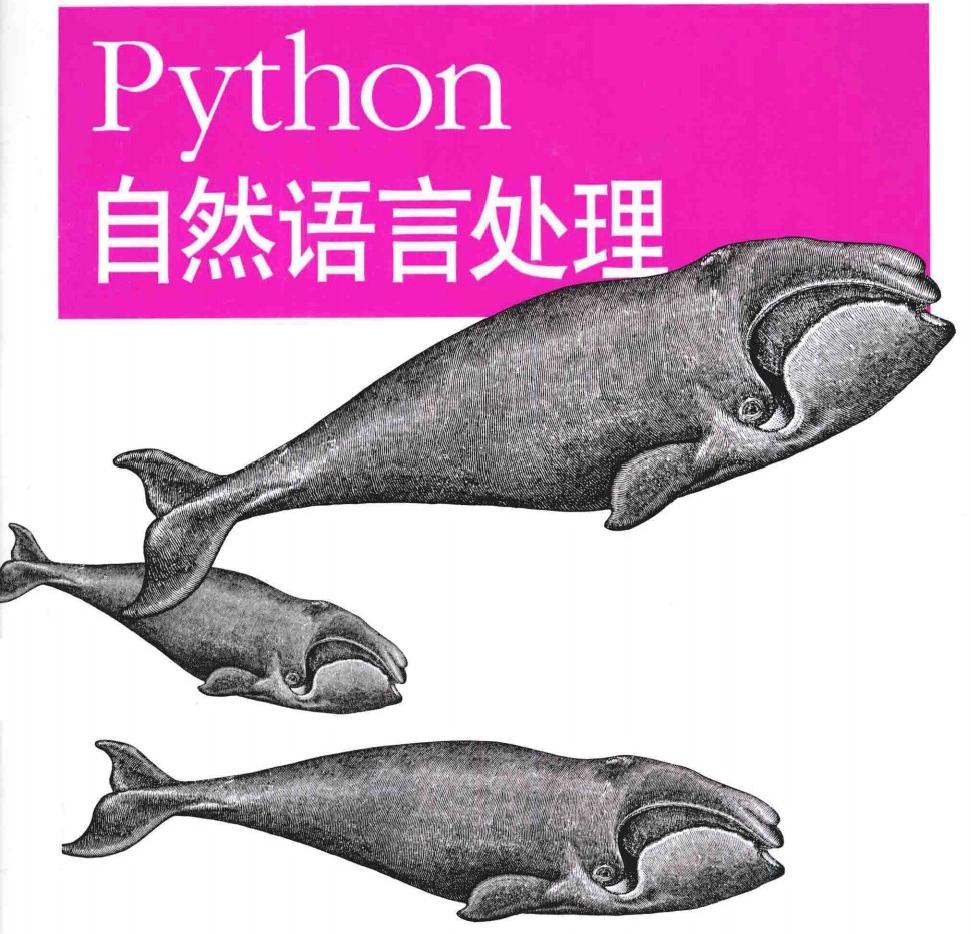
*Natural Language Processing with Python*



**O'REILLY**°

*[美]Steven Bird,Ewan Klein &Edward Loper 著*

陈涛张旭崔杨刘海平译

**人民邮电出版社**

**POSTS &TELECOM PRESS**



**O'REILLY°**

**Python 自然语言处理**

[美] Steven Bird Ewan Klein Edward Loper 著

陈 涛 张 旭 崔 杨 刘海平 译

**人** **民** **邮** **电** **出** **版** **社**

**北** **京**

著 译

**图书在版编目(CIP)** **数据**

Python 自然语言处理/(美)伯德(Bird,S.),

(美)克莱恩(Klein,E.), ( 美 ) 洛 佩 尔 (Loper,E . )著

;陈涛等译 .一北京：人民邮电出版社，2014 . 7

ISBN 978-7-115-33368-1

I.①P…Ⅱ.① 伯 … ② 克 … ③ 洛 … ④ 陈 … Ⅲ . ①

软件工具一自然语言处理 IV.①TP311.56②TP391

中国版本图书馆CIP数据核字(2013)第277137号

**版权声明**

CopyrightC 2009 by O'Reilly Media,Inc.

Simplified Chinese Edition,jointly published by O'Reilly Media,Inc.and Posts &Telecom.,Press, 2013.Authorized translation of the English edition,2012 O'Reilly Media,Inc.,the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

本书中文简体版由O'Reilly Media,Inc.授权人民邮电出版社出版。未经出版者书面许可，对本书的 任何部分不得以任何方式复制或抄袭。

版权所有，侵权必究。

◆

[美]Steven Bird Ewan 陈 涛 张 旭 崔

Klein Edward Loper

杨 刘海平

责任编辑 陈冀康

责任印制 彭志环 焦志炜

◆人民邮电出版社出版发行 北京市丰台区成寿寺路11号

邮编 100164 电子邮件315@ptpress.com.cn

网址<http://www.ptpress.com.cn>

三河市海波印务有限公司印刷

◆开本：787×10001/16

印张：31.75

字数：559千字 2014年7月第1版

印数：1-3000册 2014年7月河北第1次印刷

著作权合同登记号 图字：01-2013-2169号

定价：89.00 元

读者服务热线： (010)81055410 印装质量热线：(010)81055316

反盗版热线：(010)81055315

**内容提要**

自然语言处理 (Natural Language Processing,NLP) 是计算机科学领域与人工智能 领域中的一个重要方向。它研究实现人与计算机之间用自然语言进行有效通信的各 种理论和方法，涉及所有用计算机对自然语言进行的操作。

本书是自然语言处理领域的一本实用入门指南，旨在帮助读者学习如何编写程序来 分析书面语言。本书基于 Python 编程语言以及一个名为 NLTK 的自然语言工具包 的开源库，但并不要求读者有 Python 编程的经验。全书共11章，按照难易程度顺 序编排。第1章到第3章介绍了语言处理的基础，讲述如何使用小的 Python 程序 分析感兴趣的文本信息。第4章讨论结构化程序设计，以巩固前面几章中介绍的编 程要点。第5章到第7章介绍语言处理的基本原理，包括标注、分类和信息提取等。 第8章到第10章介绍了句子解析、句法结构识别和句意表达方法。第11章介绍了 如何有效管理语言数据。后记部分简要讨论了NLP 领域的过去和未来。

本书的实践性很强，包括上百个实际可用的例子和分级练习。本书可供读者用于自 学，也可以作为自然语言处理或计算语言学课程的教科书，还可以作为人工智能、 文本挖掘、语料库语言学等课程的补充读物。

**前言**

这是一本关于自然语言处理的书。所谓“自然语言”,是指人们日常交流使用的语 言，如英语、印地语、葡萄牙语等。相对于编程语言和数学符号这样的人工语言， 自然语言随着一代代的传递而不断演化，因而很难用明确的规则来确定。从广义上 讲，“自然语言处理”(Natural Language Processing,NLP) 包含所有用计算机对自 然语言进行的操作，从最简单的通过计数词汇出现的频率来比较不同的写作风格， 到最复杂的完全“理解”人所说的话，或至少达到能对人的话语作出有效反应的 程度。

NLP 的技术应用日益广泛。例如：手机和手持电脑对输入法联想提示和手写识别 的支持；网络搜索引擎能搜索到非结构化文本中的信息；机器翻译能把中文文本翻 译成西班牙文。通过提供更自然的人机界面和获取存储信息的高级手段，语言处理 正在这个多语种的信息社会中扮演着更核心的角色。

这本书提供自然语言处理领域的入门指南。它可以用来自学，也可以作为自然语言 处理或计算语言学课程的教科书，或是作为人工智能、文本挖掘、语料库语言学课 程的补充读物。本书实用性强，包括上百个实例和分级练习。

本书基于 Python 编程语言及名为自然语言工具包 (Natural Language Toolkit, NLTK) 的开源库。NLTK 包含大量的软件、数据和文档，所有这些都可以从 <http://www.nltk.org/>上免费下载。NLTK的发行版本支持Windows 、Macintosh 和 UNIX 平台。强烈建议你下载 Python 和 NLTk, 与我们一起尝试书中的例子和 练习。

**读者**

从科学、经济、社会和文化因素来看， NLP 十分重要。NLP 正在迅速成长，其中 很多理论和方法在大量新的语言技术中得到广泛应用。所以对很多人来说掌握 NLP 知识十分重要。在应用领域，包括从事人机交互、商业信息分析、Web 软件开发的

人；在学术界，包括从事人文计算学、语料库语言学到计算机科学和人工智能领域 的人。(学术界的很多人把NLP 称为“计算语言学”)

本书旨在帮助所有想要学习编写程序来分析书面语言的人，不管他们以前的编程经 验如何。

*初学编程?*

本书的前几章适合没有编程经验的读者，只要你不怕应对新概念和学习新的计 算机技能。书中的例子和数以百计的分级练习，你都可以亲自尝试一下。如果 你需要关于Python 的更一般的介绍，<htp://docs.python.org/> 给出了 Python 资源 列表。

*初学Python?*

有经验的程序员可以很快掌握书中的 Python 代码，而把更多精力专注于自然 语言处理。所有涉及的 Python 功能都经过精心解释和举例说明，你很快就会 体会到 Python 在这些应用领域的妙用。书中的语言索引会帮你查找书中的相 关论述。

*已经精通Python?*

你可以浏览一下 Python 的例子并且钻研从第1章开始就提到的语言分析材料。很 快你就能在这个神奇的领域展现你的技能。

**强调**

本书是一本介绍 NLP 的实用书籍。你将通过例子学习编写真正的程序，并通过实 践验证自己想法的价值。如果你没有学过编程，本书将教你如何**编程。**与其他编程 书籍不同的是，我们提供了丰富的NLP 实例和练习。我们撰写本书时讲究**探究原** 理，无论是严谨的语言学还是计算分析学，我们不回避所涉及的任何基础理论。我 们曾经试图在理论与实践之间寻求折中，确定它们之间的联系与边界。最终我们认 识到如果不能寓教于乐，几乎无法实现这个目标，所以我们竭尽所能写入了很多既 有益又有趣的应用和例子，有的甚至有些异想天开。

请注意本书并不是一本工具书。本书讲述的 Python 和 NLP是精心挑选的，并通过

教程的形式展现的。关于参考材料，请查阅 <http://python.org/>和 <htp://www.nltk.org/>, 那里有大量可搜索的资源。

本书也不是高深的计算机科学文章。书中的内容属于入门级和中级，目标读者是那 些想要学习如何使用 Python 和自然语言分析包来分析文本的人。若想学习 NLTK 中更高级的算法，你可以查阅 <http://www.nltk.org/>中的 Python 代码库，或查询在本 书中引用的其他文献。

**你将学到什么**

通过钻研本书，你将学到如下内容：

简单的程序如何帮你处理和分析语言数据，以及如何编写这些程序；

NLP 与语言学的关键概念是如何用来描述和分析语言的；

数据结构和算法是怎样在NLP 中运用的；

语言数据如何存储为标准格式，以及如何使用数据来评估NLP 技术的性能。

根据读者知识背景和学习NLP 的动机不同，从本书中获得的技能和知识也将不同， 详情见表 P-1。

**表** **P-1 目标和背景不同的读者，阅读本书可获得的技能和知识**

|  |  |  |
| --- | --- | --- |
| **目标** | **艺术与人文背景** | **科学与工程背景** |
| 语言分析 | 操控大型语料库，设计语言模型， 验证由经验得出的假设 | 使用数据建模、数据挖掘和知识发掘的技术来 分析自然语言 |
| 语言技术 | 应用NLP技术构建高效的系统来 处理语言学任务 | 在高效的语言处理软件中使用语言学算法和 数据结构 |

**篇章结构**

本书前几章按照概念的难易程度编排。先是实用地介绍语言处理，讲述如何使用小 的 Python 程序分析感兴趣的文本信息(第1～3章)。接着是结构化程序设计章节 (第4章),用来巩固散布在前面几章中学习的编程要点。之后，加快速度，我们用 一系列章节讲述语言处理的基本原理：标注、分类和信息提取(第5～7章)。接下 来 的 3 章探索了句子解析、句法结构识别和句意表达方法构建(第8～10章)。

最后一章重点讲述了如何有效管理语言数据(第11章)。本书结尾处的后记简要讨 论了NLP 领域的过去和未来。

每一章中我们都在两种不同的叙述风格间切换。 一种风格是以自然语言为主线。我 们分析语言，探索语言学概念，在讨论中使用编程的例子。我们经常会使用尚未系 统介绍的 Python 结构，这样你可以在钻研这些程序如何运作的细节之前了解它们 的用处。就像学习一门外语的惯用表达一样，你能够买到好吃的糕点而不必先学会 复杂的提问句型。另一种风格是以程序设计语言为主线。我们分析程序、探索算 法，而不以语言学例子为主。

每章结尾都有一系列分级练习，用于巩固所学的知识。练习按照如下的标准分 级： O 初级练习，对范例代码稍加修改等简单的练习；①中级练习，深入探索 所学知识的某个方面，需要仔细地分析和设计； ●高级练习，开放式的任务， 挑战你对所学知识的理解并要求你独立思考解决的方案(新学编程的读者可以 跳过这些)。

每一章都有深入阅读环节和放置在 <http://www.nltk.org/>网站上的“额外内容”部分， 用来介绍更深入的相关材料及一些网络资源。所有实例代码都可从网上下载。

**为什么使用** **Python?**

Python 是一种简单但功能强大的编程语言，其自带的函数非常适合处理语言数据。 Python 可以从<http://www.python.org/>免费下载，并能够在各种平台上安装运行。

下面的4行Python 程序就可以操作file.txt文件，输出所有后缀是 “ing” 的词。

>>>for line in open("file.txt"):

...for word in line.split():

...if word.endswith('ing'):

...print word

这段程序演示了 Python 的一些主要特征。第一，使用空白符号缩进代码，从而使 if 后面的代码都在前面一行 for 语句的范围之内；这能保证对每个单词都能进行 “ing” 结尾检测。第二，Python 是面向对象语言。每一个变量都是包含特定属性 和方法的实例。例如：变量 “line” 的值不仅仅是一行字符串，它是一个 string 对 象，包含用来把字符串分割成词的split()方法(或叫操作、函数)。我们在对象名 称后添加句号(点)和方法名称就可以调用对象的一个方法，即 line.split)。第三，

方法的参数写在括号内。例如：上面例子中的 word.endswith(ing), 参 数 “ing” 表示我们需要找的是以“ing” 结尾的词而不是别的结尾的词。最后也是最重要的， Python 的可读性非常强可以很容易地猜出程序的功能，即使你以前从未写过一行 代码。

选择Python 是因为它的学习曲线比较平缓，文法和语义都很易懂，具有强大的字符 串处理功能。作为解释性语言，Python 便于交互式编程。作为面向对象语言， Python 允许数据和方法被方便地封装和重用。作为动态语言， Python 允许属性在等到程序 运行时添加到对象，允许变量自动类型转换，提高开发效率。Python 自带强大的标 准库，包括图形编程、数值处理和网络连接等组件。

Python 在世界各地的工业、科研、教育领域应用广泛，因其提高了软件的生产效率、 质量和可维护性而备受欢迎。<http://www.python.org/about/success/>列举了许多成功使 用Python 的例子。

NLTK 定义了使用Python 进行NLP 编程的基础工具。它提供了与自然语言处理相关 的数据表示基本类，词性标注、文法分析、文本分类等任务的标准接口及这些任务 的标准实现，可以组合起来解决复杂的问题。

NLTK 自带大量文档。作为本书的补充， <http://www.nltk.org/>网站提供的 API 文档涵 盖了工具包中每一个模块、类和函数，详细说明了各种参数，以及用法示例。该网 站还为广大用户、开发人员和导师提供了很多包含大量的例子和测试用例的

HOWTO。

**软件安装需求**

为了充分利用好本书，你应该安装一些免费的软件包。 <http://www.nltk.org/> 上有这

些软件包当前的下载链接和安装说明。

*Python*

本书的例子都假定你正在使用 Python 2.4 或2.5版本。 一旦 NLTK 依赖的库支持

Python 3.0, 我们将把 NLTK 移植到 Python 3.0。

*NLTK*

本书的代码示例使用NLTK 2.0版本。NLTK 的后续版本是兼容的。

*NLTK-Data*

包含本书中所分析和处理的语言语料库。

*NumPy* *(推荐)*

这是一个科学计算库，支持多维数组和线性代数，在某些概率计算、标记、聚类和

分类任务中会用到。

*Matplotlib (推荐)*

这是一个用于数据可视化的2D 绘图库，在产生线图和条形图的程序例子中会用到。

*NetworkX (可选)*

这是一个用于存储和操作由节点和边组成的网络结构的函数库。实现可视化语义网

络还需要安装 Graphviz 库。

*Prover9 (可选)*

这是一个使用一阶等式逻辑的定理自动证明器，用于支持语言处理中的推理。

**自然语言工具包** **(NLTK)**

NLTK创建于2001年，最初是宾州大学计算机与信息科学系计算语言学课程的一部分。 从那以后，在数十名贡献者的帮助下不断发展壮大。如今，它已被几十所大学的课程所 采纳，并作为许多研究项目的基础。表P-2 列出了NLTK 的一些最重要的模块。



**表** **P-2** **语言处理任务与相应** **NLTK** **模块及功能描述**

|  |  |  |
| --- | --- | --- |
| **语言处理任务** | **NLTK模块** | **功能描述** |
| 获取语料库 | nltk.corpus | 语料库和词典的标准化接口 |
| 字符串处理 | nltk.tokenize,nltk.stem | 分词、句子分解、提取主干 |
| 搭配探究 | nltk.collocations | t-检验、卡方、点互信息 |
| 词性标识符 | nltk.tag | n-gram、backoff、Brill、HMM、TnT |
| 分类 | nltk.classify,nltk.cluster | 决策树、最大熵、朴素贝叶斯、EM、 k-means |
| 分块 | nltk.chunk | 正则表达式、n-gram、命名实体 |
| 解析 | nltk.parse | 图表、基于特征、 一致性、概率性、依 赖项 |
| 语义解释 | nltk.sem,nltk.inference | λ演算、 一阶逻辑、模型检验 |
| 指标评测 | nltk.metrics | 精度、召回率、协议系数 |
| 概率与估计 | nltk.probability | 频率分布、平滑概率分布 |
| 应用 | nltk.app,nltk.cha | 图形化的关键词排序、分析器、WordNet 查看器、聊天机器人 |
| 语言学领域的工作 | nltk.toolbox | 处理SIL工具箱格式的数据 |

NLTK 设计中的4个主要目标如下。

*简易性*

提供直观的框架和大量模块，使用户获取 NLP 知识而不必陷入像标注语言数据那 样繁琐的事务中。

*一致性*

提供具有一致的接口和数据结构并且方法名称容易被猜到的统一框架。

*可扩展性*

提供一种结构使得新的软件模块可以方便添加进来，模块包括同一任务中不同的或 相互冲突的实现方式。

*模块化*

提供可以独立使用而与工具包的其他部分无关的组件。

前言 7

对比上述目标，我们回避了工具包的潜在实用性。首先，虽然工具包提供了广泛的工 具，但它不是面面俱全的。第一，它是一个工具包而不是一个系统，它将会随着NLP 领域一起发展。第二，虽然这个工具包的效率足以支持实际的任务，但它运行时的性 能还没有高度优化。这种优化往往涉及更复杂的算法或使用C 或C++等较低一级的编 程语言来实现。这将使得工具包的可读性变差且更难以安装。第三，我们试图避开巧 妙的编程技巧，因为我们相信清楚直白的实现比巧妙却可读性差的方法好。

**对老师的话**

自然语言处理一般是在高年级本科生或研究生阶段开设的为期一个学期的课程。很 多教师都发现，在如此短的时间里涵盖理论和实践两个方面是十分困难的。有些课 程注重理论而排除掉实践练习，这剥夺了学生编写程序自动处理语言带来的挑战和 兴奋感。另一些课程仅仅教授语言学编程而不包含任何重要的 NLP 内容。最初开 发NLTK 就是为了解决这个问题，无论学生之前是否具有编程经验，都能使教师在 一个学期里同时教授大量理论和实践成为可能。

在所有NLP 教学大纲中算法和数据结构部分都十分重要。它们本身可能非常枯燥，而 NLTK 提供的交互式图形用户界面能让读者一步一步看到算法过程，使它们变得鲜活。 大多数NLTK 组件都有一个无需用户输入任何数据就能执行有趣任务的示范性例子。 学习本书的一种有效方法就是通过交互式重现书中的范例，把它们输入到Python 会话 控制台，观察它们的功能，尝试修改它们去探索经验性问题或者理论性问题。

本书包含了数百个练习，可作为学生作业。最简单的练习包括用指定的方式修改已 有的程序片段来回答具体的问题。另一方面，NLTK 为研究生水平的研究项目提供 了一个灵活的框架，包括所有的基本数据结构和算法的标准实现，几十个广泛使用 的数据集(语料库)的接口，以及一个灵活可扩展的体系结构。NLTK 网站上还有 其他资源可以支持NLTK 教学。

我们相信本书是唯一能为学生提供在学习编程的环境中学习NLP 的综合性教程。各个 章节和练习通过与NLTK 紧密结合，并将各章材料有序分割开，为学生(即使是那些 以前没有编程经验的学生)提供一个实用的NLP 的入门指南。学完这些材料后，学生 能准备好尝试一本更加深层次的教科书，例如： Speech and Language Processing(《语 音和语言处理》),作者是Jurafsky 和Martin(Prentice Hall 出版社，2008年)。

本书介绍编程概念的顺序与众不同。以一个重要的数据类型：字符串列表(链表)

开始，然后介绍重要的控制结构，如推导和条件式等。这些常用知识允许我们在一 开始就做一些有用的语言处理。当有了这样动机，我们再回过头来系统地介绍一些 基础概念，如字符串、循环、文件等。这种方法同更传统的方法相比，达到了同样 的效果而不必要求读者对编程感兴趣。

表 P-3 列出了两个课程计划表。第一个适用于艺术人文专业背景的读者，第二个适 用于科学与工程背景的读者。其他的课程计划应该涵盖前5章，然后把剩余的时间 投入单独的领域，例如：文本分类(第6、7章)、文法(第8、9章)、语义(第 10章)或者语言数据管理(第11章)。

