定为0或1。

上面公式中的分母$\log_2 i$是一种损失因子（discount or reduction factor），在增益公式中，经常被使用到。使用这种损失因子并不存在理论上的原因，虽然它确实提供了一个相对平滑的削减[⊖]。通过变化公式中的对数的基数，这种损失可以变得更加锐利或平滑。如当基数为2时，排序为4的损失值为1/2，顺序为8的损失值为1/3。例如，考察下面的一组排序，其中每个数字代表排序位置，每个文档范围为0~3（不相关~非常相关）的相关性等级：

3, 2, 3, 0, 0, 1, 2, 2, 3, 0

下面的这些数字代表了在其对应排序位置上的增益。这些损失增益为：

3, 2/1, 3/1.59, 0, 0, 1/2.59, 2/2.81, 2/3, 3/3.17, 0=

3, 2, 1.89, 0, 0, 0.39, 0.71, 0.67, 0.95, 0

在每个排序位置上的DCG，通过求和这些数字而构成，即为：

3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61

类似于p位置上的准确率的计算，p的具体取值根据评价方法而确定，DCG评价方法将会首先对每个查询的p位置上的DCG进行计算，继而将这些值进行平均。由于这种评价方法的注意力主要放在排序靠前的文档上，因此DCG的值会比较小。例如，在位置5或10上，DCG@5的值为6.89，DCG@10的值是9.61。然而，不同的查询，会返回不同数量的相关文档，为了便于平均不同查询的评价值，可以通过将每个排序位置上的DCG值与该查询的最优排序的DCG值进行比较，得出一个归一化的值。例如，对于某一查询，如果排序结果包含所有的相关文档，则最优排序的增益值表现如下：

3, 3, 3, 2, 2, 2, 1, 0, 0, 0

继而可以获取理想的DCG值：

⊖ 在某些出版书籍中，DCG还定义为

$$DCG_p = \sum_{i=1}^{p} (2^{rel_i} - 1) / \log(1 + i)$$

对于二元相关性判断，这两种定义方法是相同的，但对于多元的相关性判断而言，上面公式的定义对检索出的高相关性的文档增强了权重。这一版本的计算方法被很多搜索引擎公司所使用，也正是因为如此，这种方法或许会成为标准方法。

3, 6, 7.89, 8.89, 9.75, 10.52, 10.88, 10.88, 10.88, 10.88

可以通过将真实的DCG值除以理想的DCG值来归一化DCG值，表示为NDCG（normalized discounted cumulative gain）值：

1, 0.83, 0.87, 0.76, 0.71, 0.69, 0.73, 0.8, 0.88, 0.88

注意，在任何排序的位置上的NDCG值，都是≤1的。总而言之，对于一个给定的查询，NDCG定义为：

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$

其中IDCG是针对某一查询的理想的DCG值。

8.4.4 使用用户偏好

在8.3节介绍了如何从查询日志中挖掘用户偏好。用户偏好已经被用来训练排序算法，还有人建议将其作为评价文档相关性的另一种方法。但是，目前还没有标准的、基于用户偏好的效果评价方法。

一般而言，使用偏好描述的两个排序结果，可以通过Kendall τ 系数(τ)进行比较。如果P是两种排序中一致的偏好的数量，而Q代表不一致的数量，Kendall的τ被描述为：

$$\tau = \frac{P-Q}{P+Q}$$

这个估计值在1（当两个排序中用户偏好全部一致时）到−1（当偏好全部不一致时）之间变化。然而，当偏好来自于用户点击的数据（clickthrough data）时，只有一部分的排序是可用的。实验结果显示，这部分信息可以被用来学习高效的排序算法，这也意味着可以用这种方式进行排序算法性能的评价，当然并不是使用全部的用户偏好来计算P和Q。一种新的排序方法可以通过对比其产生的排序结果与已知的用户偏好，来对排序算法进行评价。例如，如果从点击的数据中学习到了15个偏好，排序的结果和其中的10个一致，那么τ 度量为(10−5)/15=0.33。虽然这种计算看起来很合理，但并没有研究证明，这种效果评价方法在系统对比中是有效的。

对于偏好来自二元相关判决的情况，BPREF[⊖]方法被证明可以较好地利用部分的点击数据信息对排序算法作出评价，并给出类似于召回率-准确率方法（例如MAP）的评价结果。在这种方法中，相关文档和不相关文档的数量进行了一定的平衡，来促进不同查询结果之间的平均。对于一个返回了R个相关文档的查询，我们仅考察前R个不相关文档，这也相当于使用$R \times R$个用户偏好。基于此，该方法定义为：

$$\text{BPREF} = \frac{1}{R}\sum_{d_r}\left(1 - \frac{N_{d_r}}{R}\right)$$

其中，d_r是一个相关文档，N_{dr}给出了排序高于d_r的不相关文档的数量（考察不相关文档集R）。如果从偏好的角度来解释，N_{dr}即是对不一致的用户偏好数量的统计（对于二元相关性判定而言）。由于$R \times R$是需要考察的偏好数量，BPREF的另一种定义为：

⊖ 二元偏好。

$$\text{BPREF} = \frac{P}{P+Q}$$

这种定义和Kendall的τ定义非常类似。主要的不同在于BPREF在0~1之间变动。由于BPREF是一个有效的效果评价方法，那么这将意味着同样的评价方法或Kendall's τ评价方法，可以综合多元相关性等级和用户偏好进行计算。

8.5 效率评价

相比效果评价而言，一个搜索系统的效率衡量起来好像要相对容易一些。所考虑的大部分问题可以使用一个计时器来自动完成，而并非需要进行代价较高的相关性判别。尽管如此，像效果评价一样，我们需要明确搜索引擎效率的哪些方面需要进行评估。表8-5给出了一些常用的评价方法。

最常使用的效率评测方法是基于查询流量（query throughput）的方法，记为每秒处理的查询数量。两个搜索引擎系统只有在同一个测试集、同一个查询集合并在同样的硬件条件下进行评测，该评测方法的结果才具有可比性。尽管如此，在相似的硬件条件下，也可以进行较为粗糙的对比。作为效率评价的一种单一数值表示方法，由于基于查询流量的评测方法是符合人们直觉的，并且反映了人们想要借助于效率数量（efficiency numbers）来解决其他较为普遍的问题的想法，因此是一种不错的方法。真正的系统用户希望借助于流量的数量估计系统容量，继而决定是否需要更多的硬件资源来处理大量的用户查询。由于很容易去衡量每秒钟传递给服务器的查询的数量，所以也很容易确定系统能否很好地处理现有的服务。

表8-5 一些重要的效率评测方法的定义

评价方法	描述
索引时间开销	用于评价在一个特定系统上建立文档索引需要的时间
索引处理器时间开销	用于评价建立文档索引所需的时间，与索引时间开销相似，但不包括I/O等待时间或系统并行获得的速度
查询流量	每秒钟处理查询的数量
查询延迟	用户提交一个查询之后，在获得返回结果之前需要等待的时间，以毫秒计算。可以使用平均值来进行评测，但中值或百分比通常更好
临时索引空间	创建索引所使用的临时磁盘空间的数量
索引大小	用于存储索引文件的存储空间的大小

仅仅使用流量来评估搜索引擎的性能的主要问题在于，它并不捕捉延迟（latency）。当用户向系统提交一个查询时，延迟衡量了该系统从接受查询到反馈相关文档的时间差。心理学研究显示，用户考虑某种操作并付诸实施的时间少于150毫秒。如果超过了这个界限，用户将会消极地对待他们察觉到的延迟。

这让我们重新回到查询流量的话题上来，由于延迟和查询流量并不是正交的，一般可以通过增加延迟来改进流量，而减少系统延迟则会导致较差的流量。可以考虑下面这样一个形象的例子：私人厨师和在饭馆点餐之间的区别。私人厨师拥有最小的可能延迟为你准备饭菜，这是因为在工作时间，她没有其他的任务，因此将全部的精力用于准备你的饭菜。遗憾的是，

私人厨师拥有较低的流量，因为她完全是为你服务的，当你没有任务吩咐的时候，她处于空闲状态。而餐厅拥有较高的流量操作，它有大量的厨师同时为不同的订餐而服务。拥有较多的订餐和较多的厨师无疑会产生一定的经济规模，例如，一个单独的厨师可能同一时间准备不同的订餐。请注意，这些厨师可以很好地同时处理这些订餐，是因为一些延迟已经被加入进了这些订餐中：并不是厨师们看到一个订餐就立即准备，他们往往会等几分钟看看是否有人点同样的餐品。这个结果显示，厨师们可以很高流量地准备餐品，但是需要一些延迟作为代价。

查询处理使用了和上述相同的方式。我们可以很轻松地构建一个搜索引擎系统，使得它一次仅处理一个查询，即将所有的资源倾其所能地给予目前的查询，就像私人厨师将其所有的工作时间给予了他唯一的顾客。这种系统的流量较低，因为每次只能处理一个查询，这会导致一些资源的空闲。一种完全相反的方法是大批量的处理查询。这种系统首先为到来的查询重新排序，这样，那些拥有共同表达部分的查询可以同时进行处理，从而节省宝贵的处理时间。尽管如此，互动型的用户会比较讨厌为他们的查询处理而等待。

与效果评价中的召回率和准确率一样，低延迟和高流量对于一个检索系统而言，都是非常优质的属性。然而，这两个指标总是互相冲突的，不可能在同时均达到最大值。在一个实际的搜索引擎系统里，查询流量扮演的角色并不是一个变量，而是一个基本需求：系统需要处理用户提交的每一个查询。剩下的两个变量是延迟（用户等待一个回复的时间）和硬件损耗（需要多少个处理器来完成搜索问题）。一种常见的描述延迟的方式是使用百分比的方式，如："99%的查询会在100毫秒之内完成"。搜索引擎设计者可以增加硬件资源，直至满足基本需求。

查询流量和延迟是最明显的系统效率评价方法，当然，也应该考虑索引的代价。例如，给定足够的时间和空间，我们可以存储每个可能长度的查询。这样的搜索系统会拥有完美的查询流量及查询延迟，但是存在巨大的存储和索引代价。因此，需要衡量索引结构的大小，以及创建索引所耗费的时间。因为索引通常是一个分布式的过程，我们需要知道在索引和运行中所使用的处理器时间。由于倒排索引需要进行临时存储，所以如何确定所用的临时存储的大小，也是一件值得研究的任务。

8.6 训练、测试和统计

8.6.1 显著性检验

在评价搜索引擎的性能时，通常会产生一些评测的数据结果，例如平均的准确率值或是NDCG值等。为了检测这些数据值是否可以很好地区分两个检索算法或搜索引擎，显著性检验是一种很好的检测方法。其中，每种显著性检验是基于零假设（null hypothesis）的。一个典型的搜索引擎对比实验，一般表现为使用某种评价方法对比由两个不同检索算法产生的排序结果。在这里，零假设也就意味着两个检索算法在效果评价方面没有什么差别。而其他的假设（alternative hypothesis)意味着二者之间存在差别。事实上，给定两种检索算法A和B，假设A是一种基线算法，B是一种新的算法，通常通过对比来展示B算法在某种效果评价方法中要比A算法好，而不是简单地发现二者的不同。对于这两种检索算法而言，它们是基于相同的查询集合的，这就是所谓的匹配对（matched pair）实验。

如果仅仅根据一个查询的结果，我们不能简单地认为算法B好于算法A，这是因为，在其他所有查询中，A或许好于B。那么，我们选择在多少个查询上面进行两个算法的对比呢？例如，在200个测试集中，如果B算法在90%的实例上要好于A算法，那么我们应该足以确定B算法在这种评测方法中效果好于A，但是这种确定的程度如何衡量呢？显著性检验就是一种衡量这种确定程度的方法。

一般而言，根据两个算法的实验结果，显著性检验会否定零假设，而支持其他的假设（例如，显示出算法B比A好）。否则，我们说零假设不能被否定（例如B或许并不比A好）。正如任何二元决策过程，显著性检验可以产生两种类型的错误。类型I错误是当零假设被否定的时候，结果正确。类型II错误即，当零假设成立时，结果错误[⊖]。显著性检验通常用power进行描述（衡量），也就是该检验能够正确地否定零假设的概率（例如，决定B算法比A好）。换句话说，一个带有较高power的显著性检验，将会降低类型II的错误产生的机会。随着样例大小的增加，显著性检验的值也会相应增加，这里样例大小指的是实验中查询的数量。同样，随着样例的增加，还会降低类型I错误的几率。

在一组特定的查询下，使用显著性检验对两种检索算法进行对比的流程如下所示：

1）对于每个查询，使用这两种检索算法输出排序结果，并对排序结果进行效果评价。

2）对于每个查询，对两个排序结果的效果评价值计算验证统计值。其中，验证值依赖于显著性检验的具体方法。继而，对该验证值进行分析，确定是否应该否定零假设。

3）对于多个查询的验证值，统计计算P值，P值指的是当零假设成立的情况下，验证值出现的概率。较小的P值意味着零假设并没有成立。

4）如果P值$\leqslant\alpha$，零假设（没有区别）被否定，应该选取其他种类的假设（例如，B算法比A算法高效）。α的值通常比较小，一般是0.05或0.1，为的是减少类型I错误。

换句话说，假定零假设成立，如果获取一个具体的验证统计值的概率很小，那么我们将否定这种假设。因此，结论变为，排序算法B比基线算法A更加高效。

验证统计值和相应的准确率P值一般使用表格或标准的统计软件进行计算。下面讨论的显著性检验由Galago提供。

上面描述的流程通常称为单侧检验（one-sided test）或单尾检验（one-tailed test），因为我们想验证B比A好。如果我们仅仅尝试验证B和A之间存在区别，将是双侧检验（two-sided test）或双尾检验（two-tailed test），P值也将被加倍计算。“side”和“tail”指的是概率分布的尾部。例如，图8-8展示了假定零假设成立，检验统计的可能值的分布图。其中，分布图中的阴影部分就是单侧检验的否定部分。如果一个显著性检验产生了验证统计值x（如图8-8所示），那么零假设将被否定，这是因为获取这个值或更高的值（指的是P值）的概率，比获取显著边界值0.05的概率要低。

在搜索引擎评价中使用最普遍的几种显著性检验是t检验（t-test）[⊜]、威氏符号秩次检验（Wilcoxon signed-rank test）及符号检验（sign test）。为了解释这些检验，我们将使用表8-6中显示的数据，这些数据显示了两种检索算法A和B对于10个查询的排序的效果评价值。表中的数值是人工赋予的，一般采用了平均准确率或是NDCG的效果评价方法，例如，在0～100的

⊖ 与8.4.1节中关于错误的讨论进行对比。

⊜ 也称为Student's t-test，其中的student是发明者的笔名，而并非使用该检验的人的类型。

范围之内（而非0～1之内）。这张表还展示了算法B和基线算法A在效果评价值上的差别。该实验数据中的少量的查询，并不是检索实验中典型的实验数据。

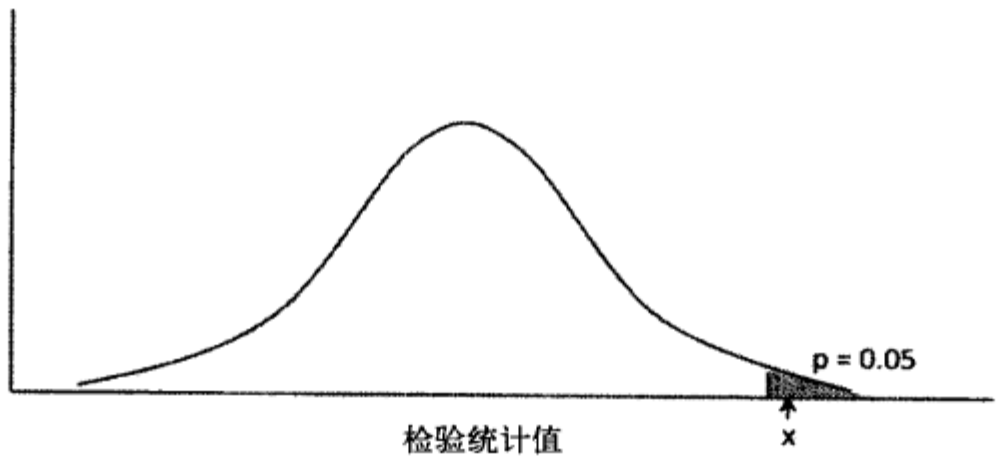


图8-8 假定零假设成立时，验证统计的可能值的分布图。阴影部分是单侧检验的否定部分

表8-6 针对10个查询，人工赋予两种检索算法（A和B）效果评价值。“B-A”列给出了两个算法之间效果评价值的不同

查询	A	B	B-A	查询	A	B	B-A
1	25	35	10	6	15	85	70
2	43	84	41	7	20	80	60
3	39	15	−24	8	52	50	−2
4	75	75	0	9	49	58	9
5	43	68	25	10	50	75	25

一般而言，*t*检验首先假定采样的数值符合正态分布。在这个匹配对实验中，这种假设指的是这两种算法的效果评价值之间的差值是正态分布的一个样例。这种情况下的零假设是指效果评价值之间的差值分布的平均值为0。两个数值之间的*t*检验的公式为：

$$t = \frac{\overline{B-A}}{\sigma_{B-A}} \cdot \sqrt{N}$$

其中，$\overline{B-A}$ 代表差值的平均值，σ_{B-A}代表标准差[⊖]（deviation），N是样例的大小（查询的数量）。对于表8-6中的数据，$\overline{B-A}$=21.4，σ_{B-A}=29.1，t=2.33。对于单侧验证，上例得到0.02的P值，这在假设α=0.05时，可以证明这两个算法的差别是显著的。因此，t检验可以否定零假设，并最终得出B比A有效的结论。

在搜索评价中使用t检验，有两点需要注意。第一点，尽管当N很大时，算法A和B效果评价值的差值的分布接近于正态分布，但是采样数据需要满足正态分布的假设对于一些效果评价方法而言，并不适合。然而，近期的实验结果却验证t检验的有效性，即在TREC数据上面，t检验产生了和随机性检验（randomization test）类似的实验结果（Smucker, Allan, & Carterette, 2007）。其中，随机性检验并没有假设数据符合正态分布，并且是非参数检验（nonparametric test[⊜]）里面最有影响力的一种检验方法。尽管如此，和t检验相比，随机性检验的计算代价很大。

⊖ 对于一组数值x_i，标准偏离值可以通过公式 $\sqrt{\sum_{i=1}^{N}(x_i-\bar{x})^2/N}$ 进行计算，其中，$\bar{x}$代表均值。

⊜ 非参数的检验比起参数检验，对数据以及潜在的数据分布考虑较少。

第二点需要注意的是，考察与效果评价方法相关的度量层次。t检验（以及随机性检验）假定待评价的数据在一定的区间尺度上进行衡量。这就意味着可以对数据进行排序（例如，效果值为54要好于效果值为53），此外，数值之间的差值也是有意义的（例如，效果值为80和70之间的差别和为20与10之间的差别是相同的）。有些人质疑效果评价是一种顺序度量表(ordinal scale)，他们认为差值的尺度衡量并不是显著的特征。威氏符号秩次检验以及符号检验都是非参数验证，对于效果评价值都不需要做什么假设。因此，他们并不使用数据中的所有信息，并且也很难去发掘两个算法的显著性区别。换句话讲，如果效果评价值确实满足使用t检验的条件，那么威氏符号秩次检验和符号检验这两种方法则具有较低的影响力。

威氏符号秩次检验假定算法A和B之间的效果评价值的差值可以被排序，但是差值的尺度并不重要。这也意味着，例如，对于表8-6中的查询8的差值将会被排为第一位，这是由于它是非0绝对值中最小的数值，但是在这种检验中，尺度2并没有被直接使用。验证公式为：

$$w = \sum_{i=1}^{N} R_i$$

其中，R_i是符号秩次，N是不为0的差值的个数。为了计算符号秩次，差值根据它们的绝对值进行升序排序，继而赋予它们排序后的位置值。其中，排序值加入了差值原始的符号。这种检验方法的零假设指的是正排序值的个数将会和负排序值的个数相同。

例如，表8-6中的9个非0的差值，将它们的绝对值进行排序，如下所示：

2, 9, 10, 24, 25, 25, 41, 60, 70

基于上面的描述，它们相应的符号秩次为：

−1, +2, +3, -4, +5.5, +5.5, +7, +8, +9

将这些有符号的排序值累加起来，得到w=35。对于单侧验证而言，将获得0.025的近似P值，这意味着零假设可以在边界设为α=0.05的显著层次（significance level）上被否定。

符号检验比威氏符号秩次验证更加深入，并且完全忽略了差值的尺度（magnitude of the differences）。对于这种验证而言，零假设是指 $P(B > A) = P(A > B) = \frac{1}{2}$。换句话说，期望通过一个很大的样例集，显示B比A“好”的数据对的数量和A比B“好”的数据对的数量是相同的。验证公式也简单地表现为B好于A的数据对的数量。对于搜索评价而言，问题在于在效果评价方法中，确定什么样的差别算是“好的差别”。可以假设，即使在平均准确率或是NDCG效果评价值中较小的差别，例如0.51与0.5进行对比，如此微小的差别也是显著的。当然也会存在某种风险，即这个差值虽然导致了算法B比A要高效得多，但事实上对于用户而言，并不是显而易见的。因此，需要为效果评价值选择一个适合的阈值。例如，旧红外法则（old rule of thumb）即为对于平均准确率而言，至少存在5%的差别才会是显著的差别（保守而言，为10%）。这将意味着0.51−0.5=0.01的差别将被认为是有符号验证的一种束缚。另一方面，如果我们选择的效果评价方法是 p@10，任何的差别都可以考虑为显著的，这是因为可以对排名前10位的新增加的相关文档做出最直接的反馈。

基于上面的描述，对于表8-6中的数据，将会考虑任何的差别都是显著的。这也意味着，10个查询中将有7个查询使用B算法比A算法好。相应的P值是0.17，也就是在10个审判中发掘7个“成功”的可能性，其中，“成功”的概率为0.5（类似于抛硬币）。使用符号验证，很难否定零假设。因为在有符号的验证中，有那么多关于效果评价的信息被丢弃，很难展示两个

算法的不同，并且需要更多的查询来增加验证的有效性。另一方面，除了t检验，符号验证可以被用来提供更多的用户关注的方面。通过t检验以及符号检验，已经可以证明出某种算法是否高效，或许使用不同的效果评价方案，结论会看起来更加合理。

8.6.2 设置参数值

几乎每一种排序算法都需要调整参数来改进最终的效果。例如，BM25算法包括参数k1、k2和b用于项加权，而使用Dirichlet平滑的查询似然度包括参数μ。网络搜索的排序算法也可能含有大量的参数，为相关的特征提供权重。这些参数值的设定对检索的最终效果具有极大的影响，对于某一应用获取最佳性能的那组参数值，不一定对另一个应用或甚至于另一个不同的数据集有效。除了提高搜索引擎性能方面，选择正确的参数意义重大，在比较两个搜索引擎算法时，也是很重要的一部分。当与一个拥有较差参数值的基线算法进行对比时，在测试集上通过调整确定参数而获取最优性能的一个算法，或许会比它的实际效果优秀得多。

为了最优化算法以及对比不同算法，最合适的设置参数的方法是，使用一个训练集以及一个测试集。训练集被用来训练最佳的参数值，测试集被用来证实这些参数的有效性以及对比各种排序算法。训练集和测试集分属两个不同的文档集、查询以及相关性判断，尽管它们有可能是由一个集合分割而成的。例如在TREC实验中，训练集通常是前几年评测时所用的文档集、查询及相关性判断。当没有足够大的数据集时，交叉检验（cross-validation）将数据集分割为k个子集。其中一个子集用来做测试集，剩余的k−1个用于做训练集。不断循环使用其中的每个子集作为测试集，最终得出的最佳参数值是k轮参数值的平均值。

使用训练集和测试集可以帮助避免过拟合（overfitting)的问题（在第7章中提及过），这种过拟合的现象表现为某一组参数值被调整得完全符合一组特殊的数据集合。如果这组特殊的数据是在应用中需要被检索的唯一的数据，那么这组参数肯定是合适的；但是更通常的一种情况是，训练语料仅是大量实际数据的一部分样例，在搜索时偶尔才会遇到。因此，过拟合将会导致参数值不能广泛地适用于其他的数据。换句话说，过拟合就是在训练语料上算法的性能改进了，而在测试语料上的性能反而变糟了。

最公平的对比两种检索算法的方法是，使用训练语料分别为这两种算法获取最佳的参数值，继而将这些参数值应用于测试集中。在训练语料的多轮检索任务中，利用效果评价方法调整参数值，而在最终进行算法对比时，可看作使用效果评价方法衡量测试集中单独的一轮检索任务的实验结果。检索实验中最忌讳的事情，就是无论在什么情况下，都要避免在训练集上进行测试。这种行为是典型的人为增强检索算法性能的方法。当一种算法在某种方式上使用测试集进行训练，那么在与其他算法对比时肯定存在问题。虽然这个问题听起来很容易避免，但是在一些复杂的实验里，总会以一些潜在的方式出现。

给定一个训练数据集，有很多技术可以为一个特定的效果评价方法找到最优的参数设置。最普通的方法是简单地使用强制法（brute-force），穷举可能的参数。这种方式需要对参数不断地进行小的变动（a parameter sweep），继而需要进行大量的检索。虽然这种方式对于所有的参数进行全部考证是不可行的，但是它对于任何给定的效果评价方法，足以确保找到最佳的参数设置。7.6节中就详细描述了为大量的候选参数有效选取最优值的一个比较成熟的过程。当需要优化的公式满足一定的条件[⊖]，通过这种方法以及一些类似的最优化技术，就会找出

⊖ 具体地，函数应该是凸函数（或者凹函数；当且仅当$-f(x)$是凸函数，$f(x)$才可能是凹函数）。一个凸函数的公式是一个满足下面的限制的连续函数：$f(\lambda x_1+(1-\lambda)x_2)\leqslant\lambda f(x_1)+(1-\lambda)f(x_2)$，对于[0, 1]中所有的$\lambda$。

最优的参数值。由于讨论的很多效果评价方法不会满足这些条件，所以需要使用一些不同的公式来求解最优化问题，当然这时很多参数值不能确保是最优的。尽管如此，这依然是一个很活跃的研究领域，人们在持续不断地尝试参数优化的新方法。

8.6.3 在线测试

到目前为止，上述所有评价方法都是在假定训练集和测试集离线处理完毕的前提下进行的。也就是说，假设所有的训练集和测试集数据已经提前准备好了。尽管如此，当使用真实的搜索引擎时，很有可能需要利用实时通信量，即在线的数据进行测试（或训练）。这通常称为“在线测试”(online testing)。例如，假设你只是开发了一个新的广告搜索算法，那么，评价该算法时与其使用人工相关评价算法，倒不如直接使用该算法，对比该算法与其他基线算法所产生的经济效益。这种利用实时通信量（在线数据）以及真实用户行为的方法，使得测试不同的搜索引擎模块（如测试排序算法、查询推荐算法、网页摘要生成算法等）成为可能。这非常类似于我们前面讲到的日志。然而，使用日志的评价是对旧的信息的一种回顾式的评价，而在线测试却是使用了实时的数据。

在线测试有很多优点。首先，它允许真实的用户参与到系统中。这些用户参与提供了一些很重要的信息，如用户点击数据等都可以被用来对系统进行各种评价。其次，在线测试并不片面，这是因为这种评价方法是在真实的用户数据集上进行的。由于那些能够反映真实用户数据和通信量的测试集很难被获取，因此这些在线的数据集是非常有价值的。最后，由于在线测试不需要任何人员去进行人工相关检验，因此可以很廉价地产生大量的数据集。

但遗憾的是，在线测试也存在一些缺陷。其中，主要的缺陷在于采集的数据噪声比较大。这是因为导致用户在线行为的原因有很多种。例如，如果一个用户没有点击某个搜索结果，这并不意味着该搜索结果是不相关的。有时候，用户反而点击了一个不相关的广告链接，或者对点击的结果感觉没什么兴趣，或者离开去吃饭了。基于此，在线测试搜集的数据需要经过去除噪声，进而才能产生有意义的结论。在线测试的另一个缺陷是，它将使得实时通信量以一种不利的方式进行潜在的改变。如果换用一种新的算法进行在线测试，则有可能大幅度地降低检索的效果，从而导致用户的离开。基于这种原因，在线测试必须谨慎执行，以免消极地影响用户体验。一种降低在线测试对用户数量的影响的方法是，仅在一小部分数据上进行测试，如实时通信量的1%或5%的数据量。最后，在线测试仅是提供了一种特殊类型的数据——用户点击数据。像我们在本章前面所提到的，点击数据对于评价搜索引擎的性能总是不那么理想，这是因为数据总是含有噪声而且具有片面性。尽管如此，对于某一特定的搜索引擎评价算法，如点击率[⊖]和收益情况，在线测试还是非常有用的。

因此，在线测试是一种有用的、廉价的训练或测试新算法的评价方法，尤其对于那些可以使用点击数据进行评价的算法。但是，需要特别注意，以确保采集的数据被正确地分析以及不影响整个系统的用户体验。

8.7 基本要点

在本章中，我们展示了一系列的效果和效率评价方法。那么，它们中的哪一个是最好的评价方法？答案是，没有哪个方法对于任意的搜索应用是万能的，尤其是对于效果评价而言。

⊖ 某一词项被点击的次数在总点击次数中的比例。

因此，一个搜索引擎应该通过多尝试一些评价方法，将它们的组合作为最终的评价。这样可以全面地对搜索引擎的不同方面进行评价。下面所有的评价方法和验证方法几乎都不需要额外的条件，可以直接执行：

- MAP——使用一个数字进行评价，是一种很受欢迎的方法，基于检索池的相关性判断。
- NDCG平均值——基于排序的位置给出一个数字进行评价，强调排序靠前的文档，仅仅在一个特定的排序个数内（通常指的是10）才需要进行相关性判断。
- 召回率-准确率折线图——比单一的数字的评价方法传递了更多的信息，使用检索池的相关性判断。
- P@10——强调排序靠前的文档，很容易理解，相关性判断仅限于排序的前10个。

使用MAP和召回率-准确率折线图方法，需要在相关性判断方面得到更多的协助，而从上面的分析中我们获知，NDCG评价和P@10评价只有在考察排序位于前10位的相关文档时，才会考虑使用相关性判断。

所有的这些评价应该相对于一个或更多的基线搜索算法，而被执行。执行一个效果评价方案，而没有一个好的基线算法，这一般是说不通的。这是因为最终得到的效果值强烈依赖于查询和测试集数据的组合。t检验可以用于平均准确率、NDCG以及P@10的显著性检验。

所有的标准评价方法和显著性检验都可以使用NIST提供的程序trec_eval。它们也作为Galago的一部分被提供。

除了这些评价方法，通过与基线系统的对比，将改进的以及改错的查询的数量进行总结，也是很有用的一件事情。图8-9给出了TREC任务的一次评测后的总结，对于某种评价方法（通常是MAP），其中随着改进的不同的百分比，查询的数量呈现出如图的分布。图中的每个柱状代表了在给定当前的百分比下，好于（或坏于）基线系统的查询的数量。这也提供了一个简单的可视的总结图，展示了改进的查询要多于改错的查询，并且这些改进是很可观的。通过对“较为可观的”的改进设定阈值，符号检验可以对这些数据进行显著性检验。

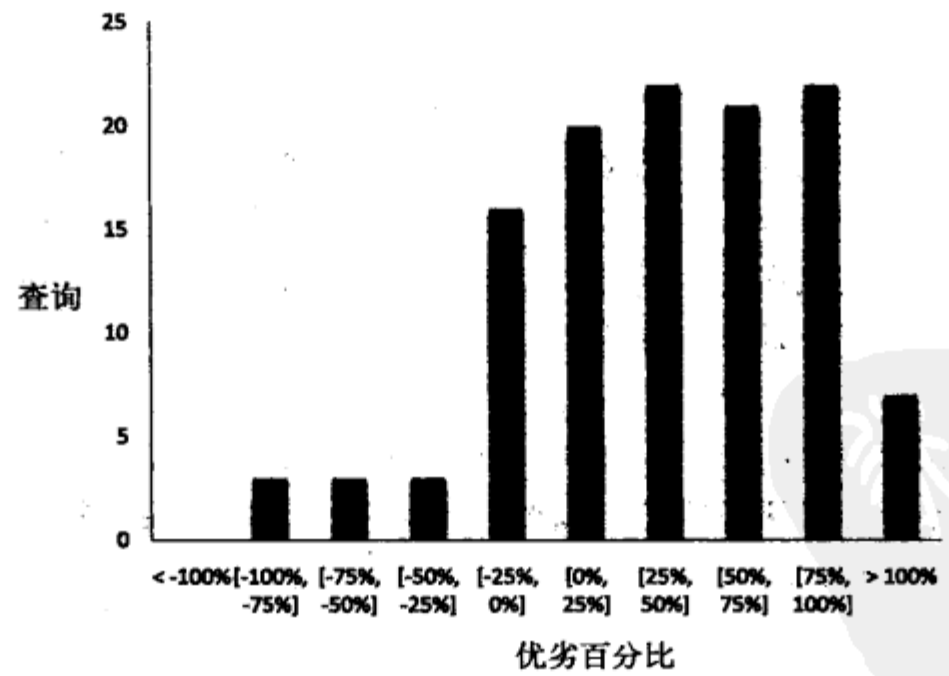


图8-9 查询效果改进的样例分布

通过上述介绍的一系列评价方法，开发者和用户都将对搜索引擎的评价有了更清晰的认识，如该搜索引擎哪些方面表现得好，哪些方面需要进一步改进等。有时候很有必要去查看个别的查询，以便更好地了解是什么原因引发了该算法的这种排序结果。像图8-9的查询数据，

有助于识别出这些有意义的查询。

参考文献和深入阅读

虽然这个问题已经探讨了40多年了，但搜索引擎中的效果评价依旧是一个热点话题，每年的主要会议中都有大量的文章发表。在van Rijsbergen（1979）的文章中，关于评价的那一章节给出了信息检索中效果评价方法的历史发展流程。另一个通常认为比较有价值的参考文献是TREC的书籍（Voorhees&Harman，2005），该书描述了TREC的测试集以及所使用的评价流程，以及它们经历了哪些演变。

Saracevic（1975）和Mizzaro（1997）所写的这两篇文章，是关于经典的话题相关性的最佳的综述性文章。获取相关性判断的过程以及检索实验的可靠的方法，都在TREC那本书中有所讨论。 Zobel（1998）的研究显示，一些相关性判断的不完整性并没有影响实验，虽然Buckley和Voorhees（2004）指出，如果存在大量的不完整性会是一个问题。Voorhees和Buckley（2002）讨论了使用不同数量的查询的相关错误率。Sanderson和Zobel（2005）的研究展示了怎样使用一个显著性检验可以影响对比实验的可靠性，同时还比较了浅层和深层的相关性判断。Carterette等人（2006）研究了一项技术，用来减少用于搜索引擎可靠性对比所需要的相关性判断的数量。Kelly和Teevan（2003）回顾了一些获取和使用潜在的相关性信息的方法。Fox等人（2005）研究了在网页检索的上下文中的一些潜在的相关性判断方法，Joachims等人（2005）介绍了获得基于点击数据的用户偏好的策略。Agichtein, Brill, Dumais和Ragno（2006）扩展了这种方案，并且完成了更多的实验来介绍点击数据分布和偏差，同时证实了大量的用户行为相关的特征有助于预测文档相关性。

F值的评价方案由van Rijsbergen（1979）首次以E=1−F的形式提出。就评价理论而言，他还为E评价提供了一种合理的理由，同时提出了效果评价方案应该是间隔的还是有序的疑问，以及指出符号检验和威氏符号秩次检验是合理的显著性检验方法。Cooper（1968）早期写了一篇重要的文章，介绍了expected search length（ESL)方法，其中ESL指的是用户期望看到的相关文档的具体数量。虽然这种方法没有被广泛应用，但是它仍然是那些重视排序位置靠前文档的评价方法如NDCG（Järvelin & Kekäläinen, 2002）方法的鼻祖。这种类型的另一种方法，最近刚刚被引入，叫做rank-biased precision（Moffat, Webber, & Zobel, 2007）。

Yao（1995）提供了用户偏好的首次讨论，并研究了怎样使用它们才能更好地评价搜索引擎。Joachims（2002b）提供的论文展示了如何使用用户偏好训练一个基于特征的线性检索模型，同时使用了Kendall's τ作为效果评价方法来获取最佳排序。由Carterette和Jones（2007）最近撰写的论文，展示了如何直接使用来自点击数据的相关性信息来进行搜索引擎的评价，而不是先将其转换为用户偏好。

与效果评价方案相关的一个很有趣的话题是查询效果的预测。Cronen-Townsend等人（2006）的文章描述了清晰度（Clarity）测量方法，即预测一个查询的返回文档排序列表是否具有较好的准确率。其他的评测方法被证实与平均准确率有甚至更好的相关性。

很少有文章讨论搜索引擎的效率评价方法。Zobel等人（1996）是文献中的一个例子。

练习

8.1 在信息检索领域的文献中，找出其他3个测试集的样例（TREC除外）。请描述它们，并用

表的形式对比它们的统计数据。

8.2 假设你想了解博客搜索引擎的效果。请详细设计这项应用实例的几项检索步骤，继而描述你想要构建的测试集，以及你怎样评价排序算法。

8.3 对于CACM数据集中的一个查询（在本书网站上提供），使用Galago生成一个排序，请手工计算平均准确率、NDCG@5、NDCG@10、P@10以及排序倒数。

8.4 对于CACM数据集中的两个查询，请生成两个没有进行插值的召回率-准确率折线图，以及在标准召回率等级上使用插值法构建准确率值表格，以及平均插值召回率-准确率折线图。

8.5 考察整个CACM查询集合，请计算平均准确率、召回率-准确率折线图、平均NDCG@5以及NDCG@10、P@10。

8.6 根据图8-3的实例，对比MAP与GMAP评价方法。请指出哪个查询对GMAP值的影响较大。

8.7 在很多评价体系中有一种评价方法R-precision经常被使用。该方法定义为R个文档中的准确率，其中R是某一查询相关文档的数量。该方法被用于不同的查询对应的相关文档数量变动非常大的情况。请为CACM查询集计算平均的R-precision，并与其他的评价方法进行对比。

8.8 假设CACM中的10个查询为简单查询，对它们手工增加查询的结构生成复杂查询。对比简单查询与复杂查询的效果评价值MAP、NDCG以及P@10值。分别使用t检验、威氏符号秩次检验以及符号检验考察它们的显著性。

8.9 对于CACM数据集中的一个查询，手工构建一个排序，计算BPREF值，并证明BPREF的两个公式将给出同样的值。

8.10 考察一个测试集，该测试集可以对大量的时间敏感查询，如“奥林匹克”或是“环球小姐”作出相关性判断。假设这些相关性判断是在2002年制定的。这会不会对搜索引擎构成一个问题？如何使用在线测试来避免这个问题？

第9章　分类和聚类

“这是什么东西呢？我需要一个明确定义。”

——Ripley，《异形》

现在我们暂时抛开搜索，来看看分类和聚类。分类和聚类在文档检索方面有很多共同之处。事实上，许多对文档相关排序有用的技术也用于分类和聚类，分类和聚类算法在最新的搜索引擎中也有重要的应用，因此，需要了解这些技术是如何工作的，以及如何解决真实世界的问题。这里主要提供一般的背景知识和普遍的看法。另外，会给出这些方法在实际中工作的例子。我们没有过深地陷入细节的描述和理论中，因为有许多其他优秀的参考书已在这些方面做了描述，在本章最后的“参考文献和深入阅读”中有所提及。在这一章结束的时候，你应该知道什么是分类和聚类，包括最常用的算法，它们在实际工作中的实例以及如何评价它们。首先我们给出一个简短的对分类和聚类的描述。

分类（classification）是指自动对数据（例如电子邮件、网页或者图像）进行标注。人们通过日常生活经验划分类别。但是，要依据一些原则手工地对互联网上的每一篇网页进行分类，如“垃圾信息”或“非垃圾信息”，是不可能的，因此迫切需要自动分类技术。本章将描述具有较广泛用途的几种分类算法，包括垃圾信息检测、情感分析以及应用语义对互联网广告分类。

聚类（clustering）是本章讨论的另一个话题，广义的定义是将相关的项目聚到一起。在分类中，每一项被划分到一个类别，例如“垃圾信息”或“非垃圾信息”。在聚类中，每一项被划分到一个或多个类别中，每个类别不一定对应一个有意义的概念。在本章后面会看到，项是根据相似度聚合到一起的。因此，聚类不是将项匹配到预先定义的标签集合，而是允许数据通过揭示与项相关的隐含结构“自己说话”。

分类和聚类是信息检索研究者多年来研究的课题，一方面以搜索的应用为目的来提高有效性或某些情况下的效率。另一方面，这两个任务是经典的机器学习问题。在机器学习领域，分别称为有监督和无监督的学习方法。有监督的学习（supervised learning）用充足的标注好的数据，通常称为训练集（training set），对模型进行学习。一旦模型学习完成，就可以应用于未标注的数据。分类常常看作是有监督的学习问题。例如，给定一个电子邮件的集合，分别标注为“垃圾邮件”和“非垃圾邮件”（训练集），分类模型可以学习，然后模型可以应用于新的邮件，将其划分为“垃圾邮件”或“非垃圾邮件”。

无监督学习（unsupervised learning）算法完全基于无标注的数据学习。由于输入数据不能映射到一个预先定义的类别集合，因此无监督学习和有监督的学习相比，会有一些不同。聚类是最常见的无监督学习的一个例子，聚类算法将无标注的数据作为输入，然后根据一些相似度的概念将这些项目进行聚合。还有许多其他类型超越了有监督和无监督学习的范例，例如半监督学习（semi-supervised learning），主动学习（active learning）以及在线学习（online learning）。但是这些内容超出了本书的范围，本章主要给出基本的分类和聚类算法以及评价方法。

9.1 分类

对我们所看到的事物进行分类是一件很自然的事情，在日常生活中，我们常常会无意识地这样做，例如，去食品店，会不自觉地将看到的食品分为“熟的”或“生的”，“健康的”或“不健康的”，“便宜的”或“昂贵的”。

这些都是二元分类的例子，因为它们只有两种情况。还可以对食品进行多元分类，例如可以划分为“淀粉类”、“肉类”、“蔬菜类”或者“水果类”。另一种分类体系是类别间具有层次性，例如“蔬菜类”可以按照颜色分为“绿色”、“红色”和“黄色”等子类。在这个体系下，食品将被分到类别层次中的某个位置。这个不同的分类及分类体系，包括二元、多值及层次关系，称为本体（ontologies）（见第6章）。

为基础任务选择合适的本体是重要的，例如检测是否为垃圾邮件，最好选定包含“垃圾”和“非垃圾”的标注集。如果有人要设计一个分类器以检测某个网页是用哪一种语言写的，那么所有可能语言的集合就是比较合适的本体。一般情况下，本体的正确选择是根据问题而定的。如果不是，则选择明确有益于基础任务标注的集合。分类是一个有监督的学习任务，不要构造过于复杂的本体，因为当很少或没有数据与类别中的一个或多个关联时，多数的学习算法会失效（即对未见到的数据不具有很好的推广性）。以网页语言分类为例，如果每一个亚洲语言只有一个样例，那么合并各种语言为一个类别，建立一个标签“亚洲语言”，胜过为每一个语言建立一个独立的标签，例如“中文”、“韩文”等，因为如果有更多的训练样例，分类器会将“亚洲语言”分得更正确。

为了了解机器学习算法的工作原理，我们先来了解一下人们是如何进行分类的。回顾食品店的例子，考虑是如何将食品分为“健康的”和“不健康的”。为了进行这个分类，需要考察饱和脂肪、胆固醇、糖和钠在食物中的数量。如果这些值单独或者合起来超出某些阈值，就可以将食物标为“健康的”或者“不健康的”。总的来说，人们在进行分类的时候，首先根据一些能帮助区分各种情况的重要特征对项目进行类别划分，从每个项目中抽取这些特征，然后按照某种方式合并证据（combine evidence），最后，用基于合并的决策机制对项分类。

在上面的例子中，特征是饱和脂肪和胆固醇的数量。这些特征抽取可以通过打印在食品包装上的信息表或者是通过实验室测试获得，通过各种方法合并这些依据以便衡量健康指数（计为H），简单的方法是衡量每一个特征的重要性，然后将各个带权重的特征值相加。例如：

$$H(food) \approx w_{fat} fat(food) + w_{chol} chol(food) + w_{sugar} sugar(food) + w_{sodium} sodium(food)$$

这里w_{fat}、w_{chol}等参数是与每个特征值相关的权重。当然，这种情况下每个特征值是负的。

一旦对一个给定的食物有了健康的得分H，就必须应用决策机制给这种食物标注是健康的还是不健康的。有很多方法可以采用，最简单的方法是用简单的阈值规则，也就是“如果$H(food) \geqslant t$，食物就是健康的”。

虽然这是一个人们如何进行分类的理想化模型，但它提供了一个洞察计算机如何进行分类的方法。实际上，下面描述的两种分类算法跟前面描述的步骤一致，两个算法唯一不同的是每一步实现操作的细节。

9.1.1 朴素贝叶斯

现在来描述项目是如何被自动分类的。最直接有效的分类技术称为朴素贝叶斯（Naïve Bayes）。在第7章将贝叶斯分类器作为检索模型的框架做了介绍。在那种情况下，只有两种类别，即相关（relevant）和不相关（non-relevant）。一般来讲，分类任务是多于两类的，基于贝叶斯分类器的贝叶斯规则（Bayes'Rule）描述如下：

$$P(C|D)=\frac{P(D|C)P(C)}{P(D)}$$
$$=\frac{P(D|C)P(C)}{\sum_{c\in\mathbb{C}}P(D|C=c)P(C=c)}$$

其中C和D为随机变量（random variable）。我们时常用随机变量模拟不确定的事件，这样的变量没有固定的（确定的）值，或者说，变量值是随机的。每一个随机变量都有一个与它相关的可能结果集，也就是结果的概率分布。例如，掷硬币结果就可以模拟为随机变量X，这个随机变量的结果可能是“正面（h）”和“反面（t）”。一个公平的硬币，正面和反面的输出概率各为0.5。因此，$P(X=h)=P(X=t)=0.5$。

考虑另一个例子，代数表达式$Y=10+2X$，如果X是确定的变量，则Y也是确定的。也就是说，对于给定的X，Y总是得到同一个值。但是如果X是随机变量，则Y也是随机变量。假设X可能输出−1（概率0.1），0（概率0.25），1（概率0.65）。Y的可能输出是8，10和12，则输出的概率为：$P(Y=8)=0.1$，$P(Y=10)=0.25$，$P(Y=12)=0.65$。

本章用大写字母表示随机变量（例如C，D），随机变量的结果用小写字母（例如c，d）表示，整个结果集用花体（例如C，D）表示。最后，为了表达方便，$P(X=x)$用$P(x)$代替。同样对于条件概率，用$P(x|y)$代替$P(X=x|Y=y)$。

贝叶斯规则是重要的，因为允许根据“相反”的条件（$P(D|C)$）写条件概率（例如$P(C|D)$）。这是非常强大的定理，因为非常容易估计和计算一个方向而不是另一个方向的条件概率。例如，来看垃圾邮件的分类，D代表文档，C代表类别（例如“垃圾邮件”或者“非垃圾邮件”）。虽然不能马上清楚计算机是如何写程序（程序由$P(C|D)$表示）计算文档是否是垃圾邮件，但是很容易发现文档是否是垃圾邮件的样例，有可能得出在给定样例或训练集情况下的$P(D|C)$的估计。贝叶斯规则的神奇在于，它告诉我们如何得到所要的$P(C|D)$，即如果不能立刻知道如何估计$P(C|D)$，则可以估计比较容易得到的$P(D|C)$，然后运用贝叶斯规则得到$P(C|D)$。

如果在垃圾邮件的例子中令C为类别的随机变量，D为文档的随机变量，可以直接使用该规则分类。给定文档d[⊖]（随机变量D的一个输出）以及类别集合$C=c_1,\cdots,c_N$（随机变量C的输出），可以用贝叶斯规则计算文档d属于每一个类别c_i的可能性$P(c_1|d),\cdots,P(c_N|d)$，然后可以将文档d标注为概率最大的那一类。对文档d的贝叶斯分类如下：

$$\text{Class}(d)=\arg\max_{c\in\mathbb{C}}P(c|d)$$
$$=\arg\max_{c\in\mathbb{C}}\frac{P(d|c)P(c)}{\sum_{c\in\mathbb{C}}P(d|c)P(c)}$$