**表P-3 课程计划建议(每一章近似的课时数)**

|  |  |  |
| --- | --- | --- |
| **章** | **艺术人文专业** | **理工科** |
| 第1章 语言处理与Python | 2～4 | 2 |
| 第2章 获得文本语料和词汇资源 | 2～4 | 2 |
| 第3章 处理原始文本 | 2～4 | 2 |
| 第4章 编写结构化程序 | 2～4 | 1～2 |
| 第5章 分类和标注词汇 | 2～4 | 2～4 |
| 第6章 学习分类文本 | 0～2 | 2～4 |
| 第7章 从文本提取信息 | 2 | 2～4 |
| 第8章 分析句子结构 | 2～4 | 2～4 |
| 第9章 建立基于特征的文法 | 2～4 | 1～4 |
| 第10章分析语句的含义 | 1～2 | 1～4 |
| 第11章 语言数据管理 | 1～2 | 1～4 |
| 总计 | 18～36 | 18～36 |

本书使用的约定

本书使用以下印刷约定。

**黑体**

表示新的术语。

*斜体*

用在段落中表示语言学例子、文本的名称和URL, 文件名和后缀名也用斜体。



等宽字体

用来表示程序清单，用在段落中表示变量、函数名、声明或关键字等程序元素。也 用来表示程序名。

*等宽斜体*

表示应由用户提供的值或上下文决定的值来代替文本中的值，也在程序代码例子中 用来表示元变量。

这个图标表示提示、建议或一般性的提醒。

这个图标表示警告。

**使用代码范例**

本书是为了帮你完成工作的。 一般情况下，你可以在你的程序或文档中使用本书中 的代码，而不需要得到我们的允许，当你需要大量地复制代码时除外。例如，编写 程序时用到书中的几段代码不需要许可。销售和分发包含 O'Reilly 书籍中例子的 CD-ROM 需要获得许可。援引本书和书中例子来回答问题不需要许可。大量地将 本书中的例子纳入你的产品文档将需要获得许可。

我们希望但不强求被参考文献引用。引用通常包括标题、作者、出版者和ISBN。例 如： “Natural Language Bocessing with Rthow,Steven Bird,Ewan Klein 和 Edward Loper。版权所有2009 Steven Bird,Ewan Klein 和 Edward Loper,978-0-596-51649-9。” 如果你觉得你使用本书的例子代码超出了上面列举的一般用途或许可，请通过 permissions@oreilly.

com 随时联系我们。

**Safari“在线丛书**

当你看到任何你喜爱的技术书的封面上印有 Safari°在线丛书的图标时，这意味着这 本书可以在OReilly 网络 Safari 书架上找到。

Safari 提供比电子书更好的解决方案。它是一个虚拟图书馆，你可以轻松搜索数以千 计的顶尖技术书籍，可剪切和粘贴例子代码，并下载一些章节，在你需要最准确最 新的信息时快速找到答案。欢迎免费试用 http://my safaribooksonline.com。

**如何联系我们**

关于本书的意见和咨询请写信给出版商。

OReilly Media 公司

1005 Gravenstein Highway North

Sebastopol,CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C 座807(100035)

奥莱利技术咨询(北京)有限公司

我们为本书的勘误表、例子等信息制作了一个网页。你可以访问这个页面：

[*http://www.oreilly.com/catalog/9780596516499*](http://www.oreilly.com/catalog/9780596516499)

作者通过NLTK 网站提供了各章的其他材料：

*htp:/rvww.nltk.org/*

要发表评论或询问有关这本书的技术问题，发送电子邮件至：

*bookquestions@oreilly.com*

欲了解更多有关我们的书籍、会议、资源中心和 O'Reilly 网络的信息，请参阅我们 的网站：

[*http://www.oreilly.com*](http://www.oreilly.com)

**致谢**

作者感激为本书早期手稿提供反馈意见的专家，他们是：DougAmold 、Michaela Atterer、 Greg Aumann、Kenneth Beesley、Steven Bethard、Ondrej Bojar、Chris Cieri、Robin Cooper、 Grev Corbett 、James Curran 、Dan Garrette 、Jean Mark Gawron 、Doug Hellmann 、Nitin Indurkhya 、Mark Liberman 、Peter Ljunglöf、Stefan Muller、Robin Munn 、Joel Nothman 、 Adam Przepiorkowski、Brandon Rhodes 、Stuart Robinson 、Jussi Salmela、Kyle Schlansker、 Rob Speer和 Richard Sproat。感谢学生和同事们，他们对课堂材料的宝贵意见演化成本

书的相关章节，其中包括巴西、印度和美国的 NLP 与语言学暑期学校的参加者。没有 NLTK 开发社区的成员的努力就不会产生这本书，他们为建设和壮大NLTK 无私奉献了 他们的时间和专业知识，他们的名字都记录在NLTK 网站上。

非常感谢美国国家科学基金会、语言数据联盟、Edward Clarence Dyason奖学金、 宾州大学、爱丁堡大学和墨尔本大学对本书相关工作的支持。

感谢 Julie Steele 、Abby Fox 、Loranah Dimant 及其他 O'Reilly 团队成员。他们组织 大量 NLP 和 Python 社区成员全面审阅我们的手稿，还主动为满足我们的需要而定 制 O'Reilly 的生成工具。感谢他们一丝不苟的审稿工作。

最后，深深地感谢我们的合伙人，他们是Kay 、Mimo 和 Jee 。感谢在我们写作本书 的几年里他们付出的关心、耐心和支持。我们希望我们的孩子——Andrew 、Alison、 Kirsten 、Leonie 和 Maaike—— 能从这些页面中感觉到我们对语言和计算的热情。

**版税**

这本书的版税将用来支持自然语言工具包的发展。

|  |
| --- |
|  |

图 P-1.Edward Loper、Ewan Klein 和Steven Bird,斯坦福大学，2007年7月

**作者简介**

**Steven Bird** 是墨尔本大学计算机科学和软件工程系副教授，宾夕法尼亚大学的语言数 据联盟高级副研究员。他于1990 年在英国爱丁堡大学完成计算音韵学博士，导师是 Ewan Klein。后来到喀麦隆开展夏季语言学研究所主持的Grassfields 班图语语言实地调 查。最近，他作为语言数据联盟副主任带领研发队伍花了几年时间，创建已标注文本 的大型数据库的模型和工具。在墨尔本大学，他建立了一个语言技术研究组，并在各 级本科计算机科学课程任教。2009年，史蒂芬成为计算语言学学会主席。

**Ewan Klein** 是英国爱丁堡大学信息学院语言技术教授。于1978年在剑桥大学完成形式 语义学博士学位。在苏塞克斯和纽卡斯尔大学工作多年后，开始在爱丁堡从事教学工 作。于1993 年他参与了爱丁堡语言科技集团的建立，并一直与之密切联系。从2000 年到2002年，他离开大学，在圣克拉拉的埃迪法公司的总部——爱丁堡的自然语言的 研究小组担任研发经理，负责处理口语对话。Ewan 是欧洲章计算语言学协会(European Chapter of the Association for Computational Linguistics) 前任主席，并且是人类语言技 术 (ELSNET) 欧洲卓越网络的创始成员和协调员。

**Edward Loper** 最近完成了宾夕法尼亚大学自然语言处理的机器学习博士学位。爱德华 是史蒂芬在2000年秋季计算语言学研究生课程的学生，也是教师助手和NLTK开发的

成员。除了 NLTK, 他帮助开发了用于记录和测试 Python 软件的两个包：epydoc 和

doctest。

**封面介绍**

本书封面上的动物是露脊鲸，所有大型鲸鱼中最稀有的。可以通过它约占身体总长 三分之一的巨大的头来识别。它生活在两个半球的大洋表面温带和凉爽的海洋中。 露脊鲸的名字被认为得自捕鲸人，他们认为它“正是”要杀死并取油的鲸鱼。虽 然自从20世纪30年代以来它就被保护，但露脊鲸仍然濒危物种。

大而笨重的露脊鲸通过其头部的老茧很容易区别于其他鲸鱼。它有一个广阔无背鳍的背 部和从眼睛上面开始的长拱嘴。它的身体是黑色的，除了肚皮上的白色补丁。伤口和疤 痕可能会呈现明亮的橙色，往往会被鲸虱子或 cyamids 感染。老茧——也在气孔附 近出现，眼睛、下巴、上唇的上面——呈黑色或灰色。它有大的、形状如桨的鳍状 肢和独特的 V 形吹气孔，由广泛分布在其头部的顶端的气孔发出，水柱可以上升 到海洋表面16英尺以上。

露脊鲸以浮游生物包括像磷虾虾和桡足类为食。作为须鲸，它们有一串从每一侧上 颌骨悬挂下来的225～250个边缘折叠板，那里应该是牙齿。呈黑色，长有7.2英 尺。露脊鲸是海中食草动物，经常张着自己的嘴巴慢慢地游。水流进嘴里再通过须， 猎物被困在舌头附近。

因为雌性要到10岁才能达到性成熟，在一年之久的妊娠后才能生下小鲸，所以露 脊鲸数量增长缓慢。小露脊鲸一般跟随母亲一年。

露脊鲸遍布世界各地，但数量很少。 一只露脊鲸通常单独或组成1～3 只的小团体 活动，但求偶时，它们可能形成30只的大队伍。与大多数须鲸一样，它们随季节 性迁徙。在寒冷水域栖息觅食，然后迁移到温暖的水域繁殖和生产。虽然它们在哺 乳季节可能游到远海，但露脊鲸会在沿海地区进行繁殖。有趣的是，许多雌性不会 每年都回到这些沿海繁殖区，而是只在繁殖年来到该地区。其他年份它们去哪里了 这仍是一个谜。

露脊鲸唯一的天敌是逆戟鲸和人类。当危机出现时， 一组露脊鲸可能在一起围成一

个圈，用尾巴指向外面，以阻止捕食。这种防御并非总是成功的，小鲸偶尔会与它 们的母亲分开而被杀害。

露脊鲸是游泳最慢的鲸之一，尽管它们短期爆发可能达到10英里每小时。它们可 以下潜到至少1000英尺，并潜水长达40分钟。即使经过多年的保护状态，露脊鲸 仍然极度濒危。只在过去的15年中有证据表明它们在南半球的数量有所恢复。仍 然不知道露脊鲸是否会在北半球存活。虽然目前没有捕杀，当前保护的问题包括船 舶碰撞、捕鱼活动的冲突、栖息地的破坏、石油钻探和与其他鲸类可能的竞争冲突。 露脊鲸没有牙齿，所以在某些情况下，耳骨和眼球晶体可以用来估计露脊鲸在死亡 时的年龄。相信露脊鲸至少能活50年，但是有关它们寿命的数据很少。

封面图片来自多佛尔画报档案。

**目录**

**第** **1** **章** **语言处理与** **Python** [1](#bookmark4)

1.1 语言计算：文本和词汇 [1](#bookmark5)

1.2 近观 Python:将文本当做词链表 [10](#bookmark6)

1.3 计算语言：简单的统计 [17](#bookmark7)

1.4 回到 Python:决策与控制 [24](#bookmark8)

1.5 自动理解自然语言 [29](#bookmark9)

1.6 小结 [35](#bookmark10)

1.7 深入阅读 [36](#bookmark11)

1.8 练 习 [37](#bookmark12)

**第** **2** **章** **获得文本语料和词汇资源** [41](#bookmark13)

2.1 获取文本语料库 [41](#bookmark14)

2.2 条件频率分布 [55](#bookmark15)

2.3 更多关于 Python:代码重用 [60](#bookmark16)

2.4 词典资源 [63](#bookmark17)

2.5 WordNet [72](#bookmark18)

2.6 小结 [78](#bookmark19)

2.7 深入阅读 [79](#bookmark20)

2.8 练 习 [80](#bookmark21)

**第** **3** **章** **处理原始文本** [84](#bookmark22)

3.1 从网络和硬盘访问文本 [84](#bookmark23)

3.2 字符串：最底层的文本处理 [93](#bookmark24)

3.3 使用Unicode 进行文字处理 [100](#bookmark25)

3.4 使用正则表达式检测词组搭配 [105](#bookmark26)

3.5 正则表达式的有益应用 [109](#bookmark27)

3.6 规范化文本 [115](#bookmark28)

3.7 用正则表达式为文本分词 [118](#bookmark29)

3.8 分割 [121](#bookmark30)

3.9 格式化：从链表到字符串 [126](#bookmark31)

3.10 小结 [132](#bookmark32)

3.11 深入阅读 [133](#bookmark33)

3.12 练 习 [134](#bookmark34)

**第** **4** **章** **编写结构化程序** [142](#bookmark35)

4.1 回到基础 [142](#bookmark36)

4.2 序列 [147](#bookmark37)

4.3 风格的问题 [152](#bookmark38)

4.4 函数：结构化编程的基础 [156](#bookmark39)

4.5 更多关于函数 [164](#bookmark40)

4.6 程序开发 [169](#bookmark41)

4.7 算法设计 [175](#bookmark42)

4.8 Python 库的样例 [183](#bookmark43)

4.9 小结 [188](#bookmark44)

4.10 深入阅读 [189](#bookmark45)

4.11 练 习 [189](#bookmark46)

**第** **5** **章** **分类和标注词汇** [195](#bookmark47)

5.1 使用词性标注器 [195](#bookmark48)

5.2 标注语料库 [197](#bookmark49)

5.3 使用 Python 字典映射词及其属性 [206](#bookmark50)

5.4 自动标注 [216](#bookmark51)

5.5 N-gram 标注 [221](#bookmark52)

5.6 基于转换的标注 [228](#bookmark53)

5.7 如何确定一个词的分类 [230](#bookmark54)

5.8 小结 [233](#bookmark55)

5.9 深入阅读 [234](#bookmark56)

5.10 练 习 [235](#bookmark57)

**第** **6** **章** **学习分类文本** [241](#bookmark58)

6.1 监督式分类 [241](#bookmark59)

6.2 监督式分类的举例 [254](#bookmark60)

6.3 评估 [258](#bookmark61)

6.4 决策树 [263](#bookmark62)

6.5 朴素贝叶斯分类器 [266](#bookmark63)

6.6 最大熵分类器 [271](#bookmark64)

6.7 为语言模式建模 [275](#bookmark65)

6.8 小 结 [276](#bookmark66)

6.9 深入阅读 [277](#bookmark67)

6.10 练 习 [278](#bookmark68)

**第** **7** **章** **从文本提取信息** [281](#bookmark69)

7.1 信息提取 [281](#bookmark70)

7.2 分块 [284](#bookmark71)

7.3 开发和评估分块器 [291](#bookmark72)

7.4 语言结构中的递归 [299](#bookmark73)

7.5 命名实体识别 [302](#bookmark74)

7.6 关系抽取 [306](#bookmark75)

7.7 小 结 [307](#bookmark76)

7.8 深入阅读 [308](#bookmark77)

7.9 练 习 [308](#bookmark78)

**第** **8** **章** **分析句子结构** [312](#bookmark79)

8.1 一些语法困境 [312](#bookmark80)

8.2 文法的用途 [316](#bookmark81)

8.3 上下文无关文法 [319](#bookmark82)

8.4 上下文无关文法分析 [323](#bookmark83)

8.5 依存关系和依存文法 [332](#bookmark84)

8.6 文法开发 [336](#bookmark85)

8.7 小结 [343](#bookmark86)

8.8 深入阅读 [344](#bookmark87)

8.9 练 习 [344](#bookmark88)

**第** **9** **章** **建立基于特征的文法** [349](#bookmark89)

9.1 文法特征 [349](#bookmark90)

9.2 处理特征结构 [359](#bookmark91)

9.3 扩展基于特征的文法 [367](#bookmark92)

9.4 小结 [379](#bookmark93)

9.5 深入阅读 [380](#bookmark94)

9.6 练习 [381](#bookmark95)

**第10** **章** **分析语句的含义** [384](#bookmark96)

10.1 自然语言理解 [384](#bookmark97)

10.2 命题逻辑 [391](#bookmark98)

10.3 一阶逻辑 [395](#bookmark99)

10.4 英语语句的语义 [409](#bookmark100)

10.5 段落语义层 [422](#bookmark101)

10.6 小结 [428](#bookmark102)

10.7 深入阅读 [429](#bookmark103)

10.8 练习 [430](#bookmark104)

**第** **11** **章** **语言数据管理** [434](#bookmark105)

11.1 语料库结构：案例研究 [434](#bookmark106)

11.2 语料库生命周期 [439](#bookmark107)

11.3 数据采集 [443](#bookmark108)

11.4 使用XML [452](#bookmark109)

11.5 使用 Toolbox 数据 [459](#bookmark110)

11.6 使用OLAC 元数据描述语言资源 [463](#bookmark111)

11.7 小结 [466](#bookmark112)

11.8 深入阅读 [466](#bookmark113)

11.9 练 习 [467](#bookmark114)

**后记** [470](#bookmark115)

**参考文献** [476](#bookmark116)

**第1章**



**语言处理与** **Python**

我们能够很容易地得到数百万数量级的文本。假设我们会写一些简單的程序，那可 以用它来做些什么?本章将解决以下几个问题。

(1)通过将技术性较简單的程序与大规模文本结合起来，我们能实现什么?

(2)如何自动地提取出关键字和词组，用来总结文本的风格和内容?

(3)Python 编程语言为上述工作提供了哪些工具和技术?

(4)自然语言处理中有哪些有趣的挑战呢?

本章分为风格完全不同的两部分。在1.1节，我们将进行一些与语言相关的编程练 习而不去解释它们是如何实现的。在1.2节，我们将系统地回顾关键的编程概念。 我们使用章节标题来区分这两种风格，而后面几章则不像前面一样，是将两种风格 混合在一起，不作明显的区分。我们希望这种介绍风格能使你对将要出现的内容有 一个真实的体会，与此同时，介绍中还涵盖了语言学与计算机科学的基本概念。如 果你对这两个方面已经有了基本的了解，可以直接从1.5节开始学习。我们将在后 续的章节中重复所有要点，如果错过了什么,你可以很容易地在<http://www.nltk.org/> 上查询在线参考材料。如果这些材料对你而言是全新的，那么本章所提出的问题比 它还要多，这些问题将在本书的其余部分中进行讨论。

**1.1** **语言计算：文本和词汇**

我们都对文本非常熟悉，因为我们每天都在进行阅读和写作。在本书中，把文本视



为编写程序的原始数据，并通过很多有趣的编程方式来处理和分析文本。但在能写 这些程序之前，必须得从了解 Python 解释器开始。

**Python** **入门**

Python 与用户友好交互的方式之一包括你可以在交互式解释器直接输入代码—— 解释器将运行你的 Python 代码的程序。你可以通过一个叫做交互式开发环境 (Interactive Development Environment,IDLE) 的简单图形接口来访问Python 解释 器。在 Mac 上，你可以在 “Applications→MacPython” 中找到；在 Windows 中，

你可以在“程序 →Python” 中找到。在UNIX 下，你可以在 shell 输 入 “idle” 来运 行 Python (如果没有安装，尝试输入 **python)** 。解释器将会输入关于你的 Python 的版本简介，请检查你是否运行在 Python 2.4或2.5(这里是2.5.1)。

Python 2.5.1(r251:54863,Apr 152008,22:57:26)

[GCC 4.0.1(Apple Inc.build 5465)]on darwin

Type "help","copyright","credits"or "license"for more information.

>>>

如果你无法运行 Python 解释器，可能是由于没有正确安装 Python 。请 访问 <http://python.org/> 查阅详细操作说明。

提示符>>>表示 Python 解释器正在等待输入。复制这本书的例子时，自己不要键入 >>>。现在，把 Python 当作计算器使用。

>>>1+5\*2

-3

8

>>>

一旦解释器完成计算并显示出答案，提示符就会重新出现。这表示 Python 解释器 在等待另一个指令。

**轮到你来**

输入几个自己的表达式。可以使用星号(\*)表示乘法，左斜线表示除法， 可以用括号括起表达式。请注意：除法并不总是像你期望的那样。当输 入1/3时，是整数除法(小数会被四舍五入),当输入1.0/3.0 时，是“浮 点数”(或十进制)除法。要想获得通常平时期望的除法(在 Python3.0 中的标准),需要输入： from\_\_future\_import division。

前面的例子展示了如何使用 Python 交互式解释器，体验 Python 语言中各种表达 式，看看它们能做些什么。现在让我们尝试一个无意义的表达式，看看解释器将 如何处理。

>>>1 +

File "<stdin>",line 1

1 +

SyntaxError:invalid syntax

>>>

结果产生了一**个语法错误。**在 Python 中，指令以加号结尾是没有意义的。Python 解释器会指出发生错误的行(<stdin> 的 第 1 行 ，<stdin> 表示“标准输入”)。

现在我们学会使用 Python 解释器了，已经准备好开始处理语言数据了。

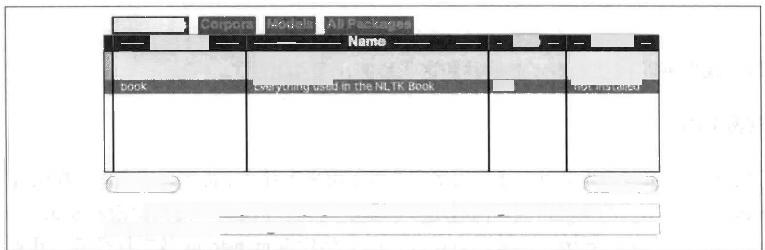
**NLTK** **入门**

首先应该安装 NLTK, 可以从 <http://www.nltk.org/> 上免费下载。按照说明下载适合 你的操作系统的版本。

一旦安装完成，便可像前面那样启动Python 解释器。在 Python 提示符后面输入下 面两行命令来安装本书所需的数据，然后选择 book, 如图1-1所示。

>>>import nltk

>>>nltk.download()



Colleetlier

all

all-corpora

Download Refresh

Server Index:<http://nltk.googlecode.com/avn/trunk/nltk>\_data/index.xml Download Directory:C:\nltk\_data

Smtus

not installed

not mstalled

packages the comore



n/a

n/a n/a

Al

Al

Sze

**图1-1** **下载** **NLTK图书合集：使用** **nltk.download()浏览可用的软件包。下载器上的Collections 选项卡显示软件包如何被打包分组，然后选择** **book 标记所在行，可以获取本书所有例子和练习** **需要的全部数据。这些数据需要100MB** **硬盘空间，包括约30个压缩文件。完整的数据集(即** **下载器中的**所有)大约是这个大小的5倍(在本书写作期间),并且还在不断扩充

一旦数据被下载到你的机器，你就可以使用 Python 解释器加载其中的一些了。第 一步是在 Python 提示符后输入一个特殊的命令，告诉解释器去加载一些我们要用 的文本： from nltk.book import\*。这条语句是说“从NLTK 的 book 模块中加载所 有的条目”。book 模块包含你阅读本章时所需的所有数据。在输出欢迎信息之后， 将会加载一些书的文本(这将需要几秒钟)。下面就是你需要输入的命令及输出的 结果，注意拼写和标点符号的正确性，记住不要输入>>>。

>>>from nltk.book import \*

\*\*\*Introductory Examples for the NLTK Book \*\*\*

Loading text1,... text9 and sentl, ..,sent9

Type the name of the text or sentence to view it.