⊖ 贯穿本章大部分，我们都假设分类的对象是文本文档。但是，此处描述的方法适用于更加普遍的情况，可以用于非文本对象，如图像和视频。了解这一点很重要。

这里 $\max_{c\in C} P(c|d)$ 的意思是："返回类别集合C中$P(c|d)$最大值对应的类别c"，这是对于给定的文档d确定其所属类别的数学表述式。

不需直接计算$P(c|d)$，通过计算$P(d|c)$和$P(c)$，然后应用贝叶斯公式可以获得$P(c|d)$。正如在前面所说，应用贝叶斯的一个原因是，由于它比较容易估计一个条件概率，而不是另一个。现在来解释在实际中如何估计这些值。

首先描述如何估计类别的先验概率$P(c)$。这个估计很简单，如下所示：

$$P(c)=\frac{N_c}{N}$$

这里N_c表示训练样例中已经标注为c的样例，N是训练样例总数。因此，$P(c)$表示训练样例中类别c的比例。

估计$P(d|c)$有点复杂，因为用与计算$P(c)$相同的方法是不可行的（为什么？见练习9.3）。为了使这个估计可以实施，必须进行简化，假设文档d可以表示为$d=w_1,\cdots,w_n$，并且对于$i\neq j$，其中w_i和w_j之间是互相独立的。简单地说，文档d可以归结为元素（词项）的集合。而且假设元素（词项）之间彼此是独立的[⊖]。这个假设是称其为朴素（naïve）的原因之一，因为将文档的表示简单化了。实际上，词项之间并不是独立的。可是，像在第11章中描述的那样，适当地模拟词项间的依赖性是可能的，但是很复杂。尽管独立性是一种假设，但是朴素贝叶斯还是对各种分类任务显示出了较强的鲁棒性和高效性。

这个独立性假设允许将一个独立随机变量集合的联合概率写为单独的条件概率的乘积，即$P(d|c)$可以写为：

$$P(d|c)=\prod_{i=1}^{n}P(w_i|c)$$

因此，需要估计在词表$\mathcal{V}$中的每一个词在每个类c的概率$P(w_i|c)$。得到这个结果要比直接估计$P(d|c)$容易得多，因为词表和类别都是有限的，而文本也许是无限的。独立性假设可以将$P(c|d)$写为：

$$P(c|d)=\frac{P(d|c)P(c)}{\sum_{c\in C}P(d|c)P(c)}=\frac{\prod_{i=1}^{\mathcal{V}}P(w_i|c)P(c)}{\sum_{c\in C}\prod_{i=1}^{\mathcal{V}}P(w_i|c)P(c)}$$

现在剩下的事情就是描述如何估计$P(w|c)$。在估计概率之前，要明确概率的真实意义。例如$P(w|c)$可以解释为"w与类别c相关的概率"、"w与类别c无关的概率"或者其他任意事情。为了使意义更明确，必须明确概率定义的事件空间。事件空间（event space）是指来自某个过程的所有可能事件（或结果）的集合。事件空间中的每一个事件被赋予一定的概率，所有事件概率的总和等于1。

概率估计及结果的分类非常依赖于事件空间的选择，在此将简要描述比较常见的两种事件空间，以及说明$P(w|c)$在每一种事件空间中是如何估计的。

⊖ 这是和第7章描述的大多数检索模型核心一样的假设。也等价于在第11章讨论的词袋假设。

1. 多重伯努利模型

描述的第一个事件空间非常简单。给定一个类c，为在词表中的每一个词项定义一个二值随机变量w_i。二值事件的输出是0或1。概率$P(w_i=1|c)$可以解释为“类别c生成w_i的概率”。相反，$P(w_i=0|c)$可以解释为“类别c不生成w_i的概率”。这是精确的采用二值独立模型（见第7章）的事件空间，也就是称为多重伯努利（multiple-Bernoulli）的事件空间。

在这个事件空间下，在一些类别c中的每一个词，估计这个词在每一个类别中的生成概率。例如，在垃圾信息分类中，$P(cheap=1|spam)$有一个很高的概率，但是$P(dinner=1|spam)$将有一个非常低的概率。

图9-1显示在事件空间中训练文本集合的表达方式。在这个例子中，有10个文本(每一个有唯一的id)，两个类别（垃圾信息、非垃圾信息），词表包括“cheap”、“buy”、“banking”、“dinner”和“the”。在这个例子里，$P(spam)=3/10$，$P(not\ spam)=7/10$。接下来，为每一对词和类别估计$P(w|c)$。最简单的方法是用最大似然估计来估计概率。即

$$P(w|c)=\frac{df_{w,c}}{N_c}$$

这里$df_{w,c}$是在类别c中包含w的文本数量，N_c是训练样本中类别c的文本数量。正如我们看到的，最大似然估计仅仅是类别c中包含词w的文本所占的比例。用最大似然估计，可以容易地计算$P(the|spam)=1$，$P(the|not\ spam)=1$，$P(dinner|spam)=0$，$P(dinner|not\ spam)=1/7$，等等。

文档id	cheap	buy	banking	dinner	the	类别
1	0	0	0	0	1	非垃圾信息
2	1	0	1	0	1	垃圾信息
3	0	0	0	0	1	非垃圾信息
4	1	0	1	0	1	垃圾信息
5	1	1	0	0	1	垃圾信息
6	0	0	1	0	1	非垃圾信息
7	0	1	1	0	1	非垃圾信息
8	0	0	0	0	1	非垃圾信息
9	0	0	0	0	1	非垃圾信息
10	1	1	0	1	1	非垃圾信息

图9-1 多重伯努利事件空间中文档的表示说明。在这个例子中，有10个文本（每一个有唯一的id），两个类别（垃圾信息、非垃圾信息），词表包括“cheap”、“buy”、“banking”、“dinner”和“the”

采用多重伯努利模型，文档的似然估计$P(d|c)$可以写为：

$$P(d|c)=\prod_{w\in V}P(w|c)^{\delta(w,d)}(1-P(w|c))^{1-\delta(w,d)}$$

这里$\delta(w,D)$为1，当且仅当w在文档d中。

实际上，由于零概率问题（zero probability problem）存在，应用最大似然估计是不可能的。为了说明零概率问题，回顾图9-1中的垃圾信息分类样例。假设收到的某个邮件中包含“dinner”，不论邮件中是否包含其他的词，由于$P(dinner|spam)=0$，则$P(d|c)=0$，并且词在文本

中出现（也就是说$\delta_{\text{dinner},d}=1$）。所以，任何包含“dinner”的文本成为垃圾邮件的可能性都为零。这个问题是比较普遍的，由于文本中包含一个词从来没有在一个或更多的类中出现，零概率就会发生。问题是，最大似然估计是基于训练集中的计数的结果（occurrence）。但是，训练集是有限的，不是每一个可能结果都会发生，这就是数据稀疏。数据稀疏在小规模的训练集中非常常见。在相对大规模的数据集中也会出现。因此，必须对所有词改变这种估计，包括在一个给定类别中没有出现的词，给出一些概率，来保证在V中的所有词的概率$P(w|c)$不为零。这样，可以避免与零概率相关的所有问题。

如在第7章描述的那样，平滑是克服零概率问题的有效方法。一个常见的平滑技术是贝叶斯平滑（Bayesian smoothing），针对模型假定一些先验概率，用最大后验概率（maximum a posteriori）估计。针对多重伯努利模型，平滑估计有如下公式：

$$P(w|c)=\frac{df_{w,c}+\alpha_w}{N_c+\alpha_w+\beta_w}$$

这里α_w、β_w是依赖于w的参数。参数的不同设置将导致不同的结果。一个常见的选择是对所有的w设置$\alpha_w=1$且$\beta_w=0$，得到如下的估计：

$$P(w|c)=\frac{df_{w,c}+1}{N_c+1}$$

另一个选择是对所有的w设置$\alpha_w=\mu\frac{N_w}{N}$和$\beta_w=\mu\left(1-\frac{N_w}{N}\right)$，这里$N_w$是训练文本中包含$w$的所有文本数，$\mu$是可调参数。有如下估计：

$$P(w|c)=\frac{df_{w,c}+\mu\frac{N_w}{N}}{N_c+\mu}$$

这个事件空间仅仅考虑是否有词出现，但是没有考虑这个词出现多少次，而次数也是一个重要信息。下面将描述一个考虑词频的事件空间。

2. 多项式模型

多重伯努利模型的二值事件空间过于简单，不能很好地模拟文本中词出现的次数。实际已经表明，词频对检索和分类都是重要的特征，尤其是用于长文本时更加明显。针对比较短的文本，大多数词的出现不会超过一次，这时多重伯努利模型是一个正确的模型。但是实际的集合中包含的文本经常是有长有短，因此考虑词频及文本长度是很重要的。

多项式（multinomial）事件空间与多重伯努利事件空间相似，除了多重伯努利模型假定词的频率是二值的（“词出现”、“词不出现”），多项式模型假定词的出现是零或者多次（“词出现0次”、“词出现1次”，等等）。

图9-2展示了垃圾信息分类样例的文本在多项式事件空间的表示。这种表示和多重伯努利模型唯一不同的是事件不是二值的。对于多项式模型，最大似然估计与伯努利模型相似。计算如下：

$$P(w|c)=\frac{tf_{w,c}}{|c|}$$

这里$tf_{w,c}$是训练集中类别c中w出现的次数。$|c|$表示训练集中类别c中词的总数。在垃圾信息分

类样例中，$P(the|spam)$=4/20，P(the|not spam)=9/15，P(dinner|spam)=0，P(dinner|not spam)=1/15。

文档id	cheap	buy	banking	dinner	the	类别
1	0	0	0	0	2	非垃圾信息
2	3	0	1	0	1	垃圾信息
3	0	0	0	0	1	非垃圾信息
4	2	0	3	0	2	垃圾信息
5	5	2	0	0	1	垃圾信息
6	0	0	1	0	1	非垃圾信息
7	0	1	1	0	1	非垃圾信息
8	0	0	0	0	1	非垃圾信息
9	0	0	0	0	1	非垃圾信息
10	1	1	0	1	2	非垃圾信息

图9-2 在多项式事件空间中文档的表示说明。在这个例子中，有10个文本（每一个有唯一的id，两个类别（垃圾及非垃圾），词表包括“cheap”、“buy”、“banking”、“dinner”和“the”

由于词是根据多项式分布来描述的，对于给定类别c，文本的最大似然估计计算如下：

$$P(d|c)=P(|d|)(tf_{w_1,d},\ tf_{w_2,d},\cdots,tf_{w_V,d})!\prod_{w\in V}P(w|c)^{tf_{w,d}}$$

$$\propto\prod_{w\in V}P(w|c)^{tf_{w,d}}$$

这里$tf_{w,d}$是文档d中w的出现次数，$|d|$是文档d中的总词数，$P(|d|)$是长度为$|d|$的文本的生成概率，$(tf_{w_1,d}, tf_{w_2,d}, \cdots, tf_{w_V,d})$是多项式系数[⊖]。注意到$P(|d|)$及多项式系数是依赖于文档的，对于分类，可以忽略。

词似然的贝叶斯平滑估计如下：

$$P(w|c)=\frac{tf_{w,c}+\alpha_w}{|c|+\sum_{w\in V}a_w}$$

这里α_w是依赖于w的参数，对于多重伯努利模型，不同平滑参数的设定导致不同的估计类型。对所有的w设置α_w=1，是其中一种选择。这个结果导致如下估计：

$$P(w|c)=\frac{tf_{w,c}+1}{|c|+|V|}$$

另一种流行的选择是设置 $\alpha_w=\mu\frac{cf_w}{|C|}$，这里$cf_w$是在训练集中$w$出现的次数，$|C|$是训练集中词的总数，$\mu$像前面一样，是一个可调参数，在这种设置下，得到如下的估计：

$$P(w|c)=\frac{tf_{w,c}+\mu\frac{cf_w}{|C|}}{|c|+\mu}$$

⊖ 多项式系数是二项式系数的泛化，这样，计算 $(N_1, N_2, \cdots, N_k)!=\frac{N!}{N_1!N_2!\cdots N_k!}$，用来计算 $\sum_i N_i$ 项（词）能被排列的每一种不同形式的总数，每一个词项i可以出现N_i次。

这个估计看起来很熟悉，实际上就是第7章描述的Dirichlet平滑语言模型估计。

实际中，多项式模型已经显示出一向优于多重伯努利模型。基于这两种模型的任意一种分类器实现都很简单，训练包括计算简单的词频统计。在多数情况下，这些统计可以保存在内存中，可以提高分类的效率。模型的简洁性以及较好的准确率，使朴素贝叶斯分类器成为一般的分类算法中较好的选择。

9.1.2 支持向量机

与完全基于概率论原理的朴素贝叶斯分类不同，下一个分类器基于几何学原理。支持向量机（Support Vector Machine），常称为SVM，将输入（如文本）看作几何空间中的一个点。为简单起见，首先描述如何将SVM应用于二值分类，就是常说的正例和反例。在这个框架下，SVM的目标是发现能分割正例和反例的超平面（hyperplane）[⊖]。

在朴素贝叶斯模型中，文本在多重伯努利模型中被看作二值向量，在多项式模型中被看作词频向量，SVM根据文档的表示提供更好的适应性。对于SVM，不是定义基础的事件空间，而是定义以文本作为输入并生成称为特征值（feature value）的特征函数（feature function）集合$f_1(.), \cdots, f_N(.)$。设给定文档d，文档由向量$x_d=[f_1(d), \cdots, f_N(d)]$表示为$N$维空间。根据给定的训练数据集，用特征函数在$N$维空间中表示文本。不同的特征函数导致不同的表示结果。由于SVM根据类别发现分离数据的超平面，因此选择有助于分离不同类别的特征函数非常重要。

两个常用的特征函数是$f_w(d)=\delta(w, d)$和$f_w(d)=tf_{w,d}$。如果词w出现在文本中，则第一个特征函数是1，可以类比于多重伯努利模型。第二个特征函数是计算w在d中出现的次数，可类比于多项式模型。注意到这些特征函数被w索引，这意味着共有$|V|$个这样的函数。这将导致文本被映射到$|V|$维空间，也有可能在bigram或者trigram上定义相似的特征函数，这将引起特征维度空间爆炸。

进一步地，其他信息可以被编码到特征函数里，例如文档长度、文档中句子数、文档最后更新时间等。

既然有了一个N维空间的表示文本的机制，我们来描述SVM如何在这个空间中进行分类，像在前面描述中提到的，SVM的目标是发现一个分离正例和反例的超平面。这个超平面从训练集中学习获得。根据测试点，落在超平面的哪一侧确定其类别。例如，如果这个点落在反例的一面，则将其判为反例，同样，如果落在正例的一面，就将其判为正例。但是，并不是总能找到完全分离正例和反例的超平面。因为对于一些互相嵌套的正例和反例，就不存在这样的超平面。例如，在图9-3中，对于左边的平面，可以找到一条线（平面）分离正例（“+”）和反例（“−”），但是在右面的平面就不可能这样做。在左面的点称为线性可分（linearly separable），所以可以画一个线性平面分离这些点。

当数据是线性可分的时候，很容易定义并找到一个好的超平面。因此，我们开始讨论在这种特殊情况下SVM是如何工作的，然后再扩展到更一般的情况，讨论数据点不是线性可分的情况。

1. 情况1：线性可分数据

假设给你一个线性可分的数据集，如图9-3所示，要求找到分离这些数据的超平面。该如何处理？可能我们首先要问怎样算最优？一个首要原则是，最优意味着能够分离训练集中的

⊖ 超平面即为N维空间平面。

正例和反例的任意（any）超平面。必须考虑到任何分类算法最根本的目标，就是对未见数据的推广性（generalize）。如果一个分类器对训练数据有很好的分离，但是不能对测试集中的数据分离，它的价值就很小，这种情况称为过拟合（overfitting）。

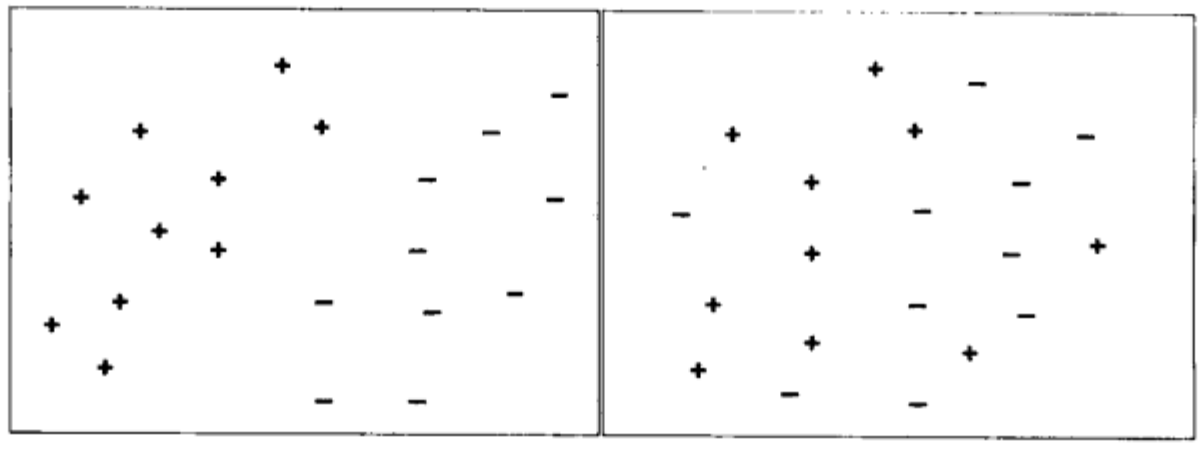


图9-3 包含两种类别的数据集（加号和减号）。左边的数据集是线性可分，右边的数据集是线性不可分

为了避免过拟合，SVM选择与正例和反例的数据点最大化距离的超平面。这个选择很直观，也得到了理论的支持。假设的超平面定义为w，要发现能够分离训练集中的正例和反例并且最大化分离数据点的w。最大化分离定义如下，假设x^-是训练集中与超平面最近的反例，x^+是训练集中与超平面最近的正例[⊖]，定义边缘（margin）为x^-到超平面加上x^+到超平面的距离。图9-4给出了边缘、超平面以及支持向量的图形解释。边缘可以计算如下：

$$\text{Margin}(w)=\frac{|w\cdot x^-|+(w\cdot x^+)}{\|w\|}$$

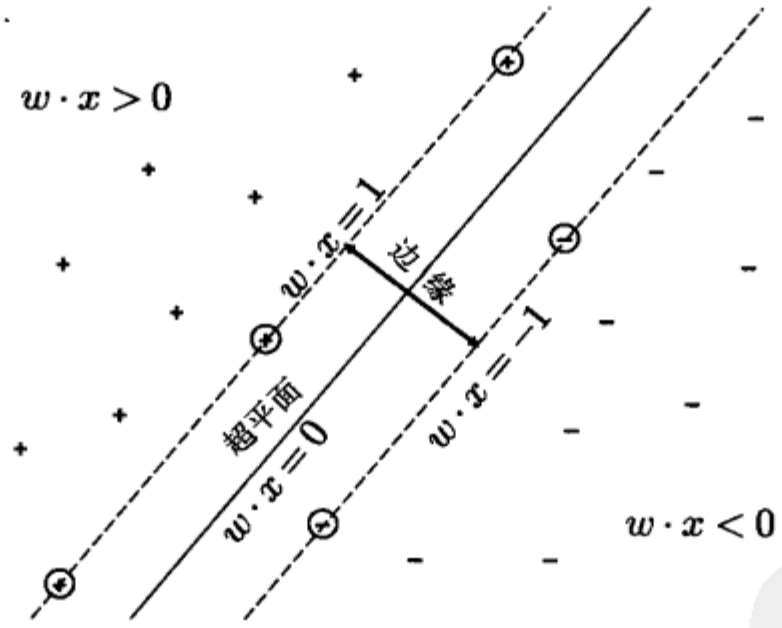


图9-4 对线性可分数据支持向量机的图示说明。这里，由w定义的超平面以及边缘、决策区域和圆圈表示的支持向量被表示出来

这里，是两个向量间的点积（内积），$\|w\|=(w\cdot w)^{1/2}$是向量w的长度。SVM算法中超平面的概念是分离数据最大边缘化的超平面w。为了简化问题，通常假设$w\cdot x^-=-1$并且$w\cdot x^+=-1$。这个假设不能改变对问题的解法，使边缘等于$2/\|w\|$。另一个等价的公式是，发现解决下列优化

⊖ 向量x^-及x^+称为支持向量。最优超平面w是这些向量的特征组合。因此，它们为决策边界提供支持，这就是支持向量机名称的来源。

问题的超平面：

$$\begin{aligned} \min: \quad & \frac{1}{2}\|w\|^2 \\ subject\ to: \quad & w\cdot x_i \geqslant 1 \quad \forall i\ s.t.\ Class(i)=+ \\ & w\cdot x_i \leqslant -1 \quad \forall i\ s.t.\ Class(i)=- \end{aligned}$$

这个公式容易解答，因此常常应用。实际上，优化问题可以通过动态规划解决，这个问题超出了本书的范围。但是，有很多开源的SVM工具包，在“参考文献和深入阅读”中，会提供这样的软件包的指示。

一旦超平面w被确认，未见的文档将按如下规则分类：

$$\text{Class}(d)=\begin{cases} + \text{如果}\ w\cdot x_d>0 \\ - \text{其他} \end{cases}$$

因此，这个规则是基于文档特征向量落到超平面的哪一侧进行分类。回顾图9-4，可以看到这个例子中，落在超平面左边的点被划为正例，落在超平面右边的点被划为反例。

2. 情况2：非线性可分数据

真实世界的数据集很少是线性可分的。因此，为了解决这个问题，需要修改SVM公式的描述。通过在不满足线性可分约束的训练实例中加入惩罚因子来完成。

假设在正例类别中的一些正例，$w\cdot x=-0.5$，这不符合正例约束$w\cdot x\geqslant 1$。事实上，是x落在超平面错误的一端。由于$w\cdot x$目标值至少要等于1，可以基于目标值和实际值的差值使用线性惩罚因子。

例如对于x，惩罚因子为$1-(-0.5)=1.5$，如果$x=1.25$，符合约束的范围，则不用惩罚因子，这种惩罚因子类型称为关键损失函数（hinge loss function），公式定义如下：

$$L(x)=\begin{cases} \max(1-w\cdot x,0) \text{如果}\ \text{Class}(i)=+ \\ \max(1+w\cdot x,0) \text{如果}\ \text{Class}(i)=- \end{cases}$$

这个损失函数合并进SVM最优化如下：

$$\begin{aligned} \min: & \frac{1}{2}\|w\|^2+C\sum_{i=1}^{N}\xi_i \\ subject\ to: & w\cdot x_i \geqslant 1-\xi_i \quad \forall i\ \text{s.t.Class}(i)=+ \\ & w\cdot x_i \leqslant -1+\xi_i \quad \forall i\ \text{s.t.Class}(i)=- \\ & \xi_i \geqslant 0 \quad \forall i \end{aligned}$$

这里ξ_i称为允许目标值被违反的松弛变量（slack variable）。这个松弛变量加强关键损失函数。注意，如果所有的约束被满足，所有松弛变量会等于0，损失函数将降到线性可分的情况。如果违反约束，不符合约束的数量加上目标函数再乘以C，C是一个自由参数，控制对分错的样例给多少惩罚不符合约束的，标准值设为1。这个优化问题就是发现一个允许一些松弛的最大边缘的超平面。对于线性可分的情况，这个优化问题用二次规划解决。另外，分类问题视为线性可分情况执行。

3. 核技巧

图9-3说明了训练数据线性不可分的情况，将它们变换（transformation）或映射（mapping）到高维空间中，得到线性可分的点的集合，多数情况下可以提高分类的效率。

有很多方法可以映射N维向量到更高的空间。例如，给定向量$[f_1(d), \cdots, f_N(d)]$，可以包括特征值的平方来增大向量。数据项将由$2N$维向量表示：

$$[f_1(d), \cdots, f_N(d), f_1(d)^2, \cdots, f_N(d)^2]$$

由于向量空间是高维的，在时间和空间的需求比较大，会降低算法的效率。

需要注意，在训练和测试SVM时，主要的数学运算是点积，如果有高效的方法计算高位空间的向量点积，这个映射就比较容易实现。事实上这种可能是存在的，可以通过核函数（kernel function）完成，一个核函数取两个N维向量并计算在更高维空间的点积。这个高维空间是隐含的（implicit），不是实际构造的。

我们来看一个例子，假设有两个二维向量$w=[w_1 w_2]$和$x=[x_1 x_2]$，定义$\Phi(\cdot)$如下：

$$\Phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

这里，$\Phi(\cdot)$将二维向量映射为三维向量。正如在前面所述，这个方法或许是有用的，由于将原始的输入通过$\Phi(\cdot)$映射到三维空间，使其有可能成为线性可分的点。还可以通过很多方法将原始输入映射到高维空间。就像现在看到的一样，某种映射在高维空间具有高效的点积运算。

给定这个映射，计算$\Phi(w) \cdot \Phi(x)$，首先要明确构造$\Phi(w)$、$\Phi(x)$，然后在三维空间，进行点积。但是很意外，我们发现这是不必要的。由于：

$$\begin{aligned}\Phi(w)\cdot\Phi(x) &= w_1^2x_1^2 + 2w_1w_2x_1x_2 + w_2^2x_2^2 \\ &= (w\cdot x)^2\end{aligned}$$

这里$w \cdot x$的计算是在原始的二维空间，因此，不必在三维空间，只需要在二维空间计算点积，然后计算取平方。这个技巧称为核技巧（kernel trick），能高效地在一些更高维空间计算点积。

当然，这里的例子是微不足道的，核技巧真正的作用是在处理映射到非常高维的空间时。事实上，一些核是在无限维空间上处理点积！

最广泛应用的核在表9-1中给出，注意到，高斯核也称为径向基函数(RBF)核。核的最佳选择依赖于数据的几何形状。这些核已经证明在文本特征上的有效性，只要是方差σ^2适当，高斯核也擅长在更广范围的数据集工作。多数的SVM软件包已经建立了这些核，用它们和用指定的命令行参数一样简单。因此，给出它们的潜在作用和易用性，对给定数据集合任务确定最好的核是有价值的。

这些软件包的可用性及表达方式的灵活性，更重要的是，已经在许多应用中被证明的有效性导致了SVM在分类中有广泛的应用。

4. 非二元分类

直至目前，所有的讨论还是集中在支持向量机如何应用在二元分类任务上。下面将描述最流行的两种方法，将二元分类器，例如支持向量机，转换为多类分类器。这些方法实现起

来很简单，并且已经证明非常有效。

表9-1 SVM中典型应用的核列表。对于每一个核，给出了名称、值及隐含维度空间

核类型	值	隐含维度
线性	$K(x_1, x_2) = x_1 \cdot x_2$	N
多项式	$K(x_1, x_2) = (x_1 \cdot x_2)^p$	$\binom{N+p-1}{N}$
高斯	$K(x_1, x_2) = \exp - \| x_1 - x_2 \|^2 / 2\sigma^2$	无穷

第一种技术称为一对多（One Versus All, OVA）方法，假设有$K \geqslant 2$类的分类问题。这个OVA方法将训练K个分类器。当训练到第k个分类器的时候，第k类将被看作正例，其他所有类将被看作反例。也就是说，每个分类器将单一类被选中的实例作为正例，余下的实例作为反例。给定测试样例x，被K个分类器分类。x被划分到的类别是分类器的值$w \cdot x$中最大的那一类。就是说，如果w_c是“类别c对非类别c的分类器”，那么将根据如下公式分类：

$$\text{Class}(x) = \arg\max_c w_c \cdot x$$

另一种技术称为一对一（One Versus One, OVO）方法，在一对多方法中，二值分类器被每一个类别对训练。例如，对于三元类别问题，类别为“优秀”、“不错”、“差”，需要训练如下的分类器：“优秀vs不错”、“优秀vs差”、“不错vs差”。一般情况下，这个OVO方法要求$K(K-1)/2$个分类器被训练，每一次x被划分到c类，对c的投票将被记录下来，最后，x被划分到具有最多投票的类别中。

OVA和OVO方法在实践中都是很好的方法，没有确定的证据说明一种方法优于另一种方法。换句话说，方法的有效性在很大程度上依赖于数据集的基础特征。

9.1.3 评价

大多数分类任务用标准的信息检索度量标准评价，例如准确率（accuracy）[⊖]、精确率（precision）、召回率（recall）、F值以及ROC曲线分析。在第8章详细描述了这些度量标准，这些度量标准最常用的是准确率和F值。

评价分类任务和评价检索任务有两个主要的不同。第一个不同是“相关”的概念被替换为“被正确分类”。另一个不同是，微平均很少被用于评价检索任务，但是广泛用于分类的评价。宏平均包括计算每个类（class）的评价参数，然后将每个类的参数平均。微平均计算的是对每个测试实例（样本）的评测，然后在所有这些实例上平均。分析宏平均和微平均非常有用，尤其是在$P(c)$分布不平衡的情况下。

9.1.4 分类器和特征选择

到目前为止，我们已经介绍了两个基本的分类器的基本要素。描述了建立在分类器之上的原理、基本假设、利与弊，以及在实践中如何应用。由于分类是一个复杂和丰富的课题，在这一小节将介绍更深入的话题，或许那些喜欢更深和更复杂的分类话题的人会更感兴趣。

1. 生成模型、判别模型和非参数模型

朴素贝叶斯分类器基于概率模型，模型要求假设文档是从根据对应一些基础事件空间的

⊖ 准确率是排序为1的精确率的另一种定义。

文档模型的文档类别产生的，朴素贝叶斯分类器是称为生成模型（generative model）的概率模型的众多种类中的一个例子。这些模型假设一些基础的概率分布生成文档和类别。在朴素贝叶斯情况下，类别和文档生成如下。首先根据$P(c)$生成一个类，然后根据$P(d|c)$生成文档，这个过程在图9-5中进行了总结。生成模型模仿人类实际生成（撰写）文档的方式。

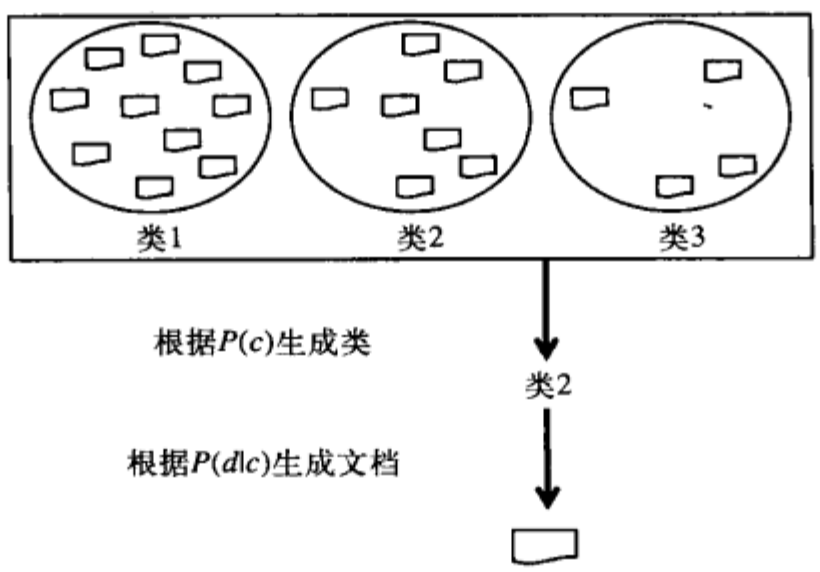


图9-5 朴素贝叶斯所用的生成模型。首先根据$P(c)$生成一个类，然后，根据$P(d|c)$选择文档

当然，生成模型的准确率很大程度依赖于概率模型如何准确地捕捉生成过程。如果模型合理地反映了生成过程，生成模型则是非常有效的，特别是训练样例非常少的时候。随着训练样例的增多，生成模型的能力可以被简化的分布假设约束，例如朴素贝叶斯分类器的独立性假设。这种情况下，判别模型（discriminative model）常常优于生成模型。判别模型不会去模拟文本和类别的生成过程，而是直接模拟对给定的输入文本赋予类别。这种方式下，它们是在类别间判别。由于这种模型不需要模拟文档生成，所以很少有分布假设。这也是在训练样例比较多的情况下，倾向于生成模型的一个原因。支持向量机是判别模型的一个例子。注意，在SVM公式中，没有文本生成过程的假设，而是由SVM直接学习到有效划分类别的超平面。

针对大规模的训练样例，非参数分类器（Non-parametric classifier）是另一个选择。非参数分类器通过去除所有分布假设让数据“自己说话”。一个简单的非参数分类器就是最近邻分类器（nearest neighbor classifier），给定一个未见样例，在训练样例中找到离它最近的邻居，该未见样例就被赋予离它最近的样例所属的类别。图9-6表示了最近邻分类器的输出。注意到分类器导致的不规则、非线性的决策边界。生成和判别模型，甚至具有非线性核的SVM将花费很多的时间来使模型适应数据。由于这个原因，当数据量接近无穷大时，最近邻分类器是最佳的。但是，这个分类器对小规模数据集合有很大的不一致，所以也限制了它的应用。

2. 特征选择

SVM分类器将输入（例如文本）嵌入到特征函数集定义的一些特征空间。就像曾经描述过的，通常对词表中的词定义一个(或多个)特征函数。这个|V|-维特征空间可以很大，尤其是对很大的词表。由于特征集的规模对分类器的效率和效果都会带来影响，研究者发明了特征空间剪枝技术，这就是特征选择（feature selection）技术。

特征选择的目的是在最初的特征空间中，找到能够代表原特征空间的主要特征的特征子集，这既能提高效率同时又不会过多地影响其有效性。事实证明，特征选择常常是提高了有效性而不是降低有效性。原因是一些去除的特征可能是噪声或不准确的因素，这会妨碍分类

模型更好的学习。

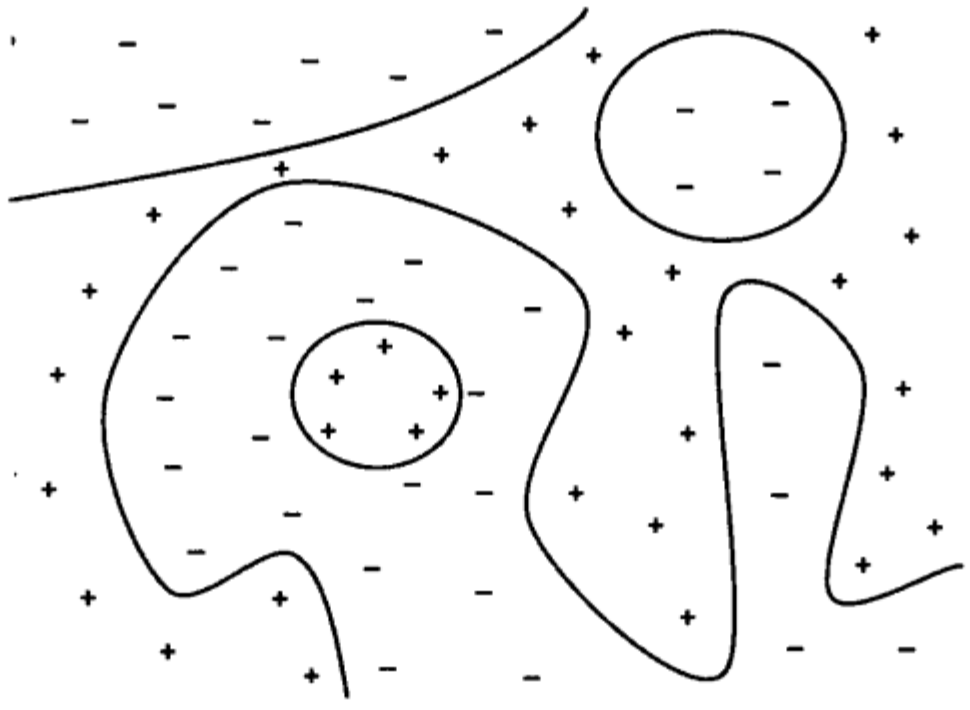


图9-6 非参数学习算法数据集合的例子，非参数学习算法，例如最近邻分类器，也许会优于参数算法。加号和减号分别代表正例和反例。灰实线表示了实际非线性决策边界

信息增益（information gain）是文本分类中应用最广泛的特征选择方法。信息增益基于信息论原理。就像其名字中包含的信息一样，该特征度量的是对类别标签增加了多少信息。我们回顾图9-1垃圾信息分类的例子，会发现特征“cheap”提供大量的信息，如果“cheap”出现，很有可能是垃圾类，如果不出现，很有可能是“非垃圾类”。在信息论中，熵（entropy）是包含某些分布的期望信息。例如类的分布$P(C)$。因此，特征f的信息增益度量的是发现f之后$P(C)$的熵的变化。假设一个多重伯努利事件空间，计算如下：

$$\begin{aligned} IG(w) &= H(C) - H(C \mid w) \\ &= -\sum_{c \in C} P(c) \log P(c) + \sum_{w \in \{0,1\}} P(w) \sum_{c \in C} P(c \mid w) \log P(c \mid w) \end{aligned}$$

这里$H(C)$是$P(C)$的熵，$H(w|C)$称为条件熵（conditional entropy）。下面以例子说明，从垃圾信息的例子中计算“cheap”信息增益：

$$\begin{aligned} IG(cheap) = & -P(spam)\log P(spam) - P(\overline{spam})\log P(\overline{spam}) \\ & + P(cheap)P(spam \mid cheap)\log P(spam \mid cheap) \\ & + P(cheap)P(\overline{spam} \mid cheap)\log P(\overline{spam} \mid cheap) \\ & + P(\overline{cheap})P(spam \mid \overline{cheap})\log P(spam \mid \overline{cheap}) \\ & + P(\overline{cheap})P(\overline{spam} \mid cheap)\log P(\overline{spam} \mid \overline{cheap}) \\ = & -\frac{3}{10}\log\frac{3}{10} - \frac{7}{10}\log\frac{7}{10} + \frac{4}{10}\cdot\frac{3}{4}\log\frac{3}{4} \\ & + \frac{4}{10}\cdot\frac{1}{4}\log\frac{1}{4} + \frac{6}{10}\cdot\frac{0}{6}\log\frac{0}{6} + \frac{6}{10}\cdot\frac{6}{6}\log\frac{6}{6} \\ = & 0.2749 \end{aligned}$$

这里$P(\overline{cheap})$是$P(cheap=0)$的简写形式。$P(\overline{spam})$就是$P(C=\text{not spam})$，假设0log0=0。“buy”、

“banking”、“dinner”和“the”的信息增益分别是0.0008、0.0434、0.3612和0.0，因此，根据信息增益，“dinner”是最有信息量的词，因为根据训练集，它是一个非常好的“非垃圾”的预测器。相反，“the”是最不好的预测器，因为该词会出现在每一篇文档中，所以没有判别功能。

相似的信息增益度量可以从其他的事件空间中导出，例如多项式事件空间。有许多不同的采用信息增益的方法来选择特征。但是，最常用的方法还是选择具有最大信息增益的K个特征，并用这些特征训练模型。也有可能按照所有特征的百分比或者采用阈值的方法选择特征。

尽管有许多其他的特征选择策略，但信息增益是一个万能的特征选择标准，尤其是基于文本的分类问题。在本章末尾的“参考文献和深入阅读”部分，提供了其他特征选择技术的导读。

9.1.5 垃圾、情感及在线广告

尽管相关排序函数是搜索引擎非常关键的部分，但分类技术在各种搜索任务中也充当着重要的角色。在这一小节，将描述几个真实世界的文本分类的应用。这些应用包括：垃圾信息检测（spam detection）、情感分类（sentiment classification）及在线广告分类（online advertisement classification）。

1. 垃圾信息

分类技术有助于检测和去除各种垃圾信息。垃圾信息一般定义为一些恶意内容[⊖]，例如一些未经同意的广告，网页的欺诈排序或者病毒传播。垃圾信息一个重要的特征是，常常不包含有用的内容。垃圾信息的定义是主观的，因为有的信息对某些人无用，但是对其他人会有用，所以很难提出一个客观的定义。

垃圾信息有很多类型，包括垃圾邮件、垃圾广告、垃圾博客和垃圾网页。垃圾邮件的制作者对不同的垃圾信息采用不同的技术。因此没有一个单一的垃圾信息分类方法来解决所有的垃圾信息问题。特定的垃圾信息分类器是根据不同的垃圾信息类型建立的，每一个都考虑到了特定域的信息。大量都是关于垃圾邮件的，例如SpamAssassin[⊜]这样的过滤程序很通用。图9-7显示了SpamAssassin对邮件处理的输出样例。SpamAssassin计算邮件的分值并与阈值（一般为5.0）比较，确定是否为垃圾邮件。这个分值基于多个特征的合并，贝叶斯分类器的输出是最重要的特征之一。假如这样，邮件中包含的URL在黑名单上，时间戳比收到的时间晚，贝叶斯分类器根据词的信息判定这个邮件40%～60%可能是垃圾信息。由于上述这三项对这个邮件的打分没有超过5，所以不是垃圾信息（属于误判）。

由于本书集中于搜索引擎，因此将关注垃圾网页的处理，这也是分布最广而且最难处理的垃圾信息类型之一。检测网页垃圾是很困难的，因为垃圾信息制造者变得日益诡异，似乎垃圾邮件制造者在信息检索领域有很高的水平，有很多方法针对网页。Gyöngyi和Garcia-Molina (2005)提出了网页垃圾分类体系，尝试将常常用于人为提高网页排序的不同网页垃圾技术进行分类，两个最常用的技术是链接垃圾（link spam）和词垃圾（term spam）。

⊖ “垃圾”一词的词源，相对于计算机侵权是非常有趣的，这个含义出自于1970Monty Python的滑稽短剧集，餐馆中，在菜单上的每一种都是垃圾食品（肉类产品）。海盗合唱团唱着歌“垃圾、垃圾、垃圾…”，不断地重复恼人的行为。

⊜ http://spamassassin.apache.org/。

```
To: ...
From: ...
Subject: non profit debt
X-Spam-Checked: This message probably not SPAM
X-Spam-Score: 3.853, Required: 5
X-Spam-Level: *** (3.853)
X-Spam-Tests: BAYES_50,DATE_IN_FUTURE_06_12,URIBL_BLACK
X-Spam-Report-rig: ---- Start SpamAssassin (v2.6xx-cscf) results
        2.0 URIBL_BLACK         Contains an URL listed in the URIBL blacklist
                        [URIs: bad-debtyh.net.cn]
        1.9 DATE_IN_FUTURE_06_12  Date: is 6 to 12 hours after Received: date
        0.0 BAYES_50            BODY: Bayesian spam probability is 40 to 60%
                        [score: 0.4857]

Say good bye to debt
Acceptable Unsecured Debt includes All Major Credit Cards, No-collateral
Bank Loans, Personal Loans,
Medical Bills etc.
http://www.bad-debtyh.net.cn
```

图9-7 Spam Assassin垃圾邮件过滤输出样例

对于链接垃圾，垃圾信息制造者用各种办法人为地提高他们网页的链接分值。特别是，互联网常常根据链接的入度和PageRank度量，完全基于链接结构度量网页的重要性。但是这些技术对垃圾信息是敏感的。一个常用的简单的链接垃圾信息的方法是，在博客或论坛中张贴目标网页的链接，另一种人为增加基于链接的分值方法是，加入一个链接交换网络。链接交换网络是彼此相互连接的大规模站点网络，用来增加进入这个站点的链接数。另一个链接垃圾信息的技术是链接耕作，与交换网络类似，不同之处在于，垃圾制造者自己买下大量的域名，建立大量的站点，然后将它们链接在一起。还有各种其他的方法，但是大多是链接垃圾。最近提出许多替换PageRank的方法，包括HostTrust（2004）和SpamRank（2005），都在试图减弱链接垃圾的潜在作用。

另一个常用的产生垃圾信息的方法是词垃圾信息。词垃圾信息试图改变文本的表示，以便能被某些查询或者关键词检索。与基于链接的打分的原理一样，基于词的打分也是对垃圾信息敏感的。多数广泛用于检索的模型，包括BM25和语言模型，主要是利用一些包含词频或文档频率的公式。因此，通过增加目标词，这些模型会很容易地被检索到的非相关文档欺骗。进一步地，许多网页相关排序函数匹配进入的锚文本和URL，修改URL以匹配给定的词或词组是很容易的。但是修改进入的锚文本还是需要很多努力，利用链接交换和链接耕作会更容易一些。另一个技术称为倾销（dumping），就是用许多不相关的词(常常是整个词典)填充文本，导致文本被任何查询检索，因为几乎包含了查询词的每一种组合。因此，这起到了加强召回率的作用。也可以和其他的垃圾技术合并，例如重复文本，都是为了有更高准确率和高召回率。短语拼接（Phrase stitching）（从多种来源合并词和句子）和编织（weaving）（将垃圾词加入到有效的文本中，例如新闻故事）是其他一些人为生成垃圾内容的技术。当开发相关排序函数防止垃圾信息时，所有这些词垃圾的类型都将应该被考虑。

图9-8给出了网页包含垃圾信息的例子，这个网页包含重复重要的词的垃圾和提到的有关垃圾网站的链接垃圾信息

至此，可以看到，垃圾信息有规模庞大的类型。这里简单地关注网页垃圾，不考虑其他

类型的垃圾。的确，很容易以垃圾信息为主题写一整本书，但是，在结束垃圾信息讨论之前，我们还将描述众多用来处理网页垃圾的不同方法之一。

WWW站点：

BETTING NFL FOOTBALL PRO FOOTBALL SPORTSBOOKS NFL FOOTBALL LINE ONLINE NFL SPORTSBOOKS NFL

Players Super Book

When It Comes To Secure NFL Betting And Finding The Best Football Lines Players Super Book Is The Best Option! Sign Up And Ask For 30 % In Bonuses.

MVP Sportsbook

Football Betting Has Never been so easy and secure! MVP Sportsbook has all the NFL odds you are looking for. Sign Up Now and ask for up to

30 % in Cash bonuses.

词垃圾：

pro football sportsbooks nfl football line online nfl sportsbooks nfl football gambling odds online pro nfl betting pro nfl gambling online nfl football spreads offshore football gambling online nfl gamblibg spreads online football gambling line online nfl betting nfl sportsbook online online nfl betting spreads betting nfl football online online football wagering online gambling online gambling football online nfl football betting odds offshore football sportsbook online nfl football gambling ...

链接垃圾：

MVP Sportsbook Football Gambling Beverly Hills Football Sportsbook
Players SB Football Wagering Popular Poker Football Odds
Virtual Bookmaker Football Lines V Wager Football Spreads
Bogarts Casino Football Point Spreads Gecko Casino Online Football Betting
Jackpot Hour Online Football Gambling MVP Casino Online Football Wagering
Toucan Casino NFL Betting Popular Poker NFL Gambling
All Tracks NFL Wagering Bet Jockey NFL Odds
Live Horse Betting NFL Lines MVP Racebook NFL Point Spreads
Popular Poker NFL Spreads Bogarts Poker NFL Sportsbook ...