Type:'texts()'or 'sents()'to list the materials.

text2:Sense and Sensibility by Jane Austen 1811

text1:Moby Dick by Herman Melville 1851

text3:The Book of Genesis

text4:Inaugural Address Corpus

text5:Chat Corpus

text6:Monty Python and the Holy Grail

text7:Wall Street Journal

text8:Personals Corpus

text9:The Man Who Was Thursday by G .K.Chesterton 1908

>>>

无论什么时候想要找到这些文本，只需要在**Python**提示符后输入它们的名字即可。

>>>text1

<Text:Moby Dick by Herman Melville 1851>

>>>text2

<Text:Sense and Sensibility by Jane Austen 1811>

>>>

现在我们可以使用 Python解释器和这些数据，准备开始了。

**搜索文本**

除了简单地阅读文本之外，还有很多方法可以用来查看文本内容。词语索引视图可 以显示指定单词的出现情况，同时还可以显示一些上下文。下面我们输入 text1 并 在后面跟一个点，再输入函数名 concordance, 然后将monstrous 放在括号里，用来 查找《白鲸记》中的词 monstrous。

>>>text1.concordance("monstrous")

Building index...



Displaying 11 of 11 matches:

ong the former,one was of a most monstrous size ....This came towards us, ON OF THE PSALMS ."Touching that monstrous bulk of the whale or ork we have r 11 over with a heathenish array of monstrous clubs and spears .Some were thick d as you gazed ,and wondered what monstrous cannibal and savage could ever hav that has survived the flood;most monstrous and most mountainous !That Himm they might scout at Moby Dick as a monstrous fable ,or still worse and more de h of Radney.'"CHAPTER 55 Of the monstrous Pictures of Whales .I shall ere 1 ing Scenes .In connexion with the monstrous pictures of whales ,I am strongly ere to enter upon those still more monstrous stories of them which are to be fo ght have been rummaged out of this monstrous cabinet there is no telling .But of Whale -Bones;for Whales of a monstrous size are oftentimes cast up dead u

>>>

**轮到你来**

尝试搜索其他词。为了方便重复输入，你也许会用到上箭头、Ctrl-↑ 或者 Alt-p 以获取之前输入的命令，然后修改要搜索的词。你也可以尝试搜索 已经列入的其他文本。例如：使用text2.concordance("affection")搜索《理 智与情感》中的词 affection; 使用 text3.concordance("lived")搜索《创 世纪》找出某人活了多久；你也可以看看 text4,《就职演说语料》,回 到1789年去看看那时英语使用的例子，并且搜索如 nation,terror,god 这样的词，看看随着时间的推移这些词的使用有何不同；同样还有 text5,《NPS 聊天语料库》,你可以在里面搜索一些网络用语，如 im,ur, lol。(注意这个语料库是未经审查的!)

通过一段时间对这些文本的研究，我们希望你能对语言的丰富性和多样性有一个新的 认识。在下一章中，你将学习如何获取更广泛的文本，包括英语以外其他语言的文本。

关键词索引让我们可以看到上下文中的词。例如：我们看到 monstrous 出现在文章中， 如 the *pictures 和the* size。还有哪些词出现在相似的上下文中?我们可以通过 在被查询的文本名后添加函数名similar, 然后在括号中插入相关词的方法来查找到。

>>>text1.similar("monstrous")

Building word-context index...

subtly impalpable pitiable curious imperial perilous trustworthy

abundant untoward singular lamentable few maddens horrible loving lazy mystifying christian exasperate puzzled

>>>text2.similar("monstrous")

Building word-context index...

very exceedingly so heartily a great good amazingly as sweet remarkably extremely vast

>>>



可以发现从不同的文本中能够得到不同的结果。Austen (奥斯丁，英国女小说家) 在词汇的使用上与 Melville 完全不同。对于她来说，monstrous 是正面的意思，有 时它的功能像词very一样用作强调成分。

我们可以使用函数 common contexts 研究共用两个或两个以上词汇的上下文，如

monstrous和 very。使用方括号和圆括号将这些词括起来，中间用逗号分割。

>>>text2.common\_contexts(["monstrous","very"])

be\_glad am\_glad a\_pretty is\_pretty a\_lucky

>>>>

**轮到你来：**

挑选另一对词，使用 similar()和 common\_contexts()函数比较它们在两 个不同文本中的用法。

自动检测出现在文本中的特定的词，并显示同一上下文中出现的其他词。我们也可以 判断词在文本中的位置：从文本开头算起有多少词出现。这个位置信息可以用**离散图** 表示。每一列代表一个单词，每一行代表整个文本。在图1-2中，我们看到在过去220 年中的一些显著的词语用法模式(在一个由就职演说语料首尾相连组合的人工文本 中)。可以利用下面的方法画出离散图。你也许会想尝试更多的单词(如： liberty, constitution) 和不同的文本。你能在看到这幅图之前预测一个单词的分布吗?如前所 述，保证引号、逗号、中括号及小括号的使用完全正确。

>>>text4.dispersion\_plot(["citizens","democracy","freedom","duties", "America"])

>>>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lexical Dispersion Plot | | | | |
| demotracy  frsdom  duties  Ammme  20000 | 40000 | 1 1 1161! | | |
| 60000 | B0000  Ward Oset | 100000 120000 140000 160000 |

**图1-2** **美国总统就职演说词汇分布图：可以用来研究随时间推移语言使用上的变化**



**重要事项：**

为了画出本书中用到的图形，你需要安装Python 的NumPy 和Matplotlib程 序包。请参阅<http://www.nltk.org/>上的安装说明。

现在轻松一下，尝试以上述不同风格产生一些随机文本。要做到这一点，我们 需要输入后面跟着函数名 generate 的文本名称。(需要带括号，但括号里什么 也没有。)

>>>text3.generate()

In the beginning of his brother is a hairy man ,whose top may reach

unto heaven ;and ye shall sow the land of Egypt there was no bread in

all that he was taken out of the month ,upon the earth .So shall thy

wages be ?And they made their father ;and Isaac was old ,and kissed

him:and Laban with his cattle in the midst of the hands of Esau thy

first born ,and Phichol the chief butler unto his son Isaac,she

>>>

请注意，第一次运行此命令时，由于要搜集单词序列的统计信息，因而执行速度比 较慢。每次运行后，输出的文本都会不同。现在尝试产生就职演说风格或互联网聊 天室风格的随机文本。虽然文本是随机的，但它重复使用了源文本中常见的单词和 短语，从而使我们能感觉到它的风格和内容。

在 generate 产生输出时，标点符号与前面的单词分开。虽然这不是正 确的英文格式，但我们之所以这么做是为了确保文字和标点符号是彼 此独立的。更多关于这方面的内容将在第3章学习。

**计数词汇**

在前面例子中出现的文本中，最明显的地方在于它们所使用的词汇不同。在本节中， 我们将看到如何使用计算机并以各种有用的方式来计数词汇。像以前一样，你可以 马上开始使用 Python 解释器进行试验，即使你可能还没有系统地研究过 Python 。 修改这些例子并测试一下你对它们的理解程度，尝试一下本章结尾的练习题。

首先，让以文本中出现的单词和标点符号为单位算出文本从头到尾的长度。使用函 数 len 获取长度，参考《创世纪》中使用的例子。

>>>len(text3)

44764

>>>

《创世纪》有44764 个单词和标点符号，也被称作“标识符”。**标识符**是表示一 组字符序列——如： hairy 、his 或者：)——的术语。当计算文本中标识符的个数 时，如 to be or not to be 这句话，我们计算的是这些序列出现的次数。因此，例 句中出现了to 和 be 各两次，or 和 not 各一次。然而在例句中只有4个不同的单 词。《创世纪》中有多少不同的单词?如果要用 Python 来回答这个问题，就不 得不稍微改变一下提出的问题。因为一个文本词汇表只是它用到的标识符的集 合，在集合中所有重复的元素都只算一个。在 Python 中可以使用命令：set(text3) 来获得 text3 的词汇表。这样做之后，屏幕上的很多词就会被掠过。现在尝试以 下操作。

>>>sorted(set(text3))①

['!',"'", '(',')', ',',',]','.', '.]', ':', ';', ';]', '?','?]',

'A','Abel','Abelmizraim','Abidah','Abide','Abimael','Abimelech',

,c②cad','Achbor','Adah',...]

2789

>>>

用 sorted) 包裹 Python 表达式 set(text3)°, 得到一个词汇条目的排序表，这个表以 各种标点符号开始，然后接着是以A开头的词汇。大写单词排在小写单词前面。通 过求集合中项目的个数，可以间接地获得词汇表的大小。再次使用命令len 来获得 这个数值”。尽管书中有44764个标识符，但只有2789个不同的词汇或“词类型”。 **词类型**是指一个词在一个文本中独一无二的出现或拼写形式。也就是说，这个单词 在词汇表中是唯一的。计数的2789个项目中包括标点符号，所以把它们称作唯一 项目**类型**而不是词类型。

现在，开始对文本词汇丰富度进行测量。下面的例子展示的结果含义为每个词平均 被使用了16次(应该确保Python 使用的是浮点除法)。

>>>from \_future\_import division

>>>len(text3)/len(set(text3))

16.05 0197203298673

>>>

接下来，专注于特定的词。计数一个单词在文本中出现的次数，再计算一个特定词 在文本中占据的百分比。



>>>text3.count("smote")

5

>>>100 \*text4.count('a') /len(text4)

1.46 43016433938312

>>>

**轮到你来：**

text5中 lol 出现了多少次?它占文本全部词数的百分比是多少?

也许你想要对几个文本重复进行这些计算，但重新输入公式是很乏味的。方法是可 以自己命名一个任务，如“lexical\_diversity”或“percentage”,然后用一个代码 块关联它。这样，你只需输入一个很短的名字就可以代替一行或多行 Python 代 码，而且想用多少次就用多少次。执行一个任务的代码段叫做函数。使用关键 字 def 给函数定义一个简短的名字。下面的例子演示的是如何定义两个新的函数， lexical\_diversity()和 percentage()。

>>>def lexical\_diversity(text):①

· · return len(text)/len(set(text))②

>>>def percentage(count,total):③

return 100 \*count /total

**注意!**

当遇到第一行末尾的冒号时， Python 解释器提示符由>>>变 为…。…提示符表示 Python 期望的是在后面出现一个**缩进代码块。** 缩进由你决定只需输入4个空格或是敲击 Tab键。要结束一个缩进 代码段，只需输入一个空行。

在 lexical\_diversityO°的定义中，指定了一个 text **参数。**这个参数是计算文本词汇多 样性时的一个“占位符”,并在使用函数时，重现在运行的代码段中②。类似地， percentage)定义了两个参数： count 和total。

只要 Python 知道了 lexical\_diversityO和 percentage()是指定代码段的名字，我们就 可以继续使用这些函数了。

>>>lexical\_diversity(text3)

16.05 0197203298673

>>>lexical\_diversity(text5)

7.42 00461589185629

>>>percentage(4,5)

80.0

>>>percentage(text4.count('a'),len(text4))

1.46 43016433938312

>>>

简要重述一下，使用或者说是**调用**一个如lexical\_diversityO这样的函数时，只要输入它 的名字并在后面跟一个左括号，再输入文本名字，然后是右括号即可。这些括号经常出 现，它们的作用是将任务名——如：lexical\_diversityO——与任务将要处理的数据—— 如：text3 分割开。调用函数时放在参数位置的数据值叫做函数的实参。

在本章中，你已经遇到了一些函数，如： len(),set() 和 sorted() 。通常会在函数 名后面加一对空括号，例如 len(), 这只是为了表明这是一个函数而不是其他的 Python 表达式。函数是编程中的一个重要概念，我们只是在一开始提到了它们， 为了是让新手体会到编程的强大和它的创造力。如果你现在觉得有点混乱，请 不要担心。

稍后将学习如何使用函数列表显示数据，见表1-1。表中每一行包含了不同数据相 同的计算，将使用函数进行这种重复性的工作。

**表1-1** **Brown** **语料库中各种文体的词汇多样性**

|  |  |  |  |
| --- | --- | --- | --- |
| **体** **裁** | **标** **识** **符** | **类** **型** | **词汇多样性** |
| 技能和爱好 | 82345 | 11935 | 6.9 |
| **幽默** | 21695 | 5017 | 4.3 |
| 小说：科学 | 14470 | 3233 | 4.5 |
| 新闻：报告文学 | 100554 | 14394 | 7.0 |
| 小说：浪漫 | 70022 | 8452 | 8.3 |
| 宗教 | 39399 | 6373 | 6.2 |

**1.2** **近观** **Python:** **将文本当做词链表**

大家已经学习过Python 编程语言的一些重要元素。下面进行简单的系统复习。

**链表**

文本是什么?一方面，它是一页纸上的符号序列，就像这页纸一样。另一方面，它



是章节的序列，每一章由小节序列组成，这些小节由段落序列组成，以此类推。然 而，对于我们而言，认为文本不外乎是单词和标点符号的序列。下面是如何展示 Python 中《白鲸记》的开篇句。

>>>sent1

>>>

=['Call','me','Ishmael','.']

在提示符后面，输入自己命名的sent1, 后跟一个等号，然后是一些引用的词汇， 中间以逗号分割并用括号包围。方括号里的内容在 Python 中叫做**链表，** 是存储文 本的方式。可以通过输入名字°来查阅文本。同样可以查询文本的长度，甚至可 以在自己的函数 lexical\_diversityO中使用°。

>>>sent1①

>>>len(sent1)②

['Call','me','Ishmael','.']



1.0

>>>

定义一些链表，将每个文本开始的句子定义为 sent2…sent9 。下面只检查其中的两 个。你可以在Python 解释器中查看其余的(如果得到的是一个错误表达：sent2 没

**有定义，你需要先输入** **from nltk.book import\*)。**

>>>sent2

,elyt'''d,'l,',

>>>sent3

['In', 'the', 'beginning', 'God', 'created','the',

'heaven','and','the','earth','.']

>>>

**轮到你来：**

通过输入名字、等号和词链表，组建一些你自己想要的句子，如 ex1 = [Monty',Python',and,the',Holy',Grail] 。 重复使用一些1.1节中的其他 Python操作，如： sorted(ex1),len(set(ex1)),ex1.count(the)。

令人惊喜的是，可以对链表使用 Python 加法运算。两个链表相加°能够创造出一个 新的链表，包括第一个链表的全部，并附着第二个链表的全部。



>>>['Monty','Python']+['and','the','Holy','Grail']①

['Monty','Python','and','the','Holy','Grail']

这种加法操作的特殊用途叫做连接；它将多个链表组合为一个链表。

**可以通过把句子连接起来组成一个文本。**

不必逐字地输入链表，可以使用简短的名字来引用预先定义好的链表。

>>sent4 +sent1

['Fellow','-','Citizens','of','the','Senate','and','of','the',

'House','of','Representatives',':','Call','me','Ishmael','.']

>>>

如果想要在链表中增加一个单独的元素该如何做?这种操作叫做追加。当对一个链 表使用 append) 时，链表自身会随着操作而更新。

>>>sent1.append("Some")

>>>sent1

['Call','me','Ishmael','.','Some']

>>>

**索引列表**

正如已经看到的， Python 中的文本是一个词汇的链表，用括号和引号来表示。就像 处理一页普通的文本，我们可以使用 len(text1)来计算 text1 的全部词数，使用 text1.count(heaven')来计算一个文本中特定词出现的次数，如 heaven。

稍微耐心些，我们可以挑选出一篇文本中的第1个、第173个甚至第14278个词。 类似的，也可以通过在链表中出现的次序找出 Python 链表的元素。表示这个位置 的数字叫做这个元素的索引。在文本名称后面的方括号里写下索引， Python 就会 显示出文本中这个索引处——例如文本中第173个词。

>>>text4[173]

'awaken'

>>>

也可以反过来做；找出一个词第一次出现时的索引。



>>>text4.index('awaken')

173

>>>

索引是一种常见的用来获取文本中词汇的方式，或者，更通俗地讲，任何列表中的 元素。通过Python 也可以获取子链表，从大文本中任意抽取语言片段，术语叫做切片。

>>>text5[16715:16735]

'because','you','can','actually','play','a','full','game','without',

['U86','thats','why','something','like','gamefly','is','so','good',

'buying', 'it']

>>>text6[1600:1625]

['We',"'",'re','an','anarcho','-','syndicalist','commune','.','We',

'take','it','in','turns','to','act','as','a','sort','of','executive',

'officer','for','the','week']

>>>

索引还有一些微妙之处，我们将在下面的句子中体会这些。

>>>sent

 'word1'

>>>sent[9] 'word10'

=['word1','word2','word3','word4','word5',

'word6','word7','word8','word9','word10']

>>>

需要注意的是，索引从零开始：第0个元素写作sent[0], 其实是第1个词 “word1”; 而句子的第9个元素是“word10”。原因很简单： Python 从计算机内存中的链表获取内 容的时候，需要告诉它向前多少个元素。因此，向前0个元素使它留在第一个元素上。

这种从零算起的做法刚开始接触会有些混乱，但这是现代编程语言普 遍使用的。19XY是20世纪中的一年，如果你已经掌握了这样的计数 世纪的系统，或者如果你生活在一个建筑物楼层编号从1 开始的国 家，你很快就会掌握它的窍门，步行n-1 级楼梯能够到达第n 层。

现在，如果我们不小心使用的索引量过大就会产生错误。

>>>sent[10]

Traceback(most recent call last):

File "<stdin>", line 1,in ?

IndexError:list index out of range

>>>

这不是语法错误，因为程序片段在语法上是正确的。相反，它是一个**运行时错误，** 它会产生一个回溯消息显示错误的上下文，并标注错误的名称： IndexError, 以及 简要的解释说明。

再次使用构造的句子仔细看看切片，这里我们发现切片5:8包含索引5、6和7的句

子元素。

>>>sent[5:8]

['word6','word7','word8']

>>>sent[5]

'word6'

>>>sent[6]

*'*>*w**r*>*d*nt[7]

'word8'

>>>

按照惯例，m:n 表示元素 m…n-1。正如下一个例子所示，如果切片从链表第一个元 素开始，可以省略第一个数字；如果切片到链表最后一个元素处结尾，则可以省略 第二个数字：

>>>sent[:3]①

['wordl','word2','word3']

>>>text2[141525:]②

['among','the','merits','and','the','happiness','of','Elinor',

'and','Marianne',

',','let','it','not','be','ranked','as','the','least',

'considerable',',',

'sigh'though','sisters',',','and','living','almost','within',

'each','other',',','they','could','live','without','disagreement', 'between',

'themselves',',','or','producing','coolness','between','their',

'husbands','.',

'THE','END']

>>>

可以通过改变它的索引值来修改链表中的元素。在接下来的例子中，把 sent[0]放在 等号左侧°。也可以用新内容替换掉整个片段°。最后一个报错的原因是这个链表只 有4个元素而要获取4后面的元素，所以产生了错误③。

>>>sent[0]='First’①

>>>sent[9] ='Last'



>>>len(sent)

10

>>>sent[1:9]=['Second','Third']②

>>>sent

>>>sent[9]③

['First','Second','Third','Last']

Traceback(most recent call last):

File "<stdin>",line 1,in ?

IndexError:list index out of range

>>>

**轮到你来：**

定义你的句子，使用前文中的方法修改个别词和词组(切片)。尝试 本章结尾关于链表的练习，检验你是否真正理解。

**变量**

从1.1节开始，已经查看过名为 text1,text2 等的文本。像这样通过只输入简短 的名字来就能引用一本250000字的书的做法节省了很多打字时间。 一般情况 下，可以对任意的计算命名。在前面的小节中已经这样做了，如下所示，定义 一个变量 sent1。

>>>sent1=['Call','me','Ishmael','.']

>>>

语句形式是：变量=表达式。Python 通过计算右边的表达式把结果保存在变量中。 这个过程被称为**赋值。**它并不产生任何输出，但只能在新的一行输入变量的名字才 能够检查它的内容。等号可能会有些误解，因为信息是从右边流到左边的。你把它 想象成一个左箭头可能会有帮助。变量的名字可以是任何你喜欢的名字，如： my\_sent 、sentence 、xyzzy 等。变量必须以字母开头，可以包含数字和下划线。下 面是变量和赋值的一些例子。

>>>my\_sent=['Bravely','bold','Sir','Robin',',','rode',

Ca\_e =my\_sent[1:4]

>>>noun\_phrase

] =sorted(noun\_phrase)

>>> wOrDs

['Robin','Sir','bold']

>>>



请记住，排序表中大写字母出现在小写字母之前。

请注意，在前面的例子中，将my\_sent 的定义分成两行。Python表达 式可以被分割成多行，只要它出现在任何一种括号内。Python 使用.. 提示符表示期望更多的输入。在这些连续的行中有多少缩进都没有关 系，因为加入缩进通常会便于阅读。

最好是选择有意义的变量名，它能提醒你代码的含义，也能帮助别人读懂你的 Python 代 码 。Python 并不理解这些名称的意义。它只是盲目地服从你的指令，如 果你输入一些令人困惑的代码，例如： one='two '或者two=3, 它也不会反对。唯 一的限制是变量名不能是 Python 的保留字，如 def、if、not 或 import 。如果你使用 了保留字，Python 会产生语法错误。

>>>not = 'Camelot'

File "<stdin>",line 1

not ='Camelot'

SyntaxError:invalid syntax

>>>

我们经常使用变量来保存计算的中间步骤，尤其是在这样做能够使代码更容易被读 懂时。因此， len(set(text1)) 也可以写作如下形式。

>>>vocab =set(text1)

>>>vocab\_size =len(vocab)

>>>vocab\_size

19317

>>>

**注意!**

为Python变量选择名称(或标识符)时请注意。首先，应该以字母开始， 后面跟数字(0到9)或字母。因此， abc23 是正确的，而23abc 会导致语 法错误。名称是明确区分大小写的。这意味着myVar 和myvar是不同的变 量。变量名不能包含空格，但可以用下划线把单词分开，如 my\_var。注意 不要插入连字符来代替下划线：my-var不对，因为 Python会把-解释为减号。

**字符串**

一些用来访问链表元素的方法也可以用在单独的词或**字符串**上。例如可以把一个字

符串指定给一个变量°,索引一个字符串②,划分一个字符串●。

>>>name ='Monty’①

>>>name[0]②

'M'

>>>name[:4]③

'Mont'

>>>

还可以对字符串执行乘法和加法。

>>>name \*2

'MontyMonty'

>>>name

+'!'