图9-8 网页垃圾示例，显示主网页、相关的一些词垃圾和链接垃圾

Ntoulas等人（2006）提出了用文本分析检测网页垃圾的方法。这个方法从每个网页中抽取大量特征并用于分类器。有一些特征包括网页中词数、标题中的词数、词的平均长度、进入锚文本的数量及可视化文本部分。这些特征尝试抓住网页文本最基本的特征。另一个使用的特征是网页压缩率，可以度量网页用压缩算法减少的数量。已经证实压缩的越多，越有可能是垃圾信息，因为包含许多重复词或短语的内容是容易被压缩的，同时也显示了该方法对垃圾邮件检测的有效性。也有人用从流行词[⊖]中提取的部分词或者出现在网页中的部分流行词作为特征。这些特征尝试捕捉网页是否用很多可能与查询词匹配的流行词填充。最后两种方法基于n-gram可能性。实验证明，包含过少或者过多的n-grams比包含n-grams平均可能性的网页更有可能成为垃圾信息。所有这些特征采用另一种有监督分类器、决策树（decision tree）的学习算法，分类准确率达到90%，同样的特征可以很容易地应用到朴素贝叶斯和支持向量

⊖ 这里“全球流行”词列表简单的指测试语料中前N个频率最高的词。

机分类器。

2. 情感

正如在第6章所描述的那样，网页查询主要有三种类型。在第7章描述的模型主要关注检索及导航查询。而第三种类型，即交易查询，则面临更多不同的挑战。如果一个用户查询商品名，搜索引擎会陈列出超出用户要求的相关列表的各种信息。例如，如果用户感兴趣购买产品，则链接及在线购物网站会帮助用户完成购物。也有可能用户已经拥有这个产品，为了搜索附件或者增强功能，搜索引擎可以通过显示相关附件或服务来赚钱。

这里要详细关注的另一种情况是，用户研究某一商品的目的是需要确定是否购买。这种情况下，检索的信息例如产品说明书、产品评论、关于该产品的博客记录都是有价值的。为了减少用户通篇阅读这些信息的数量，需要有一个自动汇集所有评论和博客的记录，以呈现给用户一个浓缩的文摘。

构建这样一个系统涉及许多步骤。每个步骤都涉及分类的某些形式。首先，当抓取或索引站点时，系统需要自动分类，以判别网页是否包含评论或者在博客记录中是否表达对一个产品的意见。判别以意见为依据的文本，与事实性文本形成对比，称为意见检测（opinion detection）。在评论和博客记录完成之后，一个分类任务是要抽取产品名称和与名称对应的评论，这称为信息抽取（information extraction）任务。对于每一个给定产品的评论，分类任务是判定网页的情感。一般地，网页情感不是正面的就是反面的，虽然分类器会给出具体的分值，例如“两颗星”、“四颗星”。最后，包括情感在内的所有数据必须集成，以某种有意义的方式呈现给用户。图9-9表示从网络业务自动生成产品评论的一部分，这些基于产品各方面的摘要的情感，例如“易用性”、“大小”、“软件”，是从独立的用户评论中产生的。

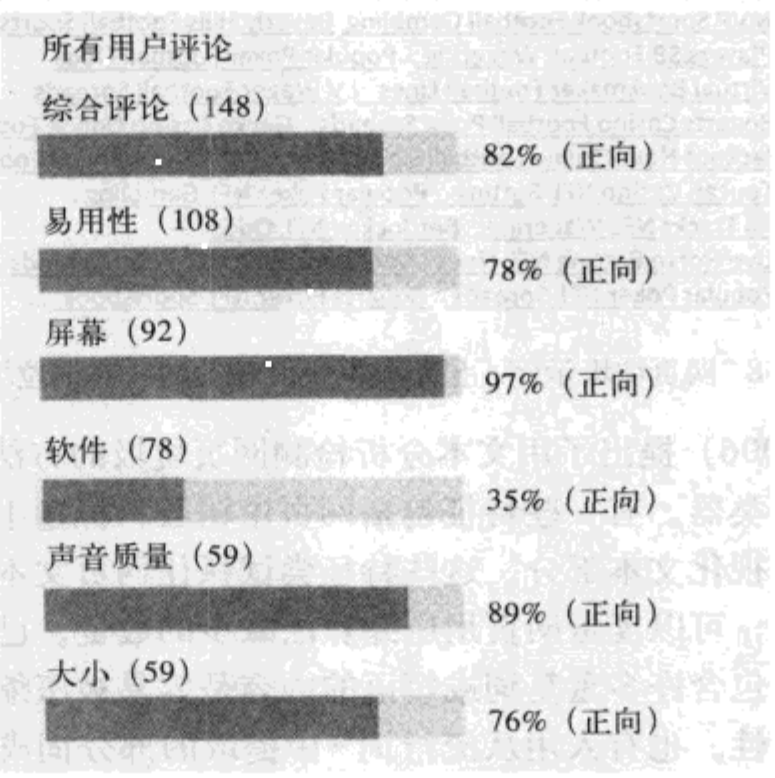


图9-9 合并情感产品评论样例

我们不是进入这些不同的分类器的细节，而是关注这些情感分类器是如何工作的。同前面的例子一样，我们考虑人是如何判定文本片断的情感的。对于大部分情况，用词汇线索确定情感。例如，正面的数码相机的评论包括内容词类似“great”、“nice”、“amazing”，而反面评论包含内容词“awful”、“terrible”、“bad”，这似乎提出了一种利用建立两个词表解决问题的可能。第一个词表包含正面情感的指示词，另一个词表包含负面情感的指示词。对于给

定的文本片断，简单地计算出包含的正面词和负面词的个数。如果正面词较多，该文本就为正面的，反之，为负面的。尽管这个方法看起来非常合理，但是人们不是很善于建立正面词和负面词的列表。因为人类语言有很大的歧义性和上下文相关性。例如“The digital camera lacks the amazing picture quality promised”，由于该句包含两个正面词（amazing, quality），只有一个负面词（lacks），所以倾向于划分为正面评论。

Pang等人（2002）提出用机器学习的方法进行情感分类，包括朴素贝叶斯、支持向量机、最大熵（maximum entropy）等。用于这些分类器的特征是unigram、bigrams、词性标注、形容词以及词在文本中的位置。作者报告SVM分类器只用unigram特征会呈现最好的性能，比用所有特征的分类器更准确。另外，也发现在这个任务上，多重伯努利事件空间优于多项式事件空间。这是因为最相关的情感词在一些文本片段中仅仅出现一次，因此词频对这个模型起的作用很小。有趣的是，机器学习方法比人工构建词表的方法要更准确。采用unigram的SVM准确率达到80%，而人工构建词表方法只有60%。

3. 分类广告

正如在第6章中所描述的，搜索竞价和内容匹配是广泛用于商务搜索引擎的两种不同的模型。前者是广告匹配查询，后者是广告匹配网页。搜索竞价和内容匹配都是点击付费。也就是说，只有用户点击广告，广告商才给搜索引擎付费。用户点击广告会有多种原因。显然，如果广告是“话题相关”，这是在本书剩余部分讨论的标准概念，用户则会点击它。但是，这不是用户点击的唯一原因，如果用户搜索“热带鱼”，她将点击宠物商店、当地水族馆甚至潜水课程，但是很少点击钓鱼、鱼餐馆或汞中毒，原因是“热带鱼”这个概念具有某种限制用户发现有趣广告类型的语义范围。

尽管可以用标准的信息检索技术（例如查询扩展和查询重构）发现这些语义来匹配广告，但也可以利用将查询划分语义类的分类器。Broder等（2007）提出了简单有效的方法，将如查询或网页正文的文本项划分为语义层次，这个层次由手工构建，包括6 000多个节点，每一个节点代表一个独立的语义类，层次越深，类别越明确。人们根据其语义含义手工地将数以千计的具有商业意向的查询放入层次中。

给定这样的层次和数以千计标注好的实例，有许多方式可以对未见查询或网页进行分类。例如，可以用朴素贝叶斯或者SVM。由于有6 000多种类别，也许某些类会有很少的实例，也就是数据稀疏问题。一个较大的问题是效率问题，朴素贝叶斯和SVM方法将一个文本分到6 000多个可能的类别中是很慢的。毫无疑问，查询必须实时分类。Broder等人提出$tf \cdot idf$用权重计算相似度，将查询（或网页）划分到语义类，他们涉及的分类问题像检索（retrieval）问题一样。查询就是要被分类的查询（或网页），文档集包含6 000多“文档”，每一个对应一个语义类，例如对于语义类“Sports”，“文档”将包含所有都标为“Sports”的查询。这些“文档”存储在倒排索引表中，进入的查询和网页可以高效地检索，这可以视为最近邻分类器的一个例子。

实际应用这个分类器时，需要对广告清单中的每一个广告进行预分类，然后，当一个新的查询（或网页）到来时，即进行分类。用语义类提高分类有许多方法。显然，如果查询语义类精确匹配广告语义类，分类器将会给一个高分。但是也有其他情况，两个事物非常相关，但是没有精确匹配到同一个语义类。因此，Broder等人提出了采用在层次中两个节点的最小公共祖先（least common ancestor）的倒数（inverse）来度量层次内两个语义类别距离的方法。

公共祖先是在层次中为了到达两个节点必须通过的节点。最小公共祖先是层次中有最大深度的节点。如果两个节点是同一个节点，距离为0。如果最小公共祖先是根节点，则距离很大。图9-10显示了如何用层次匹配将网页和广告进行语义匹配。在图中，网页和广告类的最小公共祖先是“Aquariums”，是该层次的最高节点。因此这个匹配方式，与将网页和广告划分在同一个节点相比，分数比较低。广告的全部打分可以归纳为基于层次的距离和标准的余弦相似度计算的合并。这样，广告可以根据话题相关性和语义相关性排序。

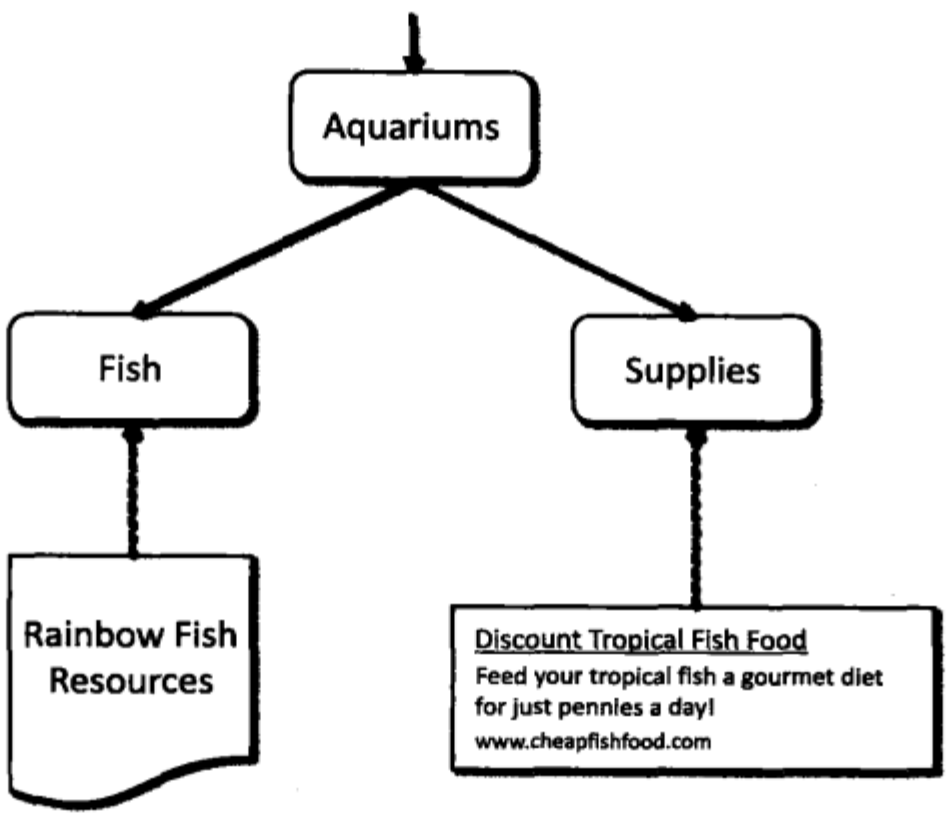


图9-10 语义类匹配关于彩虹鱼(热带鱼的一种)网页和热带鱼食物的广告的例子。节点“Aquariums”、“Fish”、“Supplies”是语义层次内的节点。网页被分为“水族馆-鱼”，广告被分为“供给-鱼”，这里，“水族馆”是最小公共祖先。尽管网页和广告没有共同的词但因为语义相似度它们能够匹配

9.2 聚类

聚类算法提供了另一种组织数据的方法。和本章中的分类算法不同，聚类算法基于无监督（unsupervised）学习，即不需要任何训练数据。聚类算法针对一个没有标注的实例集合，对所有实例进行分组（簇）。聚类的一个问题是，它常常是一个病态问题，分类有很清晰的目标，但是，对于什么是好的聚类，定义往往是非常主观的。

为了更多地了解聚类涉及的问题，我们考察一下人类是如何对事物进行分门别类的。设想你正在一个杂货店中，要求你对所有的新鲜果蔬（水果和蔬菜）进行聚类。你将如何做？首先，你需要决定聚类时应该采用什么标准。比如，你可能按照颜色、形状、维生素C含量、价格或者这些因素的其他组合对这些果蔬分组。像分类一样，聚类的标准很大程度上取决于如何表示事物。假定输入的实体是表示某个对象，如文档(或水果)的特征向量。如果你希望按照某个性质进行聚类，那么确保特征向量表示这个性质是很重要的。

聚类标准确定之后，需要决定如何将每个事物分配到不同的组中。假定你决定按照颜色

对产品聚类，并且创建了一个红色的簇（红葡萄、红苹果）和一个黄色的簇（香蕉、南瓜），当遇到一个橙子时怎么办？你会创建一个新的橙色的簇，还是将橙子分配到红色或黄色的簇？这些也是聚类算法必须要解决的重要问题。这些问题的本质是，使用多少个簇以及如何将事物分配到每个簇。

最后，将所有的产品聚类之后，该如何描述聚类效果如何？意思是，必须评价(evaluate)聚类结果。这往往是很难的，但是目前人们已经提出了一些自动化的评价方法。

在这个例子中，使用某个固定的性质集合如“红色”来定义簇。事实上，这是一种非常明确的分类方式，称为单因素聚类（monothetic）。在第6章，讨论过单因素类或簇，同时提到大多数聚类算法产生多因素（polythetic）簇，其中一个簇中的成员有很多共同的性质，但是无法给出单一的定义。换句话说，簇中的成员一般是基于表示这些对象的特征向量之间的相似度（similarity）而被聚在一起的。这意味着聚类算法关键是确定相似度的度量方法。分类和聚类文献中，经常使用距离度量（distance measure），而非相似度度量，在后面的讨论中，使用这个术语。任何相似度度量，一般为一个从0到1的值S，可以通过1−S转化为距离度量。从事信息检索和机器学习的学者，研究过各种相似度和距离度量的方法，从简单的度量方法如戴斯系数（Dice's coefficient）（见第6章），到更复杂的概率的度量方法。

阅读本节的时候，读者应该记住这些部分，因为它们将贯穿始终。本节后面的部分给出三种基于不同方法的聚类算法，讨论了评价问题，并且简单描述了聚类的一些应用。

9.2.1 层次聚类和K均值聚类

在此给出两种聚类算法，它们从数据的某个初始聚类出发，然后通过优化某个目标函数，迭代地提高聚类结果。两个算法之间最主要的差别是目标函数的不同。后面将显示，不同的目标函数会产生不同类型的簇，因此，并不存在最好的聚类算法，选择什么算法很大程度上取决于数据集和任务的特点。

在本节其余的部分，都假设目标是将某个包含N个实例（例如，可能是网页）的集合进行聚类，聚成K个簇，每个实例表示成一个特征向量，其中K是一个由先验知识（a priori）决定的常量。

1. 层次聚类

层次聚类（Hierarchical clustering）是一种以分层的方式建立簇的方法。这种方法有很多不同的聚类算法，这些算法往往根据算法的过程分成两类。

分裂聚类（Divisive clustering）从一个包含所有实例的簇开始。每一次迭代时，它从现有的簇中选择一个簇分成两个（或者可能更多）簇。不断进行这个过程，直到总共产生K个簇。算法的结果很大程度上取决于如何选择和分割簇。

分裂聚类是一种自顶向下（top-down）的方法。另一种类型的层次分类算法称为聚合聚类（agglomerative clustering），它是一种自底向上（bottom-up）的方法。图9-11和图9-12解释了两种算法的差异。一个聚合聚类算法开始时把每一个输入实例都当成单独的簇，即包含N个簇，每个簇中包含一个输入。聚类算法每次将两个（或可能更多）现有的簇合并成一个新的簇。因此，每一次迭代都使簇的总数下降。当剩余K个簇时，算法停止。与分裂聚类类似，这个算法的输出主要取决于如何选择和合并簇。

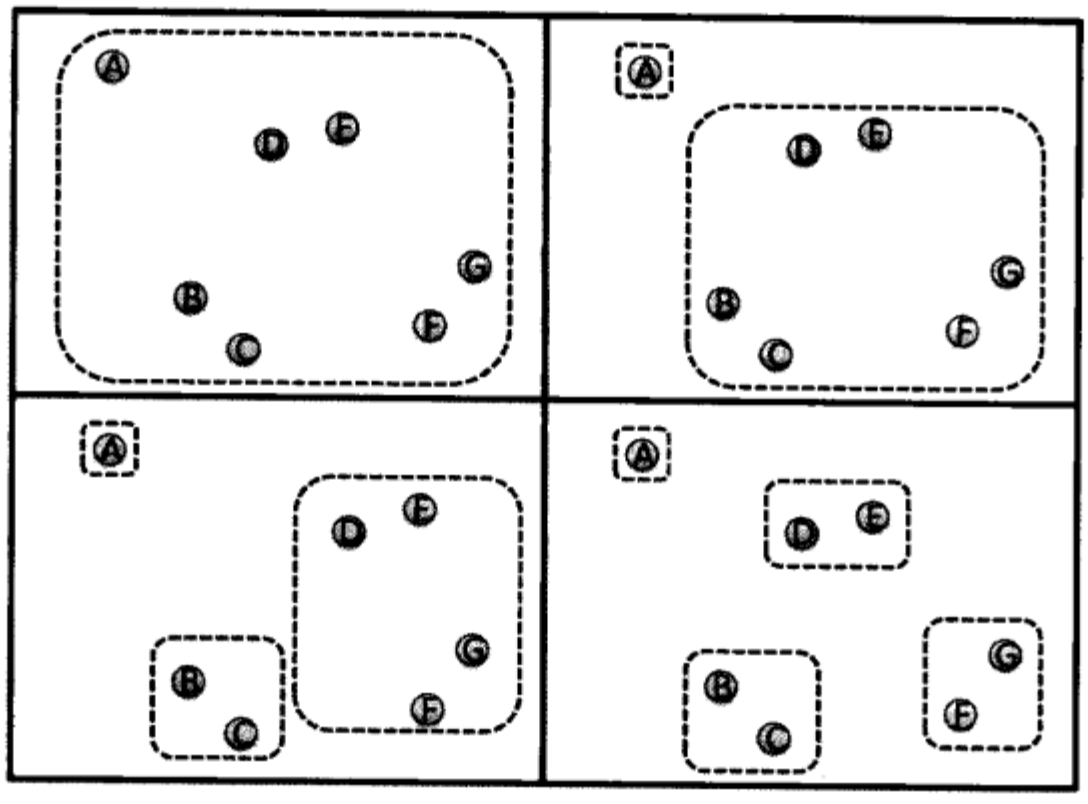


图9-11 K=4的分割聚类示例。聚类过程从左向右，从上向下进行，产生4个簇

一个聚合或分裂聚类算法产生分层的过程，可以很方便地应用树状图（dendrogram）⊖形象地展现。一个树状图以图解方式表示一个层次聚类算法如何工作。图9-13给出了一个树状图，对应图9-12中的样本点，通过聚合聚类算法得到分层结构。在这个树状图中，点D和E首先合并形成一个新的簇H。然后B和C合并成簇I。这个过程一直继续，直到产生单一的簇M，簇M由A、B、C、D、E和F构成。在一个树状图中，实例合并时，在树中所处的高度是很重要的，它表示了合并时两个簇的相似性（或距离）。例如，树状图显示D和E是最相似的实例对。

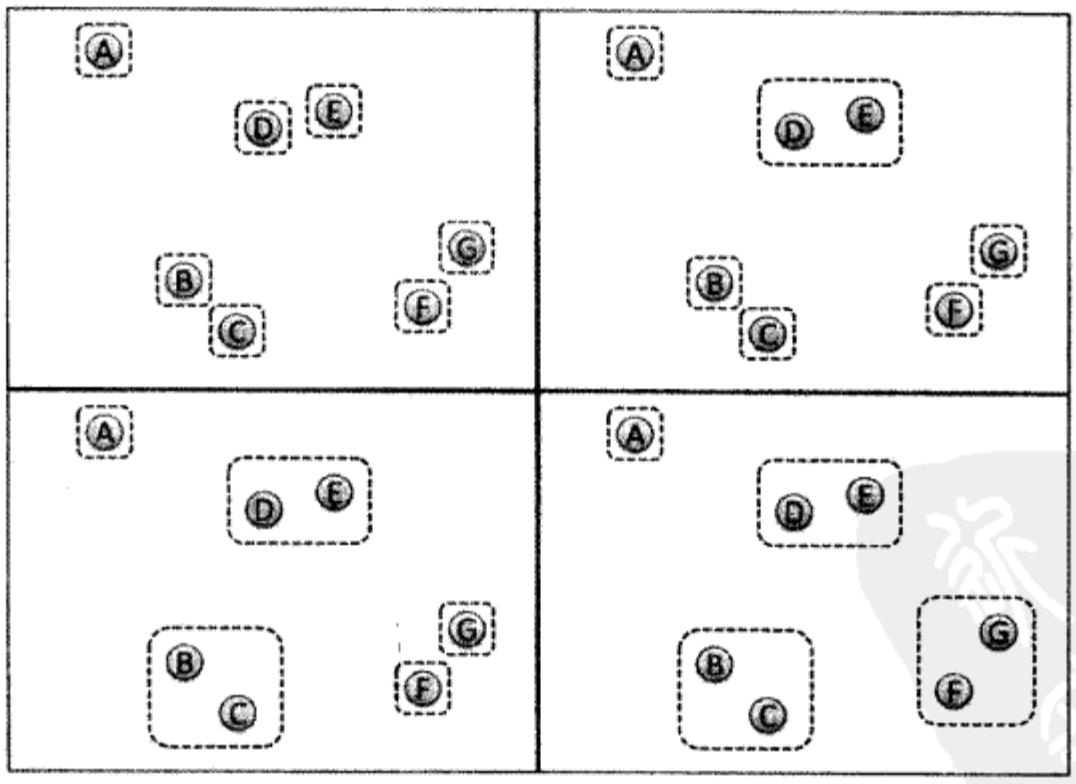


图9-12 K=4的聚合聚类示例。聚类过程从左向右，从上向下进行，产生4个簇

⊖ 来自于希腊词语“dendron”，意思是树。

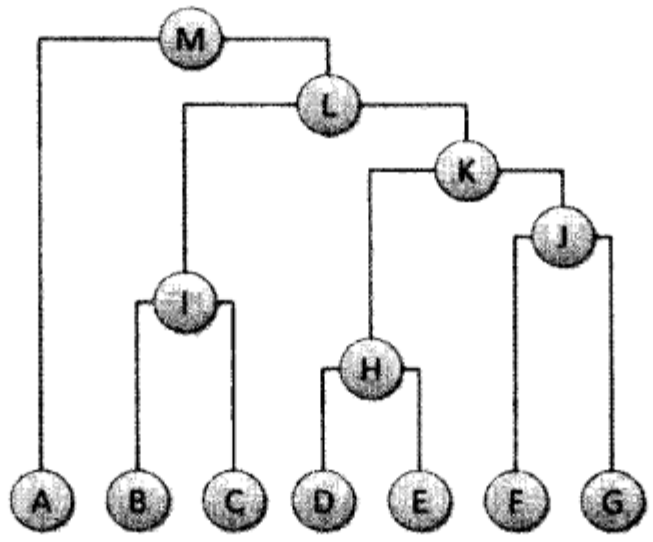


图9-13 解释对图9-12中的样本点进行聚合聚类的树状图

算法1 聚合聚类

```
1: procedure AGGLOMERATIVECLUSTER(X_1, ..., X_N, K)
2:     A[1], ..., A[N] ← 1, ..., N
3:     ids ← {1, ..., N}
4:     for c = N to K do
5:         bestcost ← ∞
6:         bestclusterA ← undefined
7:         bestclusterB ← undefined
8:         for i ∈ ids do
9:             for j ∈ ids − {i} do
10:                c_{i,j} ← COST(C_i, C_j)
11:                if c_{i,j} < bestcost then
12:                    bestcost ← c_{i,j}
13:                    bestclusterA ← i
14:                    bestclusterB ← j
15:                end if
16:            end for
17:        end for
18:        ids ← ids − {bestClusterA}
19:        for i = 1 to N do
20:            if A[i] is equal to bestClusterA then
21:                A[i] ← bestClusterB
22:            end if
23:        end for
24:    end for
25: end procedure
```

算法1是层次聚合聚类的一个简单实现[⊖]。算法使用N个向量$X_1, X_2, \cdots, X_N$表示实例，以希望得到的簇数K作为输入。数组（向量）A是结果向量，它用来记录每一个输入属于哪个簇。如果$A[i]=j$，那么输入X_i在簇j中。这个算法考虑将每两个簇合并起来，对每一个簇对(C_i, C_j)，计算出一个代价$C(C_i, C_j)$。这个代价表示将两个簇C_i和C_j合并起来需要付出的代价。我们很快会介绍如何计算代价。计算出所有的簇对的代价之后，选出合并代价最低的两个簇，算法一

⊖ 文献中经常称为*HAC*。

直进行，直到生成K个簇为止。

如算法1所示，聚合聚类主要取决于代价函数。有很多方式定义代价函数，每一个都会使最终的聚类结果具有不同的特点。现在介绍一些很流行的方法及它们的原理。

单连通（single linkage）方法通过计算C_i中每个实例和C_j中每个实例之间的距离，来衡量C_i和C_j的代价。这些距离中最小（minimum）的就是代价，可以用数学公式表示：

$$COST(C_i, C_j)=\min\{dist(X_i, X_j)|X_i\in C_i, X_j\in C_j\}$$

其中$dist$表示输入X_i和X_j的距离。通常使用欧几里得距离[⊖]计算X_i和X_j的距离，但是也可以使用很多其他距离度量方法。单连通只依赖于两个簇之间的最小距离，它不考虑两个簇中其他实例相距多远。因此，单连通可能使两个簇合并后产生长形的簇。

全连通（complete linkage）和单连通类似。它首先计算两个簇中每一对实例之间的距离。但是它使用最大距离作为代价，而不是最小距离。即代价是：

$$COST(C_i, C_j)=\max\{dist(X_i, X_j)|X_i\in C_i, X_j\in C_j\}$$

由于使用最大距离作为代价，与单连通相比，全连通产生的簇更紧凑，避免了“拉长”区域的产生。

可以通过图9-14解释单连通和全连通的差别。图中节点表示实例（如X_i），当$dist(X_i, X_j) < T$时，将对应节点连接起来，T表示某一阈值。在这个图中，簇A、B和C都是单连通簇。事实上，单连通簇是图中的相连部件，簇中每一个成员都至少和另一个成员相连。全连通簇将包括簇A、单元簇C、簇B上面的实例对以及簇B下面的实例对。全连通簇是图中的团(clique)或最大完全子图，簇中每一个成员都和其他成员相连。

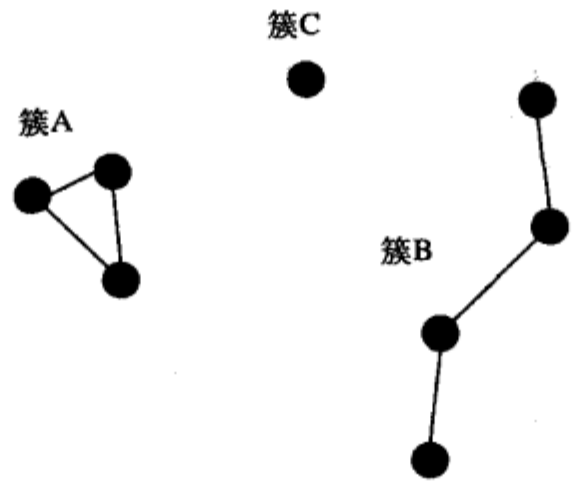


图9-14 图中节点表示实例，通过连接这些节点得到聚类结果。两个节点之间存在连接表示两个实例间的距离小于某个阈值

平均连通（average linkage）是单连通和全连通的折中方案。和之前一样，计算出C_i和C_j中每两个实例对的距离。正如其名字所隐含的，平均连通使用所有实例对代价的平均值，因此，代价为：

$$COST(C_i,\ C_j)=\frac{\sum_{X_i\in C_i,\ X_j\in C_j} dist(X_i, X_j)}{|C_i||C_j|}$$

⊖ 两个向量x和y之间的欧几里得距离公式为：$\sqrt{\sum_i (x_i-y_i)^2}$，其中下标表示向量中第$i$个元素。

其中$|C_i|$和$|C_j|$分别表示簇C_i和C_j中的实例数。使用平均连通得到的簇的样式，很大程度上取决于簇的结构，因为代价的计算基于两个簇中所有实例对的距离的平均值。

平均组连通（average group linkage）和平均连通联系紧密。其代价计算公式为：

$$COST(C_i, C_j) = dist\ (\mu_{C_i}, \mu_{C_j})$$

其中 $\mu_C = \frac{\sum_{x \in C} X}{|C|}$ 是簇C_i的质心（centroid）。一个簇的质心是簇中所有实例的平均。注意，质心是一个和输入实例维度相同的向量。因此，平均组连通使用质心表示簇，使用质心的距离计算代价。使用平均组连通得到的簇和使用平均连通得到的簇类似。

图9-15直观总结了以上介绍的四种代价函数。具体地说，它显示了对图9-11和图9-12中的点计算代价函数时，使用到了哪些实例（质心）对。

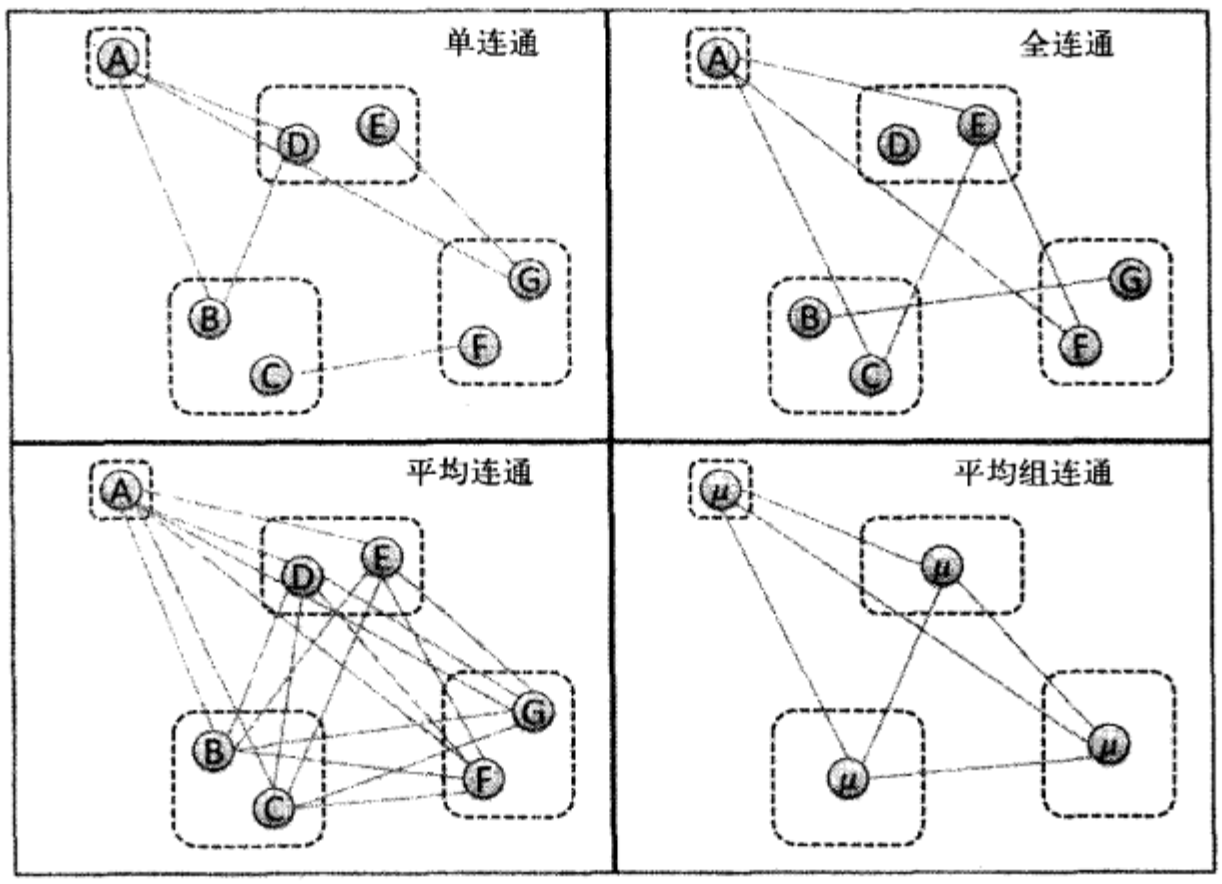


图9-15 解释如何计算各种聚类代价函数

最后介绍华德法（Ward's method）。不像之前的代价函数都是基于两个簇之间的距离，华德法基于方差（variance）统计性质。一个数字集的方差衡量数字的分散程度。华德法尝试最小化簇方差的总和。这样得到的簇会很紧凑，所有的成员最小地分散在簇的质心周围，代价的计算比之前的方法稍微复杂一些：

$$COST(C_i, C_j) = \sum_{k=i,j} \sum_{X \in C_k} (X - \mu_{C_k}) \cdot (X - \mu_{C_k}) + \sum_{X \in C_i \cup C_j} (X - \mu_{C_i \cup C_j}) \cdot (X - \mu_{C_i \cup C_j})$$

其中$C_i \cup C_j$是簇C_i和C_j中实例的并集，$\mu_{C_i \cup C_j}$是簇的$C_i \cup C_j$质心。代价函数衡量了如果将簇i和j合并起来的簇内方差。

上面5种聚合聚类方法哪个最好？答案依然取决于数据集合和具体应用。如果已知数据的

内在结构，那么可以根据这个知识决定哪个算法最好。然而一般需要通过实验和评测确定最好的方法。信息检索实验中使用层次聚类时，平均连通聚类通常是最有效的。虽然聚类是一种无监督的方法，然而最终也没有免费的午餐，可能还需要某种人工评价。

效率是所有的聚类算法需要面对的问题。因为需要计算比较每个实例和所有其他实例，所以对于N个实例，最有效率的方法也需要$O(N^2)$，这限制了聚类应用中的实例数。我们下面介绍的聚类算法，K均值（K-mean），产生一个扁平（flat）的聚类，或划分（partition），而不是层次的聚类，因此效率更高。

2. K均值

K均值算法和前面介绍的层次聚类算法本质上不同。例如，聚合聚类算法对N个簇，根据代价函数，迭代地合并两个（或更多）簇。随着算法的进行，簇的数目不断减少。此外，这个算法的特点是，一旦实例X_i和X_j被合并到一个簇中，那么之后这两个实例就不会再被分开到不同的簇中。

对于K均值算法，簇的数目不会改变。这个算法以K个簇开始，以K个簇结束。每一次迭代中，每个实例要么保留在原有簇中，要么被分配到一个其他的簇中。这个过程不断重复，直到满足某个停止标准。

K均值算法的目标是找到聚类分配方案，表示为分配向量$A[1], \cdots, A[N]$，使得如下代价函数最小：

$$COST(A[1],\cdots,A[N]) = \sum_{k=1}^{K} \sum_{i:A[i]=k} dist(X_i, C_k)$$

其中$dist(X_i, C_k)$表示实例X_i和类C_k的距离。和各种层次聚类算法中的代价函数相同，这个距离度量可选择任何合理的度量方法。但是一般它表示为如下公式：

$$\begin{aligned} dist(X_i, C_k) &= \| X_i - \mu_{C_k} \|^2 \\ &= (X_i - \mu_{C_k}) \cdot (X_i - \mu_{C_k}) \end{aligned}$$

即X_i和μ_{C_k}欧几里得距离的平方。其中μ_{c_k}是簇C_k的质心。注意，这个距离度量和用于聚合聚类的华德法中使用的代价函数相似。因此，这个方法尝试使簇内方差最小。

作为另一种选择，X_i和μ_{c_k}的余弦相似度也可以作为距离度量的一种方法。如第7章所述，余弦相似度衡量两个向量的夹角。对于某些文本应用，余弦相似度证明比欧几里得距离更有效。这种形式的K均值称为球面K均值（spherical K-mean）。

解决这个优化问题最朴素的方法之一是，尝试组合所有可能的聚类分配。然而，对于大规模数据集，这个方法是不可行的，因为它需要指数级的计算复杂度。K均值算法放弃查找全局最优解，使用一个近似的启发式的方法，使迭代的计算复杂度最小化，这个方法不能保证得到全局最优解。事实上，它甚至不保证局部最优。尽管这个算法是启发式的，但是在实际中却很有效。

算法2给出了K均值的一种可实现的伪代码。这个算法开始首先初始化聚类分配方案，或者随机分配实例到各个簇中，或者利用有关数据的一些知识，决定更有根据的分配方案。然后，算法如下迭代进行：根据距离度量dist(X_i, C_k)，每一个实例被分配到最近的簇中。变量change记录本次迭代中每个实例所属的簇是否改变。如果某些实例所属的簇改变了，那么算法继续进行，否则算法终止。另一个合理的停止标准是确定算法的迭代次数。

算法2 K-均值聚类

```
1: procedure KMeansCluster(X_1, ..., X_N, K)
2:     A[1], ..., A[N] ← initial cluster assignment
3:     repeat
4:         change ← false
5:         for i = 1 to N do
6:             k̂ ← arg min_k dist(X_i, C_k)
7:             if A[i] is not equal k̂ then
8:                 A[i] ← k̂
9:                 change ← true
10:            end if
11:        end for
12:    until change is equal to false return A[1], ..., A[N]
13: end procedure
```

实际中，K均值聚类往往很快收敛到一个解。虽然它不保证得到最优解，但是得到的解常常是接近最优。与层次聚类相比，K均值更加有效。特别地，由于每次迭代需要KN次距离计算，迭代次数很小，K均值的复杂度为$O(KN)$，而不是层次聚类的$O(N^2)$。尽管K均值产生的簇依赖于起始点选择（初始簇）和数据输入顺序，但是它产生的簇的质量一般和层次聚类相似。因此，对于很多种与搜索引擎相关的任务，尤其是大数据集，K均值都是一种通用聚类算法。

9.2.2 K近邻聚类

虽然从算法角度看层次聚类和K均值聚类是不同的，但是它们的一个共同点是，它们把每一个输入分配到唯一的簇中，即簇之间不重叠[⊖]。因此，这些算法将输入实例划分（partition）到K个部分（簇）中。然而，对于某些任务，允许簇互相重叠可能有用。产生互相重叠的簇的一种非常简单的方法，称为K近邻聚类（K nearest neighbor clustering）。一定要注意，这里的K和K均值中的K很不相同，这一点很快会变得很明了。

在K近邻聚类中，围绕每一个输入实例形成一个簇。对于输入实例x，根据某个距离衡量标准，距离x最近的K个点和x本身构成一个簇。图9-16给出一些例子，对于点A、B、C和D形成了K=5的近邻簇。虽然图中只显示了围绕这四个实例的簇，实际上每一个输入实例都对应一个簇，因此会产生N个簇。

如图9-16所示，这个算法常常无法找到有意义的簇。在输入空间的稀疏区域，如D的周围，分配给D的点距离D很远，很可能不应该和D分配在同一个簇中。然而对于输入空间的密集区域，如B的周围，簇可以很好地定义，尽管一些相关的输入由于K太小被排除在外。使用K近邻聚类的应用，通常强调找到少数紧密关联的实例（如准确率），而不是所有紧密相关的实例（召回率）。

K近邻聚类可能代价非常高，因为它需要计算每一对实例的距离。如果假设计算两个输入实例的距离需要与K和N相关的常量时间，那么这种计算需要$O(N^2)$。得到这些距离后，需要$O(N^2)$的时间得到所有点的K近邻。因此K近邻聚类总的时间复杂度为$O(N^2)$，和层次聚类相同。

对于某些应用，K近邻聚类是最好的选择，尤其当任务的输入空间很密集，同时为每一个输入找到一些相关实例很有用或很重要时。这些任务包括语言模型平滑、文档分值平滑和伪

⊖ 注意，从树状图的某一层的簇来看这是真的。树状图中，不同层的簇却互相重叠。

相关反馈。我们后续会介绍如何将聚类应用于平滑。

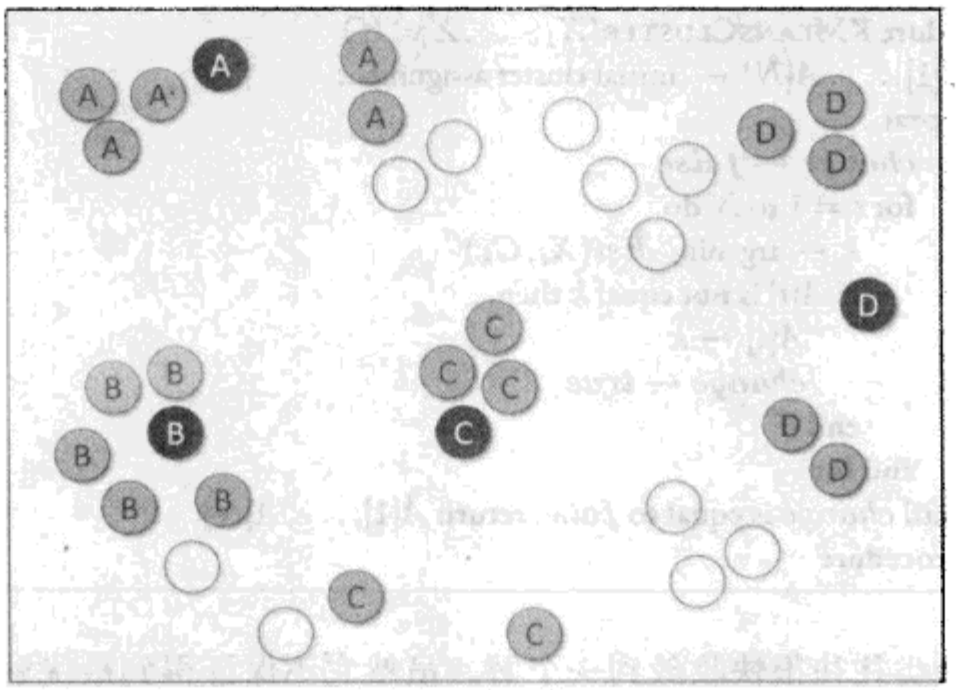


图9-16 使用K=5的近邻算法得到可重叠的簇的例子。图中给出黑点（A、B、C和D）互相重叠的簇。每个黑点的5个最近邻点染成灰色并且标记出来

9.2.3 评价

评价一个聚类算法是很有挑战的。由于聚类是一种无监督学习算法，往往没有或者只有很少用于评价的标注数据。如果没有标注的训练数据，有时可以使用一个目标函数，比如聚类算法尝试最小化的那个目标函数，从而评价聚类结果的质量。然而这是一个“先有鸡或先有蛋”的问题，原因是算法定义了评价指标，反之亦然。

如果存在一些标注数据，那么可以使用标准信息检索评价方法(如准确率和召回率)来评价聚类质量。聚类算法为每个输入实例分配一个簇标识符，簇标识符是任意的，没有明确的意义。例如，将一些电子邮件聚类成两个簇，一些邮件被分配到簇标识符1，而其他的被分配到簇2。不但簇标识符没有任何含义，簇本身也可能不存在有意义的解释。例如，有人可能希望一个簇对应垃圾邮件，另一个簇对应非垃圾邮件，但可能并不属实。因此，定义准确率、召回率时必须谨慎。

一个常用的衡量准确率的方法如下。首先，算法将输入实例聚成K=|C|个簇。然后，对每一个簇C_i，定义MaxClass(C_i)表示和C_i中大多数实例相关的类标签（由人指定）。由于在簇C_i中，与其他类标签相比，更多的相关实例的标签为MaxClass(C_i)，因此假定它是正确的标签。因此，簇C_i的准确率为C_i中标签为MaxClass(C_i)的实例比例。这个衡量指标常常使用所有实例的微平均（microaveraged）来表示，公式如下：

$$\text{簇准确率} = \frac{\sum_{i=1}^{K} |\text{MaxClass}(C_i)|}{N}$$

其中|MaxClass(C_i)|表示簇C_i中标签为MaxClass(C_i)的实例数。如果每个簇对应一个类标签，并且簇中的每一个成员的标签和这个标签相同，那么这个衡量值为1。

在很多搜索应用中，聚类只是其中一种使用到的技术。一般来讲，聚类算法的输出结果作为某个复杂的端对端系统的一部分使用。在这些情况下，分析和评价聚类算法对整个端对端系统的影响是很重要的。例如，如果聚类作为网络搜索引擎的一部分，用以提高排序质量，

那么可以通过评价它对排序效果的影响，来衡量和调节聚类算法。这样做可能会很困难，因为端到端系统往往很复杂，很难理解，并且可能存在很多影响排序的因素。

9.2.4 如何选择K

到目前为止，我们在很大程度上忽略了如何选择K。在层次聚类和K均值聚类中，K表示簇的数目。在K近邻平滑中，K表示使用的近邻数目。虽然这两种情况本质上不同，但是都很难完全自动地设定合适的K值。选择K的问题是聚类中最困难的问题之一，因为实在没有好的解决方案。不存在一个神奇的公式，可以确定任何情形下簇的最优数目。相反；K的最优选择很大程度上依赖于具体的任务和数据集。因此，K常常以实验的方式选择。

在某些情况下，应用本身会表明应该使用的簇的数目。但是这种情况很少见。大多数情况下，应用本身不会提供最优K值的线索。事实上，甚至K的范围也可能不明显。应该使用2个簇？10个？100个？1 000个？没有更好的办法去选择K，只能不断地实验，并且评价在K不同时聚类结果的效果。

对于层次聚类，可以创建整个簇的层次结构，然后根据某种机制，决定使用哪一层作为聚类结果。大多数情况下，簇的数目由人选择，甚至对于层次聚类也是如此。

形成K近邻簇后，可以使用一个有适应性的K。意思是，对于非常密集的区域中的实例，使用一个很大的K可能会有用，因为近邻点有可能相关。类似地，对于非常稀疏的区域，最好只选择很少个近邻点，因为K值太大可能会导致一些无关的近邻点包含进来。这个想法和巴尔森窗式法（Parzen window）[⊖]紧密相关。巴尔森窗式法是分类中K近邻的一种变形。在巴尔森窗式法中，近邻点的数目不固定。相反，某一个固定距离（“窗口”）内的所有实例都认为是近邻点。这样，密集区域的实例将有很多近邻点，而稀疏区域的实例则很少。图9-17给出了使用巴尔森窗式法对图9-16中的点进行聚类的结果。可以看到，更少的离群点被错误地分配到稀疏区域的点（如点C）对应的簇中，而更多的近邻点被分配给密集区域中的点（如点B）。然而，这样形成的簇并不完全正确，聚类的质量此时取决于窗口大小。因此，虽然这种方法不需要选择K，却需要选择窗口大小，而这可能是同样具有挑战性的。

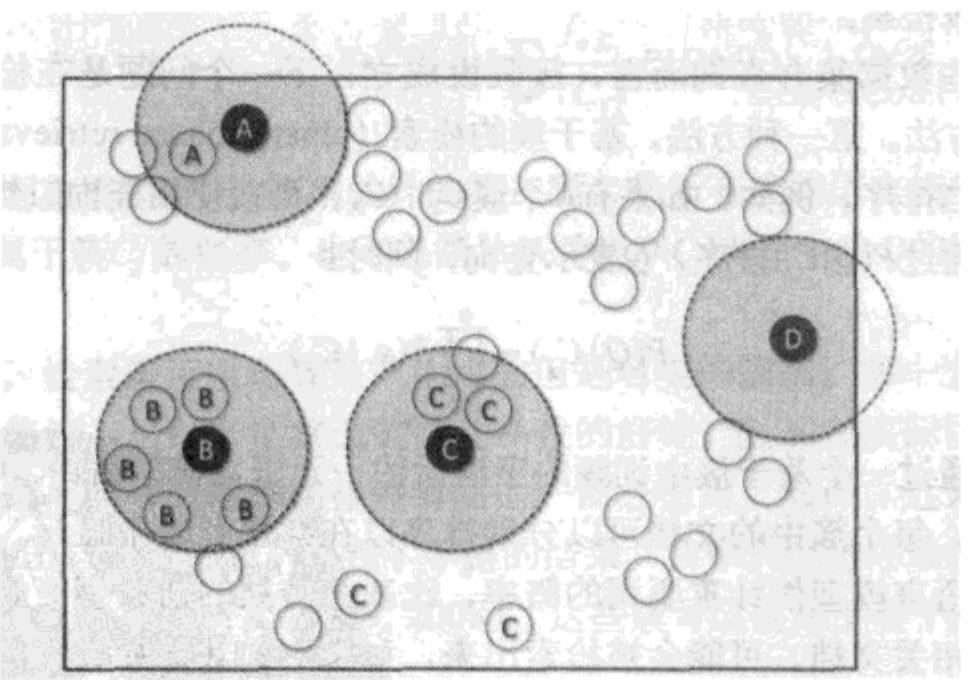


图9-17 使用巴尔森窗式法产生互相重叠的聚类结果。图中给出了对应黑点（A、B、C和D）的簇。阴影圈表示用来确定所属簇的窗口。每个黑点的近邻被染成灰色并且标记出来