'Monty!'

>>>

可以把词用链表连接起来组成单个字符串，或者把字符串分割成一个链表，如下面 所示。

>>>''.join(['Monty','Python'])

'Monty Python'

>>>'Monty Python'.split()

['Monty','Python']

>>>

我们将在第3章继续介绍字符串的内容。目前为止，已经学习了两个重要的基石一 链表和字符串——准备好重新进行语言分析了。

**1.3** **计算语言：简单的统计**

让我们重新开始探索利用计算资源处理大量文本的方法。在1.1 节已经讨论了如 何搜索文章中的词，如何汇编一个文本中的词汇，如何以相同的方式产生随机文 本等。

在本节中，我们重新考虑怎样使一个文本显得与众不同的问题，并使用程序来自动 寻找特征词汇和文本的表达方式。正如在1.1节中那样，可以通过把它们到复制 Python 解释器中来尝试 Python 语言的新特征，并将在下一节中系统地了解这些功能。

在这之前，你可能会想通过预测下面代码的输出来检验你对上一节的理解。你可以

图1-3所示。计数器可能需要几千行代 码，这将是一个极其繁琐的过程——如 此繁琐以至于我们宁愿把任务交给机 器来做。

图1-3中的表被称为**频率分布，**显示的 是每一个词项在文本中出现的频率。

what output do you expect here?

使用解释器来检查你是否正确。如果你不确定如何完成这个任务，你最好在继续学 习之前复习一下上一节的内容。

>>>saying=['After','all','is','said','and','done',

\*\* · 'more','is','said', 'than','done']

>>>tokens =set(saying)

>>>tokens =sorted(tokens)

>>>tokens[-2:]

>>>

**频率分布**

如何能自动识别文本中最能体现文本主题和风格的词汇?试想一下，要找到一本书 中使用最频繁的50个词你会怎么做?一种方法是为每个词项设置一个计数器，如

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Word Tally   |  |  | | --- | --- | | the | H I | | been | HH I | | message | Ⅲl | | persevere | l | | nation | H | |

(一般情况下，它能计数任何观察的到 **图1-3** **计数一个文本中出现的单词(频率分布)** 的事件。)这是一个“分布”,因为它告诉我们文本中词标识符的总数是如何分布在 词项中的。因为我们经常需要在语言处理中使用频率分布，NLTK为它们提供内置 支持。利用 FreqDist 寻找《白鲸记》中最常见的50个词。尝试做出下面的例子， 然后阅读接下来的解释。

>>>fdist1=FreqDist(text1)①

>>>fdist1 ②

<FreqDist with 260819 outcomes>

>>>vocabulary1 =fdist1.keys()③

>>>vocabulary1[:50]④

[',','the','.','of','and','a','to',';','in','that',"'",'-','his',

'it','I','s','is','he','with','was','as','"','all','for','this','!','at',

'by','but','not','--','him','from','be','on','so','whale','one','you',

'had','have','there','But','or','were','now','which','?','me','like']

>>>fdist1['whale']

906

>>>

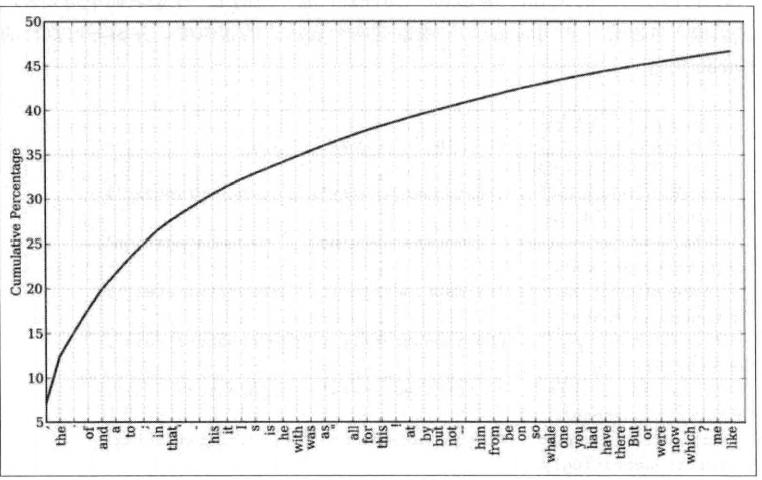


第一次调用 FreqDist 时，传递文本的名称作为参数°。计算得到的《白鲸记》中 单词的总数(“结果”)——高达260819°。表达式 keys()为我们提供了文本中所 有不同类型的链表③,可以通过切片看看这个链表的前50项。

**轮到你来；**

使用 text2 尝试前面频率分布的例子。注意正确使用括号和大写字母。 如果得到的是错误消息： NameError:name FreqDist is not defined, 则需要在一开始输入nltk.book import\*。

上一个例子中是否有什么词有助于我们把握这个文本的主题或风格呢?只有一个 词，whale, 这是仅有的信息量!它出现了超过900次。其余的词没有告诉我们关 于文本的信息，它们只是“管道”英语。这些词在文本中占多少比例?我们可以产 生关于这些词汇的累积频率图，使用 fdist1.plot(50,cumulative=True) 产生图1-4 。 这50个词占了这本书的将近一半!



**图1-4** **《白鲸记》中50个最常用词的累积频率图，这些词占了所有标识符的将近一半**

如果高频词对我们没有帮助，那么只出现了一次的词(所谓的 **hapaxes** )又如何呢? 输入fdist1.hapaxes()查看结果。此链表包括lexicographer 、cetological 、contraband,

expostulations 等9000多个词。看来低频词太多了，没看到上下文我们很可能无法 猜到点 hapaxes 的含义!既然高频词和低频词都没有帮助，那就需要尝试其他的 办法。

**细粒度的选择词**

接下来，让我们看看文本中的长词，也许它们有更多的特征和信息量。为此可以采 用集合论的一些符号。想要找出文本词汇表中长度超过15个字符的词，把它称为 特性P, 则当且仅当词w 的长度大余15个字符时P(w)为真。现在可以用(1a) 中 的数学集合符号表示我们感兴趣的词汇。它的含义是：此集合中所有w 都满足： w 是 集合V (词汇表)的一个元素且w 有特性P。

(1) a.{w l wV∈&P(w)}

b.[w for w in V ifp(w)]

(1b) 给出了对应的 Python 表达式。(请注意，它产生的是一个链表而不是集合， 这意味着可能会有相同的元素。)观察这两个表达式的相似度。并编写可执行的 Python 代码。

>>>V=set(text1)

>>>long\_words =[w for w in v if len(w)>15]

>>>sorted(long\_words)

['CIRCUMNAVIGATION','Physiognomically','apprehensiveness',

'cannibalistically',

'characteristically','circumnavigating','circumnavigation', 'circumnavigations',

'comprehensiveness','hermaphroditical','indiscriminately',

'indispensableness',

'irresistibleness','physiognomically','preternaturalness',

'responsibilities',

'simultaneousness','subterraneousness','supernaturalness',

'superstitiousness',

'uncomfortableness','uncompromisedness','undiscriminating',

'uninterpenetratingly']

>>>

对于词汇表V 中的每一个词 w, 都要检查len(w)是否大于15;所有其他词汇将被 忽略。我们将在后面更仔细地讨论这里的句法。



**轮到你来：**

在Python 解释器中尝试上面的表达式，改变文本和长度条件做一些实 验。如果改变变量名，对你的结果会产生什么影响?例如使用[word for word in vocab if …]?

让我们回到寻找文本特征词汇的问题上来。请注意， text4 中的长词反映国家主题——

*constitutionally (按宪法规定地，本质地),transcontinental (横贯大陆的)——而* *text5*

中的长词是非正规表达方式，例如 boooooooooooglyyyyyy 和 **yuaaanaaaaaammmmmm** mmmmmm。 我们是否已经成功地自动提取出文本的特征词汇了呢?那么,这些很长 的词通常是 hapaxes(即唯一的),也许对寻找长词出现的频率会更好。这样看起来 更有效，因为这样忽略了短高频词(如 the) 和长低频词(如 antiphilosophists)。以 下是聊天语料库中所有长度超过7个字符并且出现次数超过7次的词。

>>>sorted([w for w in set(text5)if len(w)>7 and fdist5[w]>7])

>>>fdist5 =FreqDist(text5)

['#14-19teens','#talkcity\_adults','((((((((((','………','Question',

'actually','anything','computer','cute.-ass','everyone','football', 'innocent','listening','remember','seriously','something','together',

'tomorrow','watching']

>>>

注意如何使用这两个条件： len(w)>7 确保词长都超过7个字母， fdist5[w]>7 确保这 些词出现次数超过7次。最后，我们成功地自动识别出与文本内容相关的高频词。 这很小的一步却是一个重要的里程碑： 一小块代码，处理数以万计的词，产生一些 有信息量的输出。

**词语搭配和双连词**

**搭配是**不经常在一起出现的词序列。因此， red wine 是一个搭配而 the wine不是。 搭配的特点是其中的词不能被类似的词置换。例如： maroon wine (粟色酒)听起 来就很奇怪。

要获取搭配，首先从提取文本词汇中的词对也就是双连词开始。使用函数 bigramsO 很容易实现这点。

>>>bigrams(['more','is','said','than','done'])

[('more','is'),('is','said'),('said','than'),('than','done')]

>>>

在这里我们看到词对 than-done 是一个双连词，在 Python中写成(than',done) 。 现在， 除了着重考虑包含生僻词的情况，搭配基本上是频繁的双连词。特别是在已知单个 词汇频率的基础上，想要找到出现频率比预期频率更频繁的双连词。collocations) 函数为我们做这些(将在以后学习它是如何实现的)。

>>>text4.collocations()

Building collocations list

United States;fellow citizens;years ago;Federal Government;General Government;American people;Vice President;Almighty God;Fellow

citizens;Chief Magistrate;Chief Justice;God bless;Indian tribes; public debt;foreign nations;political parties;State governments;

National Government;United Nations;public money

>>>text8.collocations()

Building collocations list

medium build;social drinker;quiet nights;long term;age open;

financially secure;fun times;similar interests;Age open;poss

rship;single mum;permanent relationship;slim build;seeks lady; late 30s;Photo pls;Vibrant personality;European background;ASIAN LADY;country drives

>>>

文本中出现的搭配能非常明确地体现文本的类型。为了找到 red wine 这个搭配，需 要处理更大的文本。

**计算其他东西**

计算词汇是有用的，也可以计算其他东西。例如，可以查看文本中词长的分布，通 过创造一长串数字链表的 FreqDist, 其中每个数字表示文本中对应词的长度。

>>>[len(w)for w in textl]①

[1,4,4,2,6,8,4,1,9,1,1,8,2,1,4,11,5,2,1,7,6,1,3,4,

5,2,...]

>>>fdist =FreqDist([len(w)for w in textl])②

>>>fdist③

<FreqDist with 260819 outcomes>

>>>fdist.keys()

[3,1,4,2,5,6,7,8,9,10,11,12,13,14,15,16,17,18,20]

>>>

从text1 中每个词的长度的链表开始°,然后计数 FreqDist计数链表中每个数字出现 的次数③。结果③是一个包含25万左右个元素的分布，每一个元素是一个数字，对 应文本中一个词标识符。但是只有20个不同的元素被计数，从1到20,因为只有

20个不同的词长。也就是说，有由1个字符，2个字符， ……,20个字符组成的 词，而没有由21个或更多字符组成的词。有人可能会问不同长度的词的频率是多 少?(例如，文本中有多少个长度为4的词?长度为5的词是否比长度为4的词多? 等等)。通过下面的例子回答这个问题。

>>>fdist.items()

[(3,50223),(1,47933),(4,42345),(2,38513),(5,26597),(6,17111),(7, 14399),

(8,9966),(9,6428),(10,3528),(11,1873),(12,1053),(13,567),(14,177),

1>,)it. (2)2), (17,12), (18,1), (20,1)]

3

>>>fdist[3]

50223

>>>fdist.freq(3)

0.19 255882431878046

>>>

由此可见，最频繁词的长度是3,长度为3的词有50000多个(约占书中全部词 汇的20%)。虽然我们不会在这里研究它，但关于词长的进一步分析可能帮助我 们了解作者、文体和语言之间的差异。表1-2总结了 NLTK 频率分布类中定义 的函数。

**表1-2** **NLTK** **频率分布类中定义的函数**

|  |  |
| --- | --- |
| **例** **子** | **描** **述** |
| fdist =FreqDist(samples) | 创建包含给定样本的频率分布 |
| fdist.inc(sample | 增加样本 |
| fdist['monstrous' | 计数给定样本出现的次数 |
| fdist.freq('monstrous') | 给定样本的频率 |
| fdist.N() | 样本总数 |
| fdist.keys() | 以频率递减顺序排序的样本链表 |
| for sample in fdist: | 以频率递减的顺序遍历样本 |
| fdist.max) | 数值最大的样本 |
| fdist.tabulate) | 绘制频率分布表 |
| fdist.plot) | 绘制频率分布图 |
| fdist.plot(cumulative=True) | 绘制累积频率分布图 |
| fdist1<fdist2 | 测试样本在fdist1中出现的频率是否小于fdist2 |

关于频率分布的讨论中引入了一些重要的Python 概念，我们将在1.4节系统地学习。

**1.4** **回到** **Python:决策与控制**

到目前为止，小程序有了一些有趣的特征：处理语言的能力和通过自动化节省人力 的潜力。程序设计的一个关键特征是让机器能按照我们的意愿决策，在遇到特定条 件时执行特定命令，或者对文本数据从头到尾不断循环直到条件满足。这一特征被 称为**控制，**这是本节的重点。

**条件**

Python 广泛支持多种运算符，如：<和>=,可以测试值之间的关系。全部的**关系运** 算符见表1-3。

**表1-3** **数值比较运算符**

|  |  |
| --- | --- |
| **运算** **符** | **关** **系** |
| < | 小于 |
|  | 小于等于 |
| 二二 | 等于(注意是两个“=”号而不是一个) |
| != | 不等于 |
| > | 大于 |
| >= | 大于等于 |

可以使用这些从新闻文本句子中选出不同的词。下面是一些例子——注意行与 行之间只是运算符不同。它们都使用sent7,第一句话来自text7(华尔街日报)。 像以前一样，如果得到错误结果，sent7 没有定义，需要首先输入：from nltk.book

**import\*。**

>>>sent7

['Pierre','Vinken',',','61','years','old',',','will','join','the',

'board','as','a','nonexecutive','director','Nov.','29','.']

>>>[w for w in sent7 if len(w)<4]

','e',,'a9','nt7 if len(w)<=4]

[',','61','old',',','will','join','the','as','a','Nov.','29','.']

>>>[w for w in sent7 if len(w)==4]

'rrgdn(t[yit'e,'eiiei' seotttiiyim.istitle()])

['will','join','Nov.']

>>>[w for w in sent7 if len(w) !=4]

['Pierre','Vinken',',','61','years','old',',','the','board',

'as','a','nonexecutive','director','29','.']

>>>

所有这些例子都有一个共同的模式： [w for w in text if condition], 其中 condition 是一个Python“测试”,得到真 (true) 或者假 (false) 。 在前面的代码例子中， 条件始终是数值比较。然而，也可以使用表1-4 中列出的函数测试词汇的各种 属性。

**表1-4** **词汇比较运算符**

|  |  |
| --- | --- |
| **函** **数** | **含** **义** |
| s.startswith(t) | 测试s是否以t开头 |
| s.endswith(t) | 测试s是否以t结尾 |
| t in s | 测试s是否包含t |
| s.islower() | 测试s中所有字符是否都是小写字母 |
| s.isupper() | 测试s中所有字符是否都是大写字母 |
| s.isalpha( | 测试s中所有字符是否都是字母 |
| s.isalnum() | 测试s中所有字符是否都是字母或数字 |
| s,isdigitC | 测试s中所有字符是否都是数字 |
| s.istitle) | 测试s是否首字母大写(s中所有的词都首字母大写) |

下面是一些从文本中选择词汇运算符的例子：以-ableness 结尾的词，包含 gnt 的词， 首字母大写的词，完全由数字组成的词。

>>>sorted([w for w in set(textl)if w.endswith('ableness')])

['comfortableness','honourableness','immutableness',

>>>sorted([term for term in set(text4)if'gnt'in term])

'indispensableness',...]

['A','Aaaaaaaaah','Aaaaaaaah','Aaaaaah','Aaaah','Aaaaugh','Aaagh',.]

>>>sorted([item for item in set(sent7)if item.isdigit()])

['29','61']

>>>

还可以创建更复杂的条件。如果c 是一个条件，那么 not c 也是一个条件。如果有 两个条件 c1 和 c2, 那么还可以使用合取和析取将它们合并形成一个新的条件： cl



**and c2,cl or c2。**

**轮到你来：**

**运行下面的例子，尝试解释每一条指令中所发生的事情。然后，试着** **自己组合一些条件。**

>>>sorted([w for w in set(text7)if'-'in w and 'index'in w])

>>>sorted([wd for wd in set(text3)if wd.istitle()and len(wd) >10])

>>>sorted([w for w in set(sent7)if not w.islower()])

>>>sorted([t for t in set(text2)if 'cie'in t or 'cei'in t])

**对每个元素进行操作**

在1.3节中，列举了计数词汇以外的其他项目的一些例子。让我们仔细看看之前所 使用的符号。

>>>[len(w) for w in textl]

[1,4,4,2,6,8,4,1,9,1,1,8,2,1,4,11,5,2,1,7,6,1,3,4,

5,2,...]

>>>[w.upper()for w in textl]

['[','MOBY','DICK','BY','HERMAN','MELVILLE','1851',']','ETYMOLOGY',

.' ..]

>>>

表达式形式为[f(w)for …]或[w.f()for …], 其中 f 是一个函数，用来计算词长或将字 母转换为大写。现阶段还不需要理解两种表示方法f(w)与w.f()之间的差异，而只 需学习Python 习惯用法 (idiom), 即对链表上的所有元素执行相同的操作。在前 面的例子中，遍历 text1 中的每一个词，依次地赋值给变量 w 并在变量上执行指 定的操作。

上述的表示法被称为“链表推导”,这是第一个 Python 习惯用法的例 子，是一种固定的表示法，我们习惯使用该方法，这样省去了每次分 析的烦恼。掌握这些习惯用法是成为一流Python程序员的一个重要组 成部分。

回到计数词汇的问题上，这里使用相同的习惯用法。

>>>len(text1)



19317

>>>len(set([word.lower() 17231

>>>

for word in text1]))

由于不重复计算像 This 和 this 这样仅仅大小写不同的词，就这样从词汇表计数中 抹去了2000个!更进一步，还可以通过过滤掉所有非字母元素，从词汇表中消除 数字和标点符号。

>>>len(set([word.lower()for word in text1 if word.isalpha()]))

16948

>>>

这个例子稍微有些复杂：将所有纯字母组成的词小写。也许只计数小写的词会更简

单一些，但这却是一个错误的答案(为什么?)。

如果你对链表推导不是很有信心，请不要担心，因为在下面的章节中你会学习到更

多的例子及解释。

**嵌套代码块**

大多数编程语言允许我们在**条件表达式**或者说if 语句条件满足时执行代码块。例 如[w for w in sent7 iflen(w)<4]这样的条件测试的例子。在下面的程序中，我们创建 了一个叫 word 的变量包含字符串值 “cat” 。在 if 语句中检查 len(word)<5 是否为真。 cat 的长度确实小于5,所以 if 语句下的代码块被调用，print 语句被执行，向用户 显示一条消息。别忘了要缩进，在 print 语句前输入4个空格。

>>>word

=

>>>if

'cat'

len(word)<5:

print 'word length is less than 5'

... ①

word length is less than 5

>>>

使用 Python 解释器时，必须添加一个额外的空白行°,这样它才能检测到嵌套块

结束。

如果改变测试条件为len(word)>=5, 检查词的长度是否大于或等于5,那么测试将

不再为真。此时， if 语句后面的代码段将不会被执行，没有消息显示给用户。

>>>if len(word)>=5:

print 'word length is greater than or equal to 5'

. .

>>>

if 语句被看作是**控制结构，**因为它控制缩进块中的代码是否运行。另一个控制结构 是 for 循环。尝试下面的代码，请记住输入冒号和4个空格。

>>>for word in ['Call','me','Ishmael','.']:

· print word

… …

Call

me

Ishmael

>>>

这叫做循环，因为 Python 以循环的方式执行里面的代码。它从 word=Call'赋值开 始，有效地使用变量 word命名链表的第一个元素。然后，显示word 的值给用户。 接下来回到 for 语句，执行word='me '赋值，然后把这个新值显示给用户，以此类 推。它以这种方式不断运行，直到链表中所有项都被处理完。

**条件循环**

现在，可以将 if 语句和 for 语句结合。循环链表中的每一项，只输出结尾字母是 l 的词。我们将为变量挑选另一个名字以表明 Python 并不在意变量名的意义。

>>>sent1=['Call',

>>>for xyzzy if

in

'me','Ishmael','.']

sent1:

xyzzy.endswith('l'):

·

.

·

…

Call

Ishmael

>>>

print xyzzy

你会发现在if 和 for 语句所在行末尾——缩进开始之前——有一个冒号。事实上， 所有的Python控制结构都以冒号结尾。冒号表示当前语句与后面的缩进块有关联。

也可以指定当if 语句的条件不满足时采取的行动。在这里，我们看到elif(elseif)语句 和 else 语句。请注意，这些语句在缩进代码前也有冒号。

>>>for token in sentl:

if token.islower():

elif teoiksetni()a: lowercase word'

print token, 'is a titlecase word'

… · else:

\*\* print token,'is punctuation'

is a titlecase word

me is a lowercase word

Ishmael is a titlecase word

.is

>>>

punctuation

正如你看到的，只具备少量的 Python知识，就可以构建多行的Python 程序。分块 开发程序，在整合之前测试每一块代码是否达到你的预期是很重要的。这也是 Python交互式解释器的价值所在，也是为什么你必须适应它。

最后，让我们把一直在探索的习惯用法组合起来。首先，创建一个包含 cie 或者 cei 词汇的链表，然后循环输出其中的每一项。请注意 print 语句结尾处的逗号，以便 使结果在同一行输出。

>>>tricky=sorted([w for w in set(text2)if'cie'in w or 'cei'in w]) >>>for word in tricky:

print word,

ancient ceiling conceit conceited conceive conscience

conscientious conscientiously deceitful deceive ...

>>>

**1.5** **自动理解自然语言**

我们一直在各种文本和 Python 编程语言的帮助下自下而上地探索语言。然而，我 们也对通过构建有用的语言技术，开拓语言和计算知识面的兴趣。现在，将借此 机会从代码的细节中退出来，以描绘自然语言处理的全景图。

在纯应用层面上，我们都需要帮助才能在网络上的文本中找到有用的信息。搜索引 擎在网络的发展和普及中发挥了关键作用，但也有一些缺点。它需要技能、知识和 一点运气才能找到这样一些问题的答案：“我用有限的预算能参观费城和匹兹堡的

*哪些景点?”,“专家们怎么评论数码单反相机?”,“过去的一周里可信的评论员都*

*对钢材市场做了哪些预测?”。让计算机来自动回答这些问题，涉及包括信息提取、*

推理与总结在内的广泛的语言处理任务，而且将需要在一个更大规模、更稳健的层 面上实施，这超出了我们当前的能力。

在哲学层面上，构建智能机器是人工智能长久以来的挑战，语言理解是智能行为的 重要组成部分。这一目标多年来一直被看作是太困难了。然而，随着 NLP 技术日 趋成熟，用稳定的方法来分析非结构化文本越来越广泛，对自然语言理解的期望变 成一个合理的目标再次浮现。

在本节中，我们将介绍一些语言理解技术。

**词意消歧**

在**词意消歧中**，要分析出特定上下文中的词被赋予的是哪个意思。思考存在歧义的

词serve 和 dish。

(2) a.serve:help with food or drink;hold an office;put ball into play b.dish:plate;course of a meal;communications device

在包含短语 he served the dish 的句子中，你可以知道serve 和 dish 用的都是它们与 食物相关的含义。仅仅3个词，讨论的话题不太可能从体育转向陶器。这也许会迫 使你眼前产生一幅怪异的画面： 一个职业网球手正把他的郁闷发泄到放在网球场边 上的陶瓷茶具上。换句话说，自动消除歧义需要使用上下文，利用相邻词汇的相近 含义。另一个有关上下文影响的例子是词 by, 它有几种含义，例如： the book by Chesterton (施事——Chesterton 是书的作者); the cup by the stove (位置——炉子 在杯子旁边); submit by Friday (时间——星期五前提交)。观察(3)中斜体字的 含义有助于解释 by的含义。

(3) a.The lost children were found by the searchers (施事) b.The lost children were found by the mountain (位置) c.The lost children were found by the afternoon (时间)

**指代消解**

更深刻的语言理解是解决“谁对谁做了什么”,即检测动词的主语和宾语。虽然你 在小学已经学会了这些，但它比你想象的更难。在句子 the thieves stole the paintings 中，很容易分辨出谁做了偷窃的行为。考虑(4)中句子的3种可能，尝试确定是

什么被出售、被抓和被发现(其中一种情况是有歧义的)。

(4) a.The thieves stole the paintings.They were subsequently sold. b.The thieves stole the paintings.They were subsequently caught. c.The thieves stole the paintings.They were subsequently found.

要回答这个问题涉及到寻找代词 they 的先行词 thieves 或者 paintings 。处理这个问 题的计算技术包括**指代消解** (anaphora resolution)——确定代词或名词短语指的是 什么——和**语义角色标注** (semantic role labeling)——确定名词短语如何与动词相 关联(如代理、受事、工具等)。

**自动生成语言**

如果能够自动地解决语言理解等问题，我们将能够继续进行那些包含自动生成语言 的任务，如**自动问答和机器翻译。**在自动问答中， 一台机器应该能够回答用户关于 特定文本集的问题。

(5) a.Text:...The thieves stole the paintings.They were subsequently sold..

b.Human:Who or what was sold?

c.Machine:The paintings.

机器的回答表明，它已经正确地计算出 they 是指 paintings,而不是 thieves。在机 器翻译中，机器应该能够把文本翻译成另一种语言文字，并准确传达原文的意思。 在把例子文本译成法文过程中，我们不得不在第二句中选择代词的性别：ils(男性), 如果thieves 被出售，elles (女性),如果 paintings 被出售。正确的翻译实际上取决 于对代词的正确理解。

(6) a.The thieves stole the paintings.They were subsequently found. b.Les voleurs ont voléles peintures.Ils ontététrouvés plus tard.

(the thieves)

c.Les voleurs ont voléles peintures.Elles ontététrouvées plus tard. (the paintings)

所有这些例子中，弄清楚词的含义、动作的主语及代词的先行词是确定句子含义的 步骤，也是希望语言理解系统能够做到的事情。

**机器翻译**

长久以来，机器翻译 (MT) 都是语言理解的圣杯，人们希望能找到从根本上提供

高品质且符合语言习惯的任意两种语言之间的翻译。其历史可以追溯到冷战初期， 当时自动翻译带来大量的政府赞助，它也是NLP 本身的起源。

今天，特定语言之间实用的翻译系统已经形成，而且有些已经集成到搜索引擎中了。 但是，这些系统有一些严重的缺点。我们可以在NLTK的 “babelizer” 的帮助下探 索它们(当你使用ffom nltk.book import\* 导入本章的材料时，它已经自动装载了)。 这个程序把提交的英文句子翻译成指定语言，然后把结果重新翻译回英文。这样重 复12次后结束或者得到曾经产生过的翻译时(表示一个循环)结束。

>>>babelize\_shell()

NLTK Babelizer:type 'help'for a list of commands.

Babel>how long before the next flight to Alice Springs?

Babel>german

Babel>run

0>how long before the next flight to Alice Springs?

1>wie lang vor dem folgenden Flug zu Alice Springs?

2>how long before the following flight to Alice jump?

3>wie lang vor dem folgenden Flug zu Alice springen Sie?

4>how long before the following flight to Alice do you jump?

5>wie lang,bevor der folgende Flug zu Alice tun,Sie springen?

6>how long,before the following flight to Alice does,do you jump? 7>wie lang bevor der folgende Flug zu Alice tut,tun Sie springen? 8>how long before the following flight to Alice does,do you jump? 9>wie lang,bevor der folgende Flug zu Alice tut,tun Sie springen?

10>how long,before the following flight does to Alice,do do you jump? 11>wie lang bevor der folgende Flug zu Alice tut,Sie tun Sprung?

12>how long before the following flight does leap to Alice,does you?

观察可以发现，该系统正确地把Alice Springs 从英文翻译成了德文(第1行),但 在翻译回英文时却以 Alice jump 结束(第2行)。介词 before一开始被翻译成德文 对应的介词vor, 但后来变成了连词bevor (第5行)。第5行以后的句子变得没有 意义(但请注意以逗号指示的各种分句，以及从 jump 到 leap 的变化)。翻译系统 无法识别当一个词是某个名字的一部分时，并且会弄错语法结构。语法问题在下面 的例子中更加明显。是约翰发现了猪，还是猪找到了约翰?

>>>babelize\_shell()

Babel>The pig that John found looked happy

Babel>german

Babel>run

0>The pig that John found looked happy

1>Das Schwein,das John fand,schaute gl?cklich

2>The pig,which found John,looked happy

机器翻译是困难的， 一方面原因是给定的单词可能有几种不同的解释(取决于它的 意思),另一方面原因是必须改变词序才能与目标语言的语法结构保持一致。如今， 遇到的难题是，从新闻和政府网站发布的两种或两种以上的语言文档中可以收集到 大量的相似文本。如果给出一个德文和英文双语的文档或者一个双语词典，就可 以自动配对组成句子，这个过程叫做**文本对齐。**一旦有一百万个或更多的句子对， 就可以检测出相应的单词和短语，并建立能用来翻译新文本的模型。

**人机对话系统**

在人工智能的历史中，主要的智能测试是一种语言学测试，叫做**图灵测试：** 一个响 应用户文本输入的对话系统能否表现得如此自然以致我们无法区分它是人工生成 的响应?相比之下，今天的商业对话系统能力是非常有限的，但在有限的给定领域 仍然能够发挥作用，如下面的下例子。

S:How may I help you?

U:When is Saving Private Ryan playing?

S:For what theater?

U:The Paramount theater.

S:Saving Private Ryan is not playing at the Paramount theater,but

it's playing at the Madison theater at 3:00,5:30,8:00,and 10:30.

你不能要求这个系统提供驾驶指示或附近餐馆的细节，除非所需的信息已经被保存 并且合适的问题答案已经被纳入语言处理系统。

观察可看出这个系统能够了解用户的目标：用户询问电影上映的时间，系统正确地 判断出用户是想要看电影。这一推断看起来如此明显，以至于你可能都没有注意到 它， 一个自然语言系统需要被赋予这种自然的交互能力。没有它，当问到：“你知

*道拯救大兵瑞恩什么时候上映?”时，系统可能只会回答一个冷冷的、毫无用处的*

“是的”。然而，商业对话系统的开发者使用上下文语境假设和业务逻辑确保在用户 以不同方式表达需求或提供信息时对特定应用都能有效处理。因此，如果你输入 When is…或 者I want to know when…或者 Can you tell me when…时，这些简单的规 则总是对应着放映时间，这足以使系统能够提供有益的服务了。

对话系统展示了一般的 NLP 流程。图1-5所示为一个简单的对话系统架构。沿图



的顶部从左向右是一些语言理解**组件的**“管道”。这些图是从语音输入经过文法分 析到某种意义的重现。图的中间，从右向左是这些组件将概念转换为语音的逆向 流程。这些组件构成了系统的动态方面。在图的底部是一些有代表性的静态信息： 处理组件在其上运作的语言相关数据的仓库。

**轮到你来：**

作为一个原始的对话系统的例子，尝试与 NLTK 的 chatbot 谈话。使 用之前请运行nltk.chat.chatbots() 。(记住要先输入nltk。)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 语音分析  语音合成  发音模型  语位学  Phonology | 形态和词法  分析  形态实现  形态规划  形态学  Morphology | 解析  句法实现  词汇和  语法句法  Syntax | 上下文推理  话语规划 | 应用推理和  执行 |
| 话语背景  语义  Semantics | 领域知识  推理  Reasoning |

图1-5 简单的语音对话系统的流程架构：分析语音输入(左上),识别单词，文法分析和在上 下文中解释，应用相关的具体操作(右上);响应规划，实现文法结构，然后是适当的词形变 化，最后到语音输出；处理的每个过程都蕴含不同类型的语言学知识

**文本的含义**

近年来， 一个叫做文本含义识别(Recognizing Textual Entailment,RTE) 的公开的 “共享任务”使语言理解所面临的挑战成为关注焦点。基本情形很简单：试想你想 找到证据来支持一个假设： Sandra Goudie 被 Max Purnell 击败了。而有一段简短的 文字似乎是有关的，例如： Sandra Goudie在2002年国会选举首次当选，通过击败 工党候选人Max Purnell 将现任绿党下院议员Jeanette Fitzsimons 推到第三位，以微 弱优势赢得了 Coromandel 席位。文本是否为假说提供了足够的证据呢?在这种特 殊情况下，答案是“否”。你可以很容易得出这样的结论，但使用自动方法做出正 确决策是困难的。RTE 挑战为竞赛者开发系统提供数据，但这些数据对“蛮力”机

器学习技术(将在第6章介绍)来说是不够的。因此，语言学分析是至关重要的。 在前面的例子中，很重要的一点是让系统知道 Sandra Goudie 是假设中被击败的人， 而不是文本中击败别人的人。思考下面的文本-假设对，这是任务困难性的另一个 例证。

(7) a.Text:David Golinkin is the editor or author of 18 books,and over 150 responsa,articles,sermons and books

b.Hypothesis:Golinkin has written 18 books

为了确定假说是否能得到文本的支持，该系统需要以下背景知识：

(1)如果有人是一本书的作者，那么他/她写了这本书；

(2)如果有人是一本书的编辑，那么他/她(完全)没有写这本书；

(3)如果有人是18本书的编辑或作者，则无法断定他/她是18本书的作者。

**NLP 的局限性**

尽管在很多如 RTE 这样的任务研究中取得了进展，但在现实世界的应用中已经部 署的语言理解系统仍不能进行常识推理或以一般的可靠的方式描述这个世界的知 识。在等待这些困难的人工智能问题得到解决的同时，接受一些在推理和知识能力 上存在严重限制的自然语言系统是有必要的。因此，从一开始，自然语言处理研究 的重要目标一直是使用浅显但强大的技术代替无边无际的知识和推理能力，促进构 建“语言理解”技术的艰巨任务不断取得进展。事实上，这是本书的目标之一，我 们希望你能掌握这些知识和技能，构建有效的自然语言处理系统，并为构建智能机 器这一长期的理想做出贡献。

**1.6** **小结**

● 在 Python中，文本用链表来表示： [Monty’,Python'] 。 我们可以使用索引、分 片和len()函数对链表进行操作。

· 词 “token” (标识符)是指文本中给定词的特定出现；词 “type” (类型)则是 指词作为一个特定序列字母的唯一形式。我们使用 len(text)表示计数词的标识 符，使用len(set(text))表示计数词的类型。

● 我们使用 sorted(set(t)获得文本t的词汇表。

● 我们使用[f(x)for x in text]对文本的每一项目进行操作。

● 为了获得没有大小写区分和忽略标点符号的词汇表，我们可以使用 set([w.lower()for w in text if w.isalpha()])。

· 我们使用 for 语句对文本中的每个词进行处理，例如 for win t:或者 for word in text: 。后面必须跟冒号和在每次循环都被执行的缩进代码。

● 我们使用 if 语句测试一个条件： if len(word)<5:。后面必须跟冒号和仅当条件 为真时执行的缩进代码。

● 频率分布是项目连同它们的频率计数的集合(例如： 一个文本中的词与它们出 现的频率)。

● 函数是指定了名字并且可以重用的代码块。函数通过 def 关键字定义，例如在 def mult(x,y)中 x 和 y 是函数的参数，起到实际数据值占位符的作用。

● 函数是通过指定它的名字及一个或多个放在括号里的实参来调用，就像这样： mult(3,4) 或者len(text1)。

**1.7** **深入阅读**

本章综合介绍了有关编程、自然语言处理和语言学的新概念。其中的一些将会 在下面的章节继续出现。然而，你可能也想咨询与本章相关的在线材料(在 <http://www.nltk.org/>), 包括额外的背景资料的链接及在线 NLP 系统的链接。你可能 还喜欢在维基百科中阅读一些语言学和自然语言处理相关的概念(如搭配、图灵测 试、类型-标识符的区别等)。

你应该自己去熟悉 <http://docs.python.org/> 上的Python 文档，那里给出了许多教程和 全面的参考材料。 <http://wiki.python.org/moin/BeginnersGuide> 上有《Python 初学者 指南》。关于Python 的各种问题在 <http://www.python.org/doc/faq/general/> 的 FAQ 中 都有回答。

随着对NLTK 研究的深入，你可能想订阅有关新版工具包的邮件列表。有了NLTK

用户邮件列表，用户在学习如何使用Python 和 NLTK 做语言分析工作时可以相互 帮助。在 <http://www.nltk.org/>中有这些列表的详情。

如果需要第1.5节所讲述的话题及NLP 的相关信息，你可以阅读以下的优秀图书。

· Indurkhya,Nitin 和Fred Damerau (合编，2010)自然语言处理手册 (Handbook of Natural Language Processing) (第二版),Chapman &Hall/CRC。

· Jurafsky,Daniel 和 James Martin(2008) 语音和语言处理 (Speech and Language Processing) (第二版),Prentice Hall。

● Mitkov,Ruslan (主编，2002年),牛津计算语言学手册 (The Oxford Handbook of Computational Linguistics)。牛津大学出版社。

计算语言学协会 (The Association for Computational Linguistics,ACL) 是代表 NLP领域的国际组织。ACL 网站上有许多有用的资源，包括：有关国际和地区 的会议及研讨会的信息；到数以百计有用资源的 ACL Wiki 链接；包含过去50 年以来大多数 NLP 研究文献的 ACL 选集，里面的论文全部建立索引且可免费 下载。

一些介绍语言学的优秀的教科书：(Finegan,2007),(O'Grady et al.,2004),(OSU, 2007)。LanguageLog, 是一个流行的语言学博客，其上会不定期发布一些本书中描 述的技术应用。

**1.8** **练习**

1.O 尝试使用Python 解释器作为一个计算器，输入表达式，如12/(4+1)。

2.O26 个字母可以组成26的10次方或者26\*\*10个10字母长的字符串。也就是 141167095653376L(结尾处的L 只是表示这是Python长数字格式)。100个字母长 度的字符串可能有多少个?

3.OPython 乘法运算可应用于链表。当你输入[Monty',Python']\*20 或者3\*sentl 会发生什么?

4.O 复习1.1节关于语言计算的内容。在text2中有多少个词?有多少个不同的词?

5.O 比较表格1-1中幽默和言情小说的词汇多样性得分，哪一个文体中词汇更丰富?

6.O 制作《理智与情感》中4个主角： Elinor 、Marianne 、Edward 和 Willoughby 的分布图。在这部小说中关于男性和女性所扮演的不同角色，你能观察到什么?你 能找出一对夫妻吗?

7.O 查找text5中的搭配。

8.O 思考下面的 Python 表达式： len(set(text4))。说明这个表达式的用途，并且描 述在执行此计算中涉及的两个步骤。

9.O 复习1.2节关于链表和字符串的内容。

a. 定义一个字符串，并且将它分配给一个变量，如： my\_string='My String' (在字符串中放一些更有趣的东西)。用两种方法输出这个变量的内容， 一种是通过简单地输入变量的名称，然后按回车；另一种是通过使用 print 语句。

b. 尝试使用 my\_string+my\_string 或者用它乘以一个数将字符串添加到它自 身，例如： my\_string\*3。请注意，连接在一起的字符串之间没有空格。怎 样才能解决这个问题?

10.O 使用语法my\_sent =["My","sent"], 定义一个词链表变量my\_sent (用自己

的词或喜欢的话)。

a. 使用''join(my\_sent)将其转换成一个字符串。

b. 使用 split()在你指定的地方将字符串分割回链表。

11.O 定义几个包含词链表的变量，例如： phrasel、phrase2 等。将它们连接在一 起组成不同的组合(使用加法运算符),最终形成完整的句子。len(phrase1 +phrase2) 与len(phrase1)+len(phrase2)之间的关系是什么?

12.O 考虑下面两个具有相同值的表达式。哪一个在NLP 中更常用?为什么?

a."Monty Python"[6:12]

b.["Monty","Python"][1]

13.O 我们已经学习了如何用词链表表示一个句子，其中每个词是一个字符序列。 sent1[2][2]代表什么意思?为什么?并尝试其他的索引值。

14.O 在变量 sent3中保存的是text3 的第一句话。在 sent3 中 the 的索引值是1, 因为sent3[1]的值是 “the” 。sent3 中 “the” 的其他两种出现的索引值是多少?

15.O 复习1.4节讨论的条件语句。在聊天语料库(text5) 中查找所有以字母b 开 头的词。按字母顺序显示出来。

16.O 在 Python 解释器提示符下输入表达式 range(10)。再尝试 range(10,20), range(10,20,2) 和 range(20,10,-2)。在后续章节中我们将看到这个内置函数的多种 用途。

17.①使用 text9.index()查找词sunset 的索引值。你需要将这个词作为一个参数 插入到圆括号之间。在尝试和出错的过程中，在完整的句子中找到包含这个词 的切片。

18.①使用链表加法、set和 sorted 操作，计算句子 sent1…sent8的词汇表。

19.①下面两行之间的差异是什么?哪一个的值比较大?其他文本也是同样情 况吗?

>>>sorted(set([w.lower() for w in textl]))

>>>sorted([w.lower() for w in set(text1)]

20.Ow.isupper() 和 not w.islower()这两个测试之间的差异是什么?

21.①编写一个切片表达式提取text2 中的最后两个词。

22.①找出聊天语料库(text5) 中所有4个字母的词。使用频率分布函数 (FreqDist), 以频率从高到低显示这些词。

23.①复习1.4节中的条件循环。使用 for 和 if语句组合循环遍历电影剧本《巨蟒 和圣杯》(text6) 中的词，输出所有的大写词，每行输出一个。

24.①编写表达式并找出 text6 中所有符合下列条件的词。结果应该以词链表形式 表示： [word1','word2’,…]。

a. 以 ize 结尾。

b. 包含字母 z。

c. 包含字母序列 pt 。

d. 除了首字母外是全部小写字母的词(即titlecase)。

25.①定义 sent 为词链表[she','sells,'sea','shells','by',the','sea','shore'] 。 编写代码

执行以下任务。

a. 输出所有sh 开头的单词。

b. 输出所有长度超过4个字符的词。

26.①下面的 Python代码是做什么的? sum([len(w)for w in text1]), 你可以用它来 算出一个文本的平均字长吗?

27.①定义一个名为 vocab\_size(text)的函数，以文本作为唯一的参数，返回文本的 词汇量。

28.①定义一个函数percent(word,text), 计算一个给定的词在文本中出现的频率， 结果以百分比表示。

29.①我们一直在使用集合存储词汇表。试试下面的 Python 表达式：set(sent3)< set(text1)。尝试在set) 中使用不同的参数。它是做什么用的?你能想到一个实际的 应用吗?



**第2章**

**获得文本语料和词汇资源**

在自然语言处理的实际项目中，通常要使用大量的语言数据或者语料库。本章的目 的是要回答下列问题。

(1)什么是有用的文本语料库和词汇资源，如何使用Python 获取它们?

(2)哪些 Python结构最适合这项工作?

(3)编写Python 代码时如何避免重复的工作?