⊖ 命名以纪念Emanuel Parzen，一位美国统计学家。

9.2.5 聚类和搜索

聚类算法存在的一些问题导致它们在实际中不像分类算法那样被广泛应用。这些问题包括计算代价，以及难于解释和评价聚类结果。聚类被一些搜索引擎用来组织搜索结果，如6.3.3节所讨论的。与文档集合规模相比，搜索结果的数量是很小的，因此聚类的效率问题便不那么重要了。聚类可以发现任何查询得到的搜索结果的结构，而分类则不可能做到。

在7.6.2节讨论的话题建模，也可以看成是一个聚类应用，用以提高搜索引擎的排序效果。实际上，大多数使用聚类的信息检索研究都关注这个目标。这个研究的基础是有名的簇假设(cluster hypothesis)。如van Rijsbergen(1979)最早所述，簇假设是：

> 紧密联系的文档往往和相同的请求相关。

注意这个假设实际上没有提到簇。然而，“紧密联系的”或者相似的文档，通常在同一个簇中。因此，这个假设通常解释为同一簇中的文档往往和相同查询相关。

两个不同的测试被用来验证簇假设对一个特定文档集是否适用。第一个测试比较相关文档对（针对一组查询）的相似度分值的分布，和由一个相关和一个不相关文档组成的文档对的相似度分值的分布。如果簇假设成立，将会看到这两个分布相互分离。在一些小规模语料上，如第8章提到的CACM语料，情况的确如此。然而，如果存在相关文档的一些簇，这些簇彼此不相似，那么这个测试可能不会显示任何分离。为了解决这个潜在的问题，Voorhees（1985）提出一个测试，所基于的假设是，如果簇假设成立，相关文档会有高的局部准确率(local precision)，即使它们分散到很多个簇中。局部准确率简单地衡量每个相关文档前五个近邻点中属于相关文档的数目。

图9-18给出在两个TREC数据集上的簇假设测试结果。这些数据集有相似类型的文档，包括很多新闻报道。从典型MAP值来看，数据集robust上的250个查询比trec12上的150个查询要难。图上面的两个测试显示对于两个数据集，相似度值分布的分离很小。然而下面的两个测试显示trec12上相关文档有高的局部准确率。robust上的局部准确率低一些，说明相关文档更加孤立，因此更难被检索。

假设至少对一些数据集合查询而言，簇假设成立，下一个问题是在检索模型中如何采用它。事实上有很多方法。第一种方法，基于簇的检索（cluster-based retrieval），对簇进行排序，而不是对单独的文档排序。例如，如果有K个簇$C_1 \cdots C_K$，可以使用查询似然度检索模型对簇排序。这意味着可以通过$P(Q|C_j)$排序，Q表示查询，同时：

$$P(Q \mid C_j) = \prod_{i=1}^{n} P(q_i \mid C_j)$$

概率$P(q_i|C_j)$是通过一个基于簇中词频的平滑后的一元语言模型估计，如第7章对文档排序那样。簇排序之后，每个簇中的文档可以分别排序以在结果列表中显示。这种排序方法背后的直觉是，一个簇语言模型估计重要词的概率，比基于文档的估计会更好。事实上，一个与查询没有相同项的相关文档，可能会被检索出来，如果它和其他相关文档属于一个排序很高的簇的话。

除了使用这种两阶段的过程，簇语言模型还可以按照如下方式直接嵌入到文档语言模型：

$$P(w|D)=(1-\lambda-\delta)\frac{f_{w,D}}{|D|}+\delta\frac{f_{w,C_j}}{|C_j|}+\lambda\frac{f_{w,Coll}}{|Coll|}$$

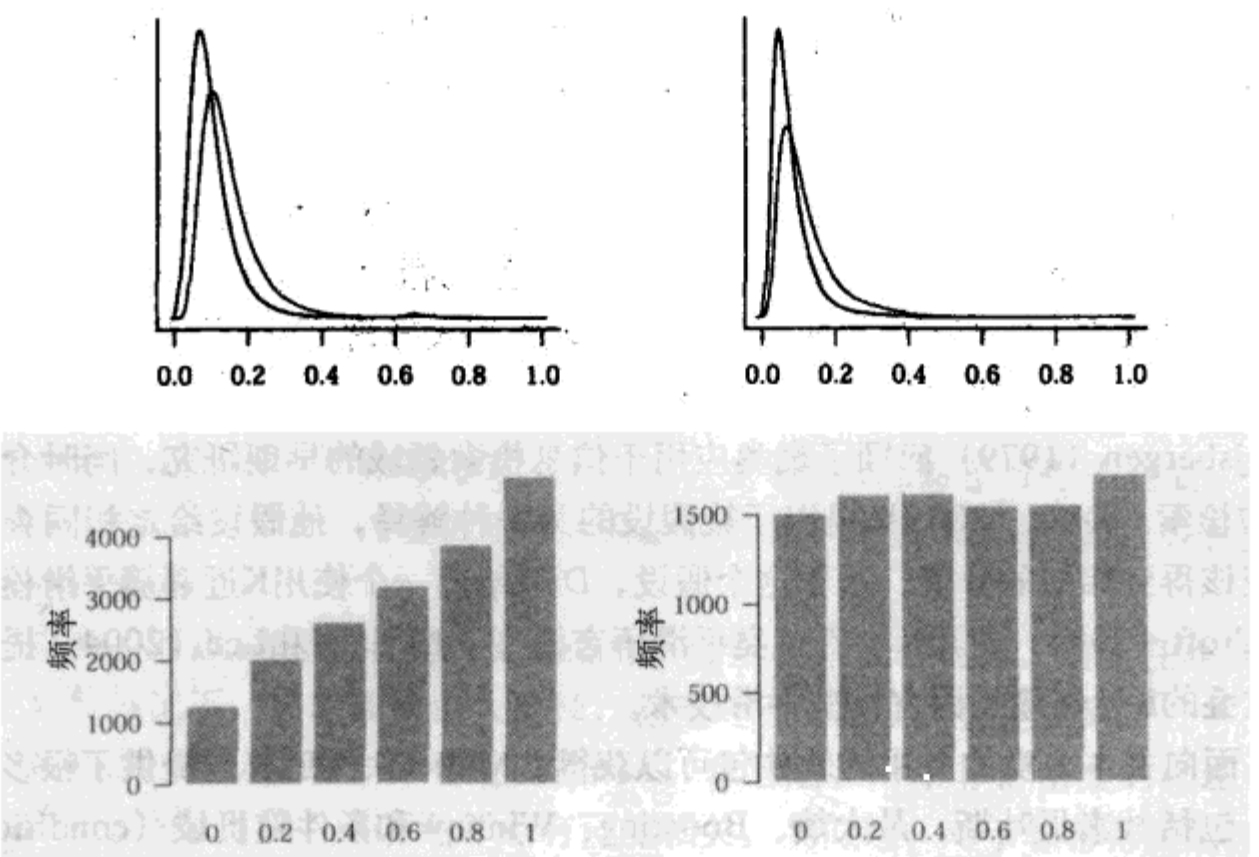


图9-18 两个TREC数据集上的簇假设测试。上面两个图比较“相关-相关”和“相关-不相关”（浅灰色）文档的相似度值分布。下面两个图给出相关文档的局部准确率

其中，λ和δ为参数，$f_{w,D}$是文档D中的词频，f_{w,C_j}为包含D的簇C_j的词频，$f_{w,Coll}$是数据集$Coll$上的词频。第二项来自于簇语言模型，它增加了在簇中频繁出现并且有可能和文档主题相关的词的概率。换句话说，簇语言模型使这个文档与簇中的其他文档更相似。这个基于簇平滑的文档语言模型可以被查询似然检索模型直接采用，如7.3.1节介绍的那样，对文档进行相关排序。

文档语言模型可以进一步推广到当文档D属于多个相互重叠的簇的情况：

$$P(w|D)=(1-\lambda-\delta)\frac{f_{w,D}}{|D|}+\delta\sum_{C_j}\frac{f_{w,C_j}}{|C_j|}P(D|C_j)+\lambda\frac{f_{w,Coll}}{|Coll|}$$

这种情况下，文档语言模型对词w概率估计的第二项，是所有簇的语言模型概率的加权和。权值($P(D|C_j)$)是文档属于簇C_j的概率。也可以简单认为$P(D|C_j)$对于包含D的簇都相同，而对其他簇为0。

检索实验证明，检索簇可以对效果产生小的但是可变的提高。另一方面，利用基于簇的估计平滑文档语言模型，可以产生重大的并且一致的好处。然而在实际中，产生簇的代价意味着它在实际的搜索应用中无法作为排序算法的一部分使用。然而，近来人们使用查询相关的聚类（query-specific clustering）得到了不错的结果，其中只对排序最靠前的（如50个）文档进行聚类（Liu & Croft, 2008; Kurland, 2008）。这些簇可以被用于基于簇的检索或者文档的平滑。很明显，这种方法的效率要高得多。

参考文献和深入阅读

分类和聚类在统计学、模式识别和机器学习这些研究领域都有深入研究。Duda等（2000）和Hastie等（2001）的书中，都介绍了范围广泛的分类和聚类技术，并且提供了很多这些技术的数学基础细节。这些书也提供了很多其他有用的机器学习技术的综述，这些技术可以在很多搜索引擎任务中应用。

如果需要更细致地了解朴素贝叶斯分类器在文本分类中的使用，请阅读McCallum和Nigam（1998）。C.J.C. Burges（1998）提供了SVM非常详细的指南，覆盖了所有基本概念和理论。但是，它的内容很难，需要一定水平的数学基础才能深入理解。此外，Joachims（2002a）这本书很全面地介绍了SVM在文本分类中的各种用法。

van Rijsbergen（1979）回顾了聚类应用于信息检索领域的早期研究，同时介绍了簇假设和基于簇的检索。Diaz（2005）提出了簇假设的另一种解释，他假设给定相同查询，紧密相关的文档应该得到相似的分值。基于这个假设，Diaz给出一个使用K近邻簇平滑检索分值的框架。Liu和Croft（2004）使用K均值聚类平滑语言模型，Kurland和Lee（2004）提出另一种基于可相互重叠的K近邻簇的语言模型平滑技术。

有很多面向文本分类的有用的软件包可以获得。Mallet软件工具包提供了很多机器学习算法的实现，包括朴素贝叶斯、最大熵、Boosting、Winnow和条件随机域（conditional random fields, CRF）。它也对文本进行解析或词素切分提供支持，从而将其转化为特征。另外一个流行的软件包是SVMLight[⊖]，它实现了SVM，支持本章中介绍过的所有核函数。很多包含聚类方法的软件包在网上都可以获取。

练习

9.1 给出一个例子，说明人们在日常生活中如何使用聚类方法。他们在表示对象时使用了哪些特征？采用了什么相似度度量方法？他们如何评价输出？

9.2 假设希望使用一个极细粒度的本体做分类，例如一个描绘了所有人类语言的本体。假设在训练之前，决定将所有亚洲语言对应的标签替换成一个名为“亚洲语言”的标签。讨论一下这个决定的负面影响。

9.3 假设根据如下公式估计$P(d|c)$：

$$P(d|c)=\frac{N_{d,c}}{N_c}$$

其中$N_{d,c}$是训练数据中文档d分配给类别c的次数，N_c是训练数据中分配给类别c的实例数。这和估计$P(c)$类似。为什么这种估计在实际中不可用？

9.4 对于某个分类数据集，使用多重伯努利和多项式模型估计所有词w的$P(w|c)$。比较多重伯努利模型和多项式模型的估计结果。他们有哪些不同？是否对于某些类型的词，它们的估计差异更大一些？

9.5 解释为什么原始SVM公式 $w=\arg\max_w \frac{2}{\|w\|}$ 的解与另一个公式 $w=\arg\min_w \frac{1}{2}\|w\|^2$ 的解等价？

⊖ http://mallet.cs.umass.edu/。

9.6 在一个多类分类数据集上，比较“一对所有”和“一对一”SVM分类器的准确率。从效果和效率的角度讨论这两个方法的不同。

9.7 什么条件下微平均等于宏平均？

9.8 使用五种聚合聚类方法将下面的二维实例集合聚成三类：

$$(-4,-2),(-3,-2),(-2,-2),(-1,-2),(1,-1),(1,1),(2,3),(3,2),(3,4),(4,3)$$

讨论这些方法的结果的差异。哪些方法得到相同的结果？和人工聚类过程相比，这些结果有何不同？

9.9 使用K均值和球面K均值对练习9.8中的点聚类，比较结果有何不同。

9.10 近邻簇不对称，意思是如果A是实例B的近邻之一，B则不一定是A的近邻之一。使用图解法解释这种情况如何发生。

9.11 一个文档的K近邻文档可以表示为指向这些文档的链接。描述在搜索引擎中使用这种表示的两种可能方法。

9.12 评价指标ClusterPrecision可能为0吗？如果是，给出一个例子。如果不是，解释原因。

9.13 使用图9-18中的两种方法，在CACM数据机上测试簇假设。可以从这些测试中得到什么结论？

第10章　社会化搜索

“你将被同化。”

——Borg Collective，《星际迷航之战斗巡逻》

10.1　什么是社会化搜索

本章将介绍社会化搜索（social search）。社会化搜索正在迅速成为互联网上关键的搜索模式之一。正如它的名字所表达的，社会化搜索处理特定社会环境内的搜索。这可以被定义为一种环境，社区（community）的用户在其中积极地参与搜索过程。如果非常宽泛地解释社会化搜索的定义，可以使任何具有相关活动的应用都包含进来。这些活动包括，定义用户的描述文件和兴趣，用户间的交互以及修正被搜索对象的表示等。在标准的搜索模式和模型中，所有用户被等同看待，重构查询的交互方式受限。与其形成鲜明对比的是，在社会化搜索中，用户充当了更活跃的角色。

用户彼此之间可以通过多种方式进行在线交互。例如，用户可以访问社会媒体站点[⊖]，这种形式近来获得了很大的人气。其中具有代表性的站点包括Digg（网站）、Twitter（状态消息）、Flickr（图片）、YouTube（视频）、Del.icio.us（书签）、CiteULike（科研论文）。社交网络站点，如MySpace、Facebook和LinkedIn等。这些站点允许朋友、同事以及具有相似兴趣的人，通过多种方式进行交互。更传统的在线社会化交互方式包括电子邮件、即时通信、大型多人在线游戏及论坛、博客等。

正如我们所看到的，网络世界是非常社会化的环境，包含形式多样的用户交互。社会互动为搜索系统提供了全新而独特的数据资源，同时也有无数的隐私问题。在第7章描述的网页搜索方法，大多仅仅考虑文档特征或网络的链接结构。而在社会化信息丰富的环境中，大量的用户交互信息有助于以新颖、有趣的方式增强用户体验。

或许能够用整本书讲述社会化交互以及如何在其中搜索。本章强调的是从搜索引擎和信息检索的角度描述社会化搜索在某些方面的特点。

本章讲述的第一个话题为用户标签（user tag）。许多社会媒体站点允许用户添加标签。例如，视频共享网站允许用户不仅可以给自己的视频而且还可以给其他用户创建的视频指定标签。例如，一个水下视频可以有“swimming”、“underwater”、“tropic”、“fish”等标签。有些站点允许多词标签，如“tropical fish”，其他站点则仅允许单个词的标签。正如即将介绍的，用户标签是一种人工索引（manual indexing）形式，项目（item）的内容通过人工添加的词项表示。许多有趣的搜索任务与用户标签相关，如利用标签搜索项目、自动标签推荐及标签集合的可视化等。

第二个话题为社区内搜索（searching within communities），将描述在线社区以及用户如何在其中搜索。在线社区是指分享共同兴趣的用户形成的虚拟的用户群组及在线环境下多样

⊖　社会媒体站点普遍指的是Web 2.0，与传统的仅包含非交互式的HTML文档的Web1.0相对应。

的社会交互。例如，一个喜欢户外运动及摄影的体育迷，可能是关于棒球、徒步旅行或是数码相机社区的成员。社区内的交互方式既有被动式的（阅读网页）也有主动式的（博客或论坛写作）。社区是虚拟的也是临时的，这意味着加入社区不必通过某些正式机制，因此人们是隐式的而非显式的社区成员。所以，对一些与搜索相关的任务，自动地确定在线环境中存在哪些社区以及用户是哪些社区的成员，是十分有价值的。其中要描述的一个任务是基于社区的问答，在这里，用户向在线系统提交一个问题，其所在社区的成员或者与其问题最相关的社区会提供答案。这种搜索任务比标准的网络搜索更具社会性、交互性和针对性。

接下来的话题是过滤（filtering）和推荐系统（recommender system）。似乎关于社会化搜索的章节不应包括这些内容，因为它们并不是典型的“Web 2.0”应用。然而，这两种类型的系统都基于对单个用户的表示，称为描述文件（profile），因此也符合社会化搜索的宽泛定义。两种系统也都结合了文档检索和文本分类。在典型的搜索任务中，系统根据不同的查询返回文档。通常情况下，查询对应短期的信息需求。在过滤中，有一个代表长期信息需求的固定查询（描述文件）。搜索系统监测新进入的文档，并从中检索与信息需求相关的文档。许多在线新闻网站提供了文档过滤的功能。例如，CNN提供一种提醒服务，允许用户指定不同的兴趣话题，如“tropical storms”，或更一般的话题，如“体育”或“政治”。当一个新的报道符合用户的描述文件时，系统会以一定方式提醒用户，如电子邮件。通过这种方式，用户不需要持续地搜索感兴趣的文章，而是由系统负责发现符合用户长期信息需求的相关文档。推荐系统与文档过滤系统相似，区别在于其目的是预测用户如何对一个项目进行评分而不是检索相关文档。例如，Amazon.com采用的推荐系统试图预测用户对某些项目的喜欢程度，如书籍、电影或音乐。推荐系统也采用社会化搜索算法，因为预测都是基于相似用户的评分，从而隐式地将用户与具有相关兴趣的用户社区联系起来。

本章最后介绍的两个话题，P2P（peer-to-peer）搜索和元搜索，则是处理社会化搜索的体系结构。P2P搜索针对给定的信息需求向“节点”社区进行查询。节点可以是单个用户、组织机构或搜索引擎。用户递交的查询会在P2P网络中传递，并在其中一个或多个节点上运行，然后将结果返回。这种类型的搜索可以完全分布在大规模的网络节点上。元搜索是一种特殊的P2P搜索，所有的节点都是搜索引擎。元搜索在一定数量的搜索引擎上执行查询、收集结果并将结果融合，其目的在于提供比单个搜索引擎更好的覆盖率和准确率。

最后要指出的是，个性化是另一个可以被视为社会化搜索组成部分的领域，因为它涉及了一系列利用用户个体的偏好和兴趣的表示来改善搜索质量的技术。其中大部分技术都是为查询提供上下文信息，在6.2.5节已经作为查询优化的一部分进行了讨论。

10.2 用户标签和人工索引

在图书馆电子搜索系统出现之前，客户（patrons）借助卡片目录查找书籍。正如名字所暗示的，卡片目录是大的卡片集合。每个卡片包括关于特定作者、题目或主题的信息。对具体的作者、题目或主题感兴趣的读者，可以到适当的目录下尝试找到描述相关书籍的卡片。因此，卡片目录的作用类似于图书馆中的信息索引。

在计算机来做这些工作之前，卡片目录长时间存在，这意味着这些卡片都是人工创建的。对于指定书籍，需要从中提取出作者、标题和主题等信息来构建各种目录。这个过程称为人工索引（manual indexing）。对于当前巨大的数字媒体集合，人工索引是不现实的，因此搜索

引擎采取自动索引（automatic indexing）技术，在构建索引的过程中将标识符（词、短语、特征等）指派给文档。既然这一过程是自动的，索引的质量与准确性可能都大大低于人工索引。然而，自动索引的优势在于它是彻底的（exhaustive），在这个意义上，文档中的每个词都被索引而没有遗漏，并且是一致的（consistent）。而人在索引的时候，会出现错误或对如何索引存在一定的偏差。对人工索引和自动索引进行比较的搜索评价发现，自动索引至少和人工索引一样有效，且经常远远优于人工索引。这些研究同时指出，两种索引方式是彼此互补的，而且在多数有效的搜索中，两种方式都被结合使用。

作为人工索引（图书馆目录）和自动索引（搜索引擎）的一种折中，社会媒体站点为用户提供人工标注项目的机会。典型的标签是描述该项目的一个单词。例如，一张老虎的图片可以被指派标签“tiger”、“zoo”、“big”、“cat”。由于由用户指派标签，一些项目被标注，其他的则没有被标注。为了使每一项目都可以被搜索，可以自动索引所有项目。因此某些项目既包括自动产生的标识符也包括人工添加的标识符。正如我们将会在本节后面介绍的，这给检索模型和相关排序函数带来独特的挑战。

社会媒体标注与卡片目录生成一样，称为人工标注（manual tagging）。这种命名冲突是不幸的，因为这两种索引方式实际上是非常不同的。卡片目录由专家人工生成，这些专家从后控词表（controlled vocabulary）（固定本体）中选取关键字、类别及其他描述。这确保获得的描述是相对标准的。与之相比，任意用户都可进行社会媒体标注，这些用户未必都是专家，因此用户标签的质量几乎是不可控的。不仅如此，并没有固定的词表提供给用户，令用户从中选择标签。用户标签自身形成对某一领域中重要概念和关系的描述。用户生成的本体（或分类体系）称为大众分类（folksonomy）。因此，大众分类可以解释为动态的、受社区影响的本体。

已有很多研究或兴趣致力于开发一种语义标注的网络版本，通常称为语义网（semantic web）。语义网的目的在于从语义层次标注网页的内容，从而能够更容易地查找、组织和共享信息。例如，一篇新闻文章可以用元数据进行标注，如标题、主题、描述、出版商、日期和语言等。然而，为使语义网能够实现并明显提高相关性，必须在大规模网页上持续地开发和使用标准化的、固定的元数据标签本体。与基于灵活的、用户驱动的大众分类的社会媒体站点日益普及相比，基于严格的、预先定义本体的语义网的数量则少得多。看起来用户对使用相对限制较少的标签集合来对数据进行标注的方式持更开放的态度。这些标签集合对用户来说是有意义的，而且反映具体的应用情况。

既然允许用户使用任何喜欢的方式去标注项目，因此存在许多不同的标签类型。例如，Golder和Huberman（2006） 描述了标签的不同分类。Z. Xu等（2006）提出了简化的五种标签分类，包括：

1）基于内容的标签。这类标签描述项目的内容。例如：“car”、“woman”和“sky”。

2）基于上下文的标签。这类标签描述一个项目的上下文。例如：“New York City”或“Empire State Building”。

3）属性标签。这类标签隐含项目的属性。如“Nikon”（相机的类型）、“black and white”（电影的类型）或者“homepage”（网页的类型）。

4）主观标签。这类标签主观地描述一个项目。如：“pretty”、“amazing”和“awesome”。

5）组织标签。这类标签能够帮助组织项目。例如：“todo”、“my pictures”和“readme”。

正如所见，标签可以应用到不同类型的项目上，从网页到视频。利用标签可以达到许多不同的目的，而不仅仅是标注内容。因此，标签和在线协同标注环境对用户搜索、组织、共享和发现新信息是非常有用的工具。标签可能会长期存在，甚至在未来会有更广泛的应用。对现在来说，理解标签周边的问题以及搜索引擎如何利用标签是非常重要的。

在本节的后续部分，将会介绍如何利用标签来进行搜索，如何通过已有的标签为指定项目预测新标签，以及如何将标签集合可视化以呈现给用户。

10.2.1 搜索标签

既然这是一本关于搜索引擎的书，下面要讨论的第一个与标签相关的任务即是在协同标注的项目集合中进行搜索。标签的一个独特性质是基本上是文本关键词，这些关键词用来描述文本或非文本项目。因此，标签可以为无法显式表示为文本的项目，如图片、视频等，提供一种文本维度。这些非文本项目的文本表示对搜索是非常有用的。可以应用许多第7章叙述的检索策略来解决这一问题。除了在标注的集合中搜索可以转化为文本搜索这一事实外，标签面临着在处理标准文档检索或网页检索时没有出现过的某些挑战。

第一个挑战，也是到目前为止最普遍的一个事实是，对于复杂的项目，标签是非常稀疏的表示方式。搜索标注的项目集合最简单的方法可能是布尔检索模型。例如，给定查询“fish bowl”，可以构造查询“fish AND bowl”，系统将仅返回既包括“fish”又包括“bowl”的项目。或者可以构造查询“fish OR bowl”，系统将返回或者被标注为“fish”或者被标注为“bowl”的项目。合取（AND）查询可能产生高质量的结果，但是会损失许多相关项目，所以这种方法会导致高准确率但是召回率低。在另一方面，析取（OR）查询将会匹配更多的相关项目，但是会以牺牲准确率为代价。

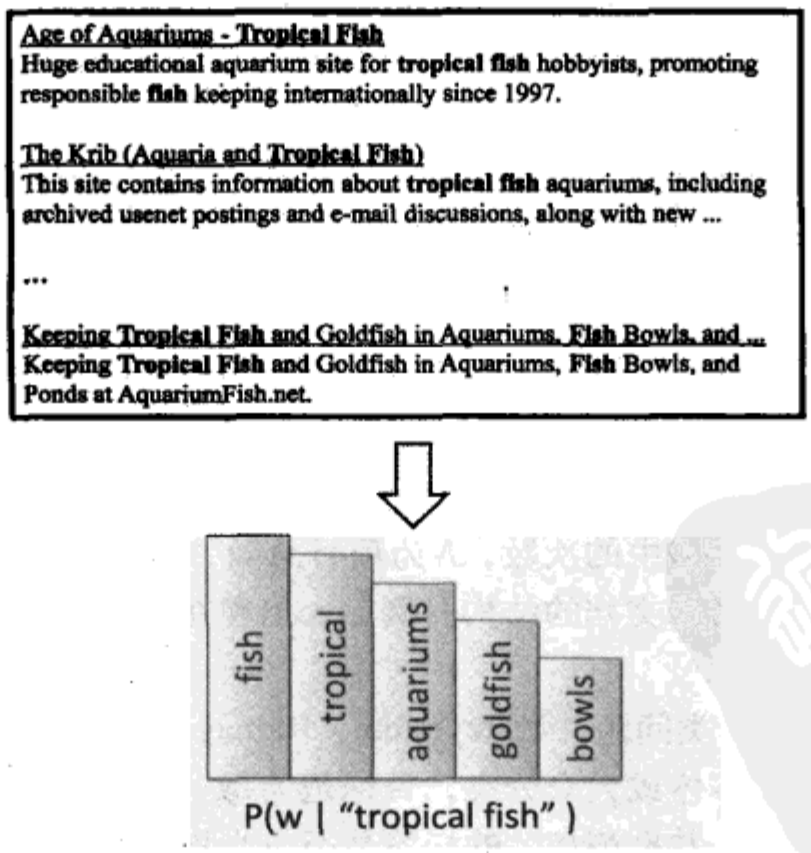


图10-1 利用搜索结果丰富标签表示。在这个例子中，被扩展的标签为“tropical fish”。将“tropical fish”作为查询在搜索引擎搜索，利用返回的页面摘要生成相关词项的分布

当然，高精确率与高召回率都是需要的，然而要做到这点却是很大的挑战。考虑查询

“aquariums”和一幅被标注为“tropical fish”与“goldfish”的鱼缸图片。多数检索模型，包括布尔检索，无法找到这幅图片，因为查询关键词与标签关键词之间没有重叠。这一问题在第6章关于广告的内容中，被描述为词表不匹配（vocabulary mismatch）问题。有很多方法试图克服这一问题，简单的方法如提取词干。还有方法试图通过伪相关反馈丰富稀疏标签（或查询）的表示。图10-1示范了如何利用网页搜索结果来丰富标签的表示。在这个例子中，标签“tropical fish”作为一个查询在搜索引擎中检索。返回的页面摘要（snippet）使用7.3.2节中任何一种标准的伪相关反馈技术进行处理，如相关性模型，该模型形成相关词项分布。在这个例子中，“fish”、“tropical”、“aquariums”、“goldfish”和“bowl”在该模型中具有最高的概率。查询也可以以这种方式进行扩展。搜索时可以使用扩展的查询和/或标签的表示，以达到较高的准确率与召回率。

第二个挑战是标签含有内在的噪声。正如前文所述，标签可以提供关于项目的有用信息并提高搜索的质量。然而，由于是由用户创建的，这些标签可能是偏离主题的、不适当的、拼写错误的甚至是垃圾信息。因此给予适当的奖励以使用户提交高质量的标签是非常重要的。例如，可以允许用户报告不恰当的或垃圾标签，从而减少生成的垃圾标签。如果用户能在一段时间内输入一定数量的（非垃圾）标签，甚至可以令他们升级或给予其更高的权限。激励策略可以促进产生更多标签，进而提高标签质量与覆盖率。

最后的挑战是，指定集合中的很多项目并没有被标注，这使得对于基于文本的搜索系统来说，这些项目是不可见的。对于这些项目，自动地预测缺失标签并使用预测的标签提高搜索的召回率，是非常有价值的，这个问题将在下一节深入探讨。

10.2.2 推测缺失的标签

如前文所述，没有标签的实体对搜索系统提出了挑战。对于许多与标签相关的搜索任务，准确率显然非常重要，但是在某些情况下，召回率同等重要。在这种情况下，为那些没有人工指派标签的项目自动推测标签，是十分重要的。

首先考虑集合中的项目都是文本的情况，如书籍、报纸、科研论文和网页等。在这种情况下，可以仅仅依靠项目的文本表示来推测标签。一个简单的方法是，对出现在文本中的每一个词项计算权值，然后选择K个权值最高的词项作为预测的标签。有许多评价词项重要性的方法，包括基于tf.idf的权值估计，如下面的公式所示：

$$wt(w) = \log(f_{w,D} + 1)\log\left(\frac{N}{df_w}\right)$$

其中$f_{w,d}$为词项w出现在项目D中的次数，N为项目总数，df_w为包含词项w的项目数目。其他词项重要性评价方法可能考虑文档的结构。例如，出现在项目标题中的词项会被赋予更高的权值。

标签推测问题也视为分类问题，如Heymann、Ramage和Garcia-Molina（2008）最近提出的方法。给定标签的固定本体或大众分类，目标是为每一个标签训练出一个二元分类器。分类器以一个项目作为输入，并推测相关的标签是否应当指派给该项目。这种方法需要为每一个标签训练一个分类器，这可能是繁重的任务并需要大量的训练数据。然而幸运的是，针对这个任务的训练数据近乎是唾手可得的，因为用户在持续地人工标注项目。所以可以使用所有已有的标签/项目对作为训练数据来训练分类器。Heymann等使用SVM分类器来推测网页标

签。他们计算了一些列特征包括网页文本和锚文本中词项的tf.idf值，以及一些基于链接的特征。实验结果表明，仅通过文本特征即可获得高准确率和高召回率，尤其是那些在集合中出现多次的标签。简单的分类方法可以应用于其他类型的项目上，如图片或视频。处理非文本项目的挑战，是从项目中提取有用的特征。

已经介绍的推测缺失标签的两种方法，在为项目选择标签时，不考虑已经指派的标签。这导致指派给某些项目的标签虽然相关但非常冗余。例如，一幅儿童图片可能有以下标签："child"、"children"、"kid"、"kids"、"boy"、"boys"、"girl"、"girls" 等。所有这些标签都是相关的，但显然也是十分冗余的。因此，选择既相关又不冗余的标签是非常重要的。这称为新颖性（novelty）问题。

Carbonell和Goldstein（1998）提出了最大边缘相关（Maximal Marginal Relevance，MMR）技术，阐述了如何选择一组多样的项目这一问题。MMR选择标签时，标签之间不是彼此独立，而是迭代地选择标签，一次为项目添加一个标签。给定一个项目i及相应的标签集合T_i，MMR技术选择下一个标签t时，最大化下面的公式：

$$MMR(t;\ T_i) = \left(\lambda Sim_{item}(t,\ i) - (1-\lambda)\max_{t \in T_i} Sim_{tag}(t_i,\ t)\right)$$

其中Sim_{item}是度量标签t与项目i相似度的方程（如在本节描述过的方法），Sim_{tag}度量两个标签间的相似度（如在10.2.1节中叙述的方法），而λ为可调系数，用来平衡相关性（λ=1）与新颖性（λ=0）。因此，与项目非常相关而与任何其他标签不相似的标签获得较大的MMR值。以这种方式迭代地选择标签，可以消除大量冗余。这不仅在将预测的标签展示给用户时有用，从利用预测的标签进行搜索的角度，也是非常有用的，因为具有多样性的标签集合进一步提高了召回率。

10.2.3 浏览和标签云

如前文所述，标签可以用来搜索协作标注的项目集合。标签也可以用来帮助用户在一个大的项目集合中浏览、探索及发现新的项目。可采取几种不同的方式利用标签帮助浏览。例如，当用户正在关注特定的项目时，这个项目的所有标签都被显示。这时，用户可能会点击其中一个标签，接着一组被标注有该标签的项目集合呈现给用户。用户可能不断重复这一过程，不断地选择一个项目，点击其中一个标签。这使得用户可以通过一个相关标签链来浏览项目集合中的项目。

这种浏览行为非常有针对性，而且不会使用户浏览集合中很大一部分项目。例如，如果一个用户由一个关于"tropical fish"的图片开始，用户需要多次点击才有可能到达一幅关于一本信息检索书籍的图片。当然，这是可取的，尤其是当用户仅仅对与"tropical fish"密切相关的项目感兴趣时。

一种将对于整个集合的全局视图提供给用户的方法是，让其看到最流行的标签。这些标签可能是整个站点或者某个特别的群组，或者是某类项目中最流行的标签。标签的流行度可以通过许多不同方法进行度量，但是最常用的方法是，计算标签指派给某些项目的次数。显示流行度的标签可以使用户以其中一个标签为起点浏览和探索项目集合。

图10-2 标签云的加权列表形式举例。标签按照字母表顺序，权值计算按照某种标准，如流行度

到目前为止，已经描述了利用标签帮助用户浏览的方法。对于浏览，最重要的方面之一是，以一种形象的并且有意义的方式将一组标签显示给用户。例如，考虑呈现50个最流行的标签给用户。最简单的方法即是按照字母序列或流行度的排序，以列表或表格的方式进行显示，这种方式不够形象化且不利于用户迅速观察到所有相关的信息。当显示标签时，按照字母表顺序显示标签列表是有用的，用户可以迅速浏览这个列表或找到他们正在寻找的标签，而且有利于描绘标签的流行度或重要性。有很多方式可以使这些信息形象化，其中应用最广泛的技术称为标签云（tag cloud）。在一片标签云中，标签显示的尺寸与它的流行度或重要性成正比。标签可能以随机顺序或按照字母顺序分布在“云”中。图10-2展示了一个标签云的例子，这里，标签按照字母顺序排列。这种表示也称作加权列表。基于这种标签云，用户可以清楚地看到标签“wedding”、“party”和“birthday”都是非常流行的。因此，标签云提供了一种方便、形象的方式来表示一组标签。

10.3 社区内搜索

10.3.1 什么是社区

刚才描述的协同标注的环境充满了隐式的社会交互。通过分析用户提交和搜索的标签，可以发现具有相关兴趣的用户群体。例如，冰球球迷可能会标注他们喜爱的冰球运动员的图片，标注他们最喜欢的冰球网页，搜索和冰球相关的标签等。标签只是利用在线互动推测实体（人）之间关系的实例之一。处于在线环境中进行交互且具有共同的目的、特性或兴趣的实体群组，形成在线社区（online community）。这个定义与传统社区的定义并没有本质区别。实际上，在线社区和传统社区非常相似，有许多相同的社会形态。此处的定义与传统社区的定义的主要差异是，在线社区可以由用户、组织、网页或任何有意义的在线实体构成。

回到标注和搜索与冰球有关的项目的例子。冰球球迷形成了一个在线社区，社区成员除了标注和搜索，还参与许多其他活动。例如，他们会发布内容到博客、新闻组和其他的论坛。他们也会发送即时消息或电子邮件给社区内的其他成员，讨论有关冰球的体验。甚至，他们还可以通过类似eBay这样的站点，在线购买或销售有关冰球的物品。所以，一个用户可以有许多方式参与到社区中来。但需要注意的是，用户的社区成员身份是隐式的。另一点需要注

意的是，用户往往具有多种爱好和兴趣，可能是多个在线社区的成员。因此，为了提高整体的用户体验，对于搜索引擎和其他在线站点来说，自动地确定与指定与用户关联的社区是非常有用的。

一些在线社区不仅仅包括人。例如，一组关于相同主题的网页形成一个在线社区，经常称为网页社区（web community）。这些页面由于具有相似的特点（例如都属于同一主题）而形成社区。既然网页是由人创建的，网页社区与由人构成的社区具有很多相同的特点。自动地识别网页社区对于提高检索质量也是十分有益的。

本节余下部分从搜索引擎的角度，讲述一些在线社区方面的内容。首先，介绍一些自动发现在线社区的有效方法，之后将讨论基于社区的问答，在这里人们提出问题并从社区的其他成员那里获得答案。最后将介绍协同式搜索，这是一种用户群体共同搜索的搜索模式。

10.3.2 社区发现

下面将介绍的第一个任务是如何自动地发现在线社区。正如前文提到的，在线社区隐式地定义为一组具有共同特性的实体及它们之间的交互。这个定义相当模糊，因而很难设计出通用算法去发现每一种类型的在线社区。相反，一些算法已被开发来，能有效地发现具有特定性质的、具体类型的社区。现在将要描述这些算法。

用于社区发现的算法多数以一组实体作为输入，如用户或者网页，描述这些实体的信息以及这些实体交互或相互关联的细节。这可以由图方便地表示。在图中，实体作为节点，实体之间的交互（或关系）作为边。图既可以是有向图也可以是无向图。有向图中的边具有方向，箭头用来标识边的起点和终点。无向图中的边则没有方向，箭头用来标识起点和终点。有向图适于描述实体间的非对称或因果关系，无向图则适于表示对称关系或简单地表明两个实体具有某种联系。

使用这种表示方法，很容易定义从图中发现社区的两个标准。一是这组实体（节点）之间按照某些相似度度量必须彼此相似。二是这组实体之间的交互要多于与其他实体的交互。第一个要求保证这些实体具有共同的性质，第二个要求保证实体之间以有意义的方式进行交互，从而使他们成为一个社区，而不是一组具有相同性质却从不彼此交互的用户群体。

这里描述的第一个算法是HITS算法，在第4章关于PageRank部分曾简要介绍。HITS算法和PageRank类似，区别在于HITS是依赖于查询的，而PageRank通常是独立于查询的。或许你想知道，HITS是如何处理社区发现问题的，因为它原本是用来改善网页搜索的。然而，无论是PageRank还是HITS，都属于一族通用、有效的算法，即链接分析（link analysis）算法。这些算法可以应用到许多不同的类型，可以表示为有向图的数据集。

给定一个实体图，必须识别出一个实体子集，该子集内节点可能是社区成员。我们称这些实体为候选实体（candidate entity）。例如，如果希望找到一个关于冰球的在线社区，必须查询图中的每个节点，并且找到所有对冰球感兴趣的节点（用户）。这可以通过找到系统中所有曾经搜索过任何冰球相关信息的用户来实现。这满足第一个标准，实体之间应该是相似的。另一个例子是找到“fractal art”网页社区的任务。这里，可以在网络上以“fractal art”作为查询，在网络上进行搜索，并且仅考虑匹配该查询的实体（网页）。同样，这也保证了所有网页之间是主题相似的。第一步找到相似的实体集合，但是无法识别在社区内积极参与多种交互方式的实体，而这正是第二个重要的标准。

对于指定的候选实体，HITS算法可以用来发现社区的“核心”。HITS算法将图G的节点集V和边集E作为输入。为了发现社区，节点集V由所有候选实体构成，边集E由候选实体间所有的边构成。对于图中每一个候选实体（节点）p，HITS算法计算其权威值（authority score, $A(p)$）和中心值（hub score, $H(p)$）。该算法基于如下假设，好的中心网页（hub）指向好的权威网页（authority），而好的权威网页被好的中心网页链接。注意定义中的循环，这意味着权威值依赖于中心值，而中心值又依赖于权威值。给定一组权威值和中心值，HITS算法依据下面的公式来更新权值：

算法3 HITS

```
1: procedure HITS(G = (V, E), K)
2:     A_0(p) ← 1 ∀p ∈ V
3:     H_0(p) ← 1 ∀p ∈ V
4:     for i = 1 to K do
5:         A_i(p) ← 0 ∀p ∈ V
6:         H_i(p) ← 0 ∀p ∈ V
7:         Z_A ← 0
8:         Z_H ← 0
9:         for p ∈ V do
10:            for q ∈ V do
11:                if (p, q) ∈ E then
12:                    H_i(p) ← H_i(p) + A_{i-1}(q)
13:                    Z_H ← Z_H + A_{i-1}(q)
14:                end if
15:                if (q, p) ∈ E then
16:                    A_i(p) ← A_i(p) + H_{i-1}(q)
17:                    Z_A ← Z_A + H_{i-1}(q)
18:                end if
19:            end for
20:        end for
21:        for p ∈ V do
22:            A_i(p) ← A_i(p) / Z_A
23:            H_i(p) ← H_i(p) / Z_H
24:        end for
25:    end for
26:    return A_K, H_K
27: end procedure
```

$$A(p) = \sum_{q \to p} H(q)$$

$$H(p) = \sum_{p \to q} A(q)$$

其中$p \to q$表示实体p（起点）与实体q（终点）之间的边。如等式所示，$A(p)$是所有指向节点p的实体的中心值之和，而$H(p)$是所有p指向的实体的权威值之和。所以，要成为强的权威网页，必须拥有许多具有相对适度的中心值的入边，或者具有较少但非常大的中心值的入边。类似地，要成为好的中心网页，必须具有许多指向较小权威页面的出边，或者指向少数但拥有较大权威值的网页的出边。

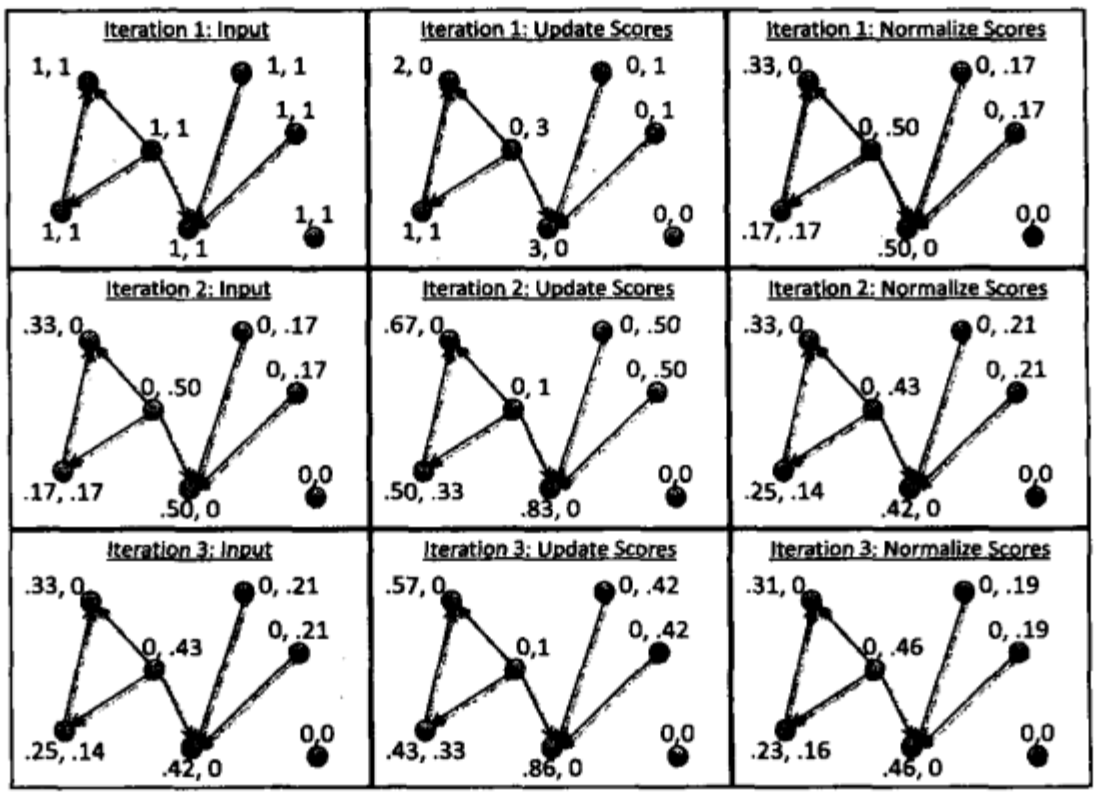


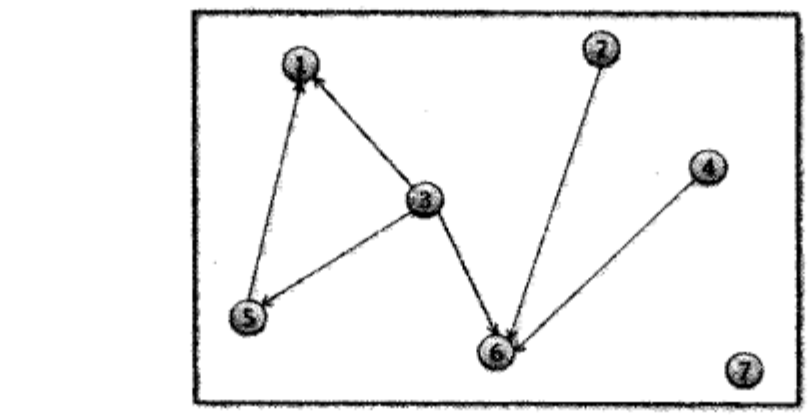
图10-3 HITS算法举例。每一行对应算法的一次迭代，每一列对应算法中具体的一步

HITS算法的一种迭代版本由算法3给出。该算法首先将所有的中心值和权威值都初始化为1，然后根据刚才介绍的方程更新中心值和权威值。接下来，中心值被归一化，以使所有中心值之和为1。采取相同的过程更新权威值。整个过程在归一化的分值上迭代固定的次数，记为K。该算法可以保证收敛且通常经过很小的迭代次数就可以达到收敛。图10-3展示了在一个具有7个节点和6条边的图上应用HITS算法的例子。算法进行了3次迭代。注意，具有许多入边的节点倾向于具有较高的权威值。HITS算法的另外一个特点是，不与任何节点连接的节点的中心值和权威值全部为0。

一旦中心值和权威值被计算出来，实体将按照权威值进行排序。该列表将会包括社区内最具有权威性的实体。这样的实体有可能成为“领袖”，或形成社区的“核心”，这将取决于他们与社区内其他成员的交互情况。例如，如果算法应用于计算机科学读者引用图，试图发现其中的信息检索研究社区，权威性最高的作者将是那些被高产作者引用很多次的人，这些是该领域的权威，并且构成这个研究社区的核心。当应用于网络图来发现网页社区时，算法将返回那些被大量高质量网页指向的页面。

聚类算法，如在第9章曾经介绍的方法，也可以用来发现在线社区。这些算法很容易地适应这类问题，因为社区发现是一种无指导学习问题。自底向上的层次聚类和K均值聚类都可以用来解决社区发现问题。两种聚类算法都需要一个度量类间距离的函数。正如在第9章讨论过的，欧式距离被经常使用。然而，如何用欧氏距离度量图（有向图或无向图）中节点间的距离仍不明确。一个简单的方法是，将图中的每一个节点（实体）表示为一个向量，这个向量有$|V|$个部件——每一个部件对应于图中的一个节点。对某一节点p，如果$p \rightarrow q$，它的向量中的部件q设置为1，否则置为0。这样做的结果是，每一个节点都由它指向的节点表示。图10-4展示了一个图中的节点如何按照这种方式来表示。每一个向量可以被归一化，当然这是可选的。利用这种表示方法，就可以使用欧式距离，从而直接将自底向上的层次聚类或K均值聚类应用到这一问题上来。根据欧氏距离的特点，当两个实体的边指向很多相同的实体时，它们之间

的相似度会很高。例如，回到发现信息检索研究社区的问题中来，如果两个作者倾向于引用相同的作者，他们之间可以被认为相似。信息检索研究社区中的大多数成员都会倾向于引用许多共同的作者，尤其是那些利用HITS算法计算被赋予较高权威值的作者们。这个假设是符合实际的。



节点： 1 2 3 4 5 6 7

向量：

$$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}\quad\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\end{bmatrix}\quad\begin{bmatrix}1\\0\\0\\0\\1\\1\\0\end{bmatrix}\quad\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\end{bmatrix}\quad\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\end{bmatrix}\quad\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}\quad\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

图10-4 有向图中节点表示为向量的例子。对指定节点p，如果$p \rightarrow q$，它的向量中的部件q设置为1

评价社区发现算法的有效性，一般比评价传统的聚类任务更困难，这是因为很难判断一些实体是否属于某一指定的社区。实际上，即使人工地识别在线社区，也会产生很多分歧，这是由于社区定义的模糊性。因此，很难说HITS算法或者自底向上的层次聚类哪一个更适合发现社区。最佳选择极大地依赖于具体的任务及应用的数据集。

现在，我们已经看到了几种自动发现在线社区的方法，或许你想知道这些信息在实际应用中是如何使用的。实际上，在社区被识别之后，有很多事情可以做。例如，如果一个用户已经被识别出是信息检索研究社区的一员，那么当用户访问一个网页，满足用户具体想去的目标内容应当被显示。搜索引擎可以使用社区信息作为额外的上下文信息，使检索结果在主题上与用户关联的社区相关，进而提高检索结果的相关性。在线社区信息也可以应用到其他方面，如加强浏览体验、专家发现、网站推荐，甚至向你推荐合适的约会对象。

10.3.3 基于社区的问答

上一节介绍了如何自动发现在线社区，这一节将叙述如何利用社区帮助回答传统搜索引擎难以回答的复杂的信息需求。例如，有人希望了解他正服用的一种药与他经常喝的草药茶是否会发生反应。他可以使用搜索引擎，花费很长时间，尝试多种查询并查看检索结果，试图从中找到相关的信息。这种方法的困难之处在于，可能不存在单独的网页完全满足用户的信息需求。现在，设想该用户直接向一组用户提出问题，其中一些人是药剂师或者草药专家，他更可能得到答案。人们向一个社区提问，该社区涉及广泛的主题且由专家和非专家构成，其中每个人都可以选择回答这个问题，这种模式称为基于社区的问答（community-based question answering，CQA）。这些系统利用人们知识的力量来满足广泛的信息需求。当今此类

受欢迎的商业系统包括Yahoo!Answers、Naver和一个韩国的搜索门户。

基于社区的问答既有优点也有不足。优点为针对复杂或晦涩的信息需求，用户可以得到答案，能够看到关于同一主题的不同观点，可以与其他具有共同兴趣、问题或目标的用户进行交流。不足之处在于，对于某一问题，可能没有任何回复，需要等待很长时间（几天）才能获取答案，以及得到的答案可能是错误的、误导的、冒犯的甚至是无用的垃圾。

正如前面所述，许多提交的答案都是低质量的。正如计算机程序设计中的一句格言所述："garbage in，garbage out"，研究表明，低质量的答案往往是回答低质量的问题。确实，人们可以并且实际提出的问题非常广泛。表10-1展示了提交到Yahoo!Answer的一些问题样例。表中的部分问题组织良好且语法正确，然而其他的却并非如此；其中一些问题有着简单且直接的答案，而其他的许多问题却不是这样的。

表10-1 提交到Yahoo!Answer的问题举例

What part of Mexico gets the most tropical storms?
How do you pronounce the french words, coeur and miel?
GED test?
Why do I have to pay this fine?
What is Schrödinger's cat?
What's this song?
Hi...can u ppl tell me sumthing abt death dreams??
What are the engagement and wedding traditions in Egypt?
Fun things to do in LA?
What lessons from the Tao Te Ching do you apply to your everyday life?
Foci of a hyperbola?
What should I do today?
Why was iTunes deleted from my computer?
Heather Locklear?
Do people in the Australian Defense Force (RAAF) pay less tax than civilians?
Whats a psp xmb?
If C(-3, y) and D(1, 7) lie upon a line whose slope is 2, find the value of y.?
Why does love make us so irrational?
Am I in love?
What are some technologies that are revolutionizing business?