本章继续通过语言处理任务的例子展示编程概念。稍后再系统地探索每一个 Python 结构。如果你看到例子中含有一些不熟悉的东西，请不要担心。只需去尝试 它，看看它能做些什么——如果你很勇敢——通过使用不同的文本或词替换代码的 某些部分来进行修改。这样，你会将任务与编程习惯用法关联起来，并会在后续的 学习中了解到怎么会是这样和为什么是这样。

**2.1** **获取文本语料库**

正如刚才提到的， 一个文本语料库是一大段文本。许多语料库的设计都要考虑一 个或多个文体间的平衡。我们曾在第1章研究过一些小的文本集合，例如美国总 统就职演说。这种特殊的语料库实际上包含了几十个单独的文本——一个人一个 演讲——但为了处理方便，我们把它们头尾连接起来当做一个文本对待。第1章中 也使用变量预先定义了一些文本，我们通过输入from book import\* 来访问它们。



然而，由于我们希望能够处理其他文本，本节将探讨各种文本语料库。我们将学习 如何选择单个文本，以及如何处理它们。

**古腾堡语料库**

NLTK包含古腾堡项目(Project Gutenberg) 电子文本档案的一小部分文本。该项目大 约有25000(现在是36000了)本免费电子图书，放在 <http://www.gutenberg.org/>上。我 们先要用 Python 解释器加载 NLTK 包，然后尝试 nltk.corpus.gutenberg.fileids), 下面 是这个语料库中的文件标识符。

>>>import nltk

>>>nltk.corpus.gutenberg.fileids()

['austen-emma.txt','austen-persuasion.txt','austen-sense.txt',

'bible-kjv.txt',

-oaoornie-.' 'gcenrbr-bt',

'chesterton-thursday.txt','edgeworth-parents.txt',

'melville-moby\_dick.txt',

'milton-paradise.txt','shakespeare-caesar.txt',

'shakespeare-hamlet.txt',

'shakespeare-macbeth.txt','whitman-leaves.txt']

挑选这些文本中的第一个——简·奥斯丁的《爱玛》——简称 emma, 然后找出它 包含多少个词。

>>>emma

>>>len(emma)

192427

=nltk.corpus.gutenberg.words('austen-emma.txt')

在1.1节中，我们讲述了如何使用textl.concordance()命令对例如 text1 这样的文本进行索引。然而，前提是你正在使用由 from nltk.book import\* 导入的9个文本中的1个。现在开始研究 nltk.corpus 中的数 据，像前面的例子一样，必须采用以下语句对来执行索引和1.1节中 的其他任务。

>>>emma =nltk.Text(nltk.corpus.gutenberg.words('austen- emma.txt'))

>>>emma.concordance("surprize")

定义emma 时，我们调用了NLTK corpus 包中的 gutenberg 对象所包含的 wordsO函 数。但因为总是要输入这么长的名字很繁琐， Python 提供了另一个版本的 import

语句，示例如下。

>>>from nltk.corpus import gutenberg

>>>gutenberg.fileids()

emb,e'r'ans- t'austen-sense.txt', ..]

写一段简短的程序，通过循环遍历前面所列出与 **gutenberg** 文件标识符相应的 fileid, 然后计算统计每个文本。为了使输出看起来紧凑，使用函数int)来确保数字都是整数。

>>>for fileid in gutenberg.fileids():

·· num\_chars =len(gutenberg.raw(fileid))①

· num\_words =len(gutenberg.words(fileid))

. . num\_sents =len(gutenberg.sents(fileid))

· num\_vocab =len(set([w.lower()for w in gutenberg.words(fileid)]))

print int(num\_chars/num\_words),int(num\_words/num\_sents),int(num\_ words/num vocab),fileid

42126 sten-emma.txt

42316

42422

43379

4185

41714

41712

41612

41711

41911

41610

41824

42415

45210

4128

4137

4136

43512

austen-persuasion.txt

austen-sense.txt

bible-kjv.txt

blake-poems.txt

bryant-stories.txt

burgess-busterbrown.txt

carroll-alice.txt

chesterton-ball.txt

chesterton-brown.txt

chesterton-thursday.txt

edgeworth-parents.txt

melville-moby\_dick.txt

milton-paradise.txt

shakespeare-caesar.txt

shakespeare-hamlet.txt

shakespeare-macbeth.txt

whitman-leaves.txt

这个程序运行结果显示出每个文本的3个统计量：平均词长、平均句子长度和本 文中每个词出现的平均次数(词汇多样性得分)。平均词长看似是英语的一个一 般属性，因为它的值总是4。(事实上，平均词长是3而不是4,因为 num\_chars 变量计数了空白字符。)相比之下，平均句子长度和词汇多样性看上去是作者个 人的特点。

前面的例子也表明我们怎样才能获取“原始”文本”而不用把它分割成标识符。raw0

函数能在没有进行过任何语言学处理之前把文件的内容分析出来。例如：



len(gutenberg.raw('blake-poems.txt) 能告诉我们文本中出现的词汇个数，包括词之间 的空格。sents()函数把文本划分成句子，其中每一个句子是一个词链表。

>>>macbeth\_sentences =gutenberg.sents('shakespeare-macbeth.txt') >>>macbeth\_sentences

[['[','The','Tragedie','of','Macbeth','by','William','Shakespeare',

'1603',']'],['Actus','Primus','.'],.]

>>>macbeth\_sentences[1037]

['Double',',','double',',','toile','and','trouble',';',

'Fire','burne',',','and','Cauldron','bubble']

>>>longest\_len =max([len(s) for s in macbeth\_sentences]) >>>[s for s in macbeth\_sentences if len(s)==longest\_len]

[['Doubtfull','it','stood',',','As','two','spent','Swimmers',',','that',

'doe','cling','together',',','And','choake','their','Art',':','The',

'mercilesse','Macdonwald',..],...]

除了 words()、raw()和 sents()以外，大多数 NLTK 语料库阅读器还包 括多种访问方法。 一些语料库提供了更加丰富的语言学内容，例如： 词性标注、对话标记、句法树等。在后面的章节中，我们将介绍。

**网络和聊天文本**

虽然古腾堡项目包含成千上万的书籍，但它代表的是既定的文学。考虑较非正式的 语言也是很重要的。NLTK 中网络文本小集合的内容包括 Firefox 交流论坛、在纽 约无意听到的对话、《加勒比海盗》的电影剧本、个人广告和葡萄酒的评论。

>>>from nltk.corpus import webtext

>>>for fileid in webtext.fileids():

print fileid,webtext.raw(fileid)[:65],'...'

firefox.txt Cookie Manager:"Don't allow sites that set removed cookies

to se...

grail.txt SCENE 1:[wind][clop clop clop]KING ARTHUR:Whoa there!

[clop...

overheard.txt White guy:So,do you have any plans for this evening?Asian

girl...

pirates.txt PIRATES OF THE CARRIBEAN:DEAD MAN'S CHEST,by Ted Elliott & Terr...

singles.txt 25 SEXY MALE,seeks attrac older single lady,for discreet

encnne... Lovely delicate, fragrant Rhone wine.Polished leather and

strawb...

还有一个即时消息聊天会话语料库，最初由美国海军研究生院为研究自动检测互联网

入侵者而收集的。语料库包含超过10000个帖子，以 UserNNN 形式的通用名替换掉 用户名，并手工编辑消除其他身份信息。语料库被分成15个文件，每个文件包含几百 个从特定日期和特定年龄的聊天室(青少年、20岁、30岁、40岁，再加上一个通用 的成年人聊天室)收集的帖子。文件名中包含日期、聊天室和帖子数量，例如： 10-19-20s\_706posts.xml 表示2006年10月19日从20多岁聊天室收集的706个帖子。

>>>from nltk.corpus import nps\_chat

>>>chatroom =nps\_chat.posts('10-19-20s\_706posts.xml')

>>>chatroom[123]

['i','do',"n't",'want','hot','pics','of','a','female',',',

'I','can','look','in','a','mirror','.']

**布朗语料库**

布朗语料库是第一个百万词级的英语电子语料库，由布朗大学于1961年创建。这个语 料库包含500个不同来源的文本，按照文体分类，如新闻、社论等。表2-1 给出了各 个文体的例子(完整列表，请参阅htp:/icame.uib.no/brown/bcm-los.html)。

**表2-1** **布朗语料库每一部分的示例文档**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **文件** | **文** **体** | **描** **述** |
| A16 | cal6 | 新闻news | Chicago Tribune:Society Reportage |
| B02 | cb02 | 社论editorial | Christian Science Monitor:Editorials |
| C17 | cc17 | 评论reviews | Time Magazine:Reviews |
| D12 | cd12 | 宗教religion | Underwood:Probing the Ethics of Realtors |
| E36 | ce36 | 爱好hobbies | Norling:Renting a Car in Europe |
| F25 | cf25 | 传说lore | Boroff:Jewish Teenage Culture |
| G22 | cg22 | 纯文学belles lettres | Reiner:Coping with Runaway Technology |
| H15 | ch15 | 政府governmen | US Office of Civil and Defence Mobilization:The Family Fallout Shelter |
| J17 | cj19 | 博览learned | Mosteller:Probability with Statistical Applications |
| K04 | ck04 | 小说fiction | W.E.B.Du Bois:Worlds of Color |
| L13 | cl13 | 推理小说mystery | Hitchens:Footsteps in the Night |
| M01 | cm01 | 科幻science\_fiction | Heinlein:Stranger in a Strange Land |
| N14 | cn15 | 探险adventure | Field:Rattlesnake Ridge |
| P12 | cp12 | 言情romance | Callaghan:A Passion in Rome |
| R06 | cr06 | 幽默humor | Thurber:The Future,If Any,of Comedy |



我们可以将语料库作为词链表或者句子链表来访问(每个句子本身也是一个词链

**表)。我们可以指定特定的类别或文件阅读。**

>>>from nltk.corpus import brown

>>>brown.categories()

['adventure','belles\_lettres','editorial','fiction','government',

'hobbies',

'humor', 'learned','lore', 'mystery','news', 'religion', 'reviews', 'romance',

'science\_fiction']

>>>brown,words(categories='news')

o'rdy'22'])

 ,tegorie'a']views'])

[['The','Fulton','County'...],['The','jury','further'...],..]

布朗语料库是一个研究文体之间的系统性差异(又叫做**文体学**的语言学研究)的资 源。让我们来比较不同文体中的情态动词的用法。第一步：对特定文体进行计数。 记住进行下面的尝试之前要输入nltk。

>>>news\_text =brown.words(categories='news')

>>>from nltk.corpus import brown

>>>fdist =nltk.FreqDist([w.lower()for w in news\_text])

>>>modals =['can', 'could','may', 'might','must','will']

>>>for m in modals:

\* \* print m+':',fdist[m],

can:94 could:87 may:93 might:38 must:53 will:389

**轮到你来：**

选择布朗语料库的不同部分，修改前面的例子对包含 wh 的词进行计

*数，如：* *what、when 、where 、who和why.*

下面，统计每一个感兴趣的文体。我们使用 NLTK 提供的条件频率分布函数。在 2.2 节中会系统地把下面的代码一行行拆开来讲解。现在，你可以忽略细节，只看 输出。

>>>cfd

…\*

… \*

· ·

=nltk.ConditionalFreqDist(

(genre,word)

for genre in brown.categories()

for word in brown.words(categories=genre))

>>>genres=['news','religion','hobbies','science\_fiction','romance', 'humor']

>>>modals =['can','could','may','might','must','will'] >>>cfd.tabulate(conditions=genres,samples=modals)

can could may might must will

news

religion

hobbies science\_fiction

romance humor

93 86 66 38 50 389

82 59 78 12 54 71

268 58 131 22 83 264

16 49 4 12 8 16

74 193 11 51 45 43

16 30 8 8 9 13

观察发现新闻文体中最常见的情态动词是 will, 而言情文体中最常见的情态动词是 could。你能预测这些吗?这种可以区分文体的词计数方法将在第6章中详细介绍。

**路透社语料库**

路透社语料库包含10788个新闻文档，共计130万字。这些文档分成90个主题，按照 “训练”和“测试”分为两组。因此，编号为“test/14826”的文档属于测试组。这样分 割是为了方便运用训练和测试算法的自动检测文档的主题，我们将在第6章中学习。

>>>from nltk.corpus import reuters

>>>reuters.fileids()

['test/14826','test/14828','test/14829','test/14832',.…]

>>>reuters.categories()

['acq','alum','barley','bop','carcass','castor-oil','cocoa',

'coconut','coconut-oil','coffee','copper','copra-cake','corn',

'cotton','cotton-oil','cpi','cpu','crude','dfl','dlr',...]

与布朗语料库不同，路透社语料库的类别是互相重叠的，因为新闻报道往往涉及多 个主题。我们可以查找由一个或多个文档涵盖的主题，也可以查找包含在一个或多 个类别中的文档。为了方便起见，语料库方法既接受单个的标示也接受标示列表作 为参数。

>>>reuters.categories('training/9865')

['barley','corn','grain', 'wheat']

>>>reuters.categories(['training/9865','training/9880'])

['barley','corn','grain', 'money-fx','wheat']

>>>reuters.fileids('barley')

['test/15618','test/15649','test/15676','test/15728','test/15871',.…] >>>reuters.fileids(['barley','corn'])

['test/14832','test/14858','test/15033','test/15043','test/15106',

'test/15287','test/15341','test/15618','test/15618','test/15648',..]



**类似的，可以根据文档或类别查找我们想要的词或句子。这些文本中最开始的几个**

词是标题，按照惯例以大写字母存储。

>>>reuters.words('training/9865')[:14]

['FRENCH','FREE','MARKET','CEREAL','EXPORT','BIDS',

'DETAILED','French','operators','have','requested','licences','to', 'export']

>>>reuters.words(['training/9865','training/9880'])

>>>reuters.words(categories='barley')

['FRENCH','FREE','MARKET','CEREAL','EXPORT',·..]

,''(,']arley','corn'])

['THAI','TRADE','DEFICIT','WIDENS','IN','FIRST',..]

**就职演说语料库**

在1.1节中，我们学习了就职演说语料库，但是仅把它当作一个单独的文本对待。 在图1-2中使用“词偏移”作为其中的一个坐标轴，是语料库中词的索引数，从第 一个演讲的第一个词开始算起。然而，语料库实际上是55个文本的集合，每个文 本都是一个总统的演说。这个集合的一个显着特性是时间维度。

>>>from nltk.corpus import inaugural

>>>inaugural.fileids()

['1789-Washington.txt','1793-Washington.txt','1797-Adams.txt',...] >>>[fileid[:4]for fileid in inaugural.fileids()]

['1789','1793','1797','1801','1805','1809','1813','1817','1821',.]

需要注意的是，每个文本的年代都出现在它的文件名中。要从文件名中获得年代， 使用 fileid[:4]提取前4个字符。

让我们来看看词汇 america 和 citizen 随时间推移的使用情况。下面的代码使用 w.lower)“将就职演说语料库中的词汇转换成小写形式。然后用 startswith)° 检查它 们是否以“目标”词汇 america 或 citizen 开始。因此，它会计算如 American's 和 Citizens 等词。我们将在2.2节学习条件频率分布，现在只考虑输出，如图2-1所示。

>>>cfd =nltk.ConditionalFreqDist(

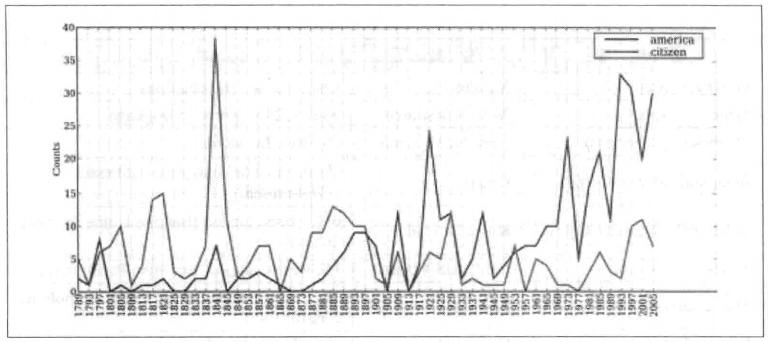
\* (target,file[:4])

· for fileid in inaugural.fileids() · for w in inaugural.words(fileid)

· if w.lower(),startswith(target))①

for target in ['america','citizen']

>>>cfd.plot()



**图2-1** **条件频率分布图：就职演说语料库中所有以** **america 或** **citizen 开始的词都将被计数。**

**每个演讲单独计数并绘制出图形，这样就能观察出随时间变化这些用法的演变趋势。计数没有与** **文档长度进行归一化处理**

**标注文本语料库**

许多文本语料库都包含语言学标注，有词性标注、命名实体、句法结构、语义角色 等。NLTK 中提供了几种很方便的方法来访问这几个语料库，而且还包含有语料库 和语料样本的数据包，用于教学和科研时，可以免费下载。表2-2列出了其中一些 语料库。有关下载信息请参阅 <http://www.nltk.org/data>。关于如何访问NLTK 语料库 的其他例子，请在 <http://www.nltk.org/howto> 上查阅语料库的 HOWTO。

**表2-2** **NLTK** **中的一些语料库和语料库样本**

|  |  |  |
| --- | --- | --- |
| **语** **料** **库** | **编** **译** **器** | **内** **容** |
| Brown Corpus | Francis,Kucera | 15 genres,1.15M words,tagged,categorized |
| CESS Treebanks | CLiC-UB | 1M words,tagged and parsed(Catalan, Spanish) |
| Chat-80 Data Files | Pereira &Warren | World Geographic Database |
| CMU Pronouncing Dictionary | CMU | 127k entries |
| CoNLL 2000 Chunking Data | CoNLL | 270k words,tagged and chunked |
| CoNLL 2002 Named Entity | CoNLL | 700k words,pos and named-entity-tagged (Dutch,Spanish) |
| CoNLL 2007 Dependency Treebanks (selections) | CoNLL | 150k words,dependency parsed(Basque, Catalan |
| Dependency Treebank | Narad | Dependency parsed version of Penn Treebank sample |
| Floresta Treebank | Diana Santos et al | 9k sentences,tagged and parsed (Portuguese) |



续表

|  |  |  |
| --- | --- | --- |
| 语 料 库 | **编** **译** **器** | 内 容 |
| Gazetteer Lists | Various | Lists of cities and countries |
| Genesis Corpus | Misc web sources | 6 texts,200k words,6 languages |
| Gutenberg (selections) | Hart,Newby,et al | 18 texts,2M words |
| Inaugural Address Corpus | CSpan | US Presidential Inaugural Addresses (1789-present) |
| Indian POS-Tagged Corpus | Kumaran et al | 60k words,tagged (Bangla,Hindi,Marathi Telugu |
| MacMorpho Corpus | NILC,USP,Brazil | 1M words,tagged (Brazilian Portuguese) |
| Movie Reviews | Pang,Lee | 2k movie reviews with sentiment polarity classification |
| Names Corpus | Kantrowitz,Ross | 8k male and female names |
| NIST 1999 Info Extr  (selections) | Garofolo | 63k words,newswire and named-entity SGML markup |
| NPS Chat Corpus | Forsyth,Martel | 10k IM chat posts,POS-tagged and dialogue act tagged |
| Penn Treebank(selections) | LDC | 40k words,tagged and parsed |
| PP Attachment Corpus | Ratnaparkhi | 28k prepositional phrases,tagged as noun or verb modifiers |
| Proposition Bank | Palmer | 113k propositions,3300 verb frames |
| Question Classification | Li,Roth | 6k questions,categorized |
| Reuters Corpus | Reuters | 1.3M words,10k news documents,categorized |
| Roget's Thesaurus | Project Gutenberg | 200k words,formatted text |
| RTE Textual Entailment | Dagan et al | 8k sentence pairs,categorized |
| SEMCOR | Rus,Mihalcea | 880k words,part-of-speech and sense tagged |
| Senseval 2 Corpus | Pedersen | 600k words,part-of-speech and sense tagged |
| Shakespeare texts  (selections) | Bosak | 8 books in XML format |
| State of the Union Corpus | CSpan | 485k words,formatted text |
| Stopwords Corpus | Porter et a | 2400 stopwords for 11 languages |
| Swadesh Corpus | Wiktionary | comparative wordlists in 24 languages |
| Switchboard Corpus  (selections) | LDC | 36 phone calls,transcribed,parsed |
| Univ Decl of Human Rights | United Nations | 480k words,300+languages |
| TIMIT Corpus (selections) | NIST/LDC | audio files and transcripts for 16 speakers |
| VerbNet 2.1 | Palmer et al | 5k verbs,hierarchically organized,linked to WordNet |
| Wordlist Corpus | OpenOffice.org et al | 960k words and 20k affixes for 8 languages |
| WordNet 3.0 (English) | Miller,Fellbaum | 145k synonym sets |

50

第2章



**其他语言的语料库**

**NLTK** 包含多国语言语料库。某些情况下在使用这些语料库之前需要学习如何在 Python 中处理字符编码(见3.3节)。

>>>nltk.corpus.cess\_esp.words()

['El','grupo','estatal','Electricit\xe9\_de\_France',..]

>>>nltk.corpus.floresta.words()

['Um','revivalismo','refrescante','0','7 e Meio',...]

>>>nltk.corpus.indian.words('hindi.pos')

['\xe0\xa4\xaa\xe0\xa5\x82\xe0\xa4\xb0\xe0\xa5\x8d\xe0\xa4\xa3',

'\xe0\xa4\xaa\xe0\xa5\x8d\xe0\xa4\xb0\xe0\xa4\xa4\xe0\xa4\xbf\xe0\xa4\ xac\xe0\xa4



['Abkhaz-Cyxillic+Abkh','Abkhaz-UTF8','Achehnese-Latin1','Achuar- Shiwiar-Latin1',

'Adja-UTF8','Afaan\_Oromo\_Oromiffa-Latinl','Afrikaans-Latin1','Aguaruna- Latin1',

'Akuapem\_Twi-UTF8', 'Albanian\_Shqip-Latinl','Amahuaca','Amahuaca- Latinl',...]

>>>nltk.corpus.udhr.words('Javanese-Latin1')[11:]

[u'Saben',u'umat',u'manungsa',u'lair',u'kanthi',...]