除了允许用户提出和回答问题，CQA的用户也可以搜索保存过去曾经被提问的问题以及对应的答案的存档集合。这种搜索功能出于两个目的。第一，如果用户找到了过去曾经问过的相似问题，那么他就不必再次提问和等待答案。第二，搜索引擎可以通过命中问答数据库中的数据来扩充传统的搜索结果。例如，如果一个用户输入查询"schrödingers cat"，搜索引擎可能将"What is Schrödinger's cat"的答案（出现在表10-1中）连同其他更标准的排序后的网页集合返回给用户。

因此，给定一个查询[⊖]，从问答数据库中自动找到潜在答案是非常重要的。搜索问答数据库的策略包括：新查询仅匹配存档的问题，或仅匹配答案，或既匹配问题又匹配答案。研究表明，查询匹配存档问题比匹配答案要好，因为一般来讲，找到相关问题（它们更可能有着相关的答案）比使用查询直接匹配答案要容易。

⊖ 在接下来的讨论中，"查询"既可指一个问题也可指一个网络查询。

匹配查询与问题可以使用任何在第7章介绍的检索模型，如语言模型或者BM25。然而，传统的检索模型可能由于词表不匹配问题而损失很多相关的问题。这里词表不匹配问题源于可以通过多种不同的方式来问相同的问题。例如，有如下问题“who is the leader of India?”。相关的问题有“who is the prime minister of India?”，“who is the current head of the Indian government?”等。注意到这些问题共有的词为“who”、“is”、“the”、“of”和“India”。盲目地应用任何标准的检索模型可能会找到不相关的问题，如“who is the finance minister of India?”和“who is the tallest person in all of India?”。在这种情况下，去除停用词帮助不大。更好的匹配可以通过泛化“leader”来包含其他概念，如“prime minister”、“head”和“government”等。

在6.4节描述了跨语言检索，用户用源语言（如英语）提交查询而用目标语言对文档进行检索（如法语）。多数针对跨语言检索提出的检索方法，都是基于翻译模型，在该模型中，需要学习翻译概率$P(s|t)$，s是源语言中的词，t是目标语言中的词。翻译模型也可以用来处理单语中的词表不匹配问题，这通过估计$P(t|t')$来实现，t和t'是同一语言的词。这个概率可以解释为可以用词t替换t'的概率。回到上面的例子，如果P(leader|minister)和P(leader|goverment)的概率值大于0，将会检索到更多的相关问题。下面介绍两种在存档集合中检索相关问题和答案的翻译模型。

第一个模型由Berger和Lafferty（1999）提出。该模型与7.3.1节介绍的查询似然模型类似，但该模型允许查询词由其他词“翻译”而来。给定一个查询，相关问题[⊖]依照下面公式排序：

$$P(Q|A)=\prod_{w\in Q}\sum_{t\in V}P(w|t)P(t|A)$$

其中Q是一个查询，A是相关问题存档集合，V是字母表。$P(w|t)$是翻译概率，$P(t|A)$是文档A生成单词t的平滑概率（详见7.3.1节）。翻译模型允许查询词w由其他的词t翻译而来，而t则可能出现在问题中。一个主要问题是，该模型并不保证问题一定会与查询相关，因为每一个词都是独立翻译的，具有最高分的句子或许是很好的词翻译，但却未必是很好的总体翻译。

第二个模型由Xue（2008）提出，是Berger模型的一种扩展。该模型中，试图通过使匹配原查询中的词比匹配翻译的词具有更高的权值，来解决上述的问题。在该模型下，问题（或答案）按照如下函数排序：

$$P(Q|A)=\prod_{w\in Q}\frac{(1-\beta)f_{w,A}+\beta\sum_{t\in V}P(w|t)f_{t,A}+\mu\frac{c_w}{|C|}}{|A|+\mu}$$

这里β为0、1之间的参数，用来控制翻译概率的影响，μ为Dirichlet平滑参数。注意，当β=0时，该模型与原始查询的似然模型是等价的。随着β不断接近于1，翻译模型对排序产生更大影响，从而更接近于Berger的模型。

利用这些模型进行排序，需要很大的计算代价，因为对每一个词计算时，都要考虑对整个词表进行求和运算，而词表可能是非常大的。针对每个查询词，仅考虑小部分翻译可以大

⊖ 这里的讨论关注于问题检索，但是同样的模型同样可以用来做答案检索。正如前面所说，问题检索通常更有效。

大提高查询处理的速度。例如，仅使用5个最可能的翻译，那么在求和的时候，需要考虑的词的数目由V减少到5。

到目前为止被忽略的一个问题是，如何计算翻译概率。在跨语言检索中，翻译概率可以使用平行语料库来自动学习。翻译概率通过文档对来估计，文档对可表示为$\{(D_1^s, D_1^t), \cdots, (D_N^s, D_N^t)\}$，其中$D_i^s$为由源语言书写的文档$i$，$D_i^t$为由目标语言书写的文档$i$。但在处理单语翻译概率时，平行语料库的概念变得有些模糊。有许多方法可以估计单语的翻译概率。为了找到相关问题，其中最成功的方法是，假设问题/答案对形成平行语料库，并利用这个平行语料库计算翻译概率。也就是说，翻译概率通过存档的问题/答案对来估计，其形式为$\{(Q_1, A_1), \cdots, (Q_N, A_N)\}$，其中$Q_i$为问题$i$，$A_i$为答案$i$。表10-2给出了使用这种方法，从一个真正的问题答案数据库中估计翻译概率的例子。具体如何从平行语料库中估计翻译概率的算法，在本章末尾“参考文献和深入阅读”一节给出。

表10-2 由一个问题答案对集合中自动学习得到的翻译。表中给出了针对“everest”、“xp”、“search”最可能的几种翻译

everest	xp	search	everest	xp	search
everest	xp	search	mt	version	web
mountain	window		ft	click	list
tallest	install	information	measure	pc	free
29 035	drive	internet	feet	program	info
highest	computer	website	mount	microsoft	page

在这一节，假设社区成员会为提出的问题提供答案，问题答案存档集合在这一过程中创建。正如第1章所述，可以设计问答系统利用大规模文档集来回答一些受限的问题。我们将在第11章详细讨论这些系统。

10.3.4 协同搜索

最后一个基于社区的搜索任务是协同搜索（collaborative searching）。协同搜索使一组具有相同搜索目的的用户共同搜索。在一些情况下，协同搜索是十分有用的。设想一组学生正在一起准备关于世界历史的报告，为了完成这个报告，学生们必须针对报告主题做背景研究。通常，大家会将这个主题分为若干个子主题，并且分派给组内的每个成员。之后每个学生通过到网络或在线图书馆独立地搜集关于各自子主题的资料。最后，这些学生会将各个子主题的资料合并起来，形成一个完整的报告。每位学生仅对于属于他的子主题十分了解，而没有哪个学生完全理解整个报告。显然，如果研究过程有更多的合作，每位学生最终会学习到更多的知识。协同搜索系统允许学生同时搜索网络和其他资源，这样组内的每一个成员都可以对报告中的每个子主题有所贡献和理解。协同检索在公司内部也非常有用，许多同事一定会收集关于某个项目的不同方面的信息。搜索休闲、娱乐信息的用户也会发现协同搜索系统十分有用。假设你和朋友们正在筹备一个聚会，协同搜索系统将会帮助每个人协调信息收集工作，如找食谱、选择装饰、挑选音乐和发送邀请等。

根据搜索参与者彼此之间所处的位置，有两种协作搜索的情景。第一种情景称为同位（co-located）协同检索，指所有的搜索参与者处于同一地点，如同一个办公室、同一个图书

馆甚至坐在同一台计算机前。另一情景称为远程（remote）协同检索，指搜索参与者处于不同地点，可能处于同一建筑中的不同房间、同一城市中的不同建筑甚至不在同一个国家。图10-5给出了这两种情景的示意图，两种情况面对不同的挑战，针对每个情景开发的系统也有着不同的要求，这取决于它们如何支持搜索。为了示范，下面简要介绍两个协同检索系统的例子。

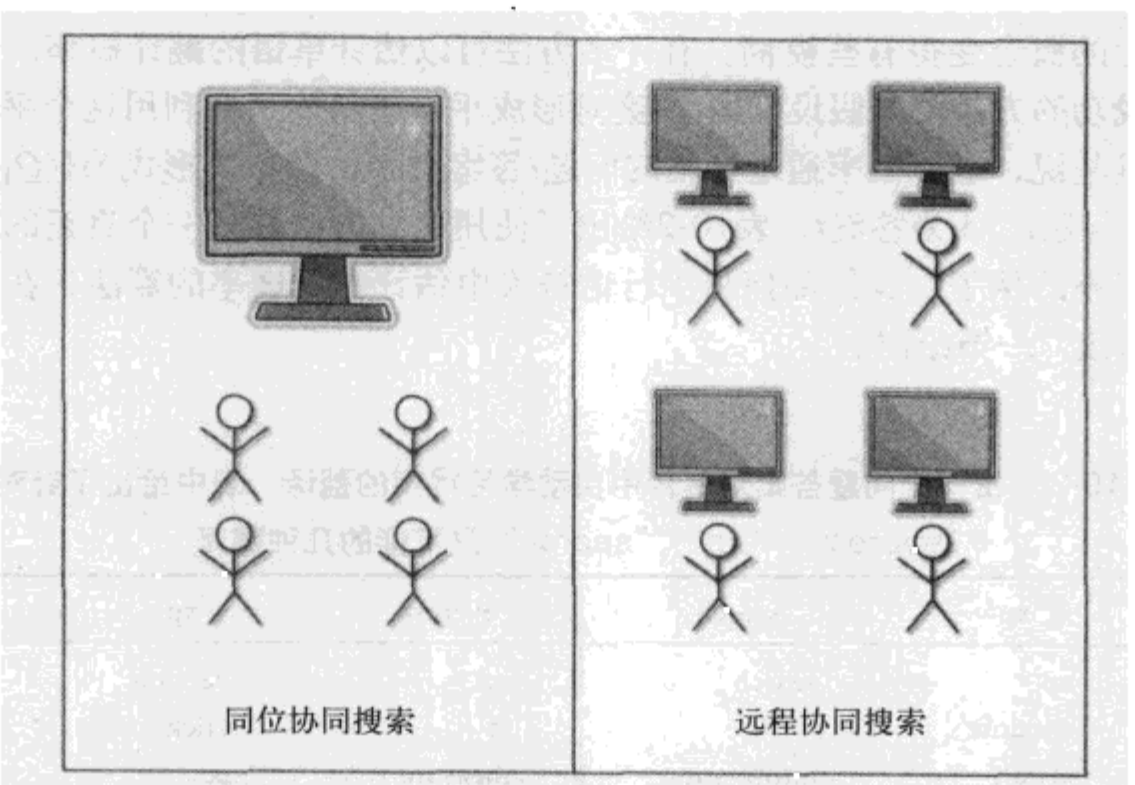


图10-5 两种通用协同搜索情景概览。左图为同位协同搜索，该情景在同一时间涉及相同地点的多个参与者。右图为远程协同检索，该情景在同一时间涉及的参与者在不同地点

由Amershi and Morris（2008）开发的CoSearch系统是一个同位协同搜索系统。该系统的主显示器、键盘和鼠标由一个称为“driver”的人控制，主导整个搜索任务。其他的参与者称为“observer”，每人有一个鼠标或者具有蓝牙[⊖]功能的手机。Driver向搜索引擎提交一个查询，开始一个会话（session），搜索结果将会显示在主显示器和任何一个拥有手机的用户的显示屏上。Observer可以点击搜索结果，对应的页面将加入一个共享的页面队列中。这使得每一位参与者都能够以方便、集中的方式推荐接下来值得关注的网页，而不是将控制权全部交给driver。除了页面队列，系统还维护一个查询队列，记录用户提交的新查询。查询队列为每一个用户提供一个查询列表，其中的查询可能是有用的、值得关注的。同时，查询队列提供给driver一个合作生成的选项集合。CoSearch系统为一组用户提供了许多有用的、使他们一起合作的搜索，它可以使每个人相互合作完成共同的任务，同时通过多种输入设备维护分工。

远程协同检索系统的例子是SearchTogether，由Morris and Horvitz（2007b）开发。在这个系统中，假设每一个参与会话的人都在不同的地点，而且有自己的计算机。与同位协同搜索假定参与者在整个对话中都在线不同，远程搜索并没有假定参与者是否同时在线。所以同位搜索对话往往是简短的，而远程搜索对话则是持续的。系统的用户可以提交查询，这些查询将被记录为日志并与所有参与者共享。这使得所有参与者都能看到其他人搜索了哪些内容，并且允许他们重新提交查询或优化查询。用户可以评价（“thumbs up”或“thumbs down”）和评论搜索过程中看过的页面，这些页面将被聚合起来且允许其他参与者访问。此外，参与

⊖ 蓝牙（Bluetooth）是一种短距离无线通信技术。允许电脑、打印机、PDA和手机之间进行无线通信。

者可以显式地推荐页面给其他参与者，这些页面将会出现在推荐页面列表中。因此，SearchTogether系统提供了CoSearch系统具有的大多数功能，此外还适应远程协同搜索的特殊要求。持续搜索对话的一个特别的好处是，原来不属于该搜索过程的新参与者，可以通过浏览查询历史、页面评价、评论和推荐的内容，迅速地加入到这一过程中来。

协同搜索为用户提供了独特的工具，可以用来在同位或远程搜索对话中有效地同他人合作。尽管这类系统具有光明前景，但在今天，只有很少的商用协同搜索系统。然而，这种系统在研究社区内获得了广泛关注。随着在线体验中的合作日益增长，协同检索系统被更广泛地应用也许只是时间问题。

10.4 过滤和推荐

10.4.1 文档过滤

如前面所述，社会化搜索应用的重要部分之一，是表示单个用户的兴趣与偏好。最早关注用户描述文件（profile）的应用是文档过滤。文档过滤经常简称为过滤，是对标准的搜索模式的一种替代或补充。在标准的搜索中，用户随着时间的推移输入很多查询，而文档集合相对静止。在过滤中，用户的信息需求视为不变的，而文档集合则是随着新文档周期性的到来而动态变化的。过滤的目的，是识别出（过滤）相关的新文档，并将其发送给用户。正如在第3章所述，过滤是一种推送（push）应用。

过滤也是有监督学习的例子，这里描述文件扮演着训练数据的角色，而进入的文本则作为测试用例，需要被判定为相关或不相关。在垃圾邮件检测模型（spam detection model）中，有大量的标注过的电子邮件作为输入，而过滤系统的描述文件可能仅仅是一个单独的查询，这使得学习任务更具挑战性。由于这个原因，过滤系统通常不仅仅使用一般的分类技术，而是采取更具体的技术来克服缺乏训练数据的问题。

尽管过滤系统没有标准的网络搜索引擎应用得广泛，但现实世界中仍有很多文档过滤系统。例如，很多新闻网站都提供过滤服务，提醒用户关注某些事件，如突发性的新闻、某些新类别（如体育或政治）中新文章的发布、有关某一特定主题的文章发布，这里的主题通常使用一个或多个关键词（如“terrorism”或“global warming”）来表示。提醒以电子邮件、短消息（文本信息）或个性化信息源的形式实现。这种方式使得用户可以持续关注某些感兴趣的主题，而不必频繁地查看新闻网站的更新或在站点的搜索引擎进行大量查询。因此，过滤通过维护长期的信息需求，提供了一种个性化的搜索体验。

文档过滤系统有两个主要的组成部分。首先，用户的长期信息需求必须精确地表示。这可以通过为每一个信息需求构建描述文件来实现。另外，对于新来的文档，必须采用一种决策机制来判别哪些描述文件与该文档相关。因为通常会有成百上千的描述文件，这个决策机制要求非常有效而且精确。过滤系统不应该遗漏相关的文档，更重要的是，不应该经常将不相关的文档推送给用户。在本节余下部分，会详细描述这两个部分。

1. 描述文件

在网页检索中，用户通常输入非常短的查询。搜索引擎面临的一个艰巨挑战是，如何从非常稀疏的信息中判断出用户的潜在信息需求。有很多原因可以解释为什么大多数搜索引擎都通过短的关键词查询来接受用户的信息需求，其中一个主要原因是，用户不愿意（或没有时间）对每一个信息需求都输入长的、复杂的查询。很多简单、非持久性的信息需求通常通

过短的查询就可以得到满足。过滤系统则是为了满足长期的信息需求。因此，用户或许愿意花费更多的时间去详细地指定他们的信息需求，从而能够确保在较长的时间内获得高度相关的结果。用户的长期信息需求通常称为过滤描述文件（filtering profile）或描述文件（profile）。

过滤描述文件的构成取决于特定领域的兴趣。描述文件可以像布尔或关键字查询那样简单，也可以包含与用户的信息需求相关或不相关的文档。更进一步地，可以包含其他项目，如社会标签和相关的命名实体。最终，描述文件可能还有一个或多个关系约束，如“发表在1990年以前”、“价格在$10～$25”等。与其他一些软约束相比，关系约束是一种硬约束，即只有满足关系约束的文档才可能被检索。

尽管有许多方法表示描述文件，但底层过滤模型通常决定真正的表示形式。过滤模型与第7章叙述的检索模型非常相似。实际上，很多广泛使用的过滤模型就是简单的检索模型，只不过查询被描述文件替代。有两种典型的过滤模型。第一种称为静态模型，静态是指用户的描述文件不随时间改变，因而模型一直被使用。第二种称为自适应模型，指用户描述文件随时间变化。该情景要求过滤模型随时间的推移是动态的，更新的信息可以被纳入描述文件。

2. 静态过滤模型

如上面所述，静态过滤模型基于以下假设：过滤描述文件不随时间推移而变化。在很多方面，这使得过滤过程变得简单，但在其他方面，使得过滤过程不够健壮。所有流行的静态过滤模型都由第7章描述的标准检索模型演化而来。然而，与网页检索不同，过滤系统并不是对每个描述文件返回排序后的文档列表，而是当一篇新文档到来时，过滤系统必须决定这篇文档与每个描述文件相关与否。图10-6展示了静态过滤系统是如何工作的。当新文档到来时，他们将与每一个描述文件进行比较，由文档指向描述文件的箭头，表示该文档是相关的并被返回给用户。

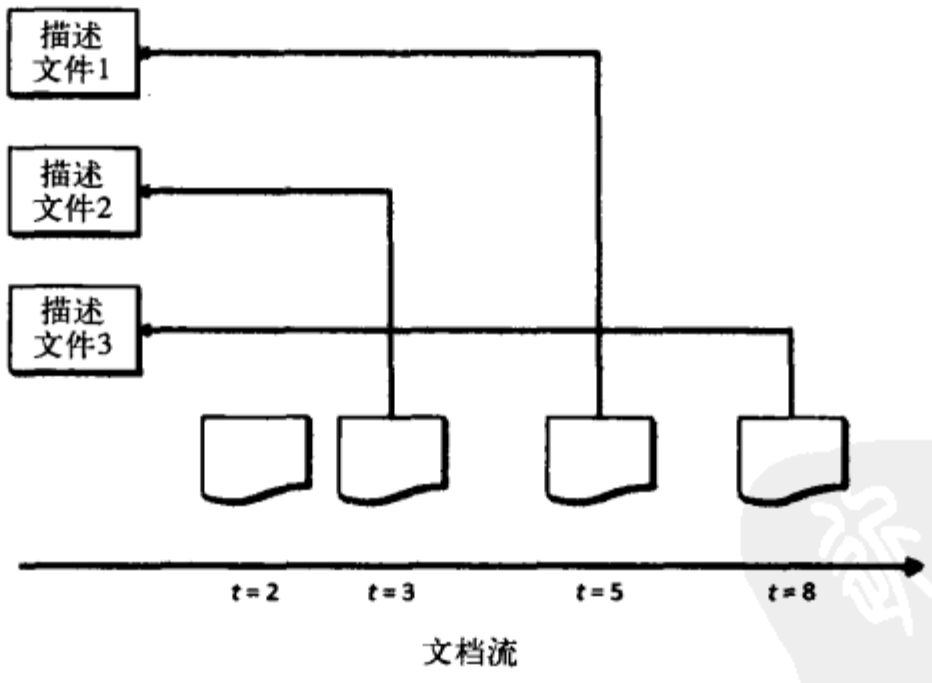


图10-6 静态过滤系统举例。文档随时间推移不断进入系统，并与每个描述文件进行比较。由文档指向描述文件的箭头，表示该文档匹配该描述文件并被检索

对简单的情况，可以使用布尔检索模型。此时，过滤描述文件简单地表示为布尔查询，当且仅当其满足该查询时新到文档被检索。布尔模型尽管简单，却可以有效地应用于文档过滤，尤其是在要求准确率更重要的情况下。实际上，许多基于网页的过滤系统都采用布尔检索模型。

布尔模型一个最大的缺陷是召回率低。对某些需要过滤的领域，用户可能希望在高准确率的基础上，也有很好的覆盖率。针对这一问题有很多的解决方案，包括使用向量空间模型、概率模型、BM25或者语言模型。通过将描述指定为关键词查询集合或文档集合，这些模型都可以被扩展并应用到过滤中。直接将这些模型应用到过滤会有些麻烦，因为这些模型都是返回得分，而不是像布尔模型那样返回一个答案，检索或不检索。为了克服这一问题，最常用的技术是设置一个阈值来决定是否检索一个文档，即只有相似度大于阈值的文档才会被检索。阈值根据实际的效果进行调整。使用全局阈值时，会产生很多复杂的问题，包括如何保证对所有的描述文件和随时间推移设置合适的阈值。

作为一个具体例子，下面描述如何利用语言模型框架来实现静态过滤。给定一个静态的描述文件，如一个关键词查询、多个查询、一个文档集合，或以上情况的组合，必须估计一个描述文件语言模型，记为P。下式为其中一种实现方法：

$$P(w|P)=\frac{(1-\lambda)}{\sum_{i=1}^{K}\alpha_i}\sum_{i=1}^{K}\alpha_i\frac{f_{w,T_i}}{|T_i|}+\lambda\frac{c_w}{|C|}$$

其中$T_1, \cdots, T_k$，为构成描述文件的文本片段（如查询、文档），α_i为关于T_i的权值（重要性）。其他的变量和参数在第7章进行了具体定义。

当指定一个传入的文档，需要估计文档语言模型（D）。按照第7章的讨论，采用下面的公式估计D：

$$P(w|D)=(1-\lambda)\frac{f_{w,D}}{|D|}+\lambda\frac{c_w}{|C|}$$

随后文档按照描述文件语言模型（P）与文档模型（D）的负KL距离进行排序：

$$-KL(P\|D)=\sum_{w\in V}P(w|P)\log P(w|D)-\sum_{w\in V}P(w|P)\log P(w|P)$$

如果$-KL(p\|D)\geqslant t$，文档D将被指派给描述文件P，其中t为相关性阈值。

文档过滤也可视为机器学习问题。其核心是将过滤视为具有少量训练数据（描述文件）的分类问题。该任务为构建二元分类器，判断到来的文档是否与描述文件相关。为了训练出合适的模型，训练数据是必需的。对于该任务，训练数据以进行了二元相关性判定的描述文件/文档对的形式表示。在第9章描述的任何分类技术都可以应用到这里。假设使用线性核函数支持向量机，评分方程为下面的形式：

$$s(P;\ D)=w\cdot f(P,\ D)=w_1f_1(P,\ D)+w_2f_2(P,\ D)+\cdots+w_df_d(P,\ D)$$

其中，$w_1, \cdots, w_d$为SVM训练过程学习到的参数集合，$f_1(P, D), \cdots, f_d(P, D)$为从描述文件/文档对中提取的特征集合。很多成功应用到文本分类中的特征，如unigrams和bigrams，也可以应用到过滤。在具有大量的训练数据的情况下，机器学习方法的表现很可能超过刚才描述的简单语言模型。然而，如果只有少量或没有训练数据，语言模型则是更好的选择。

3. 自适应过滤模型

静态过滤假设描述文件不随时间推移而变化。在这种设置下，用户描述可以被创建却无法更新以更好地反映其信息需求。唯一的选择即是删除旧的描述文件，重新创建一个新的描

述文件，但这种系统不够灵活和健壮。

自适应过滤是另一种过滤技术，采用动态的描述文件，该技术提供随时间推移自动更新描述文件的机制。

描述文件可以由用户输入进行更新，也可以基于用户行为自动更新。用户行为包括点击或浏览模式等。有很多原因可以解释为什么随时间推移自动更新描述文件是非常有益的。例如，用户希望微调他们的信息需求，以获得更具体类型的信息。因此，自适应过滤技术比静态过滤技术更加健壮，可以在描述文件的生命周期内自适应地找到更相关的文档。图10-7展示了针对与图10-6相同的描述文件集合和进入的文档，自适应过滤是如何工作的。与静态过滤的情况不同，当一个文档指定给某个描述文件时，用户针对该文档进行反馈，随后描述文件将进行更新，用来匹配未来进入的文档。

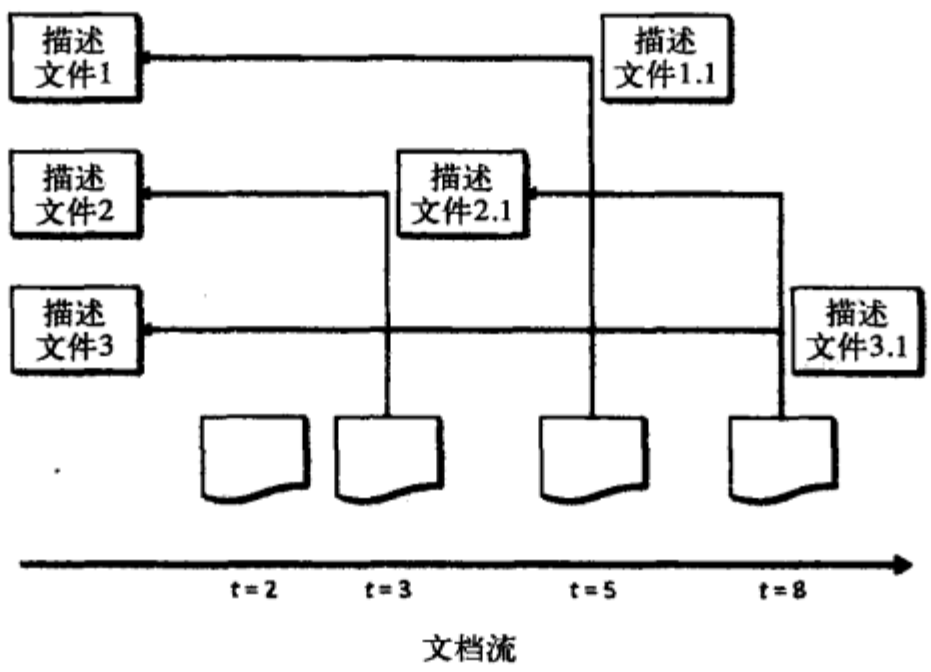


图10-7 自适应过滤系统举例。随时间推移，文档到达并与描述文件进行比较。由文档指向描述文件的箭头表示该文档匹配描述文件并被检索。与静态过滤中描述文件随时间推移是静态的情况不同，自适应过滤的描述文件是动态更新的（如有新的匹配时）

如10.7节所指出的，自适应更新描述文档最常用的方法即是根据用户的反馈。用户反馈可以有多种方式，每一种都可以用来以不同的方式更新用户描述文档。为了提供一个具体例子说明描述文档是如何根据用户反馈自适应更新的，考虑用户提供相关反馈（relevance feedback，见第6章）文档的情况。在这种情况下，针对一个文档集合，如根据指定描述文件检索到的文档集合，用户显式地指出每篇文档是相关的或是不相关的。和以前一样，如何表示描述文档以及随后如何更新，极大地依赖于使用的检索模型的类型。

如第7章所述，Rocchio算法可以使用向量空间模型来进行相关反馈。因此，如果描述文件表示为向量空间模型中的向量，当用户提供相关反馈信息的时候，Rocchio算法就可以用来更新描述文件。对于描述P，非相关文档集合（记为Nonrel）和相关文档集合（记为Rel），自适应的描述文件P'可以按照下式进行计算：

$$P' = \alpha \cdot P + \beta \cdot \frac{1}{|Rel|} \sum_{D_i \in Rel} D_i - \gamma \cdot \frac{1}{|Nonrel|} \sum_{D_i \in Nonrel} D_i$$

其中，D_i为文档i的向量表示，α、β、γ作为参数，用来平衡初始描述文件、相关文档和不相

关文档之间的权值。

第7章同样介绍了相关性模型（relevance model）是如何结合语言模型来处理伪相关反馈的问题。然而，相关性模型同样也可以用来处理真正的相关反馈，如下面公式所示：

$$P(w|P)=\frac{1}{|Rel|}\sum_{D_i\in Rel}\sum_{D\in C}P(w|D)P(D_i|D)$$

$$\approx\frac{1}{|Rel|}\sum_{D_i\in Rel}P(w|D_i)$$

这里C为数据集中的文档集合，Rel为被判定为相关的文档集合，D_i为文档i，$P(D_i|D)$为文档D_i由文档D的语言模型生成的概率。可以采用近似（≈）的原因是，D_i是一个文档，因此当$D_i=D$时，$p(D_i|D)$为1或非常接近于1，而对其他文档，则接近于0。因此描述文件中词w的概率，可以简单地通过计算相关文档语言模型中w概率的平均值来获得。与Rocchio算法不同，这里并没有考虑非相关文档。

如果采用第9章所介绍的分类方法来处理过滤，当用户提供新的反馈时，可以使用在线学习算法来修正分类模型。在线算法通过一个或一批新来的项目更新模型的参数，如SVM中的超平面w。这类算法与标准的有监督学习方法不同，因为它们没有“记忆”，一旦输入被用作训练，随即被清除而无法继续显式地用来更新模型参数，仅使用新的训练输入来训练。在线学习方法的细节超出了本书的范围。本章最后的“参考文献和深入阅读”中，给出了若干参考文献。

静态和自适应过滤都可以视为第6、7、9章描述的许多检索模型和技术的特例。表10-3总结了不同的过滤模型，包括描述文件如何表示和更新。实际中，向量空间模型和语言模型已经被证明针对静态和自适应过滤都是有效且易于实现的。分类模型对于高度动态的环境则更具健壮性。然而，所有的分类技术都需要训练数据来学习有效的模型。

表10-3 静态和自适应过滤模型总结。对每种模型，给出了描述文件的表示和更新的算法

模型	描述文件的表示	描述文件更新
布尔模型	布尔表达式	N/A
向量空间模型	向量	Rocchio
语言模型	概率分布	相关性模型
分类	模型参数	在线学习

4. 针对大量描述文件的快速过滤算法

在大规模实际系统中，可能会有数千甚至百万级的描述需要与进入的文档进行匹配。幸运的是，应用标准的信息检索索引和查询评价策略，可以有效地进行匹配。在大多数情况下，描述文件表示为关键词集合或特征值集合，使得每个描述文件可以用第5章讨论的策略进行索引。可扩展的基本索引技术可以轻易处理百万级甚至十亿级的描述文件。一旦这些描述文件被索引，进入的文档可以转化为一个“查询”，同样也表示为关键词集合或特征集合。然后，使用该查询在被索引的描述文件中进行检索，返回描述文件的排序列表。每篇文档将被指定给与查询相关度大于指定阈值的描述文件。

5. 评价

第8章介绍的评价方法可以用来评价过滤系统。然而选择合适的评价标准是十分重要的，

这是因为过滤与新闻搜索、网页搜索等标准的搜索任务有所不同。其中最重要的区别是，过滤系统并不为每个描述文件返回排序的文档，而是当相关文档到达时，将其指定给描述文件。因此排序准确率和平均准确率（MAP）不适合用来评价过滤系统，而通常使用基于集合的评价方法。

表10-4与表8-3类似，展示了一篇文档针对某个描述文件对应的所有可能。文档可以是相关或不相关的，这可从列标题看出。此外，文档可以被过滤系统检索或未被检索，可从行标题看出。所有过滤评价标准都可以根据该表格计算。

表10-4 针对过滤系统可能输出的联列表。其中，*TP*（true positive）为检索到的相关文档数目，*FN*（false negative）是未检索到的相关文档数目，*FP*（false positive）是检索到的不相关文档的数目，*TN*（true negative）为未被检索到的不相关文档数目

	相关的	非相关的
被检索的	*TP*	*FP*
未被检索的	*FN*	*TN*

评价过滤系统最简单的方式是，采用经典的精确率与召回率的评价标准，分别对应于$\frac{TP}{TP+FP}$和$\frac{TP}{TP+FN}$。*F*值通常为精确率与召回率的调和平均值。一般对每个描述文件，分别计算评价指标而后取平均值，作为对系统的评价[⊖]。

也可以通过将表10-4中的4个表项结合，定义更一般化的评价标准：

$$U = \alpha \cdot TP + \beta \cdot TN + \delta \cdot FP + \gamma \cdot FN$$

系数α、β、δ、γ可以通过多种方式设置，以平衡评价的各个组成部分。其中一种设置方式在过滤实验中被广泛采用：α=2，β=0，δ=−1，γ=0，即*TP*（检索到的相关文档）被赋予权值2，而*FN*（未检索到的相关文档）则给予惩罚1。当然，不同的系数可以根据具体任务的实际情况来确定。

10.4.2 协同过滤

静态和自适应过滤并不属于社会化的任务，这是因为描述文件被认为是彼此独立的。如果现在考虑描述文件之间存在复杂的关系，那么可以获得额外的有用信息。例如，假想一个自适应过滤系统有两个描述文件，称为描述文件A（对应于用户A）和描述文件B（对应于用户B）。如果用户A和用户B根据各自的描述文件对大量文档具有相同的相关性判断（相关或不相关），可以推测这两个描述文件是相似的。可以利用这一信息来改善用户A和用户B的相关性匹配。例如，如果用户A判定一篇文档对于描述文件A是相关的，那么该文档很可能与描述文件B也相关，因而该文档也应当被检索，即使自适应过滤系统指定给它的分值低于预先指定的阈值。这个系统是社会化的，因为返回给用户的文档不仅基于文档与描述文件的主题相关性，还与拥有相似描述文件的用户对该文档的评价或反馈有关。

考虑描述文件（或用户）之间的关系，并且利用这些信息改善进入的项目与描述文件（或用户）间匹配的过滤技术，称为协同过滤（collaborative filtering）。协同过滤通常是推荐系统的组成部分之一。推荐系统利用协同过滤算法推荐项目（如书籍或电影）。许多主要的商

⊖ 回想一下，这称为宏平均（macroaveraging）。

业站点，如Amazon.com和Netflix，都充分利用推荐系统为用户提供推荐的产品列表，以希望用户能够看到他可能喜欢却尚未了解的产品，并可能因此购买。所以，这类系统对终端用户和搜索引擎公司都非常有价值。对用户来说，他们可以看到从前没有考虑到的相关产品，对搜索引擎公司来说，此类系统有助于增加收入。

本节余下的部分关注于推荐系统中的协同过滤算法。要指出的是，这些算法和静态或自适应过滤在很多方面都不同。首先，当使用协同过滤算法进行推荐时，通常每一个描述文件与一个用户相关联，即用户就是描述文件。其次，静态或自适应过滤系统对每一个进入的文档进行二元决策（检索或不检索），而推荐系统中的协同过滤算法则是为项目评分（rating）。这些评分可以是0（相关）和1（不相关）或者更复杂，例如评分在1至5范围内。最后，推荐系统中的协同过滤算法不仅为新到的项目进行评分，而且对数据库中所有当前用户没有提供显式判定的项目进行评分。与之相比，静态和自适应过滤算法仅仅决定是否将进入的文档发送给用户，而从不反过来判断已有的文档是否应该被检索。

图10-8表示用户的虚拟空间，具有相似偏好和品味的用户彼此接近。每个用户头顶的对话框表示他们对某些项目的兴趣，如关于热带鱼的电影。没有对电影进行评分的用户的对话框里为问号。协同过滤算法的工作即是尽可能准确地预测用户将会如何对电影进行评分。

协同过滤在概念上很简单，细节上却很难。例如，必须决定如何表示用户以及如何度量用户之间的相似度。当相似用户被识别之后，用户评分必须按照一定方式合并起来。另一个重要的问题是，如何评价协同过滤和推荐系统。本节余下的部分，将在详细介绍两种成功应用于推荐系统的协同过滤算法的同时，阐述这些问题。

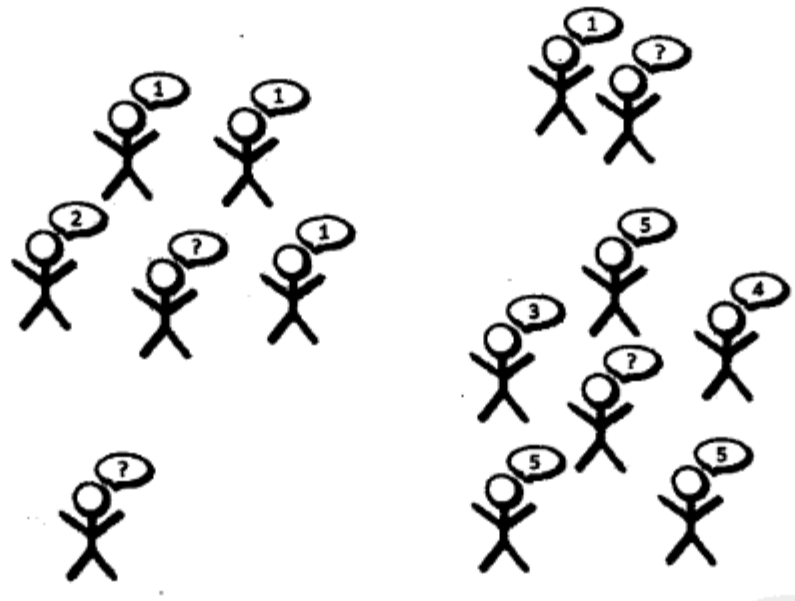


图10-8 推荐系统中的一组用户。用户和他们的评分被给出。头顶有问号的用户表示他们尚未对项目评分。推荐系统的目的即是填写问号的内容

1．用户群评分

在接下来的两个算法中，假定用户集合为U，项目集合为I。进一步地，$r_u(i)$表示用户u对项目i的评分，$\hat{r}_u(i)$为系统对用户u对项目i评分的预测值。注意，在用户没有对项目i评分时，$r_u(i)$是未定义的。正如稍后将描述的，这种情况其实是不必要的。因此，协同过滤的一般任务即为对每一个没有显式评分的用户/项目对计算$\hat{r}_u(i)$。假设仅用显式的评分作为输入来进行预测。进一步地，出于简化的目的，假设评分为范围在1至M之间的整数，尽管多数算法在连续评分上同样工作得很好。

一个简单的方法是，首先对用户集合应用某个第9章介绍的聚类算法。通常，用户被表示为他们的评分向量$r_u=[r_u(i_1), \cdots, r_u(i_{|U|})]$。然而，既然并不是所有用户都对所有项目进行评分，向量r_u并不是每一维都被定义了，这对计算距离的标准，如余弦相似度，提出了挑战。因此，距离度量的标准必须被修改，以能够处理缺失的数值。最简单的方法是，将缺失的数值都赋予某个值，如0。其他的可能是使用用户的平均评分来替代，记为$\bar{r}_u$，或者使用项目的平均评分。

常用的用于用户聚类的相似度度量方法为相关系数（correlation），其计算公式如下：

$$\frac{\sum_{i\in I_u\cap I_{u'}}(r_u(i)-\hat{r}_u)\cdot(r_{u'}(i)-\hat{r}_{u'})}{\sqrt{\sum_{i\in I_u\cap I_{u'}}(r_u(i)-\hat{r}_u)^2\sum_{i\in I_u\cap I_{u'}}(r_{u'}(i)-\hat{r}_{u'})^2}}$$

这里I_u和I_u分别表示用户u和用户u'已经评分的项目集合。这说明求和运算仅针对两个用户共同评价过的项目进行。相关系数的取值在−1到1之间，其值为1时，表示两个用户对同一项目集合具有相同的评分，其值为−1时，表示两个用户的评分截然相反。

在图10-9中，给定了一个假想的用户聚类，由虚线边框表示。用户聚类之后，在某一类中的任一用户未评分的项目，将会被指定为该类内其他用户对这个项目评分的平均值。例如，类A中未对热带鱼电影评价的用户将被指定评分为1.25，这个数值正是类A中其他四个用户对该电影评分的平均值。该计算过程的数学表示为：

$$\hat{r}_u(i)=\frac{1}{|Cluster(u)|}\sum_{u'\in Cluster(u)}r_{u'}(i)$$

其中$Cluster(u)$表示用户u所属的用户群。

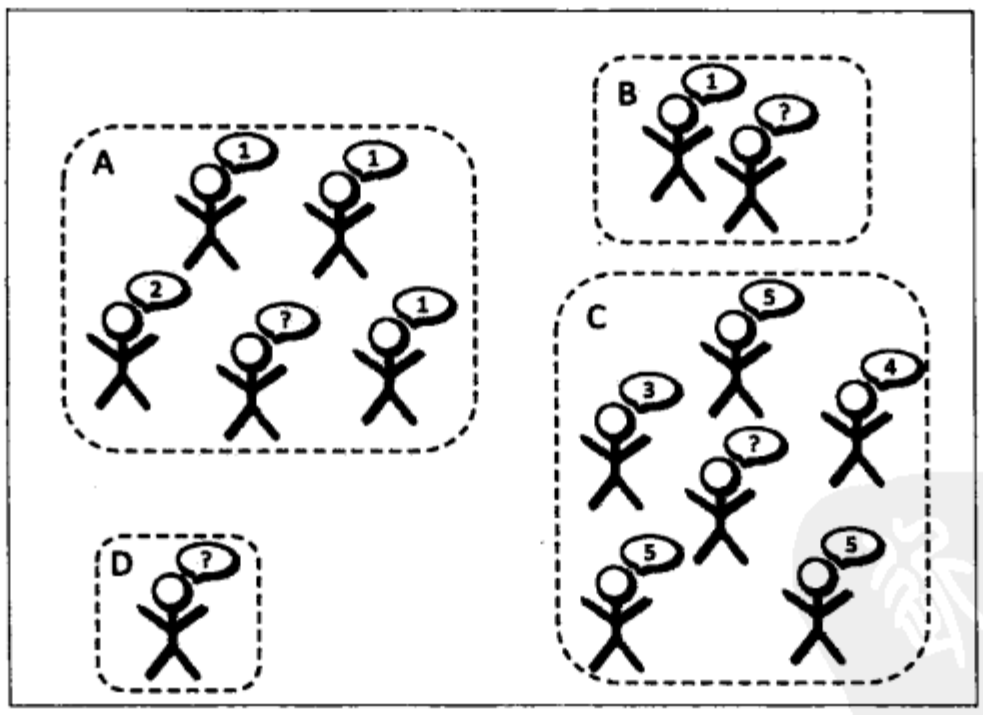


图10-9 基于聚类的协同过滤举例。相似用户群用虚线边框表示。用户对某些项目的评分已经给出。在每一个用户群中，有一个用户尚未评价该项目，这些用户未评分的项目将会被指派一个基于相似用户的评分自动生成的评分

对一个用户群的用户评分进行平均，是一种整合用户群评分的简单方法。另一个选择是使用项目的期望评分，给定用户群内其他用户的评分，其计算公式为：

$$\hat{r}_u(i) = \sum_{x=1}^{M} x \cdot P(r_u(i) = x \mid C = Cluster(u))$$
$$= \sum_{x=1}^{M} x \cdot \frac{|u' : r_{u'}(i) = x|}{|Cluster(u)|}$$

这里$P(r_u(i)=x|C=Cluster(u))$为给定用户群$Cluster(u)$，用户u对项目评分为x的概率。通常用$\frac{|u' : r_{u'}(i) = x|}{|Cluster(u)|}$来估计该概率，即$Cluster(u)$中，对项目$i$评分为$x$的用户所占的比例。例如，如果$Cluster(u)$中的用户都评分为5，那么$\hat{r}_u(i)$将等于5。如果$Cluster(u)$中有一个用户评分为1，5个用户评分为5，那么 $\hat{r}_u(i) = 1 \cdot \frac{5}{10} + 5 \cdot \frac{5}{10} = 3$。

基于聚类预测评分的问题在于用户群是非常稀疏的，如图10-9中的用户群D。如果用户具有独特的兴趣和品味而不能聚到某个合适的用户群中，那么应当如何预测该用户的评分呢？这是个复杂且有挑战性的问题，并没有直接的答案。一个简单但非常有效的方法是，将项目的平均评分指派给用户所有未评价的项目。遗憾的是，这种直接假设往往并不符合实际。

2. 最近邻评分

利用聚类评分的一个替代策略是，利用最近邻来预测评分。这种方法采用第9章描述的K近邻聚类技术。为了预测用户u的评分，首先依据某种相似度度量方法确定与其最相近的K个用户，一旦找到这些最相近的邻居，利用他们的评分（和相似度）来进行预测，公式如下：

$$\hat{r}_u(i) = \bar{r}_u + \frac{1}{\sum_{u' \in N(u)} sim(u, u')} \sum_{u' \in N(u)} sim(u, u')(r_{u'}(i) - \bar{r}_{u'})$$

这里$sim(u, u')$是用户u与u'的相似度，$N(u)$为最相近邻居集合。算法预测用户u对项目i的评分，首先包括用户评价过的项目的平均得分（$\bar{r}_u$）。然后，对用户u的每个最相近邻居，$r_{u'}(i) - \bar{r}_{u'}$利用$sim(u, u')$来加权并被加到最终预测的分数中。你可能想知道为什么使用$r_{u'}(i) - \bar{r}_{u'}$而不是$r_{u'}(i)$。这是因为用户的评分都是相对的。有些用户可能很少对任何项目都评分为1，而其他用户的评分可能从来不会低于3。因此，最好用相对用户的平均得分来进行预测。

尽管基于聚类的方法和基于最近邻的方法在本质上是相似的，但最近邻算法对噪音更具健壮性。而且，使用最近邻方法不需要选择聚类代价函数，只需要选择一个相似度方程，因而使问题简化。实验结果证明，在很多数据集上，使用最近邻算法和相关系数相似度预测评分的方法，要超出基于聚类的方法的表现。据此，采用相关系数相似度的最近邻算法，对于许多实际的协同过滤任务都是很好的选择。

3. 评价

协同过滤推荐系统可以通过多种方式评价。标准的信息检索指标，如第8章所介绍的精确率、准确率、召回率和F值，都可以用来评价。

然而，标准的信息检索指标非常严格，即要求系统预测必须完全正确。例如，如果用户的评分为4而系统A预测值为3，系统B预测值为1，那么系统A和系统B的精确率都是0，因为它们都未能精确地预测正确结果。然而，系统A要比系统B更接近于正确答案。基于这个原因，评分与预测评分之间的差异常被用作评测。其中一种方法称为绝对误差（absolute error），其计算公式如下：

$$ABS = \frac{1}{|\mathcal{U}||\mathcal{I}|}\sum_{u\in U}\sum_{i\in I}|\hat{r}_u(i) - r_u(i)|$$

这里求和操作针对集合中所有预测过的用户/项目对进行。另一个评价指标称为均方误差(mean squared error, MSE)，按照下式计算：

$$MSE = \frac{1}{|\mathcal{U}||\mathcal{I}|}\sum_{u\in U}\sum_{i\in I}(\hat{r}_u(i) - r_u(i))^2$$

绝对误差与均方误差最大的区别是，均方误差对错误预测的惩罚更大，这是由于惩罚被平方了。

以上是针对协同式推荐系统最常用的评价指标。那么到底应该选用哪种评价指标呢？遗憾的是，这并不是容易回答的问题。正如整本书反复指出的那样，要根据具体的任务选择合适的评测指标。

10.5 P2P搜索和元搜索

10.5.1 分布式搜索

我们已经介绍的社会化搜索的应用中，包括的是人构成的网络或社区，但是有很多发现和共享信息的工具，是在“节点”社区上实现的，每一个节点存储并搜索信息，节点之间也可以相互通信。最简单的分布式搜索环境为元搜索引擎（metasearch engine），此时每个节点是一个完全的搜索引擎。元搜索引擎从不同的搜索引擎采集相对小规模的搜索结果，并将搜索结果融合以获得更好的搜索效果。另一方面，一个P2P（peer-to-peer）搜索应用则通常拥有大批节点，每一个节点有相对较小的数据，而且对其他节点的情况知之甚少。

与仅使用单一的文档集合的搜索应用相比，所有的分布式搜索[⊖]应用必须实现3个额外的功能：

- 资源表示（resource representation）：生成信息资源的描述（例如文档）存储在节点上。
- 资源选择（resource selection）：基于对资源的描述，选择一个或多个资源进行搜索。
- 结果融合（result merging）：融合由包含所选择的信息资源的节点检索到的结果列表。

这些功能通过特定节点来执行，并依赖于应用的体系结构。最简单的假设是，有一个特殊的节点提供选择和融合的目录服务，而其他每个节点负责提供自身的资源表示。对于一个具有图10-10所示的体系结构的元搜索应用，资源表示和选择功能是简单的。并非针对查询特点来选择搜索引擎，而是由元搜索引擎将查询广播给使用的所有搜索引擎。对每个搜索引擎，通过应用程序接口（Application Programming Interface, API）将查询转化为搜索引擎可以理解的适当形式。由于多数搜索引擎的查询语言都很相似，这种转换一般是很简单的。

更一般地，在分布式搜索环境中，每个节点可以表示为在节点上存储的文档中词项出现的概率，即第7章介绍的用来表示文档的unigram语言模型。在这种情况下，仅用一个语言模型表示节点上的所有文档，概率通过将所有文档中的词频相加来估计，也就是说，节点上存储的所有文档被当做一个大文档来估计语言模型。这种表示是紧凑的，并且在分布式检索实验中被证明效果良好。在一些应用中，节点可以不必服从分布式检索协议，而仅仅提供搜索

⊖ 通常称为分布式信息检索（distributed information retrieval）或联合搜索（federated search）。

API即可。节点内容的语言模型描述可以通过基于查询的采样（query-based sampling）生成。这包括生成一系列查询并从节点上检索文档样本，这些样本被用来估计语言模型。选择查询词有不同的策略，但是即使是从检索的文档中随机产生的查询，也被证明能够生成准确的语言模型。

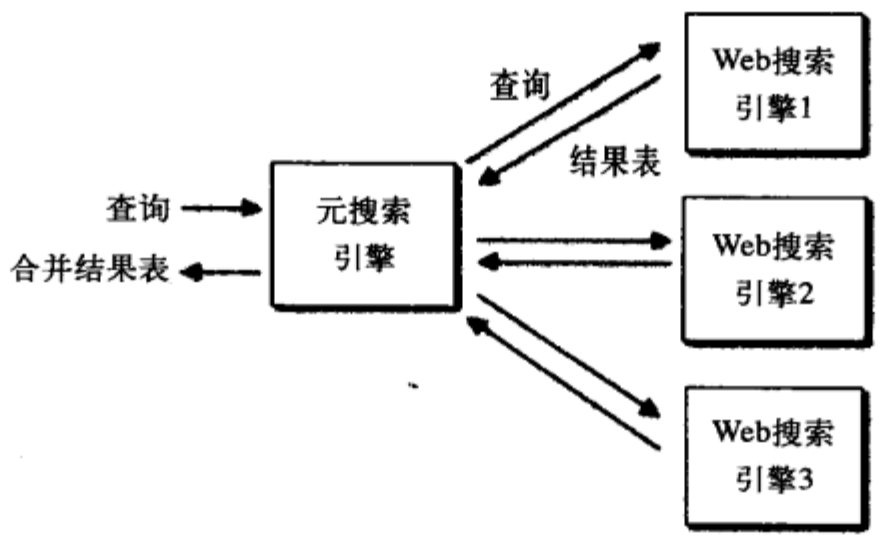


图10-10 元搜索引擎体系结构。查询被广播到多个搜索引擎并融合得到的结果列表

在一般的分布式搜索应用中，资源选择包括，首先利用节点的表示对节点进行排序，然后选择排序最前的K个节点或者所有分数超过一定阈值的节点。既然节点被表示为语言模型，我们很自然地想到使用查询似然（query likelihood）对文档进行排序。在分布式搜索的文献中，有时也称为KL距离资源排序算法，因为查询似然是KL距离的一种特殊情况。根据7.3.1节给出的查询似然值，节点N将按照下式进行排序：

$$\log P(Q|N) = \sum_{i=1}^{n} \log \frac{f_{q_i,N} + \mu P(q_i|C)}{|N| + \mu}$$

当节点被选定之后，在这些节点上执行本地搜索（local search）。这些搜索的结果将被融合为一个统一的排序。如果使用相同的检索模型（如查询似然）和相同的全局统计（如背景模型），融合仅基于本地搜索的分值。如果每个节点上采取不同的全局统计，如仅利用本地节点上的文档计算idf权值，那么融合之前，将通过共享这些统计重新计算本地搜索的分值。如果使用不同的检索模型，或者全局统计无法共享，那么在融合之前，分数必须被归一化。一个常用的启发式归一化方法为，利用文档d所在节点在节点排序时的分值R_d来修正本地文档分值S_d。如下式所示：

$$S'_d = S_d(\alpha + (1-\alpha)R'_d)$$

其中α为常量，R'_d为根据其他资源分值归一化后的资源排序分值。归一化资源分值的一种方法是，为计算针对指定查询的最小和最大可能分值R_{min}、R_{max}，而后计算：

$$R'_d = (R_d - R_{\min})/(R_{\max} - R_{\min})$$

也可以通过比较文档集合中的采样与本地分值来学习一个归一化方程（Si&Callan, 2003）。

元搜索应用中的结果融合与一般的分布式搜索不同。元搜索的两个特点是，本地搜索的文档分值一般是无法获得的，以及本地搜索经常在内容相似的文档集合上进行。例如，元搜索引擎使用的多个网络搜索引擎基本上在相同的数据集（互联网）上使用不同的检索模型。

这种情况下，各节点本地搜索产生的排序列表之间，在文档上有很大重叠。有效的文档融合算法应该特别处理这种情况。研究最多的方法可以用下面的方程描述，给定修正后的文档分值S'_d，作为分值$S_{d,i}$的函数，其中$S_{d,i}$为第i个搜索引擎产生的分值：

$$S'_d = n_d^{\gamma} \sum_{i=1}^{k} S_{d,i}$$

其中n_d为结果列表中返回文档d的搜索引擎数，γ=(−1, 0, 1)，有k个搜索引擎返回结果。当γ=−1时，修正分值为所有本地分值的平均值；当γ=0时，修正分值为本地分值之和；当γ=1时，修正值为本地分值加权求和，权值为返回文档d的搜索引擎数。最后一个版本称为CombMNZ (combine and multiply by the number of non-zero results)，该方法在许多搜索引擎结果融合的实验中被证明有效。

通常，在典型的元搜索应用中，分值是不可知的，因而常用文档排名来代替。在这种情况下，可以使用CombMNZ方程基于排名计算分值。如果结果列表中有m个文档，排名为r的文档的分数将是$(m-r+1)/m$。基于排名的CombMNZ产生的融合排序，尽管要比基于分值的方法要差一些，但也相当有效。更有效的基于排名的融合算法，可以利用投票的技术来实现(Montague&Aslam, 2002)。

一般来说，分布式地搜索非重叠的文档集与搜索所有文档集合并得到的单一文档集的效果相当。当然，大多数应用无法构建这样的文档集，但这可以作为有用的评价效果基准。TREC数据集上的实验表明，当采用R-准确率，R=10作为评价标准时，从200个数据集中选择5或10个数据集的分布式搜索，至少和集中式搜索一样有效，有时甚至更有效（Powell 等，2000）。另一方面，在一个P2P测试平台上，数据集被分布在2500个节点上，仅选择其中1%的节点，R=10时的R-准确率要比集中式搜索低25%（Lu&Callan，2006）。

元搜索融合了在相同或近似的数据集上的不同搜索，与单独搜索相比，一般提高了检索效果。在TERC上进行的元搜索实验证明，当使用4个不同的搜索引擎进行结果融合时，比使用其中最好的搜索引擎返回的结果，在平均准确率指标上（依赖于测试使用的查询集合）提高了5%～20%（Montague&Aslam，2002）。

10.5.2 P2P网络

P2P网络在一系列包含用户社区的应用中被采用，尽管曾经被推广到音乐和视频的文件共享系统中，如KaZaA和BearShare。在文件共享应用中进行搜索，一般通过限定具体的标题或其他属性，如艺术家，来找到文件。换句话说，它们支持简单的完全匹配搜索（见第7章）。许多不同的网络体系结构或覆盖网络（overlays）[⊖]被开发出来，以支持这种类型的搜索。图10-11给出了其中3种体系结构。

P2P网络中的每个节点都扮演客户端、服务器或既是客户端又是服务器的角色。客户端（信息消费者）提出查询进行初步搜索。服务器（信息提供者）返回文件（如果有匹配的文件）对查询进行响应，也可将查询路由到其他节点。负责维护其他节点信息并提供目录服务的服务器称为hub节点。图10-11中体系结构的差异，主要在于查询是如何被路由到提供商的。第一个体系结构是文件共享应用先驱Napster的基础，一个中心hub节点提供目录服务。消费者发

⊖ 覆盖网络描述了物理网络（通常是互联网）上层实现的节点间的逻辑连接。

送查询到hub节点，随后hub节点将这些查询路由到包括匹配的文件的节点上。尽管这个体系结构很有效，但如果中心hub节点遇到故障或者受到攻击，整个体系结构将受到影响。

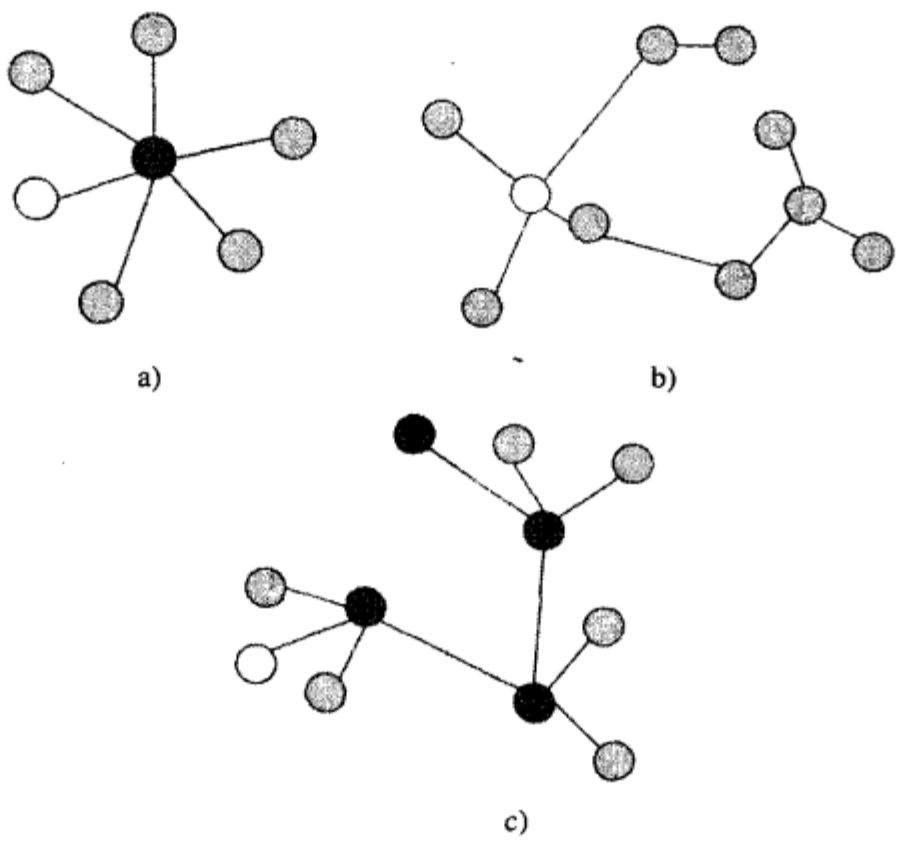


图10-11 分布式搜索的网络体系结构。a）中央hub；b）纯粹的P2P；c）层次的P2P。黑色的圆为hub或超级节点，灰色的圆表示提供商节点，白色的圆表示用户节点

第二个体系结构称为“纯粹的”（pure）P2P（例如Gnutella 0.4[⊖]），这里没有hub节点，用户生成的查询基于泛洪（flooding）算法被广播到网络上的其他节点，这意味着一个节点将查询发送给所有与它相连的节点，这些节点再将查询发送给所有相连的节点，以此类推。在过期之前，查询具有有限视野，这限制了网络跳数（hop）。节点间的连接是随机的，每个节点仅仅知道它的邻居。这种体系结构的问题是扩展性不够好，因为网络流量随着连接的用户数呈指数性增长。

第三个体系结构是层次P2P网络或超级节点网络，这种网络是对纯粹的P2P网络的一种改进。例如Gnutella 0.6标准。在层次网络中，是有一个hub节点和叶节点的两层结构。叶节点可以是提供商或者消费者，仅与hub节点相连。hub节点为与其连接的叶子节点提供目录服务，并能向其他hub节点推送查询。

所有这些网络体系结构都可以作为分布式搜索的基础，而不仅仅是文件共享中的完全匹配。如前面所述，层次网络具有健壮性和可扩展性的优点（Lu&Callan, 2006）。在分布式搜索应用中，网络中的每一个提供商节点在本地文档集合进行搜索。消费者节点为用户提供接口以提交查询，hub节点获取邻接的hub节点和提供商的资源描述，这些资源描述用来提供资源选择和结果融合服务。特别地，邻接的资源描述可以用来比泛洪更有效地路由查询，提供商的资源描述可以用来对本地文档集合进行排序。P2P系统中，每个hub节点必须能够决定使用多少提供商来响应查询，而不是选择排名靠前、数目固定的提供商。

邻接的资源描述是查询路由过程重要的组成部分。hub节点H_i在朝向hub节点H_j的方向的

⊖ 表示 Gnutella标准的0.4版本。见http://en.wikipedia.org/wiki/Gnutella。

邻居，为一个查询在一定跳数之内可以达到的hub节点集合。图10-12展示了一个有三个邻居的hub节点的例子，这里设定最大跳数为3，这样定义邻居的好处是通过转移几个跳数获得提供商的信息。与仅使用最近节点相比，提高了查询路由的效果。

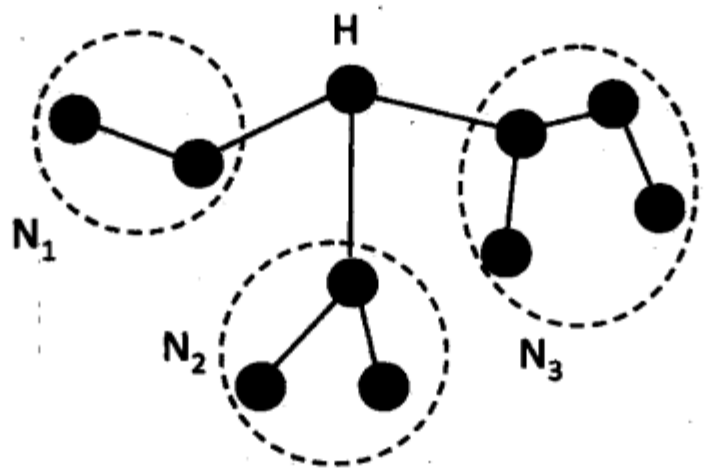


图10-12 层次P2P网络中，hub节点（H）及其邻居（N_i）

hub节点的资源描述是所有与其连接的提供商节点的资源描述的整合。换句话说，是一个语言模型，记录了词语出现的概率。当前hub节点的邻居资源描述是邻近的hub节点的资源描述的整合，但是随着距当前hub节点的跳数增加，邻近hub的贡献减少。换句话说，最近的邻居hub节点对邻居资源描述贡献最多。

在层次P2P网络上实现的分布式搜索实验表明，其效果与上节介绍的使用一个中心hub节点的体系结构相当。更具体地，使用邻居和提供商资源描述，在P2P实验平台上选择2500个节点的1%的方式与使用中心hub节点选择1%的节点的方法，具有相同的平均准确率，但是网络流量为使用查询泛洪协议的三分之一（Lu&Callan，2006）。

文件共享系统另一个流行的体系结构为结构化（structured）网络。这些网络以一个键值关联于每个数据项，并且使用分布式散列表（distributed hash table, DHT）散发这些键值。分布式散列表仅支持完全匹配搜索，既然它应用于很多实际问题，在这里就简要介绍如何利用分布式散列表来定位文件。

在DHT中，所有的关键字和节点都表示为m位的数字或者标识符。文件的名字利用哈希函数转化为一个键值。这个键值和相关联的文件存储在一个或多个节点上，这些节点的标识符在数值上与这个键值接近。键值之间距离的定义依赖于具体的DHT算法。例如，在Chord DHT中，距离为两个m位数字的数值差异（Balakrishnan et al.，2003）。

为了找到文件，包含该文件名字键值的查询被提交到任一节点。存储与检索文档都依赖于一个节点，该节点可以将请求发送到标识符与键值相近的节点。这保证了请求最终可以找到最近的节点。在Chord中，键值被认为是环上的一个点，如果k_1和k_2是两个邻接节点的标识符，标识符为k_2的节点将负责所有键值落在k_1和k_2之间的请求。每个节点都维护一个包括IP地址的路由表，这些IP地址对应的标识符大概是该节点标识符的一半、四分之一、八分之一，以此类推。一个节点将键值k的查询请求推送到表中标识符不超过k的最高节点。路由表的结构保证，对于N个节点，负责键值k的节点可以在$O(\log N)$跳数内被找到。

参考文献和深入阅读

社会化搜索在信息检索研究中越来越流行。尤其是在过去几年中，社会化标签和协同在线环境吸引了越来越多的关注。考虑到研究投入和新应用的开发仍在进行，这种关注很可能将继续增长。

关于人工索引和自动索引相对效果的主题，在信息检索领域已经讨论多年，最早可以追溯到Cranfield的研究（Cleverdon, 1970）。许多论文，如Rajashekar和Croft（1995），指出自动索引与人工索引相比具有的优势，以及两者相结合可以提高检索的效果。一些学者已经看到标签和其他元数据对提高搜索效果的作用。Heymann、Koutrika和Carcia-Molina（2008）的研究表明社会标签，如deli.cio.us上的标签，对提高搜索是没用的，主要因为较低的覆盖率，以及多数标签已经表示成锚文本。Hawking和Zobel（2007）报告了其他元数据上得到相似的结果，并讨论了语义网的含义。

Sahami和Heilman（2006）以及Metzler等（2007）采取了相似的技术处理短文本匹配。他们利用网页搜索的结果来扩展文本的表示。尽管这些方法在计算查询相似度的背景下进行评价，但是可以方便地应用到标签相似度的计算中。

与图10-2相似的标签云，可以通过网络上已有的软件来生成，如Wordle[⊖]。

本章描述的用于社区发现的HITS算法的细节，可以参考Gibson等（1998）。Hopcroft等（2003）描述了不同的增量式层次聚类的方法。其他关于在线社区发现的方法包括Flake等（2000），该文介绍了许多社区发现算法是如何高效地实现的。Borgs等（2004）研究了如何识别新闻组中的社区结构问题。Almeida和Almeida（2004）则实现了一个社区相关的搜索引擎。最后Leuski和Lavrenko等（2006）描述了如何利用聚类和语言模型分析虚拟世界中用户的行为及交互。

Jeon等（2005）描述了基于社区的问答中问题排序的方法。Xue等（2008）扩展了该方法，他们使用了更有效的方法来估计翻译概率，并且指出将问题和答案结合可以带来更好的排序结果。Jeon等（2006）阐述了基于社区的问答中答案质量及其效用的问题。Agichtein等（2008）结合了更多的特征来预测问题和答案的质量，并指出从描述用户提问与回答的社区图中获得的特征非常重要。

基于社区的问答的概念源于数字参考（digital reference）服务（Lankes，2004）。除了基于社区的问答外，还有其他的搜索任务，如用户搜索、人们已生成的问题答案。其中一些任务如信息检索社区已经有所研究。在FAQ[⊖]中，搜索问题的答案是多篇论文的主题。例如，Burke等（1997）和Berger等（2000）都试图通过考虑同义词和翻译，尝试解决FAQ检索中的词表不匹配问题。Jijkoun和de Rijke（2005）描述了从来源于网络的FAQ数据中检索。Riezler等（2007）描述了基于翻译模型的网络FAQ检索。论坛是问答的另一个资源，Cong等（2008）描述了如何从论坛中抽取问答对来支持CQA服务。

有很多协同检索系统超出了本书我们所介绍的，包括CIRE（Romano et al.，1999）和S^3（Morris&Horvitz，2007a）。Morris（2008）提供了一个很好的关于从业者如何使用协同式网

⊖ http://www.wordle.net/.

⊖ FAQ是Frequently Asked Questions的缩写。

络搜索系统的概述。Pickens等（2008）评价了支持协同式检索的迭代融合排序列表的算法。

Belkin和Croft（1992）对于ad hoc检索与文档过滤之间的联系给出了一种观点。Y. Zhang和Callan（2001）描述了自动确定过滤阈值的有效方法。Schapire等（1998）描述了boosting、一种高级机器学习技术和Rocchio算法可以应用于过滤。最后，Allan（1996）展示了增量式反馈，类似于在线算法，可以用来提高过滤的效果。信息检索的一个研究领域，话题检测与跟踪（topic detection and tracking），关注于新闻文章的话题过滤（跟踪），尽管没有在本章涉及，但可参考Allan（2002）对该研究的概述。

对本章提到的系统过滤算法更全面的了解，请参看Breese等（1998）。更进一步地，Herlocker等（2004）介绍了评价协同式过滤系统的很多方面的细节。

Callan（2000）给出了关于分布式搜索研究的精彩概述。更近的论文，Si和Callan（2004）比较了资源选择技术的效果。一个不同的基于查询采样的资源描述生成算法，称为查询探测（query probing），在Ipeirotis和Gravano（2004）中有介绍。该工作关注于提供对深层网络（deep Web）数据的访问，深层网络数据指只有通过搜索接口才能从网络访问的数据库。

有很多论文描述了融合多个搜索引擎或检索模型输出的技术。Croft（2000）给出了这项研究的概述，Montague和Aslam（2002）给出了更近一些的研究工作。

对P2P搜索的概述以及如何在层次网络体系下实现分布式搜索的详细情况，参见Lu和Callan（2006, 2007）。

练习

10.1 指出社会媒体标签与锚文本的相似点与不同点。

10.2 实现两个度量标签之间相似度的算法。第一个算法要求采用一个标准的检索模型，如语言模型。第二个算法要求利用网络或其他资源来扩展标签的表示。利用有10～25个标签构成的数据集合测试两种算法的效果。请描述算法、评价标准、标签集合及结果。

10.3 计算HITS（见算法3）和PageRank（见图4.11）算法在图10-3上的五步迭代。比较PageRank值和HITS算法得到的权威值和中心值。

10.4 描述两种本章没有介绍的在线社区的例子。怎样运用本章提到的社区发现算法来检测这两种社区呢？

10.5 在网络上找到一个基于社区的问答网站，问两个问题，一个低质量，一个高质量。描述这两个问题的答案的质量。

10.6 举出网络上两个文档过滤系统的例子。它们是如何针对你的信息需求建立描述文件的？该系统是静态的还是自适应的？

10.7 列出索引器的基本操作以支持以下任务：1）静态过滤；2）动态过滤；3）协同过滤。

10.8 实现基于最近邻的协同过滤算法。使用一个公开的协同过滤数据集，采用均方误差、欧式距离和相关系数相似度作为评价标准，比较实验效果。

10.9 本章介绍的基于聚类与基于最近邻的协同过滤算法，都利用用户/用户相似度进行预测。利用项目/项目相似度设计这两类算法。如何度量项目之间的相似度呢？

10.10 组建一个2～5人组成的用户群，使用一个已有的协同搜索系统。描述你们的体验，包括该系统的优点和缺点。

10.11 针对如何估计指定查询的最大资源排序分值和最小资源排序分值R_{max}和R_{min}，提出你的

建议。

10.12 针对一组查询样例，使用CombMNZ基于排名的版本融合两个搜索引擎的结果。评价融合后的排序结果，并与两个搜索引擎单独的结果列表进行比较。

10.13 选择你最喜欢的文件共享应用，并且了解其如何工作。描述这个应用，并与本章提到的P2P网络进行比较。

10.14 具有小世界（small-world）属性的P2P网络中，任何两个节点都由小数目的跳数连接。这些网络的特点是，一个节点有到较近节点的本地（local）连接和到远距离节点的长距离（long-range）连接，这里距离通过内容相似度或其他属性如等待时间来计算。你认为Gnutella 0.4或0.6是否具有基于内容的小世界属性？基于Chord的结构化网络呢？

第11章 超越词袋

“这意味着未来就在这里，原来的预测不算数了。”

——Mulder，《X档案》

11.1 概述

“词袋”（bag of words）用来作为检索和分类模型中文本的一种简单表示。在这种表示中，文档被考虑为一组无序词语的集合，它们之间没有句法或者统计上的关系[⊖]。第7章讨论过的许多检索模型，都是基于词袋表示的，例如查询似然模型、BM25模型、向量空间模型等。从语言学家的角度来看，词袋的表示能力非常有限。没人能够读懂一个排序后的词袋表示，并获得与正常文本表达一样的意思。例如，“No one could read a sorted bag of words representation and get the same meaning as normal text”排好序的版本是“a and as bag could get meaning no normal of one read representation same sorted text the words”。

尽管具有明显的限制，词袋表示仍然已经在检索实验中比更加复杂的文本表示方案获得了更大的成功。在基于词语的表示中可以结合简单的短语和邻近词语。这看起来似乎具有明显的优点，但是在检索模型具有显著一致的有效性之前，这种结合却花费了很多年的研究。然而，搜索应用已经超越了用词袋表示文档和查询的阶段。对于这些应用，需要基于许多不同特征来进行表示和相关排序。从词袋中衍生出来的特征仍然很重要，但是语言学、结构化、元数据和非文本特征，也都能有效地应用到检索模型中，例如推理互联网或者Ranking SVM。本章将从检验基于特征的检索模型的一般属性开始。

在前面的章节中，已经讨论了很多种特征表示以及如何将它们用于排序。在本章中，将在四个方面更详细地考察表示方法，并阐述它们是如何影响搜索引擎未来发展的。词袋模型假设词语之间没有关系，所以首先查看词项依赖性如何被捕捉并用于线性的基于特征的模型中。在词袋表示中，文档结构是被忽略的，但是已经看到它对于网络搜索的重要性。表示方法的第二个方面，就是数据库系统中的结构化表示如何用于搜索引擎。在词袋表示中，查询像文档那样被处理。但是，在自动问答应用中，查询的句法结构具有特殊的重要性。表示方法的第三个方面是，查询结构是如何用于自动问答的。最后，词袋是基于词语的，在许多应用中，用来表示被检索对象的特征可能不是词语，例如图像检索或音乐检索。因此，第四个方面就是，这些非文本特征都有哪些，以及它们如何用于相关排序。

在本章的最后一节，将对搜索的未来进行一些适当的构想（不是妄想）。

11.2 基于特征的检索模型

在第7章中，简要介绍了基于特征的检索模型，这里提供更加详细的介绍，因为这个模型对现代搜索引擎的重要性越来越大。

⊖ 在数学中，一个袋就像一个集合，但是允许重复（即一个词语可以多次出现）。

对于文档集合D和查询集合Q，可以定义包含参数向量Λ的打分或者排序函数$S_\Lambda(D; Q)$。给定一个查询Q_i，打分函数$S_\Lambda(D; Q_i)$对每个$D \in \mathcal{D}$计算一次，然后所有文档根据它们的得分降序排序。对于基于特征的线性模型，限定文档的打分函数为如下形式：

$$S_\Lambda(D;Q) = \sum_j \lambda_j \cdot f_j(D,Q) + Z$$

其中$f_j(D; Q)$是一个特征函数，将一对查询和文档映射为实数；Z是一个常量，不依赖于D（但可能依赖于Λ或者Q）。特征函数对应于前面提到的那些特征。虽然一些模型允许特征的非线性组合，但是迄今用于研究和应用的打分函数，都是基于线性组合的。由于这个原因，这里关注基于特征的线性模型。注意，这个排序函数是第5章中描述的抽象排序模型的泛化。

为了定义打分函数的形式，还需要明确寻找最优参数的方法。因此，需要一组训练数据T和一个评价函数$E(R_\Lambda; T)$，其中R_Λ是所有查询根据打分函数获得的一组排序结果。给定排序列表和训练数据，评价函数返回实数值的输出。注意，E只需要考虑文档的排序而不需要文档的得分，这是第8章中描述的评价度量的标准特点，例如平均精确率、P@10、NDCG。

基于线性特征的检索模型的目标是，寻找一个参数设置Λ，使得可以对训练数据最大化评价函数E。这一点可以形式化表示为：

$$\hat{\Lambda} = \arg\max_\Lambda E(R_\Lambda; T)$$

其中R_Λ是线性打分函数$\Sigma_j \lambda_j \cdot f_j(D, Q)+Z$返回的相关排序结果。

对于数量较少的特征，最佳参数值可以在所有可能值的整个空间中进行暴力搜索来获得。对于数量较多的特征，就需要一种优化步骤，例如Ranking SVM模型中所采用的优化。基于特征的线性模型对比其他检索模型的核心优势，就是可以方便地在模型中加入新的特征，并且在给定训练数据时，可以进行有效的优化。这些优点使基于特征的线性模型成为对本章中讨论的各种特征表示进行整合的理想框架。

一个数量相对较少的特征集合，已经被用来作为第7章中描述的、主题相关的检索模型的基础。它们包括：

- 词项出现：一篇文档中某个词项是否出现。
- 词项频率：一篇文档中某个词项出现的次数。
- 倒置文档频率：包含特定词项的文档数目的倒数。
- 文档长度：文档中词项的个数。
- 词项邻近：一篇文档中的词项共现模式（融合词项依赖性的最普通的形式）。

Galago查询语言（以及基于此的推理网络模型）提供了一种确定特征范围，并根据这些特征的线性加权组合对文档打分的方法。更一般地，Galago的#feature操作可以支持任意特征的定义和组合。例如，使用这个操作，可以将根据BM25的词项加权函数的特征作为打分函数的一部分。像推理网络模型那样，Galago并没有指明特定的优化方法来寻找最佳的参数值（即特征权值）。

11.3 词项依赖模型

第4章中讨论了作为短语的一部分的词语之间关系的潜在重要性。在第5章，展示了词项邻近如何被融入到索引中。第6章描述了度量词语之间关联性的技术。第7章介绍了词项关系

如何采用Galago查询语言进行表示。探究词语之间的关系，显然是构建高效搜索引擎的一个重要部分，尤其是对于网络搜索这类应用，大量文档包含了全部或者绝大多数的查询词。采用词项关系的检索模型经常被称作词项依赖模型。因为它们不假设词语的出现是彼此独立的。更一般地，词项依赖信息可以被整合到大量用于排序算法的特征中。

7.5节描述的Galago实现的网络搜索基于一种特定的基于特征的线性模型，称为马尔可夫随机域（Markov Random Field, MRF）模型（Metzler & Croft, 2005b）。这个模型除了允许任意特征外，还明确表示词项之间的依赖性。尽管已经有人提出了大量的词项依赖模型，这里仍然介绍一下MRF模型。因为它在文档排序和伪相关反馈的相关过程中，都具有显著的性能改进（Metzler & Croft, 2007a）。

MRF模型首先构建一个图，由一个文档节点和每个查询词项对应的节点组合而成。这些节点表示马尔可夫随机域中的随机变量。这是一种表示联合分布的一般方法。由此，在MRF模型中，覆盖文档随机变量和查询词项随机变量的联合分布开始由模型处理。马尔可夫随机域一般表示为图，因此被看成是一种图模型。特定情况下，MRF是无向图模型，图中的边都是无向的。第7章中描述的推理网络模型就是有向图模型的例子。

MRF模型中，通过绘制一条随机变量之间的边来描述它们之间的依赖性。由于查询词项的重要性依赖于文档，文档节点总是连接着每个查询词项节点。直接描述查询词项依赖性的方法是，在查询词项节点之间连线。有很多种可能方式来确定哪个词项节点需要绘制一条边，图11-1中概括了这些不同的情况。在最简单的情况下，查询词项之间没有绘制任何边，这对应于全独立假设，即查询词项之间不存在任何依赖性，这等同于第7章中描述的一元语言模型或者词袋模型中的任何词语。另外一种可能是对相邻的查询词项画边，这就是序列依赖假设。这个假设认为邻近的词项之间是相互依赖的，但是更远的词项就没有关系了。这种类型的假设类似于二元语言模型。还有一种可能的假设是，所有的词项都依赖于其他所有词项，这就是全依赖假设。最后一种可能是，查询词项之间的边根据某些有意义的方式绘制，例如自动或者手工识别那些相互依赖的词项，这称为一般依赖。但是，实践证明，使用序列依赖假设是最佳选择。事实上，所有尝试手工或者自动确定哪些词项之间具有依赖性的方式，都不如使用最简单的假设相邻词项之间相互依赖。

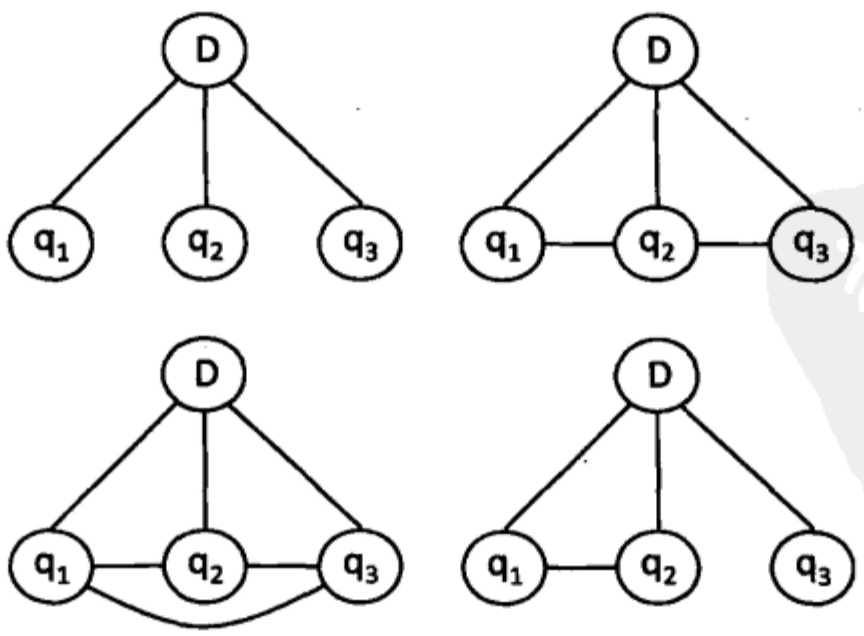


图11-1 马尔可夫随机域模型假设的例子，包括全独立（左上）、序列依赖（右上）、全依赖（左下）、一般依赖（右下）

MRF图建立以后，必须在图中的团（cliques）上定义一组势函数。势函数就是团中随机变量观察值之间相容性的度量。例如，在图11-1中的序列依赖图中，由词项q_1、q_2和文档D构成的团的势函数可以计算文档D中短语“q_1 q_2”精确出现的次数，或者这两个词项在彼此的某个窗口中一起出现的次数。因此，这些势函数都非常通用，可以计算文本中大量的不同特征。基于这种方式，MRF模型比其他模型更加强大，例如语言模型或者BM25模型。因为它允许依赖性和任意特征被直接加入到模型中。

通过构建具有特定形式的查询，Galago搜索引擎可以用来效法TREC实验中非常有效的MRF模型。这种方法也用在7.5节中。例如，给定查询president abraham lincoln，全独立MRF模型可以采用下述查询来计算：

```
#combine(president abraham lincoln)
```

注意，这是最基本的形式，没有考虑词项之间的任何依赖。序列依赖的MRF模型可以通过采用下述查询计算：

```
#weight(0.8 #combine(president abraham lincoln)
        0.1 #combine(#od:1(president abraham)
                     #od:1(abraham lincoln)
        0.1 #combine(#uw:8(president abraham)
                     #uw:8(abraham lincoln)
```

这个查询形式由三部分构成，每部分都对应用于MRF模型的一个确定的特征类型。第一部分计算匹配单独词项的贡献得分，第二部分计算查询中匹配子串短语的贡献得分。这允许精确匹配“president abraham”和“abraham lincoln”的文档获得更高的得分。注意，这些短语只由邻近的查询词项构成。因为序列依赖模型只考虑邻近查询词项的依赖性。形式中的第一部分计算邻近查询词项在无序窗口中的匹配贡献得分。特定情况下，如果词项“president”和“abraham”一起出现在8个词项的窗口中，文档的得分就会增加。注意，每个部分都是被加权的，在单独词项部分权值为0.8，精确短语和无序窗口部分的权值为0.1。这些来自经验的权值表明了，文本中匹配单独的词项比匹配完整的短语和邻近特征具有相对更大的重要性。虽然其他特征也显示了重要性，但是单独的词项是最重要的匹配。

在Galago中，将纯文本的查询Q转换为序列依赖的MRF查询是非常简单的。第一个部分（对于单独的词项）是简单的#combine(Q)，第二部分是将每对邻近的查询词项放入#od:1操作符中，并组合这些操作符到一个#combine中，第三部分和最后部分将每对邻近查询词项放入一个#uw:8操作符中，并采用#combine结合起来。三个部分分别给定权值0.8、0.1、0.1，采用前面介绍过的#weight操作符整合起来。

最后，全依赖MRF模型最为复杂，因为考虑了更多的依赖性。但是，为了完整性，展示全依赖MRF模型的查询如下：

```
#weight(0.8 #combine(president abraham lincoln)
        0.1 #combine(#od:1(president abraham)
                     #od:1(abraham lincoln)
                     #od:1(president abraham lincoln))
        0.1 #combine(#uw:8(president abraham)
                     #uw:8(abraham lincoln)
                     #uw:8(president lincoln)
                     #uw:12(president abraham lincoln)))
```

值得重视的是，为了计算不同的单独词项、精确短语和无序短语特征，这里所有例子中使用的#combine操作可以方便地由#feature操作替换。例如，可以使用#feature来实现BM25加权，并基于Galago的默认权值来计算依赖模型。

MRF模型还能用来模拟伪相关反馈中的依赖性。伪相关反馈在7.3.2节中是一种重要的查询扩展技术。图11-2比较了用于伪相关反馈的相关性模型的图模型和被称为潜在概念扩展[⊖]的MRF方法。相关模型的图（上面那个）表示一个词袋或者一元模型，是在给定一个特定文档词语的出现后彼此无关的模型。伪相关反馈使用一组排序较高的文档来估计给定查询词语时扩展词语（那些问号）的概率。那些具有最高概率的词语被加入到查询中。

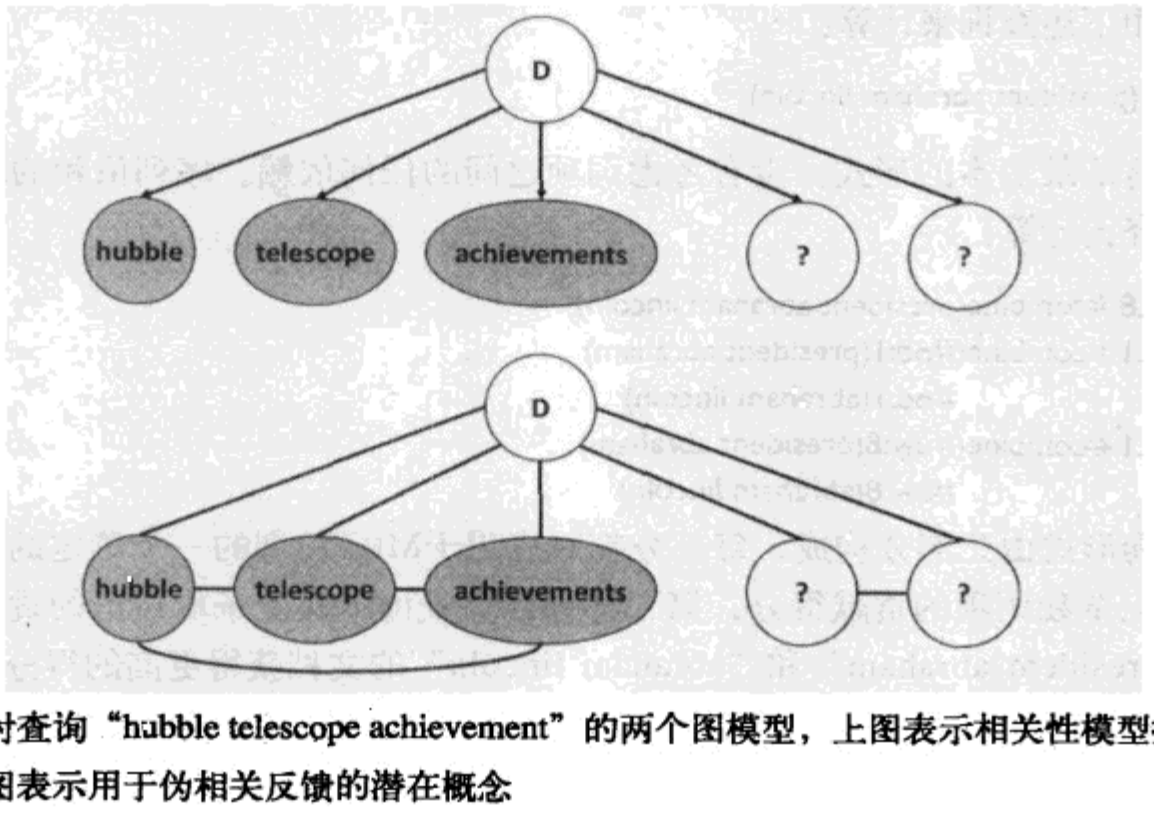


图11-2 针对查询“hubble telescope achievement”的两个图模型，上图表示相关性模型技术，下图表示用于伪相关反馈的潜在概念

下图是隐含概念扩展图，展示了查询词语和扩展词语之间的依赖性。伪相关反馈过程将排序靠前的文档用来估计可能的扩展词语的概率，但是依赖性会改变估计那些概率的方式，一般会得到更好的结果。为了对扩展词语之间的依赖性建模，隐含概念扩展模型能够产生多词短语来作为扩展词项，而不仅仅是词语。例如，表11-1展示了对于查询“hubble telescope achievements”采用隐含概念扩展模型获得的排序靠前的一词和二词概念。

表11-1 对于TREC ROBUST数据集中的查询“hubble telescope achievements”，采用隐含概念扩展模型在最靠前的25个文档上获得的排序靠前的一词和二词概念

一词概念	二词概念	一词概念	二词概念
telescope	hubble telescope	test	hubble mirror
hubble	space telescope	new	NASA hubble
space	hubble space	discovery	telescope astronomy
mirror	telescope mirror	time	telescope optical
NASA	telescope hubble	universe	hubble optical
launch	mirror telescope	optical	telescope discovery
astronomy	telescope NASA	light	telescope shuttle
shuttle	telescope space		

⊖ 潜在概念或者隐含概念是用户在表达查询时头脑中思考的词语或者短语，并没有明确提到。

概括一下，MRF模型是一种基于特征的线性检索模型，是一种在排序文档使用的打分函数中，有效融合基于词项依赖特征的方法。在MRF框架中，隐含概念扩展支持伪相关反馈。隐含概念扩展可以看成是“特征扩展”技术，因为它能通过基于扩展的查询得到新特征，并加入到原来的特征集合中。