这些语料库的最后一个是udhr, 包含有超过300种语言的世界人权宣言。这个语料库 的 fileids 包括有关文件所使用的字符编码信息，如： UTF8 或者 Latin1。利用条件 频率分布来研究“世界人权宣言”(udhr) 语料库中不同语言版本中的字长差异。 图2-2所示及输出(自己运行程序可以看到一个彩色图)。注意：True 和 False 是 Python 内置的布尔值。

>>>from nltk.corpus import udhr

>>>languages =['Chickasaw','English','German\_Deutsch', 'Greenlandic\_Inuktikut','Hungarian\_Magyar','Ibibio\_Efik']

>>>cfd =nltk.ConditionalFreqDist(

· · (lang,len(word))

· for lang in languages

for word in udhr.words(lang +'-Latin1'))

>>>cfd.plot(cumulative=True

**轮到你来：**

**在** **udhr.fileidsO中选择一种感兴趣的语言，定义一个变量** **raw\_text= udhr.raw(Language-Latinl)。使用** **nltk.FreqDist(raw\_text).plot()画出此** **文本的字母频率分布图。**

|  |  |  |
| --- | --- | --- |
| Cumulative Percentage | | |
| 16  Samples | 京 | 33  品 |

**图2-2** **累积字长分布：内容是“世界人权宣言”的6个翻译版本；此图显示：5个或5个以下** **字母组成的词在** **lbibio语言的文本中占约80%,在德语文本中占60%,在** **Inuktitut文本中占25%**

遗憾的是，许多语言没有大量的语料库。通常是政府或工业对发展语言资源的支持 不够。个人的努力是零碎的，难以发现或重用。有些语言没有既定的书写系统，或 濒临灭绝。(2.7节有关于如何寻找语言资源的建议。)

**文本语料库的结构**

到目前为止，我们已经学习了大量的语料库结构。在图2-3中有总结。最简单的一种没 有任何结构，仅仅是一个文本集合。通常，文本会按照其可能对应的文体、来源、作者、 语言等分类。有时，这些类别会重叠，尤其是在按主题分类的情况下，因为一个文本可 能与多个主题相关。偶尔的，文本集会有时间结构，新闻集合是最常见的例子。

|  |  |  |  |
| --- | --- | --- | --- |
| isolated categorized overlapping temporal | | | |
| e.9.gutenberg, webtext,uanr | e.9.Drown | 0.9 reuters | e.g inaugural |

**图2-3** **文本语料库的常见结构：最简单的一种语料库是一些孤立的没有什么特别结构的文本集**

合； 一些语料库按如文体(布朗语料库)等分类成组织结构； 一些分类会重叠，如主题类别(路 透社语料库);另外一些语料库可以表示随时间变化，语言用法的改变(就职演说语料库)

NLTK 语料库阅读器能高效地访问大量语料库，并且能用于处理新的语料库。 表2-3列出了语料库阅读器提供的函数。

表2-3 NLTK 中定义的基本语料库函数：使用help(nltk.corpus.reader)可以找到 更多的文档，也可以阅读 <http://www.nltk.org/howto>上的在线语料库的HOWTO

|  |  |
| --- | --- |
| **示** **例** | **描** **述** |
| fileids( | 语料库中的文件 |
| fileids([categories]) | 分类对应的语料库中的文件 |
| categories() | 语料库中的分类 |
| categories([fileids]) | 文件对应的语料库中的分类 |
| raw() | 语料库的原始内容 |
| raw(fileids=[fl,f2,f3]) | 指定文件的原始内容 |
| raw(categories=[cl,c2]) | 指定分类的原始内容 |
| words) | 整个语料库中的词汇 |
| words(fileids=[fl,f2,f3]) | 指定文件中的词汇 |
| words(categories=[cl,c2]) | 指定分类中的词汇 |
| sents)) | 指定分类中的句子 |
| sents(fileids=[f1,f2,f3]) | 指定文件中的句子 |
| sents(categories=[cl,c2]) | 指定分类中的句子 |
| abspath(fileid) | 指定文件在磁盘上的位置 |
| encoding(fileid) | 文件的编码(如果知道的话) |
| open(fileid) | 打开指定语料库文件的文件流 |
| root() | 到本地安装的语料库根目录的路径 |
| readme() | 语料库的README文件的内容 |

这里说明了一些语料库访问方法之间的区别。

>>>raw =gutenberg.raw("burgess-busterbrown.txt")

>>>raw[1:20]

'The Adventures of B'

>>>words =gutenberg.words("burgess-busterbrown.txt")

>>>words [1:20]

['The','Adventures','of','Buster','Bear','by','Thornton','W','.',

'Burgess','1920',']','I','BUSTER','BEAR','GOES','FISHING','Buster', 'Bear']

>>>sents =gutenberg.sents("burgess-busterbrown.txt")

>>>sents[1:20]

[['I'],['BUSTER','BEAR','GOES','FISHING'],['Buster','Bear','yawned','as',

'he','lay','on','his','comfortable','bed','of','leaves','and','watched', 'the','first','early','morning','sunbeams','creeping','through',

...]

…],

**载入你自己的语料库**

如果你有自己收集的文本文件，并想使用前面讨论的方法来访问它们，在NLTK 中 的PlaintextCorpusReader 帮助下，你能很容易地载入它们。在你的文件系统中检查 文件的位置；在下面的例子中，假定你的文件在/usr/share/dict 目录下。不管是什么 位置，将变量 corpus\_root°的值设置为这个目录。PlaintextCorpusReader 初始化函数 °的第二个参数可以是一个如[a.txt,test/b.txt] 这样的fileids 链表，或者一个匹配所 有标示的模式，如："[abc].\*\.txt (关于正则表达式的信息见3.4节)。

s\_nrlotcorpus m'/upshaCorpusReader

>>>wordlists =PlaintextCorpusReader(corpus\_root,'.\*')② >>>wordlists.fileids()

['README','connectives','propernames','web2','web2a','words']

>>>wordlists.words('connectives')

['the','of','and','to','a','in','that','is',...]

下面是另一个例子，假设你在本地硬盘上有自己的宾州树库(第3版)的副本，放 在C:lcorpora。我们可以使用BracketParseCorpusReader 来访问这些语料。我们指定 corpus\_root 来存放语料库中已解析过的《华尔街日报》部分°,并指定 file\_pattern 与它的子文件夹中所包含的文件匹配②(用前斜杠)。

rlmpeu\_spa\_nlr". \_tcoboarnpku\rg\wsj"①

>>>ptb =BracketParseCorpusReader(corpus\_root,file\_pattern) >>>ptb.fileids()

['00/wsj\_0001.mrg','00/wsj\_0002.mrg','00/wsj\_0003.mrg',

'00/wsj\_0004.mrg',..]

>>>len(ptb.sents())

.sents(fileids='20/wsj\_2013.mrg')[19]

['The','55-year-old','Mr.','Noriega','is',"n't",'as','smooth','as',

'the',

'shah','of','Iran',',','as','well-born','as','Nicaragua',"'s",

'Anastasio',

'Phi'Sloi,mperial','as','Ferdinand','Marcos','of','the',

'or','as','bloody','as','Haiti',"'s",'Baby',Doc','Duvalier','.']

**2.2** **条件频率分布**

在1.3 节中介绍了频率分布，并学习了在给定某个词汇或其他项目的链表变量 mylist,FreqDist(mylist) 就会计算链表中每个项目出现的次数。在这里，我们将推 广这一想法。

当语料文本分为几类(文体、主题、作者等)时，可以计算每个类别独立的频 率分布。这样，就可以研究类别之间的系统性差异。在上一节中，我们是用 NLTK 的 ConditionalFreqDist 数据类型来实现的。**条件频率分布**是频率分布的集 合，每个频率分布有一个不同的“条件”。这个条件通常是文本的类别。图2-4 描绘了一个仅带两个条件的条件频率分布的片段， 一个是新闻文本， 一个是言 情文本。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Condition:News Condition:Romance   |  |  | | --- | --- | | the | HHHH I | | cute |  | | Monday | 州 | | could | | | | will | #Ⅲ |  |  |  | | --- | --- | | the | #H | | cute | Ⅲ | | Monday | | | | could | HHH | | will |  | |

**图2-4** **计算文本集合中单词出现的次数(条件频率分布)**

**条件和事件**

频率分布计算观察到的事件，如文本中出现的词汇。条件频率分布需要给每个事件 关联一个条件，所以不是处理一个词序列°,而是要处理一系列配对序列②。

>>>text

>>>pairs

=['The','Fulton','County','Grand','Jury','said',..]① =[('news','The'),('news','Fulton'),('news','County'),..]②

每对的形式是：(条件，事件)。如果我们按文体处理整个布朗语料库，将有15个 条件(一个文体一个条件)和1161192个事件(一个词一个事件)。

**按文体计数词汇**

在2.1节中，我们学习了一个以布朗语料库的某一部分为条件的条件频率分布，并按 照每个条件计数词汇。FreqDist)以一个简单的链表作为输入， ConditionalFreqDist)

**以一个配对链表作为输入。**

>>>from nltk.corpus import brown

>>>cfd





·

..

=nltk.ConditionalFreqDist(

(genre,word)

for genre in brown.categories()

for word in brown.words(categories=genre))

让我们拆开来看，只看两个文体：新闻和言情。对于每个文体②,遍历文体中的每 个词以产生文体与词的配对°。

>>>genre\_word

\* ·

·

=[(genre,word)①

**f** wrien bwn[('rcoamteagn]e②s=genre)]③

>>>len(genre\_word)

170576

因此，在下面的代码中可以看到：链表genre\_word的前几个配对将是(news,word)

◎的形式，而最后几个配对将是(romance',word)² 的形式。

>>>genre\_word[:4]

[('news','The'),('news','Fulton'),('news','County'),('news','Grand')]① >>>genre\_word[-4:]

[('romance','afraid'),('romance','not'),('romance',"''"),('romance',

'.')]②

现在，可以使用此配对链表创建一个 ConditionalFreqDist, 并将它保存在一个变量 cfd 中。像往常一样，可以输入变量的名称来检查它”,并确认它有两个条件②:

>>>cfd

>>>cfd ①

=nltk.ConditionalFreqDist(genre\_word)

<ConditionalFreqDist with 2 conditions>

>>>cfd.conditions()

['news','romance']②

访问这两个条件，它们每一个都只有一个频率分布。

>>>cfd['news']

<FreqDist with 100554 outcomes>

>>>cfd['romance']

<FreqDist with 70022 outcomes>

>>>list(cfd['romance'])

[',','.','the','and','to','a','of','',"'",'was','I','in',

'he','had',

?','her','that','it','his','she','with','you','for','at','He',

'on','him',

'said','!','--','be','as',';','have','but','not','would','She',

"The', .]

>>>cfd['romance']['could']

193

**绘制分布图和分布表**

除了组合两个或两个以上的频率分布及更容易初始化之外， ConditionalFreqDist 还为 制表和绘图提供了一些有用的方法。

图2- 1 是根据下面的代码重制出来的一个条件频率分布绘制的。条件是词 america 或citizen², 绘图中的计数是指在特定演讲中出现该词的次数。它利用了每 个演讲的文件名——例如1865-Lincoln.txt——前 4 个字符包含了年代信息的特 点“。这段代码为文件1865-Lincoln.txt 中每个以 america 小写形式开头的词——如： Americans——产生一个配对(america','1865)。

>>>from nltk.corpus import inaugural

>>>cfd





\*

(target,fileid[:4])①

=nltk.ConditionalFreqDist(

for fileid in inaugural.fileids()

for w in inaugural.words(fileid)

for target in ['america','citizen']②

if w.lower().startswith(target))

图2-2也是基于下面的代码重制出来的一个条件频率分布图。条件是语言的名称， 图中的计数来源于单词长度°。它利用了这样一个特点：即每一种语言的文件名是 语言名称及后面紧跟着'-Latin1' (字符编码)。

>>>from nltk.corpus import udhr

>>>languages=['Chickasaw','English','German\_Deutsch',

.. 'Greenlandic\_Inuktikut','Hungarian\_Magyar','Ibibio\_Efik']

>>>cfd =nltk.ConditionalFreqDist(

(lang,len(word))①

... for lang in languages

for word in udhr.words(lang +'-Latin1'))

在 plot)和 tabulate)方法中，可以使用 conditions= 参数来指定显示哪些条件。如果



我们忽略它，所有条件都会显示出来。同样，可以使用 samples= 参数来限制要显 示的样本。这能将大量数据载入到一个条件频率分布，然后通过选定条件和样品， 对完成的绘图或制表进行探索。这也使我们能全面控制条件和样本的显示顺序。例 如：可以为两种语言和长度少于10个字符的词汇绘制累计频率数据表，如下所示。 我们可以解释最上排最后一个单元格中数值的含义是英文文本中9个或少于9个字 符长的词有1638个。

>>>cfd.tabulate(conditions=['English','German\_Deutsch'],

samples=range(10),cumulative=True)

0 1 2 3 4 5 6 7 8 9!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| English German\_Deutsch | 0  0 | 185  171 | 525  263 | 883 99711661283144015581638  614 717 8941013111012131275 |

**轮到你来：**

处理布朗语料库的新闻和言情文体，找出一周中最有新闻价值并且是最 浪漫的日子。定义一个变量days 其中包含星期的链表，如[Monday,…]。 然后使用 cfd.tabulate(samples=days)为这些词的计数制表。接下来用绘图 替代制表尝试同样的事情。你可以在另一个参数conditions=[Monday',…] 的帮助下控制星期输出的顺序。

你可能已经注意到：已经使用的条件频率分布多行表达式看上去像链表推导，但是 不带括号。通常，我们使用链表推导作为一个函数的参数，如：set([w.lower for w in t]), 我们可以忽略掉方括号而只写set(w.lower)for w in t)。(更多的讲解请参见4.2 节的“产生器表达式”。)

**使用双连词生成随机文本**

我们可以使用条件频率分布创建一个双连词表(词对，在1.3节介绍过)。bigrams) 函 数能接受一个词汇链表，并建立起一个连续的词对链表。

>>>sent=['In','the','beginning','God','created','the','heaven',

'and','the','earth','.']

>>>nltk.bigrams(sent)

[('In','the'),('the','beginning'),('beginning','God'),('God',

'created'),

('created','the'),('the','heaven'),('heaven','and'),('and','the'), ('the','earth'),('earth','.')]

在例2-1 中，将每个词作为一个条件，对于每个词都有效地依据后续词的创建频率 分布。函数 generate\_model) 包含简单的循环以生成文本。在调用函数时，选择一 个词(如 living”) 作为初始内容。进入循环后，输入变量word 的当前值，重新设 置 word为上下文中最可能的标识符(使用max)) 。 下一次进入循环时，将这个词 作为新的初始内容。通过检查输出可以发现，这种简单的文本生成方法往往会在循 环中卡住。另一种方法是从可用的词汇中随机选择下一个词。

例2-1.产生随机文本：此程序获得了《创世纪》文本中所有的双连词，然后构造 一个条件频率分布来记录哪些词汇最有可能会跟在给定词的后面，例如： living 后 面最可能的词是 creature 。generate\_model)函数使用这些数据和种子词来产生随机 文本。

def generate\_model(cfdist,word,num=15):

for i in range(num):

st[word].max()

text =nltk.corpus.genesis.words('english-kjv.txt')

bigrams =nltk.bigrams(text)

cfd =nltk.ConditionalFreqDist(bigrams)①

>>>print cfd['living']

<FreqDist:'creature':7,'thing':4,'substance':2,',':1,'.':1,'soul':1>

>>>generate\_model(cfd,'living')

living creature that he said,and the land of the land of the land

条件频率分布是一个对许多 NLP 任务都有用的数据结构。表2-4总结了它们常用

的方法。

**表2-4** **NLTK 中的条件频率分布：定义、访问和可视化一个计数条件**

**频率分布的常用方法和习惯用法**

|  |  |
| --- | --- |
| **示** **例** | **描** **述** |
| cfdist=ConditionalFreqDist(pairs) | 从配对链表中创建条件频率分布 |
| cfdist.conditionsO | 将条件按字母排序来分类 |
| cfdist[condition] | 此条件下的频率分布 |
| cfdist[condition][sample] | 此条件下给定样本的频率 |
| cfdist.tabulate( | 为条件频率分布制表 |
| cfdist.tabulate(samples,conditions) | 在指定样本和条件限制下制表 |

续表

|  |  |
| --- | --- |
| **示** **例** | **描** **述** |
| cfdist.plot) | 为条件频率分布绘图 |
| cfdist.plot(samples,conditions) | 在指定样本和条件限制下绘图 |
| cfdist1<cfdist2 | 测试样本在cfdist1中出现次数是否小于在cfdist2中出现次数 |

**2.3** **更多关于** **Python: 代码重用**

目前为止，你可能已经在 Python 交互式解释器中输入和重新输入过大量的代码。 如果在输入一个复杂的例子时弄得一团糟，那么就必须要重新输入。虽然使用箭头 键来访问和修改以前的命令可能会有所帮助，但也很有限。在本节中，我们将讲述 代码重用的两个重要方式：文本编辑器和 Python 函数。

**使用文本编辑器创建程序**

Python 交互式解释器在输入命令后就会立即执行。通常，使用文本编辑器来编写多 行程序，然后让Python 一次运行整个程序会更好。使用IDLE, 你就能通过“文件” 菜单打开一个新窗口来实现这些。现在就输入下面的程序，尝试一下。

print 'Monty Python'

保存这段程序，命名为 monty.py。然后打开“运行”菜单，选择运行模块命令。(我 们将很快学习到什么是模块。)IDLE 主窗口的结果如下。

>>>=== RESTART=

>>>

Monty Python

>>>

也可以输入 from monty import \*,这样能达到同样的效果。

从现在起，你可以选择使用交互式解释器或文本编辑器来创建你的程序。使用解释器测 试你的想法往往比较方便，修改代码直到达到你期望的效果。测试好之后，你就可以将 代码粘贴到文本编辑器(去除所有>>>和…提示符),然后继续扩展它。最后将程序保存 为文件，这样以后就不用再重新输入了。给文件一个简短而描述性的名字，使用所有的 小写字母，并用下划线分割词汇，再以py 文件名后缀，例如： monty\_python.py。



**要点：**

联机代码的例子中包含>>>和...提示符，仿佛我们正在直接与解释器交互。 随着程序变得更加复杂，应该在编辑器中输入程序，不管提示符，如前面所 示的那样在编辑器中运行。在这本书中提供更长的程序时，将省略提示符以 提醒你在文件中输入程序而不是使用解释器，如示例2-1所示。请注意，例 子中还有几行代码带有 Python 提示符，这是任务的互动部分，在该部分你 可以观察数据，并调用函数。请记住，诸如例2-1的所有示例代码都可以从 http://www.nltkorg/下载。

**函数**

假设你正在分析一些文本，这些文本中包含同一个词的不同形式，而且需要利用一 部分程序将给定的单数名词变成复数形式。假设需要在两个地方做这样的事， 一个 是在处理文本时，另一个是在处理用户输入时。

比起多次重复运用相同的代码，把代码放在**函数**中会更有效和可靠。函数是指带命 名的代码块，可执行一些明确的任务，就像我们在1.1节中所看到的那样。函数通 常使用称为**参数**的变量进行输入，并且可能会生成一些结果，也称为**返** **回** **值** **。**我

们使用关键字 def 加函数名及所有输入参数来定义一个函数，接下来是函数的主 体。下面是我们在1.1节中看到的函数(包括 import 声明语句来运行所期望的除

法运算)。

>>>from \_future\_import division

>>>def lexical\_diversity(text):

return len(text)/len(set(text))

使用关键字 return 来表示函数输出所产生的值。在这个例子中，函数所有的工作都 在 return 语句中完成。下面是一个等价的定义，使用多行代码进行同样的运算。我 们将参数的名称从text 变为my\_text\_data, 注意这是任意选择的。

>>>def lexical\_diversity(my\_text\_data):

· · word\_count =len(my\_text\_data)

zye\_score =(dset\_ext\_ ab\_size

. · return diversity\_score

请注意，我们已经在函数体内部创造了一些新的变量。这些是**局部变量，**不能在函

数体外访问。定义一个名为 lexical\_diversity 的函数。但只定义它，并不会有任何 输出!函数在被“调用”之前不会起任何作用。

回到前面的场景，实际定义一个简单的函数来处理英文的复数名词。例2-2中的函 数 pluralO在输入单数名词后可产生其复数形式，虽然它并不总是正确的。(我们将 在4.4节中进行更详细的介绍。)

例2-2.一个 Python 函数：这个函数试图生成任意英语名词的复数形式。关键字 def(define) 的后面是函数的名称，然后是包含在括号内的参数和一个冒号；函数 体是缩进代码块；它试图识别词内的模式，并相应地对词进行处理；例如，如果这 个词以y 结尾，删除它们并添加 ies。

def plural(word):

if word.endswith('y'):

elif rd[-1]inwox[:'iord[-2:]in ['sh','ch']:

return word + 'es'

elif word.endswith('an'):

return word[:-2]+'en'

else:

return word +'s'

>>>plural('fairy')

'fairies'

>>>plural('woman')

'women'

endswith() 函数总是与字符串对象一起使用(如：例2-2中的 word) 。 要调用此 函数，首先输入对象的名字和“点”,然后是函数的名称。这些函数通常被称为

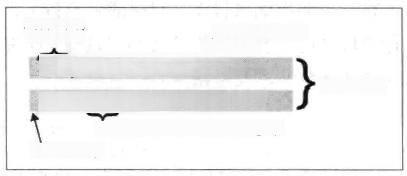
**方法。**

**模块**

随着时间的推移，你将会发现你创建了大量简短而实用的文字处理函数，并不停地 把它们从老程序复制到新程序中。哪个文件中包含的才是你要使用的函数的最新版 本?如果你能把你的劳动成果收集在一个处，并且不通过复制便可调用之前定义好 函数，工作将会更加轻松。

要做到这一点，需要将函数保存到文件：textproc.py 中。现在，可以通过从文件导 入轻松访问函数。





or lemma *,Sense defnition,*

*saw,Iverbl,past tense ofsee.*

*Homonyms*

saw,[noun],cutting instrument.

*Part of speech,or lexical category*

*Lexical entry*

**图2-5** **词典术语：两个拼写相同的词条(同音**

**异义词)的词汇项，包括词性和注释信息**

orgloss

*Headword,*

>>>from textproc import plural

>>>plural('wish')

wishes

>>>plural('fan')

fen

显然，我们的复数函数存在错误，因为 fan 的复数是 fans 。不必重新输入这个函数 的新版本，简单地编辑现有的即可。因此，在任何时候复数函数只有一个版本，不 用担心使用的是哪个版本。

一个文件中的变量和函数定义的集合被称为 Python 模 块 (module) 。 相关模块 的集合称为包 (package)。 处理布朗语料库的 NLTK 代码是模块，处理各种不 同语料库的代码的集合是包。NLTK 的本身是包的集合，有时被称为库 (library)。

注意!