11.4 再谈结构化

建立一个常规平台来处理结构化数据和非结构化数据，是长期存在的目标，可以追溯到20世纪60年代。从数据库和信息检索角度，已经提出了许多方法，但是寻找解决方案的动机却在不断增长，因为海量规模的网络数据库诞生了，曾经一度只是信息检索感兴趣的领域，例如统计推理和相关排序，现在都变成了数据库研究人员的重要话题。并且都在共同关心对互联网规模数据提供有效索引和优化技术。探究文档结构是网络搜索的关键部分，有效融合不同来源的证据是许多数据库应用的重要部分。整合有许多方式，例如扩展数据库模型，来更有效地处理概率。扩展信息检索模型用来处理更加复杂的结构和多元关系，或者开发通用的模型和系统，诸如网络搜索、电子商务和数据挖掘等应用，提供了测试平台，基于这些测试平台，我们可以对这些系统进行评测和对比。

在第7章中，展示了文档结构是如何采用Galago查询语言处理的。从传统数据库的角度来看，使用Galago来表示和查询数据存在问题。按照关系数据库术语来说，这里没有模式[⊖]，不能定义属性的数据类型，没有关系之间的联合操作[⊖]。相反，正如第7章所述，一篇文档表示为一个由标签对定义的上下文（可能嵌套）集合。文档简单地存储在数据库中，只能通过主键访问，主键就是文档的标识符。查询语言支持基于文档结构和内容的搜索特征的定义和组合，带有不同上下文的不同文档，可以整合到一个单独的Galago数据库中。每个文档都只被它包含的上下文索引。虽然没有办法定义上下文的数据类型，但是可定义关联特定数据类型的操作符，并应用到特定的上下文中。例如，时间范围操作可以应用到包含文档创建日期的上下文中。

虽然和全关系数据库系统的功能非常不同，但是在许多包含搜索引擎的应用中，这种额外的功能并不需要。例如，第3章中介绍的BigTable存储系统，就没有数据类型或连接操作。另外，它只有一个非常简单的元组和属性名称的规范。诸如BigTable的系统专注于提供对许多组件可能失效的环境下的数据一致性和可靠数据访问，以及采用分布式计算资源的可扩展能力。数据访问提供简单的API，从允许客户端应用到读、写、删除数值。图11-3概括了诸如网络搜索或者电子商务等搜索引擎和数据库系统应用中提供的功能。注意，搜索引擎建立的索引不存储在数据库系统中。

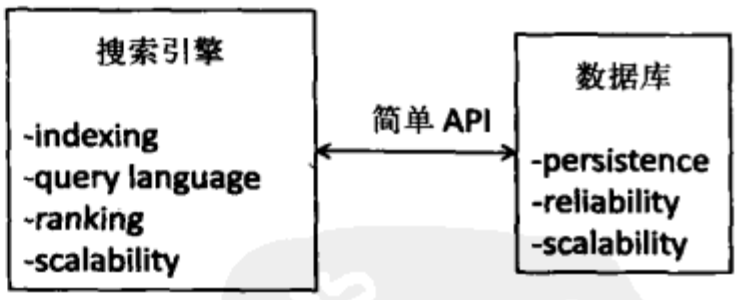


图11-3 搜索引擎和简单数据库系统交互时需要的功能

⊖ 模式是数据库中逻辑结构的一种描述，这种描述可以是关系（表）的名称和每种关系的相关属性。

⊖ 联合操作根据一个或多个普通属性来连接不同关系的元组（行）。一个例子是，根据产品数量属性连接产品信息到厂商信息上。

11.4.1 XML检索

XML是一项重要的标准，用于在应用之间交换数据并对文档编码。为了更支持面向数据的操作，数据库领域已经定义了用于描述XML数据结构的语言（XML模式，Schema），以及对XML数据的查询和操作（XQuery和XPath）。XQuery是一种类似于数据库语言SQL的查询语言，最主要的区别就是，它必须处理层次化的XML数据结构，而SQL处理简单关系数据中的表格结构。XPath是XQuery的子集，用于将搜索限定在单类型XML的数据或文档上。例如，XPath可以用于在XML电影数据库中查找由特定人导演或者在特定年代发布的电影。XQuery可以用来组合有关演员的信息，假设XML电影数据库包含电影和演员“文档”。一个例子可以是，查找澳大利亚出生的明星演员演出的电影。

复杂的数据库查询语言，例如XQuery，专注于数据的结构，组合那些文本搜索应用一般很少使用的数据。针对XML文档数据库查询中哪些部分有用的扩展研究，出现在INEX项目中[⊖]。这个项目采用了TREC中搜索评测的方法。这意味着定义了许多XML搜索任务，并且为了测试这些任务，构造了适当的测试数据集。这些评测中的一类查询叫做内容和结构（content-and-structure, CAS）查询。这些查询包含一个话题的描述和XML结构中的明确引用。CAS查询使用的简化版的XPath叫做NEXI[⊜]。这种查询语言中，两个重要的结构是路径（paths）和路径过滤（path filters）。路径是XML树形结构中对元素（或者节点）的限定。一些NEXI的路径限定例子如下：

//A//B ： XML树中A元素的任何后代元素B[⊜]。后代元素被父元素包含。

//A/* ： 一个A元素的任意后代元素。

路径过滤用于限制结果满足文本或者数值约束。例如：

//A[about(.//B, “topic”)] ：A元素包含关于（about）“topic”的B元素。about谓词没有定义，但是可以通过一些类型的检索模型实现。.//B是一条检索路径。

//A[.//B = 777] ：A元素包含数值等于777的B元素。

早先INEX实验中使用的数据库由计算机科学领域的科技文章组成。下面是一些CAS查询的样例：

```
//article[.//fm/yr < 2000]//sec[about(., “search engines”)]
```

- 查找2000年前发表的包含话题“search engines”相关章节的文章（fm是文章的扉页）。

```
//article[about(.//st,+comparison) AND about (.//bib, “machine learning”)]
```

- 查找章节标题包含词语“comparison”并且参考书目中提到“machine learning”的文章。

```
//*[about:(.//fgc, corba architecture) AND about(.//p, figure corba architecture)]
```

- 查找任何元素包含图片标题关于“corba architecture”并且一个段落提到“figure corba architecture”的文章。

虽然这些查询看起来是合理的，但是INEX实验和前期的研究表明，人们不会在他们的查询中使用结构化线索，或者如果他们被迫需要做的话，一般来说不能正确使用。本质上没有

⊖ XML检索评价计划（INitiative for the Evaluation of XML Retrieval），http://inex.is.informatik.uniduisburg.de/。

⊜ 限定扩展XPath（Narrowed Extended XPath）(Trotman & Sigurbjörnsson, 2004)。

⊜ 这意味着一个元素从<B>标签开始，结束于</B>。

证据表明，用户查询中的结构化能提高搜索效率。由于这个原因，INEX项目已经更加专注于只有内容的查询，就是和贯穿本书讨论的查询一样，并且基于自动技术来对XML元素排序而不是对文档进行处理。

小结一下，结构化对于高效排序中定义特征是很重要的部分，但不是对于用户查询。在诸如网络搜索的应用中，相对简单的用户查询被转换为包含很多特征的查询，包括基于文档结构的特征。Galago查询语言是能用于指定用于排序的特征的例子语言，用户查询能被转换到Galago查询中。NEXI中采用的绝大多数结构化线索，都能采用Galago整合到特征中。

数据库系统被用于很多搜索应用。但是，这些应用的需求和那些典型的数据库应用不一样，例如相关排序。这导致存储系统从数据库模式和查询语言方面发展，这很简单，但也是有效、可靠和可扩展的。

11.4.2 实体搜索

除了挖掘文档中已经存在的结构，还可以通过分析文档内容来创建结构。在第4章介绍的信息抽取技术，能够用来识别文本中的人物、机构和地点。实体搜索使用这种结构来对一个查询返回排序的实体列表，而不是文档列表。为了做到这一点，每个实体的表示都基于文档文本中这些实体周围出现的词语生成。构建这些表示最简单的方法就是，通过累积实体每次出现的一个特定文本窗口（例如20个词语）内的文本来创建“伪文档”。例如，如果机构实体“California institute of technology”在一个语料库中出现了65次，那么这65次出现周围的20个词语中的每个词语，都会被累计到一个伪文档表示中。这种方法用在早期实体搜索的研究中，例如Conrad and Utt (1994)。这些大量的基于词语的表示，能够被存储在搜索引擎中并随后用来根据查询排序实体。

图11-4展示了Conrad论文中实体检索的一个例子。对于这个查询排序靠前的机构名，会和语料库中例如“biomedical”等词语共同出现很多次。

Query:
biomedical research and technology

Top Ranked Results:
minneapolis research
signs inc.
syntex
california institute of technology
massachusetts institute of technology
therapeutic products

图11-4 TREC华尔街日报1987年数据集上实体搜索机构名时的例子

通过目标词语或短语的上下文或局部文本中出现的词语来创建表示的方法，已经用来在查询扩展时自动构建检索词语和短语的辞典（Jing & Croft, 1994）。有趣的是，它也用于认知科学家，作为语义记忆模型的基础（Lund & Burgess, 1996）。

最近，许多关于实体搜索的研究都集中在寻找特定领域或主题的专业人士的方法上。这个任务也称作专家搜索，已经研究了一段时间，但是自从2005年成为TREC的一项子任务后，它才被更加细致地评测。这个研究主要的贡献在于基于语言模型方法发展了对于实体（或者专家）的概率检索模型（Balog et al., 2006）。一般地，给定一个文档集合D和查询q，可以根

据实体和查询词项的联合分布$P(e, q)$来对候选实体e排序。这个分布可以表示为：

$$P(e,q) = \sum_{d \in D} P(e,q \mid d)P(d)$$

如果集中讨论$P(e, q|d)$项，实体排序问题可以分解为两个部分：

$$P(e,q \mid d) = P(q \mid e,d)P(e \mid d)$$

其中$P(e|d)$对应于寻找提供关于实体信息的文档，$P(q|e, d)$涉及那些关于查询的文档中的实体排序。这些概率不同的估计方法会得到不同的实体排序算法。如果假定给定一篇文档，词语和实体是相互独立的，即$P(e, q|d) = P(q|d)P(e|d)$，那么这两个部分就分别通过将q和e用于概率检索实现分开计算。但是，这个假设忽略了出现在同一文档中的词语和实体之间的关系（通过上下文向量方法来获得），并且会使这种方法遇到有效性质疑。相反，可以使用文档中查询词项和实体的邻近贡献来估计e和q之间的关联强度。实现此目的的一种方法是，假设查询由一个词项构成，并且文档中只有一个实体出现，就可以估计$P(q|e, d)$为：

$$P(q \mid e,\ d) = \frac{1}{Z} \sum_{i=1}^{N} \delta_d(i,\ q)k(q,\ e)$$

其中δ_d是指示函数，当出现在d中位置i的词项为q时，结果为1，否则为0；k是一个邻近核函数；$Z = \sum_{i=1}^{N} k(q,\ e)$是一个归一化常量；$N$是文档的长度。

如果查询含有多个词项，可以计算排序得分如下：

$$P(e,\ q) \overset{rank}{=} \prod_{q_i \in q} \left\{ \sum_{d \in D} P(q_i \mid e,\ d)P(e \mid d) \right\}$$

Petkova和Croft (2007)证明，最有效的核函数是高斯核，如表9.1所示，也就是

$$\exp - \| q - e \|^2 / 2\sigma^2$$

其中$q-e$是查询q和实体e之间的词语距离。这篇论文同时证明，对于专家搜索，精确的命名实体识别在性能上并没有巨大的影响，并且使用简单的诸如#od:2 (<first name><last name>)的Galago查询，就可以很好地估计人物实体的概率$P(e|d)$。

11.5 问题越长，答案越好

近来在电影或电视上看到的关于未来的所有幻想中，搜索引擎已经演变为类似于人的能够回答关于任何主题的复杂问题的助理，例如《2001：太空漫游》中的HAL9000计算机系统或者《星际迷航》系列中的计算机。虽然网络搜索引擎提供了访问海量信息的入口，但是距离实现这些智能助理的能力，仍然还有很长的距离。一个明显的差异就是，网络搜索引擎一般都使用较少数量的关键词，而不是采用自然语言表达的实际问题。在第10章中，描述了基于社区的问答系统的用户如何采用句子甚至段落来描述他们的信息需求。因为他们知道，如果问题被很好描述的话，其他人就会阅读并给出更好的回复。相比而言，同样长的查询从网络搜索引擎那里一般会获得非常差的回复，或者根本没有人回答。人们被强制将他们的问题翻译为一个或者多个恰当的关键词来获得合理的结果列表。信息检索研究的一个长期目标，就是开发检索模型来从更长更具体的查询中获得精确的结果。

在第1章中简要提到并在第10章提到的自动问答任务，包括对用户的查询提供一个具体的

答案，而不是一堆文档的排序。这项任务在自然语言处理和人工智能领域已经有很长的历史。早期的自动问答系统依赖于对非常特定的小规模领域进行详细的逻辑表示，例如篮球、月亮岩石或者玩具积木。近期重点转移到信息检索方面，这个任务包括在大规模文本语料上识别并抽取答案。

图11-5展示了从文本语料库上检索答案的自动问答系统的典型模块。这样的系统处理的问题范围往往限定在那些基于事实的简单而且答案简短的问题，例如谁、哪里、何时等以人名、机构名、地点和日期为答案的问题。下面的问题是TREC自动问答评测中的问题样例[⊖]：

谁发明了纸夹？

什么是帝王谷？

圣海伦火山最近一次大爆发是什么时候？

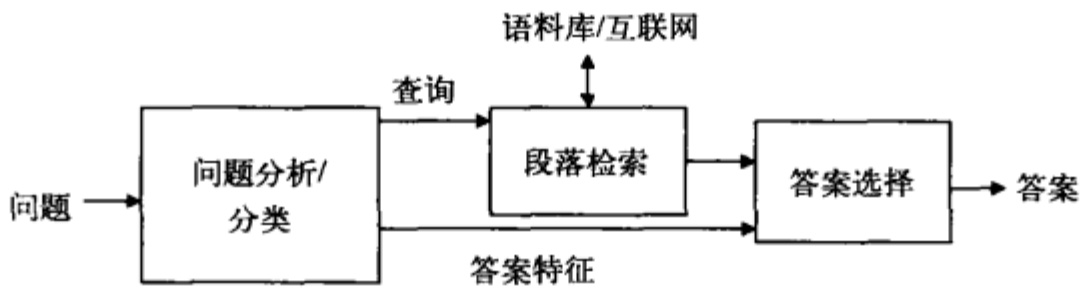


图11-5 自动问答系统框架

当然，还有其他类型的基于事实的问题可能被问到。它们可以采用很多种方式被问及。系统中的问题分析和分类模块的任务，就是根据期望的答案类型对问题分类。对于TREC问答的问题，常用的一种问题划分包括31个不同的主要类别[⊖]。许多都对应于文本中能自动识别的命名实体（见第4章）。表11-2给出了这些类别中每种问题在TREC中的样例，由于问题的多种形式，问题分类是一个比较难的任务。例如，问题词what就可以用到很多种不同类型的问题中。

问题分析和分类得到的信息用于答案选择模块，以在候选文本段落中识别答案。这种答案往往是句子。候选的文本段落是由从问题生成的查询来进行段落检索的模块获得的。文本段落是从特定的语料或者互联网中检索得到的。在TREC自动问答实验中，候选答案段落从TREC新闻语料中检索得到。互联网常常被用来作为一种额外资源。许多自动问答系统中的段落检索模块都是简单地查找那些包含问题中所有非停用词的段落。但是，一般来说，段落检索类似于其他类型的检索。因为与好的段落相关的特征可以结合起来，获得有效的排序。这类特征可以基于问题分析，包含和问题类别相关的命名实体的文本段落，如果同时包含重要的问题词，就显然应该被排在前面。

例如，对于问题“where is the valley of the kings”，那些包含标记为配置并含有词语“valley”和“kings”的句子，应该被优先考虑。一些系统识别与问题类别相关的、可能作为答案的文本模板，通过使用文本挖掘技术对互联网数据处理得到或者直接就是预定义的规则。例如，<location>中的模板<question-location>常被用来寻找答案段落。在这种情况下的question-location就是“valley of the kings”。这种模板的存在应该可以改进文本段落的排序。

⊖ TREC自动问答的问题来自于大量搜索应用的查询日志(Voorhees & Harman, 2005)。

⊖ 这个问题分类是BBN建立的。这个分类和其他内容在Metzler and Croft (2005a)中有讨论。

对于排序段落，另外一种被证明有效的特征是在诸如Wordnet等辞典中的相关词语。例如，使用Wordnet关系，如当考虑问题“who manufactures magic chef appliances.”的段落时，“fabricates”、“constructs”和“makes”等词语可以和“manufacture”相关。基于特征的线性检索模型提供了合适的框架来组合与答案段落相关的特征，并学习有效的权值。

最后一步，从文本段落中选择答案可以潜在地包含比排序文本段落更偏语言学的分析和推理。但是，绝大多数情况下，自动问答系统的用户都希望看到答案的上下文，甚至是多个答案，以便于验证答案可能是正确的，或者可能决定哪个是最佳答案。例如，对于帝王谷(the Valley of the Kings)的问题，系统可能返回“Egypt”作为答案。但是一般来说，更有用的是返回段落“The Valley of the Kings is located on the West Bank of the Nile near Luxor in Egypt”。从这个角度来说，可以认为搜索引擎需要对不同类型的查询提供一系列反馈，从专注的文本段落到整个文档。更长更精确的问题应该产生更准确集中的反馈。这在面向事实的问题实例中，一般来说是正确的，例如表11-2中的那些问题。

表11-2 TREC自动问答问题样例及其对应的问题类型

问题样例	问题类型
What do you call a group of geese?	Animal
Who was Monet?	Biography
How many types of lemurs are there?	Cardinal
What is the effect of acid rain?	Cause/Effect
What is the street address of the White House?	Contact Info
Boxing Day is celebrated on what day?	Date
What is sake?	Definition
What is another name for nearsightedness?	Disease
What was the famous battle in 1836 between Texas and Mexico?	Event
What is the tallest building in Japan?	Facility
What type of bridge is the Golden Gate Bridge?	Facility Description
What is the most popular sport in Japan?	Game
What is the capital of Sri Lanka?	Geo-Political Entity
Name a Gaelic language.	Language
What is the world's highest peak?	Location
How much money does the Sultan of Brunei have?	Money
Jackson Pollock is of what nationality?	Nationality
Who manufactures Magic Chef appliances?	Organization
What kind of sports team is the Buffalo Sabres?	Org. Description
What color is yak milk?	Other
How much of an apple is water?	Percent
Who was the first Russian astronaut to walk in space?	Person
What is Australia's national flower?	Plant
What is the most heavily caffeinated soft drink?	Product
What does the Peugeot company manufacture?	Product Description
How far away is the moon?	Quantity
Why can't ostriches fly?	Reason
What metal has the highest melting point?	Substance
What time of day did Emperor Hirohito die?	Time
What does your spleen do?	Use
What is the best-selling book of all time?	Work of Art

自动问答系统中用到的技术展示了句法和语义特征是如何用来帮助一些查询获得更加准确的答案的。但是这些技术不能解决信息检索中更加困难的挑战。例如一个TREC查询为“Where have dams been removed and what has been the environmental impact?”看起来像是基于事实的问题，但是答案需要比一个地点列表或者句子排序更具有广泛性。另一方面，使用自动问答技术识别不同的“dam removal”文本表达式，应该有益于对答案段落或者文档进行排序。类似地，TREC查询“What is being done to increase mass transit use?”显然不是基于事实的问题，也可能从识别关于“use of mass transit”的讨论的技术中受益。但是，这些潜在的益处尚未在检索实验中被验证。这表明，将这些技术应用到大量查询中，还有重要的技术问题。搜索引擎目前依赖于用户学习，基于他们的经验来提交例如“mass transit”的查询，而不是更精确的问题。

11.6 词语、图片和音乐

虽然信息检索通常专注于文本，但是人们查找的许多信息是图片、视频或者音频等形式，尤其是在互联网上查找时。网络搜索引擎和大量其他网站，提供了关于图片和视频的搜索，在线音乐商店是查找音乐的非常流行的方式。所有这些服务都是基于文本的，依赖于标题、说明、用户提交的“标签”和其他相关文本，以便为搜索创建非文本媒体的表示。这种方法很有效，也是相对直接的实现方式。但是，很多情况下，没有任何相关的文本，或者文本不能反映被表示对象的重要方面。例如，许多视频共享网站上的视频商店，没有好的文本描述，并且由于视频太多，用户的标签也不能解决这个问题。另外一个例子是，在搜索特定曲调时，标题不能提供待检索音乐文件的恰当描述。基于这些原因，研究人员一直在对非文本媒体开发基于内容的检索技术。

一些非文本媒体使用词语来传达信息，可以转换为文字。例如，光学字符识别（Optical character recognition, OCR）技术就用来将包含手写或打印文本的扫描文档转换为机器可读的文本。语音识别技术[⊖]用来将录制好的语音（或者发声文档）转换为文本。OCR和语音识别都会产生“噪声”文本，也就是说，相对于原始的打印文本或者语音脚本，输出文本可能相对会有错误。图11-6展示了两个OCR产生的错误样例。两个例子中，OCR的输出都来自于标准的OCR软件。第一个例子基于文本段落，由文字处理软件创建、打印，然后被复印了很多次（为了降低质量），并最终被OCR扫描。输出结果包含少量的错误，但是一般来说，对高质量的打印文本，OCR的错误率是很低的[⊜]。第二个例子使用了较低质量的输入，通过扫描一份很早的会议论文的影印版来创建。在这种情况下，OCR输出包含明显的错误，带有例如“sponsorship”和“effectiveness”等根本不可读的词语。注意，OCR错误出现在字符层面。换句话说，错误是单独字符混淆的结果，并导致输出不正确的字符。

图11-7展示了一段新闻广播通过高质量语音识别软件中的输出结果。绝大多数词语都被正确识别。但是，当系统遇到以前没有见过的词语时（叫做例外词汇，或者OOV），就会产生一些明显的错误。许多OOV词语都是人名或者机构名，例如这个例子中的“Pinochet”。已经有大量的研究致力于此问题。注意，语音识别错误倾向于在输出中创建新的词语，例如例子中的“coastal fish”，这是因为，软件试图寻找已知的词语来最佳地满足声音模板和语言模型。

⊖ 有时叫做自动语音识别（Automatic Speech Recognition, ASR）。

⊜ “质量”是指图片质量，根据对比度、锐利度、背景纯净度来考察。

这种类型的错误和OCR中字符级别的错误都会潜在地降低搜索的效果。

Original:
The fishing supplier had many items in stock, including a large variety of tropical fish and aquariums of all sizes.
OCR:
The fishing supplier had many items in stock, including a large variety of tropical fish and aquariums ot aH sizes~

Original:
*** This work was carried out under the sponsorship of National Science Foundation Grants NSF-GN-SBO (Studies in Indexing Depth and Retrieval Effectiveness) and NSF-GN-482 (Requirements Study for Future Catalogs).**
OCR:
This work was carried out under the sp011J10rship 01 NatiolUl1 Setenee Foundation 0rant. NSF·0N·SB0 (Studl .. In Indexing Depth and Retrieval Eflccth"ene&&) and NSF·0N·482 (Requirements Study lor Future 'Catalogs)•

图11-6 OCR错误的例子

Transcript:
French prosecutors are investigating former Chilean strongman Augusto Pinochet. The French justice minister may seek his extradition from Britain. Three French families whose relatives disappeared in Chile have filed a Complaint charging Pinochet with crimes against humanity. The national court in Spain has ruled crimes committed by the Pinochet regime fall under Spanish jurisdiction.

Speech recognizer output:
french prosecutors are investigating former chilean strongman of coastal fish today the french justice minister may seek his extradition from britain three french families whose relatives disappeared until i have filed a complaint charging tenants say with crimes against humanity the national court in spain has ruled crimes committed by the tennessee with james all under spanish jurisdiction

图11-7 语音识别器错误的例子

TREC和其他研讨会已经进行了许多基于OCR和ASR数据的评测。这些研究显示，检索的效果一般不会受到OCR和ASR错误的显著影响。主要原因是数据中存在的大量冗余性被用于评价。例如，在ASR评测中，经常出现许多相关的口语文档，每个文档都包含对于查询来讲是重要的词语的许多实例。即使一些词语实例没有被识别，其他实例也可以识别。另外，即使一个查询词一直都未被识别，往往也会有查询中的其他词语能被用来估计相关性。类似的情况发生在OCR数据上。一些词语的实例在扫描文档中也许具有比其他图像更高的质量，因此会被成功识别。唯一使OCR和ASR错误都显著降低效果的是短文档，它们具有很小的冗余性，具有较高的错误率。字符n-gram索引和扩展相关词项等技术，能够改进这些情况下的性能。

相对于能够转换为噪音文本的媒体，基于内容的图片检索[⊖]是一项更具有挑战性的难题。能够在图像中抽取到的特征，例如颜色、纹理、形状等，相对于词语都只有很少的语义内容。例如，在图像检索应用中被用到的一个普通特征就是色彩直方图。图像中的颜色使用特殊的

⊖ 也叫做基于内容的图像检索（CBIR）。

颜色模型来表示，例如RGB[⊖]模型。RGB模型将颜色表示为红、蓝、绿的混合。一般对每种颜色都有256种值（8位）。图像的色彩直方图可以通过一轮色彩值量化来降低直方图中可能的bin的数量。例如，如果RGB数值量化到8个级别而不是256个级别，可能的颜色组合数量就从256×256×256降低到8×8×8=512。那么，对图像中的每个像素，颜色数值对应的bin的数量从1开始增加。生成的直方图可以用来表示数据集中的图像，并且可以用于索引，实现快速检索。给定一个新的图像作为查询，该图像的颜色直方图可以用来和图像数据集中的直方图采用一些相似度度量的方法进行比较。图像可以根据这些相似度值进行排序。

图11-8展示了两幅样例图像及其色彩直方图。这种情况下，直方图是基于色调值的（而不是RGB值），因此直方图中的顶峰对应于颜色的峰值。鱼和花都是黄色占主导，因此两个直方图中在频谱的那个区域（在左侧）具有类似的峰值。其他类似的峰值是关于绿色和蓝色的。

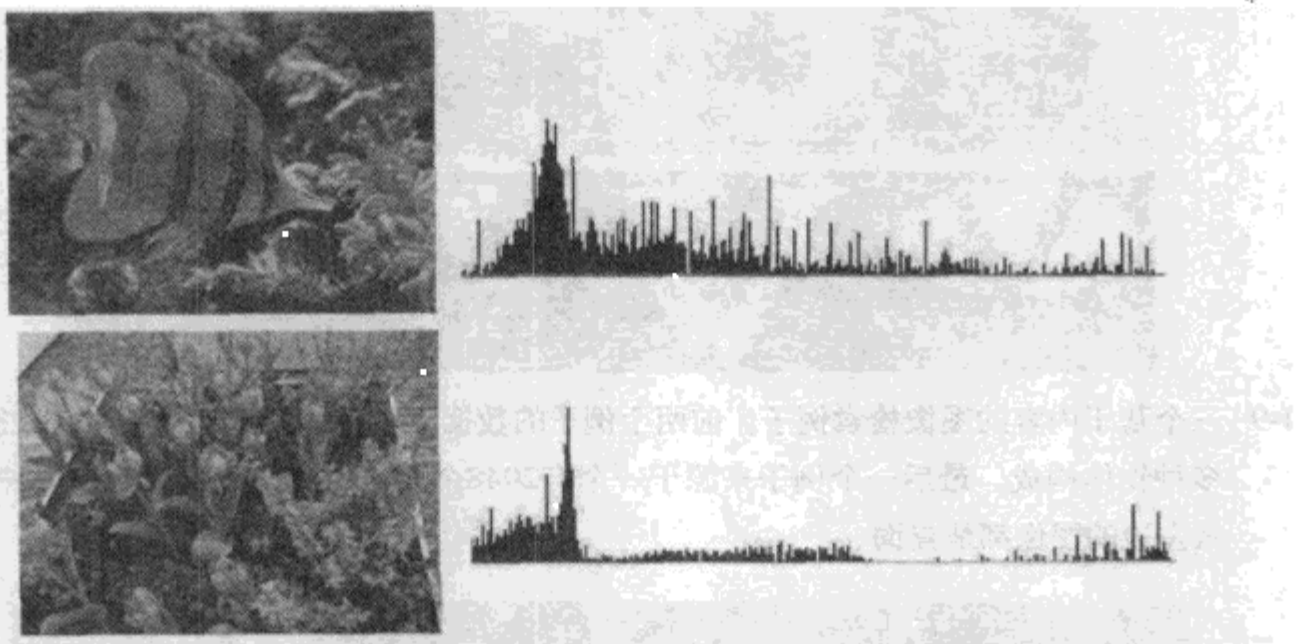


图11-8 两幅图像（一条鱼和一朵红花）的色彩直方图。水平轴是色调值

色彩特征对于寻找色彩相似度的图片很有用，例如日落的图片。但是两幅内容完全不相同的图片，可能会只因为它们的颜色而被认为非常相关。例如，图11-8中的花的图片可能在与鱼的图片比较时排序非常靠前，就是因为直方图中相似的峰值。寻找语义相关图片的能力，可以在结合色彩特征和纹理、形状特征后得到改进。图11-9展示了基于纹理特征的（汽车和火车）图像检索的例子，以及基于形状特征检索的例子（商标）。纹理被广泛地定义为空间的图像上的灰阶层次安排。形状特征描述物体边界和边缘的形式。虽然第二个例子明显说明基于纹理的相似度不能保证图像语义相似度，但是样例显示带有相似外貌的图像能够通过使用这些类型的表示来找到。在基于文本的搜索中，排序靠前的文档虽然不相关，但是可以被用户很容易的理解。在基于内容的图像检索中，例如图11-9中的猿猴的检索错误，很难向正在寻找火车图像的用户解释。

检索实验已经证明，最有效的融合图像特征的方式就是概率检索模型。正如前面所讨论的，如果图像有文本说明或者用户标签，这些特征可以容易地加入到排序中。视频检索除了具有更多的特征外，其他方面与图像检索类似，例如相关的字幕文本或者从语音识别生成的文本。视频中的图像部分一般表示为一系列关键帧图像。为了生成关键帧，视频首先被分类

⊖ 另外一个普通模型就是HSV（色调、饱和度和数值）。

为镜头或场景。一个视频镜头可以定义为一个具有视觉连贯性的连续帧序列。边界可以通过视觉不连续点来探测，例如一个形状连续的两帧相似度降低。给定分割好的镜头，一个单独帧（图片）被选为关键帧。这可以通过简单地选择每个镜头中的第一帧，或者基于视觉相似度和动作分析的更复杂的技术来选择。图11-10展示了从一段新闻会议视频中抽取到的四个关键帧样例。

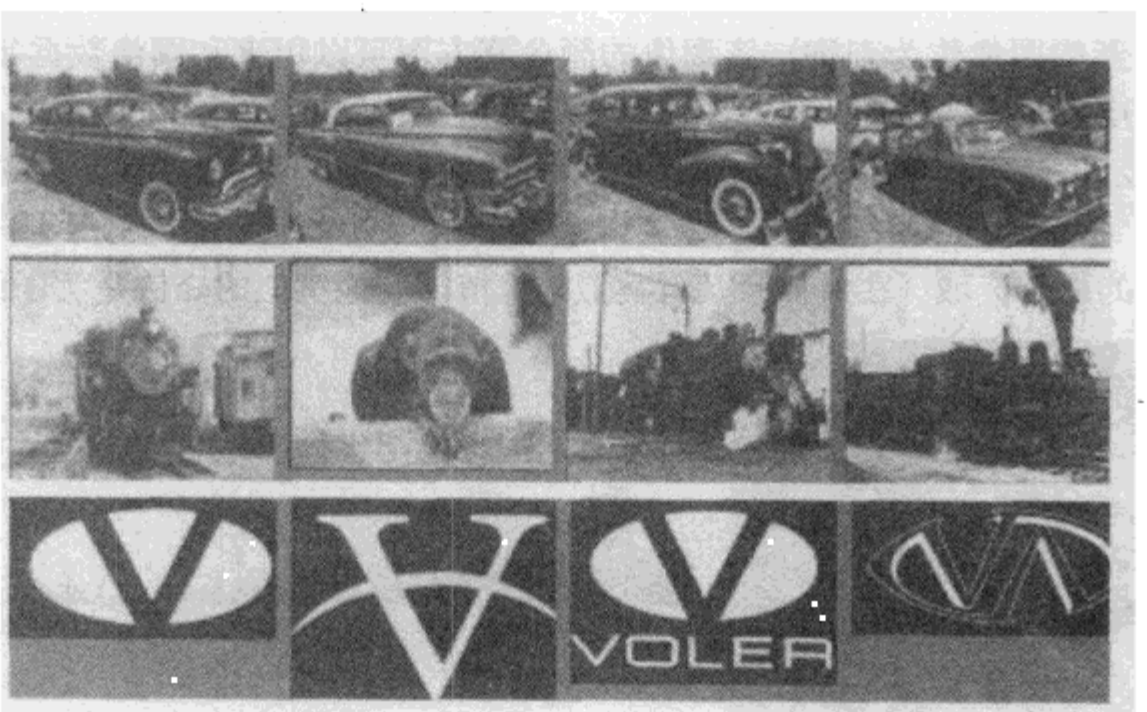


图11-9 三个基于内容的图像检索例子。前两个例子的数据集由1560张汽车、人脸、猿猴和其他多种物体构成。最后一个例子来源于一个有2048个商标图案的数据集。在每种情况中，最左边的图像都是查询

图11-10 从TREC视频剪辑中抽取到的四个关键帧

已经讨论的图像检索技术都假设查询是一幅图像。在许多应用中，用户更愿意采用文本查询来描述他们查询的图片。词语不能和从图像中衍生出来的特征直接进行比较。但是，最近的研究表明，给定足够的训练数据，概率检索模型能够学习得到词语或者类别与基于图像的特征的联系，并且事实上实现了自动的图像标注。使用这个模型时，有两个实际问题需要解决：

- 给定一幅没有文本标注的图像，如何能够自动给那幅图像赋予有含义的关键词？
- 给定文本查询$\{q_1, \cdots q_n\}$，如何检索和这个查询相关的图像？

7.3.2节描述的相关性模型被证明能对这个任务有效。取代通过观察词语w对查询词估计联合概率$P(w, q_1, \cdots q_n)$的步骤，相关性模型被用来估计$P(w, i_1, \cdots i_m)$，其中一幅图像假设表示为一组图像词项$\{i_1, \cdots i_m\}$。这种方法中，图像词项的“词表”通过使用图像分割和聚合类似的区域来构建[⊖]。作为例子，在一个包含5000幅图像的测试语料库中，一个包含500个图像词项的词表被用来表示这些图像，每个图像被1～10个词项描述。使用已经用文本标注好的训练图像

⊖ 有这样一种技术，表示图像为“滴”（blob）(Carson et al., 1999)。

集来估计联合概率，这个模型能用来回答上面提到的两个问题。为了对文本查询返回图像，下面是一个类似于伪相关反馈的过程：

1）使用文本查询来对包含文本标注的图像进行排序。

2）给定排序靠前的图像来估计图像词表联合概率。

3）使用带有图像词项的查询扩展来对那些没有文本标注的图像进行重排序。

还有一种办法是，从训练集上估计的联合概率，可以用来对那些没有标注的图像分配关键词。图11-11展示了采用这种方法标注图像的例子。用于描述图像的绝大多数词语都是合理的，但是对熊的图片的标注说明，明显的错误也可能会产生。使用自动标注技术的检索实验证明，这种方法是有前途的，能够在一些应用中增加图像和视频的检索性能。主要的问题是，什么样的文本和图像词表的类型和大小最有效。

图11-11 自动文本标注图像的例子

相比于图像，音乐是一种有较少关联词语的媒体。图像至少能够用文本描述，自动标注技术能用来通过文本查询进行检索。但是，除了标题、作曲家、演奏家、歌词，很难使用词语来描述一段音乐。音乐具有很多用于不同目的的表示。图11-12展示了巴赫作曲的“十号赋格”的三种表示。第一种是音乐性能的音频信号。这就是被压缩存储为MP3格式的内容。第二种是MIDI[⊖]表示，提供了音乐中“事件”的数字规范，例如音乐笔记中的音高、强度、持续时长、速率。第三种就是传统的乐谱，包含最详细的信息，特别是和弦音乐组成的多个部分或声音。

很多方法已经为搜索这些基本表示建立了索引词项。在音频的情况中，最成功的就是通过散列音乐的表示来作为“签名”的技术。例如，签名可以基于音频在一个时间片上的声谱图上的峰值来创建[⊜]。这种类型的索引，是基于使用手机捕获的简短录音来识别音乐服务的基础（例如Wang, 2006）。

另外一种流行的基于内容的音乐检索方法，就是哼唱式查询（query-by-bumming）。这种类型的系统中，用户逐字地唱、哼或者演奏一首乐曲，在一个音乐数据集上就会搜索类似的曲子。这种类型的搜索基于具有单独旋律（单声道）的音乐。查询被转换为乐曲的一种表示，由序列音符、相对音高和音符间的停顿构成。数据集中的音乐必须也被转换为同样类型的表示，最容易的方式就是使用MIDI作为起始点。查询将会是相关乐曲有噪音的表示，所以发展出了许多用于文本检索的检索模型来满足这个任务，例如n-gram匹配和语言模型（Dannenberg et al., 2007）。基于概率模型的复调（polyphonics）音乐搜索技术，同样用来检索传统乐谱的音乐。

总结一下，基于文本检索模型的搜索技术已经被用来处理多种多样的非文本媒体。在扫

⊖ 音乐乐器数字接口（Musical Instrument Digital Interface, MIDI）。

⊜ 声谱图表示音频信号在给定时间下每个频率上的能量和振幅。

描文档和音频文档的情况中，检索性能类似于文本文档，因为OCR和语音识别工具一般具有较低的错误率。基于内容的图像和视频检索是很有前途的，但是这些媒体的搜索应用必须依赖于描述和用户标签等来源的关联文本，以达到好的性能。音乐难以用文字来描述，但是有效的音乐搜索应用已经开发出来，因为和用户查找目标具有很强关系的索引项，可以从音频或者MIDI表示中衍生出来。

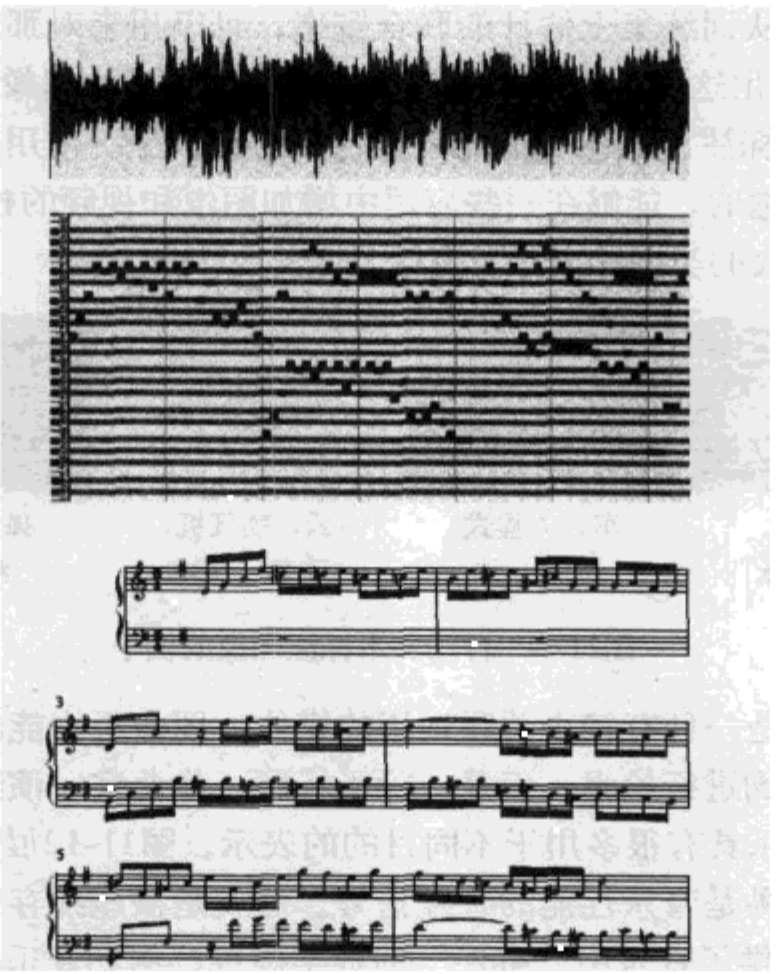


图11-12 巴赫的“十号赋格”的三种表示：音频、MIDI和传统的乐谱

11.7 搜索能否适用于所有情况

那么，搜索的未来是什么？很显然，看起来似乎不能在短期内建造无所不知的辅助系统。换句话说，正如前面讨论的那样，可以将那种类型的系统作为目标，使得我们可以清楚地认识到目前的搜索引擎还需要走多远。对搜索和性能的认识和理解在稳步提高，尽管这个领域已有长达40年的历史。有趣的是，这导致各种各样的搜索服务，而不是一个搜索引擎具有越来越多的能力。在一个流行的网络搜索引擎主页上，可以找到不同搜索引擎的连接，包括网页、图像、博客、地图、学术论文、产品、专利、新闻、书籍、金融信息、视频、政府文档和照片等。另外，这里有不同工具的链接，包括桌面搜索、企业搜索和广告搜索。没有简单地采用同一个搜索引擎的多个版本，显然说明它们使用了一些不同的特征、排序算法和界面。如果调查一下相关文献，关于搜索的一个更大范围的应用、媒体和方法正在被发展和评价着。一个稳妥的预测是，这种新想法的扩展将继续下去。

虽然专门的搜索引擎的数量在激增，但是它们之间一致的共识却在不断增加。信息检索和相关领域的研究人员，例如机器学习和自然语言处理的研究人员，已经开发了与表示文本和检索过程模型类似的方法。这些方法被迅速地扩展以来含结构化数据和非文本数据。基于新应用或者数据的方法共同增强了这种观点。文本的概率模型和基于特征的线性模型为理解

搜索提供了一个强大而有效的框架。另外一种预测就是，搜索的这种隐含“理论”的复杂性，会一直增加，并在一定范围的应用中稳步地改进效率。

这种“理论”会怎样影响搜索引擎呢？如果文本表示和其他类型的数据方面能够更加一致，针对问题在答案排序时也更加一致，那么开发人员对搜索工具将会更加熟悉。今天能够使用的开源搜索引擎、网络搜索引擎、桌面搜索引擎和企业搜索引擎，都使用了不同的词项权重、不同的特征、不同的排序算法和不同的查询语言。这是因为在做这些事情时，还没有关于正确方式的一致性，但这种情况会改变。当隐含模型具有更好的一致性时，仍然会有多种可用的搜索工具，但是对它们之间的选择将会更加注重基于实现的效率、对新应用的灵活性和适应性，以及扩展到由模型建议的更复杂算法的实现工具。这一点类似于数据库系统发生的情况，在20世纪80年代，有多家厂商和机构提供了基于关系模型的工具。

搜索的另外一方面，也是在本书中讨论不充分的内容，就是用户和搜索引擎之间交互的重要性，以及在这个过程中用户任务（task）的影响。尤其是情报科学家已经专注于这些方面，并对理解人如何查找相关信息进行了重要的观察。随着现在社会化搜索和社会化网络应用的不断增长，交互的研究已经拓展到包括用户之间以及用户和搜索引擎之间。在不久的将来，可以期望见到比现在的模型更加复杂的融合用户和交互的搜索理论和模型。Fuhr（2008）就是这类研究的一个近期例子。

不计其数的电影和电视都对未来有一个普遍的幻想，即简单的自然语言输入并输出多媒体信息。这种接口中的交互是基于对话的。由于它的简单性，这看起来像是一个合理的长期目标。在当前的搜索界面中，查询当然简单，但是正如曾讨论的那样，还需要相当大的努力来将这些简单的查询转换为那些能够返回相关信息的查询。另外，那些很小的对话或者交互，和诸如相关反馈等更加简单的想法，都没有使用。更糟糕的是，由于有许多不同搜索引擎可用，导致另外一个决策（到底用哪个搜索引擎？）被增加到交互上。

搜索引擎公司的开发人员及研究人员正在研究改进交互的技术。一些界面将多个类型搜索引擎的结果放到一个单独的结果中显示。简单的例子包括，当查询中包括一个地址时，将一个地图搜索引擎的结果放置到结果列表的顶端，或者当查询能够近似匹配一篇学术论文标题时，在结果顶端放置一个论文的链接。但是，很明显，将来的界面必须不断地将用户和他们的知识融入到搜索过程中。

最后的一个预测就是自我服务。搜索在它所有的形式中，将会不断地在未来软件应用中显示其非凡的重要性。训练人们理解搜索引擎中的原则、模型和评价技术，是不断提高他们效果与效率的重要部分。现在还没有足够的课程专注于此主题，但是通过本书或者其他这样的书，更多的人会更多地理解搜索。

参考文献和深入阅读

基于特征的线性检索模型在Metzler和Croft（2007b）中有所讨论。这篇论文包含第7章中讨论过的其他模型的参考文献。另外一篇最近讨论线性模型的论文是Gao等（2005）。

许多词项依赖模型都被信息检索文献提出，尽管很少获得有趣的结果。van Rijsbergen（1979）提出了引用最多之一的依赖模型，是贝叶斯分类方法在检索上的扩展。在另外一篇早期的论文中，Croft等（1991）证明短语和词项邻近性可以通过在推理互联网模型中假设它们具有依赖性，以获得潜在的性能改进。Gao等（2004）描述了一个依赖模型，证明了显著的性

能改进，尤其是序列依赖（或者n-gram）。最近在大规模测试集上的其他研究表明，词项邻近性信息是一种极其有用的特征。

在信息检索的角度，处理数据中的结构开始于上世纪70年代的限定在布尔域上的商业搜索服务，例如MEDLINE和DIALOG。在20世纪70年代和80年代，大量论文描述了使用关系数据库的搜索引擎实现（例如Crawford, 1981）。尽管对象管理系统被成功地应用于支持索引（例如Brown et al., 1994），这种方法的效率问题一直持续到20世纪90年代。对于搜索引擎发展出数据库模型的概率扩展的重要工作，也是在20世纪90年代完成的。Fuhr和他的同事描述了一种概率关系代数（Fuhr and Rölleke, 1997）和一种概率数据日志系统（Fuhr, 2000）。

在商业世界中，文本检索已经在20世纪90年代早期成为诸如Oracle等数据库系统中的标准操作。但是后来对互联网数据的挖掘以及不断增加的基于文本的互联网应用，使得那个十年高效处理文本的能力成为绝大多数信息系统中的重要部分。关于数据库和搜索引擎在数据库角度的整合的有趣讨论，可以在Chaudhuri等（2005）中找到。

另外一条重要的研究路线就是，使用结构化数据和XML文档进行检索。这个领域早期的工作就是处理办公文档（Croft, Krovetz, & Turtle, 1990）和文档标记（Croft et al., 1992）。对XML数据的XQuery查询语言在Chamberlin（2002）中有介绍。Kazai等（2003）描述了对XML文档的检索方法进行评价的INEX项目。Trotman和Lalmas （2006）介绍了NEXI语言。

在自动问答领域，Metzler和Croft（2005a）给出了问题分类技术的综述。自动问答的概率方法被证明很有效，包括最大熵模型（Ittycheriah et al., 2001）和翻译模型（Echihabi and Marcu, 2003）。二者都非常类似于第7章中介绍的检索模型。

Taghva等（1996）描述了第一组全方面的实验，并证明OCR错误一般来说对检索效果只有很小的影响。Harding等人（1997）证明了在显著的OCR错误率下，n-gram是如何进行有效补偿的。

Coden等（2002）包含了关于语音文档检索的一些论文。Singhal和Pereira（1999）介绍了一种能够显著改进性能的语音文档扩展技术。TREC语音文档的数据和主要结果可以参见Voorhees和Harman（2005）的描述。

关于基于内容的图像检索，已经发表了很多论文。Flickner等（1995）介绍了QBIC。这是最早的商业系统之一，整合了对色彩、纹理和形状特征的检索。Photobook系统（Pentland et al., 1996）也对其他基于内容的检索项目产生了重要影响。Ravela和Manmatha （1997）描述了最早之一的基于纹理的检索技术，其中采用了一种信息检索方法来进行评价。SIFT（尺度不变性特征转换）算法（Lowe, 2004）是目前针对搜索来表示图像的流行方法。Vasconcelos（2007）给出了最近基于内容的信息检索领域的概述。

在基于内容的音乐检索领域，绝大多数的研究都发表在音乐信息检索国际会议（ISMIR）[⊖]上。Byrd和Crawford （2002）对这个领域的研究话题给出了很好的介绍。Midomi[⊜]是基于哼唱的音乐检索的例子。

关于搜索和交互的信息科学方面，Belkin撰写了很多重要论文，例如Koenemann和Belkin（1996）、Belkin（2008）。Ingwersen和Järvelin （2005）撰写的书包括了颇具代表性的搜索中交互和上下文的深入讨论。Marchionini（2006）在搜索界面的重要性方面讨论了类似的话题。

⊖ http://www.ismir.net/。

⊜ http://www.midomi.com。

练习

11.1 你能在互联网、书籍、电影或者电视中，找到其他关于搜索引擎的“未来版本”么？介绍一下这些系统和它们可能拥有的任何一个独特的特点。

11.2 你喜爱的网络搜索引擎使用的是词袋表示么？你如何能够确定究竟它是不是这样的？

11.3 使用Galago的#feature操作来创建一个排序算法，要用到BM25和查询似然特征。

11.4 说明基于特征的线性排序函数是如何和第5章的抽象排序模型相关的。

11.5 在SIGIR 2000年的会议中，有多少篇是处理词项依赖性的？列出它们的引用。

11.6 编写一个程序将文本查询转换为Galago中的序列依赖MRF查询，就像文章中介绍的那样。在一个索引上运行一些查询，并比较使用和不使用词项依赖结果的质量。哪种查询是使用依赖模型提高最多的？哪种又是降低最多的？

11.7 想出当你搜索XML结构化文档时的五种查询。这些查询必须包括结构和内容特征。采用英语和NEXI语言来书写查询（如果XML结构不明显的话，解释一下）。你认为查询中的结构化部分能够改进性能么？给出详细的例子。

11.8 找出一个数据库系统（可以是商业的或者开源的）的文本搜索功能。尽量详细地描述，包括查询语言。和搜索引擎对比一下这种功能。

11.9 找出一个运行在互联网上的自动问答演示系统。使用一组问题测试集，区分一下对于这个系统，哪些类型的问题容易解答，哪些问题不行。使用MRR或者其他评价指标来报告性能。

11.10 使用一个网络搜索引擎中的基于文字的图像检索，找出你认为可以基于色彩找到的图片的例子。使用诸如Photoshop的工具来将测试图片生成基于色调的色彩直方图（或者RGB中的任意一种颜色都行）。使用一些相似度度量来对比这些直方图（例如坐标点上对应值平方差的和）。这些相似度在多大程度上符合你的视觉感觉？

11.11 查看已经被用户标注的图像或者视频的样例，将这些标签分为三组：那些你认为最终可以被图像处理和物体识别自动完成，那些你认为不可能被图像处理完成，或者是垃圾。然后对这些图像的相关查询确定，哪些标签应该是最有用的。总结你的发现。

11.12 对于个人数码图像和视频，哪些特征你愿意用来支持索引和检索？这些特征中，哪些可以从目前标准的软件中获得？这些特征中，哪些在研究论文中被讨论过？

11.13 从同一首歌的两个版本（例如不同的艺术家）对应的MP3文件开始，使用网上能用到的工具来分析和对比它们。你应该能够找到工具来从音频中生成MIDI和声谱图。你能从这些文件中找到任何来自于曲调的相似度吗？你还可以尝试使用麦克风录制这首歌，然后将录音创建的音频文件和原始文件进行对比。

参考文献

AbdulJaleel, N., & Larkey, L. S. (2003). Statistical transliteration for English-Arabic cross language information retrieval. In *CIKM '03: Proceedings of the twelfth international conference on information and knowledge management* (pp. 139–146). ACM.

Agichtein, E., Brill, E., & Dumais, S. (2006). Improving web search ranking by incorporating user behavior information. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 19–26). ACM.

Agichtein, E., Brill, E., Dumais, S., & Ragno, R. (2006). Learning user interaction models for predicting web search result preferences. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 3–10). ACM.

Agichtein, E., Castillo, C., Donato, D., Gionis, A., & Mishne, G. (2008). Finding high-quality content in social media. In *WSDM '08: Proceedings of the international conference on web search and web data mining* (pp. 183–194). ACM.

Allan, J. (1996). Incremental relevance feedback for information filtering. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 270–278). ACM.

Allan, J. (Ed.). (2002). *Topic detection and tracking: Event-based information organization*. Norwell, MA: Kluwer Academic Publishers.

Almeida, R. B., & Almeida, V. A. F. (2004). A community-aware search engine. In *WWW '04: Proceedings of the 13th international conference on World Wide Web* (pp. 413–421). ACM.

Amershi, S., & Morris, M. R. (2008). CoSearch: A system for co-located collaborative web search. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on human factors in computing systems* (pp. 1,647–1,656). ACM.

Anagnostopoulos, A., Broder, A., & Carmel, D. (2005). Sampling search-engine results. In *WWW '05: Proceedings of the 14th international conference on World Wide Web* (pp. 245–256). ACM.

Anh, V. N., & Moffat, A. (2005). Simplified similarity scoring using term ranks. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 226–233). ACM.

Anh, V. N., & Moffat, A. (2006). Pruned query evaluation using pre-computed impacts. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 372–379). New York: ACM.

Baeza-Yates, R., & Ramakrishnan, R. (2008). Data challenges at Yahoo! In *EDBT '08: Proceedings of the 11th international conference on extending database technology* (pp. 652–655). ACM.

Baeza-Yates, R., & Ribeiro-Neto, B. A. (1999). *Modern information retrieval.* New York: ACM/Addison-Wesley.

Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., & Stoica, I. (2003). Looking up data in P2P systems. *Communications of the ACM*, *46*(2), 43–48.

Balog, K., Azzopardi, L., & de Rijke, M. (2006). Formal models for expert finding in enterprise corpora. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 43–50). ACM.

Barroso, L. A., Dean, J., & Hölzle, U. (2003). Web search for a planet: The Google cluster architecture. *IEEE Micro*, *23*(2), 22–28.

Beeferman, D., & Berger, A. (2000). Agglomerative clustering of a search engine query log. In *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 407–416). ACM.

Belew, R. K. (2000). *Finding out about.* Cambridge, UK: Cambridge University Press.

Belkin, N. J. (2008). (Somewhat) grand challenges for information retrieval. *SIGIR Forum*, *42*(1), 47–54.

Belkin, N. J., & Croft, W. B. (1992). Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, *35*(12), 29–38.

Belkin, N. J., Oddy, R. N., & Brooks, H. M. (1997). ASK for information retrieval: Part I.: background and theory. In *Readings in information retrieval* (pp. 299–304). San Francisco: Morgan Kaufmann. (Reprinted from *Journal of Documentation*, 1982, *38*, 61–71)

Benczúr, A., Csalogány, K., Sarlós, T., & Uher, M. (2005). Spamrank – fully automatic link spam detection. In *AIRWeb: 1st international workshop on adversarial information retrieval on the web* (pp. 25–38).

Berger, A., Caruana, R., Cohn, D., Freitag, D., & Mittal, V. (2000). Bridging the lexical chasm: Statistical approaches to answer-finding. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval* (pp. 192–199). ACM.

Berger, A., & Lafferty, J. (1999). Information retrieval as statistical translation. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval* (pp. 222–229). ACM.

Berger, A., & Mittal, V. O. (2000). Ocelot: a system for summarizing web pages. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval* (pp. 144–151). ACM.

Bergman, M. K. (2001). The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, *7*(1).

Bernstein, Y., & Zobel, J. (2005). Redundant documents and search effective-

ness. In *CIKM '05: Proceedings of the 14th ACM international conference on information and knowledge management* (pp. 736–743). ACM.

Bernstein, Y., & Zobel, J. (2006). Accurate discovery of co-derivative documents via duplicate text detection. *Information Systems, 31*, 595–609.

Bikel, D. M., Miller, S., Schwartz, R., & Weischedel, R. (1997). Nymble: A high-performance learning name-finder. In *Proceedings of the fifth conference on applied natural language processing* (pp. 194–201). Morgan Kaufmann.

Bikel, D. M., Schwartz, R. L., & Weischedel, R. M. (1999). An algorithm that learns what's in a name. *Machine Learning, 34*(1–3), 211–231.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research, 3*, 993–1,022.

Borgs, C., Chayes, J., Mahdian, M., & Saberi, A. (2004). Exploring the community structure of newsgroups. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 783–787). ACM.

Breese, J., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *UAI '98: Proceedings of the uncertainty in artifical intelligence conference* (pp. 43–52).

Brill, E. (1994). Some advances in transformation-based part of speech tagging. In *AAAI '94: National conference on artificial intelligence* (pp. 722–727).

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems, 30*(1–7), 107–117.

Broder, A. (2002). A taxonomy of web search. *SIGIR Forum, 36*(2), 3–10.

Broder, A., Fontoura, M., Josifovski, V., & Riedel, L. (2007). A semantic approach to contextual advertising. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 559–566). ACM.

Broder, A., Fontura, M., Josifovski, V., Kumar, R., Motwani, R., Nabar, S., et al. (2006). Estimating corpus size via queries. In *CIKM '06: Proceedings of the 15th ACM international conference on information and knowledge management* (pp. 594–603). ACM.

Broder, A., Glassman, S. C., Manasse, M. S., & Zweig, G. (1997). Syntactic clustering of the Web. *Computer Networks and ISDN Systems, 29*(8–13), 1157–1166.

Brown, E. W., Callan, J., Croft, W. B., & Moss, J. E. B. (1994). Supporting full-text information retrieval with a persistent object store. In *EDBT '94: 4th international conference on extending database technology* (Vol. 779, pp. 365–378). Springer.

Buckley, C., & Voorhees, E. M. (2004). Retrieval evaluation with incomplete information. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 25–32). ACM.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., et al. (2005). Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on machine learning* (pp. 89–96). ACM.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*(2), 121–167.

Burke, R. D., Hammond, K. J., Kulyukin, V. A., Lytinen, S. L., Tomuro, N., & Schoenberg, S. (1997). *Question answering from frequently asked question files: Experiences with the FAQ finder system* (Tech. Rep.). Chicago, IL, USA.

Büttcher, S., & Clarke, C. L. A. (2007). Index compression is good, especially for random access. In *CIKM '07: Proceedings of the sixteenth ACM conference on information and knowledge management* (pp. 761–770). ACM.

Büttcher, S., Clarke, C. L. A., & Lushman, B. (2006). Hybrid index maintenance for growing text collections. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 356–363). ACM.

Byrd, D., & Crawford, T. (2002). Problems of music information retrieval in the real world. *Information Processing and Management*, *38*(2), 249–272.

Callan, J. (2000). Distributed information retrieval. In *Advances in information retrieval: Recent research from the CIIR* (pp. 127–150). Norwell, MA: Kluwer Academic Publishers.

Callan, J., Croft, W. B., & Broglio, J. (1995). TREC and Tipster experiments with Inquery. *Information Processing and Management*, *31*(3), 327–343.

Callan, J., Croft, W. B., & Harding, S. M. (1992). The Inquery retrieval system. In *Proceedings of DEXA-92, 3rd international conference on database and expert systems applications* (pp. 78–83).

Cao, Y., Xu, J., Liu, T.-Y., Li, H., Huang, Y., & Hon, H.-W. (2006). Adapting ranking SVM to document retrieval. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 186–193). ACM.

Carbonell, J., & Goldstein, J. (1998). The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 335–336). ACM.

Carson, C., Thomas, M., Belongie, S., Hellerstein, J. M., & Malik, J. (1999). Blobworld: A system for region-based image indexing and retrieval. In *VISUAL '99: Third international conference on visual information and information systems* (pp. 509–516). Springer.

Carterette, B., Allan, J., & Sitaraman, R. (2006). Minimal test collections for retrieval evaluation. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 268–275). ACM.

Carterette, B., & Jones, R. (2007). Evaluating search engines by modeling the relationship between relevance and clicks. In *NIPS '07: Proceedings of the conference on neural information processing systems* (pp. 217–224). MIT Press.

Chakrabarti, S., van den Berg, M., & Dom, B. (1999). Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks*, *31*(11-16), 1,623–1,640.

Chamberlin, D. (2002). XQuery: An XML query language. *IBM Systems Journal, 41*(4), 597–615.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., et al. (2006). Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th symposium on operating systems design and implementation* (pp. 205–218). USENIX Association.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the annual ACM symposium on theory of computing* (pp. 380–388). ACM.

Chaudhuri, S., Ramakrishnan, R., & Weikum, G. (2005). Integrating DB and IR technologies: What is the sound of one hand clapping? In *CIDR 2005: Second biennial conference on innovative data systems research* (pp. 1–12).

Chen, Y.-Y., Suel, T., & Markowetz, A. (2006). Efficient query processing in geographic web search engines. In *SIGMOD '06: Proceedings of the ACM SIGMOD international conference on management of data* (pp. 277–288). ACM.

Cho, J., & Garcia-Molina, H. (2002). Parallel crawlers. In *WWW 2002: Proceedings of the 11th annual international world wide web conference* (pp. 124–135). ACM.

Cho, J., & Garcia-Molina, H. (2003). Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems, 28*, 390–426.

Church, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the second conference on applied natural language processing* (pp. 136–143). Association for Computational Linguistics.

Church, K. W., & Hanks, P. (1989). Word association norms, mutual information, and lexicography. In *Proceedings of the 27th annual meeting on Association for Computational Linguistics* (pp. 76–83). Association for Computational Linguistics.

Clarke, C. L., Agichtein, E., Dumais, S., & White, R. W. (2007). The influence of caption features on clickthrough patterns in web search. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 135–142). ACM.

Cleverdon, C. (1970). Evaluation tests of information retrieval systems. *Journal of Documentation, 26*(1), 55–67.

Coden, A., Brown, E. W., & Srinivasan, S. (Eds.). (2002). *Information retrieval techniques for speech applications*. London: Springer-Verlag.

Cong, G., Wang, L., Lin, C.-Y., Song, Y.-I., & Sun, Y. (2008). Finding question-answer pairs from online forums. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 467–474). ACM.

Conrad, J. G., & Utt, M. H. (1994). A system for discovering relationships by feature extraction from text databases. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 260–270). Springer-Verlag.

Cooper, W. S. (1968). Expected search length: A single measure of retrieval effec-

tiveness based on the weak ordering action of retrieval systems. *American Documentation, 19*(1), 30–41.

Cooper, W. S., Gey, F. C., & Dabney, D. P. (1992). Probabilistic retrieval based on staged logistic regression. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 198–210). ACM.

Cowie, J., & Lehnert, W. (1996). Information extraction. *Communications of the ACM, 39*(1), 80–91.

Crawford, R. (1981). The relational model in information retrieval. *Journal of the American Society for Information Science, 32*(1), 51–64.

Croft, W. B. (2000). Combining approaches to information retrieval. In *Advances in information retrieval: Recent research from the CIIR* (pp. 1–36). Norwell, MA: Kluwer Academic Publishers.

Croft, W. B., Krovetz, R., & Turtle, H. (1990). Interactive retrieval of complex documents. *Information Processing and Management, 26*(5), 593–613.

Croft, W. B., & Lafferty, J. (2003). *Language modeling for information retrieval.* Norwell, MA: Kluwer Academic Publishers.

Croft, W. B., Smith, L. A., & Turtle, H. R. (1992). A loosely-coupled integration of a text retrieval system and an object-oriented database system. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 223–232). ACM.

Croft, W. B., & Turtle, H. (1989). A retrieval model incorporating hypertext links. In *Hypertext '89: Proceedings of the second annual ACM conference on hypertext* (pp. 213–224). ACM.

Croft, W. B., Turtle, H. R., & Lewis, D. D. (1991). The use of phrases and structured queries in information retrieval. In *SIGIR '91: Proceedings of the 14th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 32–45). ACM.

Cronen-Townsend, S., Zhou, Y., & Croft, W. B. (2006). Precision prediction based on ranked list coherence. *Information Retrieval, 9*(6), 723–755.

Cucerzan, S., & Brill, E. (2004). Spelling correction as an iterative process that exploits the collective knowledge of web users. In D. Lin & D. Wu (Eds.), *Proceedings of EMNLP 2004* (pp. 293–300). Association for Computational Linguistics.

Cui, H., Wen, J.-R., Nie, J.-Y., & Ma, W.-Y. (2003). Query expansion by mining user logs. *IEEE Transactions on Knowledge and Data Engineering, 15*(4), 829–839.

Dannenberg, R. B., Birmingham, W. P., Pardo, B., Hu, N., Meek, C., & Tzanetakis, G. (2007). A comparative evaluation of search techniques for query-by-humming using the MUSART testbed. *Journal of the American Society for Information Science and Technology, 58*(5), 687–701.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM, 51*(1), 107–113.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., et al. (2007). Dynamo: Amazon's highly available key-value store. In *SOSP '07: Proceedings of the twenty-first ACM SIGOPS symposium on op-*

erating systems principles (pp. 205–220). ACM.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., & Harshman, R. A. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science, 41*(6), 391–407.

Deutsch, P. (1996). *DEFLATE compressed data format specification version 1.3* (RFC No. 1951). Internet Engineering Task Force. Available from `http://www.rfc-editor.org/rfc/rfc1951.txt`

Diaz, F. (2005). Regularizing ad hoc retrieval scores. In *CIKM '05: Proceedings of the 14th ACM international conference on information and knowledge management* (pp. 672–679). ACM.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern classification* (2nd ed.). Wiley-Interscience.

Dunlop, M. D., & van Rijsbergen, C. J. (1993). Hypermedia and free text retrieval. *Information Processing and Management, 29*(3), 287–298.

Echihabi, A., & Marcu, D. (2003). A noisy-channel approach to question answering. In *ACL '03: Proceedings of the 41st annual meeting on Association for Computational Linguistics* (pp. 16–23). Association for Computational Linguistics.

Efthimiadis, E. N. (1996). Query expansion. In M. E. Williams (Ed.), *Annual review of information systems and technology (ARIST)* (Vol. 31, pp. 121–187).

Elmasri, R., & Navathe, S. (2006). *Fundamentals of database systems* (5th ed.). Reading, MA: Addison-Wesley.

Fagin, R., Lotem, A., & Naor, M. (2003). Optimal aggregation algorithms for middleware. *Journal of Computer and Systems Sciences, 66*(4), 614–656.

Feng, J., Bhargava, H. K., & Pennock, D. M. (2007). Implementing sponsored search in web search engines: Computational evaluation of alternative mechanisms. *INFORMS Journal on Computing, 19*(1), 137–148.

Fetterly, D., Manasse, M., & Najork, M. (2003). On the evolution of clusters of near-duplicate web pages. In *LA-WEB '03: Proceedings of the first conference on Latin American Web Congress* (pp. 37–45). IEEE Computer Society.

Finn, A., Kushmerick, N., & Smyth, B. (2001). Fact or fiction: Content classification for digital libraries. In *DELOS workshop: Personalisation and recommender systems in digital libraries.*

Flake, G. W., Lawrence, S., & Giles, C. L. (2000). Efficient identification of web communities. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 150–160). ACM.

Flickner, M., Sawhney, H. S., Ashley, J., Huang, Q., Dom, B., Gorkani, M., et al. (1995). Query by image and video content: The QBIC system. *IEEE Computer, 28*(9), 23–32.

Fox, S., Karnawat, K., Mydland, M., Dumais, S., & White, T. (2005). Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems, 23*(2), 147–168.

Fuhr, N. (2000). Probabilistic datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science and Technology, 51*(2), 95–110.

Fuhr, N. (2008). A probability ranking principle for interactive information retrieval. *Information Retrieval*, *11*(3), 251–265.

Fuhr, N., & Buckley, C. (1991). A probabilistic learning approach for document indexing. *ACM Transactions on Information Systems*, *9*(3), 223–248.

Fuhr, N., & Rölleke, T. (1997). A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, *15*(1), 32–66.

Fujii, H., & Croft, W. B. (1993). A comparison of indexing techniques for Japanese text retrieval. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 237–246). ACM.

Gao, J., Nie, J.-Y., Wu, G., & Cao, G. (2004). Dependence language model for information retrieval. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 170–177). ACM.

Gao, J., Qi, H., Xia, X., & Nie, J.-Y. (2005). Linear discriminant model for information retrieval. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 290–297). ACM.

Garcia-Molina, H., Ullman, J. D., & Widom, J. D. (2008). *Database systems: The complete book*. Prentice Hall.

Gibson, D., Kleinberg, J., & Raghavan, P. (1998). Inferring web communities from link topology. In *HYPERTEXT '98: Proceedings of the ninth ACM conference on hypertext and hypermedia* (pp. 225–234). ACM.

Golder, S. A., & Huberman, B. A. (2006). Usage patterns of collaborative tagging systems. *Journal of Information Science*, *32*(2), 198–208.

Goldstein, J., Kantrowitz, M., Mittal, V., & Carbonell, J. (1999). Summarizing text documents: sentence selection and evaluation metrics. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval* (pp. 121–128). ACM.

Grefenstette, G. (1998). *Cross-language information retrieval*. Norwell, MA: Kluwer Academic Publishers.

Guo, J., Xu, G., Li, H., & Cheng, X. (2008). A unified and discriminative model for query refinement. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 379–386). ACM.

Gupta, S., Kaiser, G., Neistadt, D., & Grimm, P. (2003). DOM-based content extraction of HTML documents. In *WWW '03: Proceedings of the 12th international conference on World Wide Web* (pp. 207–214). ACM.

Gyöngyi, Z., & Garcia-Molina, H. (2005). Web spam taxonomy. In *AIRWeb: 1st international workshop on adversarial information retrieval on the web* (pp. 39–47).

Gyöngyi, Z., Garcia-Molina, H., & Pedersen, J. (2004). Combating web spam with TrustRank. In *VLDB 2004: Proceedings of the thirtieth international conference on very large data bases* (pp. 576–587). Morgan Kaufmann.

Ha, L. Q., Sicilia-Garcia, E. I., Ming, J., & Smith, F. J. (2002). Extension of Zipf's

law to words and phrases. In *Proceedings of the 19th international conference on computational linguistics* (pp. 1–6). Association for Computational Linguistics.

Harding, S. M., Croft, W. B., & Weir, C. (1997). Probabilistic retrieval of OCR degraded text using n-grams. In *ECDL '97: Proceedings of the first European conference on research and advanced technology for digital libraries* (pp. 345–359). Springer-Verlag.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning: Data mining, inference, and prediction*. Springer.

Hatcher, E., & Gospodnetic, O. (2004). *Lucene in action*. Manning Publications.

Haveliwala, T. H. (2002). Topic-sensitive PageRank. In *WWW '02: Proceedings of the 11th international conference on World Wide Web* (pp. 517–526). ACM.

Hawking, D., & Zobel, J. (2007). Does topic metadata help with web search? *Journal of the American Society for Information Science and Technology*, *58*(5), 613–628.

He, B., Patel, M., Zhang, Z., & Chang, K. (2007). Accessing the deep web. *Communications of the ACM*, *50*(5), 94–101.

Heaps, H. (1978). *Information retrieval: Computational and theoretical aspects*. New York: Academic Press.

Hearst, M. A. (1999). User interfaces and visualization. In *Modern information retrieval* (pp. 257–324). ACM/Addison-Wesley.

Hearst, M. A. (2006). Clustering versus faceted categories for information exploration. *Communications of the ACM*, *49*(4), 59–61.

Hearst, M. A., & Pedersen, J. O. (1996). Reexamining the cluster hypothesis: scatter/gather on retrieval results. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 76–84). ACM.

Henzinger, M. (2006). Finding near-duplicate web pages: A large-scale evaluation of algorithms. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 284–291). ACM.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, *22*(1), 5–53.

Heymann, P., Koutrika, G., & Garcia-Molina, H. (2008). Can social bookmarking improve web search? In *WSDM '08: Proceedings of the international conference on web search and web data mining* (pp. 195–206). ACM.

Heymann, P., Ramage, D., & Garcia-Molina, H. (2008). Social tag prediction. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 531–538). ACM.

Hiemstra, D. (1998). A linguistically motivated probabilistic model of information retrieval. In *ECDL '98: Proceedings of the second european conference on research and advanced technology for digital libraries* (pp. 569–584). Springer-Verlag.

Hoad, T., & Zobel, J. (2003). Methods for identifying versioned and plagiarised documents. *Journal of the American Society of Information Science and Technology*, *54*(3), 203–215.

Hobbs, J., Douglas, R., Appelt, E., Bear, J., Israel, D., Kameyama, M., et al. (1997). Fastus: A cascaded finite-state transducer for extracting information from natural-language text. In *Finite state language processing* (chap. 13). Cambridge, MA: MIT Press.

Hofmann, T. (1999). Probabilistic latent semantic indexing. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval* (pp. 50–57). ACM.

Hopcroft, J., Khan, O., Kulis, B., & Selman, B. (2003). Natural communities in large linked networks. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 541–546). ACM.

Ingwersen, P., & Järvelin, K. (2005). *The turn: Integration of information seeking and retrieval in context.* Secaucus, NJ: Springer-Verlag.

Ipeirotis, P. G., & Gravano, L. (2004). When one sample is not enough: Improving text database selection using shrinkage. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on management of data* (pp. 767–778). ACM.

Ittycheriah, A., Franz, M., Zhu, W.-J., Ratnaparkhi, A., & Mammone, R. J. (2001). Question answering using maximum entropy components. In *NAACL '01: Second meeting of the North American Chapter of the Association for Computational Linguistics on language technologies* (pp. 1–7). Association for Computational Linguistics.

Järvelin, K., & Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, *20*(4), 422–446.

Jeon, J., Croft, W. B., & Lee, J. H. (2005). Finding similar questions in large question and answer archives. In *CIKM '05: Proceedings of the 14th ACM international conference on information and knowledge management* (pp. 84–90). ACM.

Jeon, J., Croft, W. B., Lee, J. H., & Park, S. (2006). A framework to predict the quality of answers with non-textual features. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 228–235). ACM.

Jijkoun, V., & de Rijke, M. (2005). Retrieving answers from frequently asked questions pages on the web. In *CIKM '05: Proceedings of the 14th ACM international conference on information and knowledge management* (pp. 76–83). ACM.

Jing, Y., & Croft, W. B. (1994). An association thesaurus for information retrieval. In *Proceedings of RIAO-94, 4th international conference "recherche d'information assistee par ordinateur"* (pp. 146–160).

Joachims, T. (2002a). *Learning to classify text using support vector machines: Methods, theory and algorithms.* Norwell, MA: Kluwer Academic Publishers.

Joachims, T. (2002b). Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference*

on knowledge discovery and data mining (pp. 133–142). ACM.

Joachims, T., Granka, L., Pan, B., Hembrooke, H., & Gay, G. (2005). Accurately interpreting clickthrough data as implicit feedback. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 154–161). ACM.

Jones, R., Rey, B., Madani, O., & Greiner, W. (2006). Generating query substitutions. In *WWW '06: Proceedings of the 15th international conference on World Wide Web* (pp. 387–396). ACM.

Jurafsky, D., & Martin, J. H. (2006). *Speech and language processing* (2nd ed.). London: Prentice Hall.

Kazai, G., Gövert, N., Lalmas, M., & Fuhr, N. (2003). The INEX evaluation initiative. In H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, & G. Weikum (Eds.), *Intelligent XML retrieval* (pp. 279–293). Springer.

Kelly, D., & Teevan, J. (2003). Implicit feedback for inferring user preference: A bibliography. *SIGIR Forum, 32*(2).

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM, 46*(5), 604–632.

Knuth, D. E. (1998). *The art of computer programming: Sorting and searching* (2nd ed., Vol. 3). Redwood City, CA: Addison-Wesley Longman.

Koenemann, J., & Belkin, N. J. (1996). A case for interaction: a study of interactive information retrieval behavior and effectiveness. In *CHI '96: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 205–212). ACM.

Kraaij, W., Westerveld, T., & Hiemstra, D. (2002). The importance of prior probabilities for entry page search. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 27–34). ACM.

Krovetz, R. (1993). Viewing morphology as an inference process. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 191–202). ACM.

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

Kurland, O. (2008). The opposite of smoothing: A language model approach to ranking query-specific document clusters. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 171–178). ACM.

Kurland, O., & Lee, L. (2004). Corpus structure, language models, and ad hoc information retrieval. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 194–201). ACM.

Kurland, O., & Lee, L. (2005). Pagerank without hyperlinks: structural reranking using links induced by language models. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 306–313). ACM.

Lafferty, J., & Zhai, C. (2001). Document language models, query models, and risk minimization for information retrieval. In *SIGIR '01: Proceedings of*

the 24th annual international ACM SIGIR conference on research and development in information retrieval (pp. 111–119). ACM.

Lankes, R. D. (2004). The digital reference research agenda. *Journal of the American Society for Information Science and Technology*, *55*(4), 301–311.

Larkey, L. S., Ballesteros, L., & Connell, M. E. (2002). Improving stemming for Arabic information retrieval: Light stemming and co-occurrence analysis. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 275–282). ACM.

Lavrenko, V., & Croft, W. B. (2001). Relevance based language models. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 120–127). ACM.

Lawrie, D. J., & Croft, W. B. (2003). Generating hierarchical summaries for web searches. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 457–458). ACM.

Leouski, A., & Croft, W. (1996). *An evaluation of techniques for clustering search results* (Tech. Rep. Nos. IR–76). Department of Computer Science, University of Massachusetts Amherst.

Lester, N., Moffat, A., & Zobel, J. (2005). Fast on-line index construction by geometric partitioning. In *CIKM '05: Proceedings of the 14th ACM international conference on information and knowledge management* (pp. 776–783). New York: ACM.

Leuski, A., & Lavrenko, V. (2006). Tracking dragon-hunters with language models. In *CIKM '06: Proceedings of the 15th ACM international conference on information and knowledge management* (pp. 698–707). ACM.

Liu, X., & Croft, W. B. (2004). Cluster-based retrieval using language models. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 186–193). ACM.

Liu, X., & Croft, W. B. (2008). Evaluating text representations for retrieval of the best group of documents. In *ECIR '08: Proceedings of the 30th European conference on information retrieval* (pp. 454–462). Springer.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *60*(2), 91–110.

Lu, J., & Callan, J. (2006). Full-text federated search of text-based digital libraries in peer-to-peer networks. *Information Retrieval*, *9*(4), 477–498.

Lu, J., & Callan, J. (2007). Content-based peer-to-peer network overlay for full-text federated search. In *RIAO '07: Proceedings of the eighth RIAO conference.*

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

Lund, K., & Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instrumentation, and Computers*, *28*(2), 203–208.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information*

retrieval. New York: Cambridge University Press.

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing*. Cambridge, MA: The MIT Press.

Marchionini, G. (2006). Exploratory search: from finding to understanding. *Communications of the ACM*, *49*(4), 41–46.

McBryan, O. A. (1994). GENVL and WWWW: Tools for Taming the Web. In *WWW '94: Proceedings of the first international World Wide Web conference* (p. 15). CERN, Geneva.

McCallum, A. (2005). Information extraction: distilling structured data from unstructured text. *Queue*, *3*(9), 48–57.

McCallum, A., & Nigam, K. (1998). A comparison of event models for naive Bayes text classification. In *AAAI-98 workshop on learning for text categorization*.

Menczer, F., & Belew, R. K. (1998). Adaptive information agents in distributed textual environments. In *AGENTS '98: Proceedings of the second international conference on autonomous agents* (pp. 157–164). ACM.

Metzler, D., & Croft, W. B. (2004). Combining the language model and inference network approaches to retrieval. *Information Processing and Management*, *40*(5), 735–750.

Metzler, D., & Croft, W. B. (2005a). Analysis of statistical question classification for fact-based questions. *Information Retrieval*, *8*(3), 481–504.

Metzler, D., & Croft, W. B. (2005b). A Markov random field model for term dependencies. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 472–479). ACM.

Metzler, D., & Croft, W. B. (2007a). Latent concept expansion using Markov random fields. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 311–318). ACM.

Metzler, D., & Croft, W. B. (2007b). Linear feature-based models for information retrieval. *Information Retrieval*, *10*(3), 257–274.

Metzler, D., Dumais, S. T., & Meek, C. (2007). Similarity measures for short segments of text. In *ECIR '07: Proceedings of the European conference on information retrieval* (pp. 16–27). Springer.

Metzler, D., Lavrenko, V., & Croft, W. B. (2004). Formal multiple-Bernoulli models for language modeling. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 540–541). ACM.

Metzler, D., Strohman, T., & Croft, W. B. (2008). A statistical view of binned retrieval models. In *ECIR 2008: Proceedings of the 30th European conference on information retrieval* (pp. 175–186). Springer.

Metzler, D., Strohman, T., Turtle, H., & Croft, W. (2004). Indri at TREC 2004: Terabyte track. In *NIST special publication 500–261: Text REtrieval Conference proceedings (TREC 2004)*. National Institute of Standards and Technology.

Miller, D. R. H., Leek, T., & Schwartz, R. M. (1999). A Hidden Markov Model

information retrieval system. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval* (pp. 214–221). ACM.

Mizzaro, S. (1997). Relevance: The whole history. *Journal of the American Society of Information Science, 48*(9), 810–832.

Moffat, A., Webber, W., & Zobel, J. (2007). Strategic system comparisons via targeted relevance judgments. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 375–382). ACM.

Montague, M., & Aslam, J. A. (2002). Condorcet fusion for improved retrieval. In *CIKM '02: Proceedings of the eleventh international conference on information and knowledge management* (pp. 538–548). ACM.

Morris, M. R. (2008). A survey of collaborative web search practices. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on human factors in computing systems* (pp. 1,657–1,660). ACM.

Morris, M. R., & Horvitz, E. (2007a). S^3: Storable, shareable search. In *Interact (1)* (pp. 120–123).

Morris, M. R., & Horvitz, E. (2007b). SearchTogether: an interface for collaborative web search. In *UIST '07: Proceedings of the 20th annual ACM symposium on user interface software and technology* (pp. 3–12). ACM.

Ntoulas, A., Najork, M., Manasse, M., & Fetterly, D. (2006). Detecting spam web pages through content analysis. In *WWW '06: Proceedings of the 15th international conference on World Wide Web* (pp. 83–92).

Ogilvie, P., & Callan, J. (2003). Combining document representations for known-item search. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on research and development in informaion retrieval* (pp. 143–150). ACM.

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?: sentiment classification using machine learning techniques. In *EMNLP '02: Proceedings of the ACL-02 conference on empirical methods in natural language processing* (pp. 79–86). Association for Computational Linguistics.

Peng, F., Ahmed, N., Li, X., & Lu, Y. (2007). Context sensitive stemming for web search. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 639–646). ACM.

Peng, F., Feng, F., & McCallum, A. (2004). Chinese segmentation and new word detection using conditional random fields. In *COLING '04: Proceedings of the 20th international conference on computational linguistics* (p. 562). Association for Computational Linguistics.

Pentland, A., Picard, R. W., & Sclaroff, S. (1996). Photobook: Content-based manipulation of image databases. *International Journal of Computer Vision, 18*(3), 233–254.

Petkova, D., & Croft, W. B. (2007). Proximity-based document representation for named entity retrieval. In *CIKM '07: Proceedings of the sixteenth ACM conference on information and knowledge management* (pp. 731–740). ACM.

Pickens, J., Golovchinsky, G., Shah, C., Qvarfordt, P., & Back, M. (2008). Algorithmic mediation for collaborative exploratory search. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 315–322). ACM.

Pinto, D., Branstein, M., Coleman, R., Croft, W. B., King, M., Li, W., et al. (2002). QuASM: a system for question answering using semi-structured data. In *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on digital libraries* (pp. 46–55). ACM.

Ponte, J. M., & Croft, W. B. (1998). A language modeling approach to information retrieval. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 275–281). ACM.

Porter, M. F. (1997). An algorithm for suffix stripping. In *Readings in information retrieval* (pp. 313–316). San Francisco: Morgan Kaufmann.

Powell, A. L., French, J. C., Callan, J., Connell, M., & Viles, C. L. (2000). The impact of database selection on distributed searching. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval* (pp. 232–239). ACM.

Pritchard-Schoch, T. (1993). WIN–WESTLAW goes natural. *Online, 17*(1), 101–103.

Rajashekar, T. B., & Croft, W. B. (1995). Combining automatic and manual index representations in probabilistic retrieval. *Journal of the American Society of Information Science, 46*(4), 272–283.

Ravela, S., & Manmatha, R. (1997). Image retrieval by appearance. In *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 278–285). ACM.

Riezler, S., Vasserman, A., Tsochantaridis, I., Mittal, V., & Liu, Y. (2007). Statistical machine translation for query expansion in answer retrieval. In *Proceedings of the 45th annual meeting of the association of computational linguistics* (pp. 464–471). ACL.

Robertson, S. E. (1997). The probability ranking principle in IR. In *Readings in information retrieval* (pp. 281–286). Morgan Kaufmann. (Reprinted from *Journal of Documentation*, 1977, *33*, 294–304)

Robertson, S. E. (2004). Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation, 60*, 503–520.

Robertson, S. E., & Walker, S. (1994). Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 232–241). Springer-Verlag.

Robertson, S. E., Zaragoza, H., & Taylor, M. (2004). Simple BM25 extension to multiple weighted fields. In *CIKM '04: Proceedings of the thirteenth ACM international conference on information and knowledge management* (pp. 42–49). ACM.

Rocchio, J. J. (1971). Relevance feedback in information retrieval. In G. Salton (Ed.), *The SMART retrieval system: Experiments in automatic document processing* (pp. 313–323). Englewood Cliffs, NJ: Prentice-Hall.

Romano, N. C., Roussinov, D., Nunamaker, J. F., & Chen, H. (1999). Collaborative information retrieval environment: Integration of information retrieval with group support systems. In *HICSS '99: Proceedings of the thirty-second annual Hawaii international conference on system sciences-volume 1* (pp. 1,053). IEEE Computer Society.

Sahami, M., & Heilman, T. D. (2006). A web-based kernel function for measuring the similarity of short text snippets. In *WWW '06: Proceedings of the 15th international conference on World Wide Web* (pp. 377–386). ACM.

Salton, G. (1968). *Automatic information organization and retrieval.* New York: McGraw-Hill.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management, 24*(5), 513–523.

Salton, G., & McGill, M. J. (1983). *Introduction to modern information retrieval.* New York: McGraw-Hill.

Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM, 18*(11), 613–620.

Sanderson, M., & Zobel, J. (2005). Information retrieval system evaluation: effort, sensitivity, and reliability. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 162–169). ACM.

Saracevic, T. (1975). Relevance: A review of and a framework for the thinking on the notion in information science. *Journal of the American Society for Information Science, 26*(6), 321–343.

Saraiva, P. C., de Moura, E. S., Ziviani, N., Meira, W., Fonseca, R., & Riberio-Neto, B. (2001). Rank-preserving two-level caching for scalable search engines. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 51–58). New York: ACM.

Schapire, R. E., Singer, Y., & Singhal, A. (1998). Boosting and Rocchio applied to text filtering. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 215–223). ACM.

Shannon, C. (1951). Prediction and entropy in printed English. *Bell System Technical Journal, 30*, 50–64.

Shannon, C., & Weaver, W. (1963). *A mathematical theory of communication.* Champaign, IL: University of Illinois Press.

Shneiderman, B., Byrd, D., & Croft, W. B. (1998). Sorting out searching: a user-interface framework for text searches. *Communications of the ACM, 41*(4), 95–98.

Si, L., & Callan, J. (2003). A semi-supervised learning method to merge search engine results. *ACM Transactions on Information Systems, 21*(4), 457–491.

Si, L., & Callan, J. (2004). Unified utility maximization framework for resource selection. In *CIKM '04: Proceedings of the eleventh international conference on information and knowledge management.* ACM.

Singhal, A., & Pereira, F. (1999). Document expansion for speech retrieval. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR con-*

ference on research and development in information retrieval (pp. 34–41). ACM.

Smucker, M., Allan, J., & Carterette, B. (2007). A comparison of statistical significance tests for information retrieval evaluation. In *CIKM '07: Proceedings of the 14th ACM international conference on information and knowledge management.* ACM.

Song, F., & Croft, W. B. (1999). A general language model for information retrieval. In *CIKM '99: Proceedings of the eighth international conference on information and knowledge management* (pp. 316–321). ACM.

Song, R., Liu, H., Wen, J.-R., & Ma, W.-Y. (2004). Learning block importance models for web pages. In *WWW '04: Proceedings of the 13th international conference on World Wide Web* (pp. 203–211). ACM.

Sparck Jones, K., Walker, S., & Robertson, S. E. (2000). A probabilistic model of information retrieval: development and comparative experiments. *Information Processing and Management*, *36*(6), 779–808.

Strohman, T. (2007). *Efficient processing of complex features for information retrieval.* Unpublished doctoral dissertation, University of Massachusetts Amherst.

Strohman, T., & Croft, W. B. (2006). Low latency index maintenance in Indri. In *OSIR 2006: Proceedings of the second international workshop on open source information retrieval* (pp. 7–11).

Strohman, T., Metzler, D., Turtle, H., & Croft, W. B. (2005). Indri: A language model-based search engine for complex queries. In *Proceedings of the international conference on intelligence analysis.*

Sun, J.-T., Shen, D., Zeng, H.-J., Yang, Q., Lu, Y., & Chen, Z. (2005). Webpage summarization using clickthrough data. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 194–201). ACM.

Sutton, C., & McCallum, A. (2007). An introduction to conditional random fields for relational learning. In L. Getoor & B. Taskar (Eds.), *Introduction to statistical relational learning.* Cambridge, MA, USA: MIT Press.

Taghva, K., Borsack, J., & Condit, A. (1996). Evaluation of model-based retrieval effectiveness with OCR text. *ACM Transactions on Information Systems*, *14*(1), 64–93.

Trotman, A., & Lalmas, M. (2006). Why structural hints in queries do not help XML-retrieval. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 711–712). ACM.

Trotman, A., & Sigurbjörnsson, B. (2004). Narrowed Extended XPath I (NEXI). In *INEX workshop proceedings* (pp. 16–40). Springer.

Turpin, A., Tsegay, Y., & Hawking, D. (2007). Fast generation of result snippets in web search. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 127–134). ACM.

Turtle, H. (1994). Natural language vs. Boolean query evaluation: a comparison of retrieval performance. In *SIGIR '94: Proceedings of the 17th annual*

international ACM SIGIR conference on research and development in information retrieval (pp. 212–220). Springer-Verlag.

Turtle, H., & Croft, W. B. (1991). Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems, 9*(3), 187–222.

Turtle, H., & Flood, J. (1995). Query evaluation: strategies and optimizations. *Information Processing and Management, 31*(6), 831–850.

Unicode Consortium. (2006). *The Unicode standard, version 5.0.* Addison-Wesley Professional.

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

Vasconcelos, N. (2007). From pixels to semantic spaces: Advances in content-based image retrieval. *Computer, 40*(7), 20–26.

Voorhees, E. M. (1985). The cluster hypothesis revisited. In *SIGIR '85: Proceedings of the 8th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 188–196). ACM.

Voorhees, E. M., & Buckley, C. (2002). The effect of topic set size on retrieval experiment error. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 316–323). ACM.

Voorhees, E. M., & Harman, D. (Eds.). (2005). *TREC: Experiment and evaluation in information retrieval.* Cambridge, MA: MIT Press.

Wang, A. (2006). The Shazam music recognition service. *Communications of the ACM, 49*(8), 44–48.

Wei, X., & Croft, W. B. (2006). LDA-based document models for ad-hoc retrieval. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 178–185). ACM.

Wei, X., & Croft, W. B. (2007). Investigating retrieval performance with manually-built topic models. In *RIAO '07: Proceedings of the eighth RIAO conference.*

Welch, T. A. (1984). A technique for high-performance data compression. *Computer, 17*, 8–19.

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing Gigabytes: Compressing and indexing documents and images* (2nd ed.). San Francisco, CA, USA: Morgan Kaufmann.

Xia, F., Liu, T., Wang, J., Zhang, W., & Li, H. (2008). Listwise approach to learning to rank – theory and algorithm. In *ICML '08: Proceedings of the 25th annual international conference on machine learning* (pp. 1,192–1,199). Omnipress.

Xu, J., & Croft, W. B. (1998). Corpus-based stemming using cooccurrence of word variants. *ACM Transactions on Information Systems, 16*(1), 61–81.

Xu, J., & Croft, W. B. (2000). Improving the effectiveness of information retrieval with local context analysis. *ACM Transactions on Information Systems, 18*(1), 79–112.

Xu, Z., Fu, Y., Mao, J., & Su, D. (2006). Towards the semantic web: Collaborative tag suggestions. In *WWW2006: Proceedings of the collaborative web tagging*

workshop. Edinburgh, Scotland.

Xue, X., Jeon, J., & Croft, W. B. (2008). Retrieval models for question and answer archives. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 475–482). ACM.

Yang, S., Zhu, H., Apostoli, A., & Cao, P. (2007). N-gram statistics in English and Chinese: Similarities and differences. In *ICSC '07: International conference on semantic computing* (pp. 454–460). IEEE Computer Society.

Yao, Y. (1995). Measuring retrieval effectiveness based on user preference of documents. *Journal of the American Society for Information Science, 46*(2), 133–145.

Yih, W., Goodman, J., & Carvalho, V. R. (2006). Finding advertising keywords on web pages. In *WWW '06: Proceedings of the 15th international conference on World Wide Web* (pp. 213–222). ACM.

Yu, S., Cai, D., Wen, J.-R., & Ma, W.-Y. (2003). Improving pseudo-relevance feedback in web information retrieval using web page segmentation. In *WWW '03: Proceedings of the 12th international conference on World Wide Web* (pp. 11–18). ACM.

Zamir, O., & Etzioni, O. (1999). Grouper: a dynamic clustering interface to web search results. *Computer Networks, 31*(11–16), 1,361–1,374.

Zeng, H.-J., He, Q.-C., Chen, Z., Ma, W.-Y., & Ma, J. (2004). Learning to cluster web search results. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 210–217). ACM.

Zhai, C., & Lafferty, J. (2004). A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems, 22*(2), 179–214.

Zhang, V., Rey, B., Stipp, E., & Jones, R. (2006). Geomodification in query rewriting. In *GIR '06: Proceedings of the workshop on geographic information retrieval, ACM SIGIR 2006.*

Zhang, Y., & Callan, J. (2001). Maximum likelihood estimation for filtering thresholds. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 294–302). ACM.

Zhou, Y., Xie, X., Wang, C., Gong, Y., & Ma, W.-Y. (2005). Hybrid index structures for location-based web search. In *CIKM '05: Proceedings of the 14th ACM international conference on information and knowledge management* (pp. 155–162). ACM.

Zobel, J. (1998). How reliable are the results of large-scale information retrieval experiments? In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 307–314). ACM.

Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys, 38*(2), 6.

Zobel, J., Moffat, A., & Ramamohanarao, K. (1996). Guidelines for presentation and comparison of indexing techniques. *ACM SIGMOD Record, 25*(3),

10–15.

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

Zukowski, M., Héman, S., Nes, N., & Boncz, P. A. (2006). Super-scalar RAM-CPU cache compression. In *ICDE: International conference on data engineering* (p. 59). IEEE Computer Society.

搜索引擎 信息检索实践

本书介绍了信息检索（IR）中的关键问题，以及这些问题如何影响搜索引擎的设计与实现，并且用数学模型强化了重要的概念。对于网络搜索引擎这一重要的话题，书中主要涵盖了在网络上广泛使用的搜索技术。

本书适用于高等院校计算机科学或计算机工程专业的本科生、研究生，对于专业人士而言，本书也不失为一本理想的入门教材。

作者简介

W. Bruce Croft 马萨诸塞大学阿默斯特分校计算机科学特聘教授、ACM会士。他创建了智能信息检索研究中心，发表了200余篇论文，多次获奖，其中包括2003年由ACM SIGIR颁发的Gerard Salton奖。

Donald Metzler 马萨诸塞大学阿默斯特分校博士，是位于加州Santa Clara的雅虎研究中心搜索与计算机广告组的研究科学家。

Trevor Strohman 马萨诸塞大学阿默斯特分校博士。他开发了Galago搜索引擎，也是Indri搜索引擎的主要开发者。

书号：978-7-111-28247-1
定价：45.00元

上架指导：计算机 信息检索 搜索引擎

ISBN 978-7-111-28808-4

9 787111 288084

定价：56.00元

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

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

封面设计：锡 彬

PEARSON

www.pearsonhighered.com

PEARSON Education