如果你正在创建一个包含一些你自己的 Python 代码的文件， 一定不 要将文件命名为nltk.py。因为这可能会在导入时占据“真正的”NLTK 包。当Python 导入模块时，它先查找当前目录(文件夹)。

**2.4** **词典资源**

词典或者词典资源是一个词和/或短语及其相关信息的集合，例如：词性和词意定 义等相关信息。词典资源附属于文本，而且通常在文本的基础上创建和丰富。例如： 如果定义一个文本 my\_text, 然后通过 vocab =sorted(set(my\_text))建立 my\_text 的词汇表，再利用 word\_freq =FreqDist(my\_text)计数文本中每个词的频率。vocab

和 word\_freq 都是简单的词汇资源。

同样，如在1.1节中学习到的，词汇索

引为我们提供了有关词语用法的信

息，对编写词典有用。图2-5中描述了

词汇相关的标准术语。词项包括词目

(也叫**词条)** 及其他附加信息，例如：

词性和词意定义。两个含义不同但拼

写相同的分词被称为**同音异义词。**

一种简单的词典资源仅有一个词汇列表。复杂的词典资源包括单个条目内部或相互

之间的复杂结构。在本节， 一起来学习NLTK 中的一些词典资源。

**词汇列表语料库**

NLTK 中包括一些仅仅包含词汇列表的语料库。词汇语料库是 UNIX 中的/usr/dict/ words 文件，被一些拼写检查程序所使用。我们可以用它来寻找文本语料中不常见 的或拼写错误的词汇，如例2-3所示。

例2-3.过滤文本：此程序能计算文本的词汇表，然后删除所有在现有的词汇列表

中出现的元素，只留下罕见的或拼写错误的词汇。

def unusual\_words(text):

text\_vocab =set(w.lower()for w in text if w.isalpha())

english\_vocab =set(w.lower()for w in nltk.corpus.words.words()) unusual =text\_vocab.difference(english\_vocab)

return sorted(unusual)

>>>unusual\_words(nltk.corpus.gutenberg.words('austen-sense.txt'))

['abbeyland','abhorrence','abominably','abridgement','accordant',

'accustomary',

'adieus','affability','affectedly','aggrandizement','alighted','allenham',

'amiably','annamaria','annuities','apologising','arbour',

'archness',...]

>>>unusual\_words(nltk.corpus.nps\_chat.words())

['aaaaaaaaaaaaaaaaa','aaahhhh','abou','abourted','abs','ack','acros',

'actualy','adduser','addy','adoted','adreniline','ae','afe','affari',

'afk',

'agaibn','agurlwithbigguns','ahah','ahahah','ahahh','ahahha','ahem', 'ahh', ..]

还有 一**个停用词**语料库，包括的是高频词汇，如： the、to 和 and, 有时在进一步进 行处理之前需要将它们从文档中过滤。停用词通常没有什么词汇内容，而它们的出

现会使区分文本变得困难。

>>>from nltk.corpus import stopwords

>>>stopwords.words('english')

['a',"a's",'able','about','above','according','accordingly','across',

'actually','after','afterwards','again','against',"ain't",'all','allow',

'allows','almost','alone','along','already','also','although',

'always',...]

定义一个函数来计算文本中不包含在停用词列表中的词所占的比例。

>>>def content\_fraction(text):

…

\* ·

·

stopwords =nltk.corpus.stopwords.words('english')

content =[w for w in text if w.lower()not in stopwords] return len(content)/len(text)

·

.

>>>content\_fraction(nltk.corpus.reuters.words())

0.65997695393285261

因此，在停用词的帮助下，筛选掉文本中三分之一的词。请注意，我们在这里结合 了两种不同类型的语料库，需要使用词典资源来过滤文本语料的内容。

词汇列表对解决如图2-6所示的词谜很有用。运行程序遍历每一个词，检查每一个 词是否符合条件。检查必须出现的字母②和长度限制°是很容易的(这里我们只查 找6个或6个以上字母的词)。只使用指定的字母组合作为候选方案，尤其是在一 些指定的字母出现了两次(如字母 v) 时，这样的检查是很棘手的。利用 FreqDist 比较法③检查候选词中的每个字母出现的频率是否小于或等于其相应在词谜中 出现的频率。

>>>puzzle\_letters =nltk.FreqDist('egivrvonl')

>>>obligatory ='r'

rstw lodrlpid(iw.t=(①n w②

\* and nltk.FreqDist(w)<=puzzle\_letters]③

['glover','gorlin','govern','grovel','ignore','involver','lienor', 'linger','longer','lovering','noiler','overling','region','renvoi', 'revolving','ringle','roving','violer','virole']

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 母在每个词中只能被用一次。每个词包括中间的字母并且至少有一个9 字母的词。没有复数以 “s” 结尾；没有外来词；没有姓名。能组出21  **个词就是“好”;32个词，“很好”;42个词，“非常好”。**   |  |  |  | | --- | --- | --- | | E | G | I | | V | R | V | | 0 | N | L |   **用这里显示的字母你能组成多少个4字母或者更多字母的词?每个字** |

**图2-6** **词谜：在由随机选择的字母组成的网格中，选择里面的字母组成单词。这个谜题叫** **做“目标”**

另一个词汇列表是名字语料库，包括8000个按性别分类的名字。男性和女性的名字存 储在单独的文件中。找出同时出现在两个文件中的名字即分辨不出性别的名字。

for fileid in names.fileids()

>>>names =nltk.corpus.names

>>>names.fileids()

['female.txt','male.txt']

>>>male\_names =names.words('male.txt')

>>>female\_names =names.words('female.txt')

>>>[w for w in male\_names if w in female\_names]

['Abbey','Abbie','Abby','Addie','Adrian','Adrien','Ajay','Alex','Alexis', 'Alfie','Ali','Alix','Allie','Allyn','Andie','Andrea','Andy','Angel',

'Angie','Ariel','Ashley','Aubrey','Augustine','Austin','Averil',.]

众所周知，以字母a 结尾的名字几乎都是女性。在图2-7中不仅可以看到这一点， 还可以看到其他的例子，该图是由下面的代码产生的。请记住 name[-1]是name 的最 后一个字母。

>>>cfd =nltk.ConditionalFreqDist(

… (fileid,name[-1])

\* for name in names.words(fileid))

>>>cfd.plot()

|  |  |  |
| --- | --- | --- |
| 1800  Counts  1600  1400|  1200  100  800  600  400  200  A U 安 | 8 # ○ A  Samples | female.txt  male.txt  2 ¥ 网 A |

**图2-7条件频率分布：此图显示男性和女性名字的结尾字母；大多数以a、e或i结尾的名字是** **女性；以h 和1结尾的名字男性和女性同样多；以** **k、o、r、s和t 结尾的更可能是男性**

**发音的词典**

表格(或电子表格)是一种略微丰富的词典资源，在每一行中含有一个词及其一些 性质。NLTK 中包括美国英语的CMU 发音词典，它是为语音合成器而设计的。

('fire',['F','AY1','R'])

>>>entries =nltk.corpus.cmudict.entries()

>>>len(entries)

127012

>>>for entry in entries[39943:39951]:

· print entry

('f·ir', ['F', 'ER1'])

('fire', ['F','AY1','ERO'])

('firearm',['F','AYl','ERO','AA2','R','M'])

('firearm',['F','AY1','R','AA2','R','M'])

('firearms',['F','AY1','ER0','AA2','R','M','Z'])

**(**''fib',[a'Fl',l'A'1'R'',F'A'A,2Y,'M1',''Z,'])'ER0','B','AO2','L'])

对任意一个词，词典资源都有语音的代码——不同的声音有着不同的标签——称做 音素。fire有两个发音(美国英语中):单音节FAY1R 和双音节FAY1ER0。 CMU 发音词典中的符号是从 Arpabet 来的，更多的细节请参考 http://en.wikipedia. org/wiki/Arpabet。

每个条目由两部分组成，我们可以用一个复杂的 for 语句分别进行处理。我们没有 用for entry in entries:, 而是用两个变量名word和 pron°替换了 entry。现在，每次 通过循环时， word 被分配到条目的第一部分；pron 被分配到条目的第二部分。

>>>for word,pron in entries:①

if len(pron)==3:②

… · ph1,ph2,ph3 =pron③

… · if ph1 =='P'and ph3 =='T':

… ·· print word,ph2,

\* ·

pait EYl pat AE1 pate EYl patt AE1 peart ER1 peat IYl peet IYl peete IYl pert ER1 petEH1 pete IY1 pettEH1 piet IYl pietteIYl pit IHl pitt IHl pot AAlpote OWl pott AA1 pout AW1 puett UWl purt ER1 put UHl putt AH1

上面的程序用于扫描词典中发音包含三个音素°的条目。如果条件为真，就将 pron 的内容分配给3个新的变量： ph1、ph2和 ph3。请注意实现这个功能的语句形式并 不多见。

以下是使用同样 for 语句的另一个例子，这次使用内部的链表推导。这段程序能够 找到所有发音结尾与 nicks相似的词汇。通过此方法也可以找到押韵的词。

>>>syllable=['N','IHO','K','S']



>>>[word for word,pron in entries if pron[-4:]==syllable]

["atlantic's",'audiotronics','avionics','beatniks','calisthenics', 'centronics',

'chetniks',"clinic's",'clinics','conics','cynics','diasonics',

"dominic's",

'ebonics','electronics',"electronics'",'endotronics',"endotronics'", 'enix',..]

请注意，可以通过多种方法来拼读一个读音： nics 、niks 、nix 甚至ntic's,t 不发音， 如词 atlantic's。让我们来看看其他发音与书写之间的不匹配情况。你可以总结一下 下面的例子有何功能，并解释它们是如何实现的吗?

>>>[w for w,pron in entries if pron[-1]=='M'and w[-1]=='n']

['autumn','column','condemn','damn','goddamn','hymn','solemn']

>>>sorted(set(w[:2]for w,pron in entries if pron[0]=='N'andw[0]!='n'))

['gn','kn','mn','pn']

音素包含数字表示主重音(1)、次重音(2)和无重音(0)。定义一个函数来提取 重音数字，然后扫描词典，找到具有特定重音模式的词汇。

>>>def stress(pron):

return [char for phone in pron for char in phone if char.isdigit()]

>>……>[w for w,pron in entries if stress(pron)==['0','1','0','2','0']]

['abbreviated','abbreviating','accelerated','accelerating','accelerator',

'accentuated','accentuating','accommodated','accommodating','accommodative', 'accumulated','accumulating','accumulative','accumulator',

'acto ...,pron in entries if stress(pron)==['0','2','0','1','0']]

['abbreviation','abbreviations','abomination','abortifacient', 'abortifacients',

'academicians','accommodation','accommodations','accreditation',

'accreditations',

'accumulation','accumulations', 'adjudication',..]

'acetylcholine','acetylcholine',

这段程序的精妙之处在于：用户自定义函数 stress() 在一个内含条件 的链表推理中被调用，还有一个双层嵌套循环。这里有些复杂，可 等你有了更多使用链表推导的经验后，再重新阅读这部分内容。

使用条件频率分布寻找相应词汇的最小对比集。找到所有p 开头的三音素词?,并按

**照它们的第一个和最后一个音素来分组°**。

>>>p3=[(propdrn

if pron[0]=='P'and len(pron)==3] **②**

>>>cfd=nltk.ConditionalFreqDist(p3)

>>>for template in cfd.conditions():

if len(cfd[template])>10:

·

·

words =cfd[template].keys()

wordlist =''.join(words)

print template,wordlist[:70]+"..."

\*……

P-CH perch puchepochepeach petsche poach pietsch putsch pautsch piche pet... P-K pik peek pic pique paque polk perc poke perk pac pock poch purk pak pa.. P-L pil poehl pille pehl pol pall pohl pahl paul perl pale paille perle po … P-N paine payne pon pain pin pawn pinn pun pine paign pen pyne pane penn p... P-P pap paap pipp paup pape pup pep poop pop pipe paape popp pip peep pope... P-R paar poor par poore pear pare pour peer pore parr por pair porr pier... P-S pearse piece posts pasts peace perce pos pers pace puss pesce pass pur.. P-T pot puett pit pete putt pat purt pet peart pott pett pait pert pote pa … . P-Z pays p.s pao's pais paws p.'s pas pez paz pei's pose poise peas paiz p..

通过查找特定词汇来访问它，而不必遍历整个词典。在5.3 节中，我们将系统地学 习如何使用 Python 的词典数据结构。通过指定词典的名字及后面带方括号的**关键** 字(例如：词 fire) 来查词典°。

>>>prondict =nltk.corpus.cmudict.dict()

>>>prondict['fire']①

[['F','AY1','ER0'],['F','AY1','R']]

>>>prondict['blog']②

Traceback(most recent call last):

File "<stdin>", line 1,in <module>



>>>prondict['blog']

*[['B','L','AA1','G']]*

如果试图查找一个不存在的关键字”,就会得到一个 KeyError 。这与使用过大的整 数索引链表时产生 IndexError 是类似的。词 blog 在发音词典中没有，所以需要对 词典稍作调整，为这个关键字分配一个值(这对NLTK 语料库是没有影响的；下 一次访问它时， blog依然是空的)。

我们可以用任何词典资源来处理文本，如：过滤掉具有某些词典属性的词(如名词),

或者映射文本中每一个词。例如：下面的“文本到发音”函数在发音词典中查找文 本中每个词。

>>>text =['natural','language','processing']

>>>[ph for w in text for ph in prondict[w][0]]

['N','AE1','CH','ERO','AHO','L','L','AE1','NG','G','W','AHO','JH',

'P','R','AAl','S','EHO','S','IHO','NG']

**比较词表**

表格词典的另一个例子是**比较词表。**NLT K 中包含了所谓**的斯瓦迪士核心词列表** (Swadesh wordlists),包括几种语言的约200个常用词的列表。语言标识符使用 **ISO639** 双字母码。

>>>from nltk.corpus import swadesh

>>>swadesh.fileids()

['be','bg','bs','ca','cs','cu','de','en','es','fr','hr','it','la',

'mk',

'nl','pl','pt','ro','ru','sk','sl','sr','sw','uk']

'th',,'you(singular),thou','he','we','you(plural)','they','this',

>>>swadesh.words('en')

'here','there','who','what','where','when','how','not','all',

'many','some',

'wid','other','one','two','three','four','five','big','long',

可以通过使用 entries) 方法来指定一个语言链表来访问多语言中的同源词。而且， 还可以把它转换成一个简单的词典(在5.3节将学到 dict()函数)。

>>>fr2en =swadesh.entries(['fr', 'en'])

>>>fr2en

s','you(sil','he'),.]

>>>translate['chien']

'dog'

>>>translate['jeter']

'throw'

通过添加其他源语言，这个简单的翻译器可以变得更为有用。使用 dict()函数把德 语-英语和西班牙语-英语对相互转换成一个词典，然后用这些添加的映射更新原有

的 translate 词 典 。

>>>de2en =swadesh.entries(['de','en']) >>>es2en =swadesh.entries(['es','en']) >>>translate.update(dict(de2en))

.[u'dict(es2en))



'dog'

#German-English

#Spanish-English

**比较德语族和拉丁语族的不同。**

>>>languages =['en','de','nl','es','fr','pt','la']

>>>for i in [139,140,141,142]:

. · print swadesh.entries(languages)[i]

('say','sagen','zeggen','decir','dire','dizer','dicere')

('sing','singen','zingen', 'cantar','chanter','cantar','canere') ('play','spielen','spelen','jugar','jouer','jogar,brincar','ludere')

('float','schweben','zweven','flotar','flotter','flutuar,boiar',

'fluctuare')

**词汇工具：** **Toolbox 和** **Shoebox**

目前最流行的语言学家用来管理数据的工具是 Toolbox(工具箱),以前叫做 Shoebox (鞋柜),因为它占据了语言学家曾用来装档案卡片的旧鞋盒。Toolbox 可以免费从 <http://www.sil.org/computing/toolbox/>下载。

Toolbox 文件由一些条目的集合组成，其中每个条目由一个或多个字段组成。大多 数字段都是可选的或重复的，这意味着这个词汇资源不能作为一个表格或电子表 格来处理。

下面是罗托卡特语 (Rotokas) 的词典。只看第一个条目，词 kaa, 意思是“窒息”。

>>>from nltk.corpus import toolbox

>>>toolbox.entries('rotokas.dic')

[('kaa',[('ps','v'),('pt','A'),('ge','gag'),('tkp','nek

i

pas'),

('dcsv','true'),('vx','1'),('sc','???'),('dt','29/0ct/2005'),

('ex','Apoka ira kaaroi aioa-ia reoreopaoro.'),

('xp','Kaikai i pas long nek bilong Apoka bikos em i kaikai na toktok.'), ('xe', 'Apoka is gagging from food while talking.')]), ..]

条目包括一系列的“属性-值”对，如(ps,'V'),表示词性是‘V’(动词),('ge','gag)



表示英文注释是'gag'。最后的3个配对包含一个罗托卡特语例句及其巴布亚皮钦语 和英语的翻译。

Toolbox 文件松散的结构使我们在现阶段很难更好地利用它。XML 提供了一种强 有力的方式来处理这种语料库，我们将在第11章继续学习。

罗托卡特语是巴布亚新几内亚的布干维尔岛上使用的一种语言。这个 词典资源由 Stuart Robinson贡献给NLTK。罗托卡特语以仅有12个音 素(彼此对立的声音)而闻名。详情请参考：<http://en.wikipedia.org/wiki/> Rotokas\_language

**2.5** **WordNet**

**WordNet** 是面向语义的英语词典，与传统辞典类似，但结构更丰富。NLTK 中包括 英语 WordNet, 共有155287个单词和117659个同义词。我们将从寻找同义词和如 何在 WordNet 中访问它们开始。

**意义与同义词**

考虑句子(1a) 。如果用automobile 替换掉(1a) 中的词 motorcar, 变成(1b), 句 子的意思几乎保持不变。

(1) a.Benz is credited with the invention of the motorcar.

b.Benz is credited with the invention of the automobile.

因为句子中其他成分都保持不变，可以得出结论： motorcar 和 automobile 有相同的 含义即它们是**同义词。**在 WordNet 的帮助下，我们可以探索这些词。

>>>from nltk.corpus import wordnet as wn

>>>wn.synsets('motorcar')

[Synset('car.n.01')]

因此，motorcar 只有一个可能的含义，它被定义为 car.n.01,car 的第一个名词意义。 carn.01 被称为**synset**或 **“同义词集”**,即意义相同的词(或“词条”)的集合。

>>>wn.synset('car.n.01').lemma\_names

['car','auto','automobile','machine','motorcar']

同义词集中的每个词可以有多种含义，例如： car 可能是火车车厢、货车或电梯厢。 但对于这个同义词集中的所有单词来说，最感兴趣的是其最常用的意义。同义词集 也有一些一般的定义和例句。

>>>wn.synset('car.n.01').definition

'a motor vehicle with four wheels;usually propelled by an internal combustion engine'

['he needs a car to get to work']

>>>wn.synset('car.n.01').examples

虽然定义可以帮助人们了解一个同义词集的本意，但往往是同义词集中的词对 程序是更有用的。为了消除歧义，将这些词标注为 car.n.01.automobile 、 car.n.01.motorcar 等。这种同义词集和词的配对叫做**词条。**可以得到指定同义词 集的所有词条°,查找特定的词条”,得到一个词条所对应的同义词集，也可以 得到一个词条的“名字”④。

>>>wn.synset('car.n.01').lemmas①

[Lemma('car.n.01.car'),Lemma('car.n.01.auto'),Lemma('car.n.01.automobi le'),Lemma('car.n.01.machine'),Lemma('car.n.01.motorcar')]

>>>wn.lemma('car.n.01.automobile')②

Lemma('car.n.01.automobile')

>>>wn.lemma('car.n.01.automobile').synset③

Synset('car.n.01')

>>>wn.lemma('car.n.01.automobile').name ④

'automobile'

与词automobile 和 motorcar 这些意义明确且只有一个同义词集的词不同，词 car 是 含糊的，共有5个同义词集。

>>>wn.synsets('car')

[Synset('car.n.01'),Synset('car.n.02'),Synset('car.n.03'),

Synset('car.n.04'),

Synset('cable\_car.n.01')]

>>>for synset in wn.synsets('car'):

 print synset.lemma\_names

['cr','auto','automobile','machine','motorcar']

['car','railcar','railway\_car', 'railroad\_car']

['car','gondola']

['car','elevator\_car']

['cable\_car', 'car']

为了方便起见，可以用下面的方式访问所有包含词 car 的词条。



>>>wn.lemmas('car')

[Lemma('car.n.01.car'),Lemma('car.n.02.car'),Lemma('car.n.03.car'),

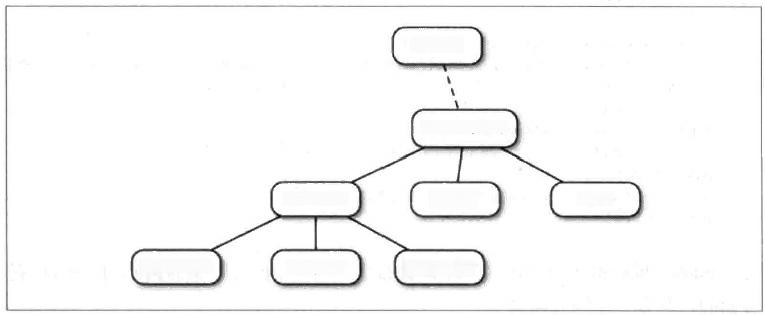
Lemma('car.n.04.car'),Lemma('cable\_car.n.01.car')]

**轮到你来：**

写下你能想到关于词 dish 的所有意思。现在，在 WordNet 的帮助下按 照前面所示的操作来探索这个词。

**WordNet 的层次结构**

**WordNet** 的同义词集相当于抽象的概念，它们并不总是有对应的英语词汇。这些概 念在层次结构中相互联系在一起。 一些概念综合性很强，如实体、状态、事件；这 些被称为**独一无二的根同义词集。**其他的，如： gas guzzler 和hatchback 等就具体 得多。图2-8展示了一个层次概念的一小部分。

artefoct

motor vehidle

*motorcar*

*90-kart*

truck

hatch-bock *compact gas guzzler*

**图2-8** **WordNet** **概念的层次片段：每个节点对应一个同义词集；边表示上位词/下位词关系，即** **上级概念与从属概念的关系**

**WordNet 使我们能容易驾驭各种概念。例如：摩托车，可以看到更加具体(直接)** **的概念——下位词。**

>>>motorcar =wn.synset('car.n.01')

>>>types\_of\_motorcar =motorcar.hyponyms()

>>>types\_of\_motorcar[26]

Synset('ambulance.n.01')

>>>sorted([lemma.name for synset in types\_of\_motorcar for lemma in synset.lemmas])

['Model\_T','S.U.V.','SUV','Stanley\_Steamer','ambulance','beach\_waggon',

'beach\_wagon','bus', 'cab','compact','compact\_car', 'convertible',



'coupe','cruiser','electric','electric\_automobile','electric\_car',

'estate\_car','gas\_guzzler','hack','hardtop','hatchback','heap',

'horseless\_carriage','hot-rod','hot\_rod','jalopy','jeep','landrover',

'limo','limousine','loaner','minicar','minivan','pace\_car','patrol\_car',

gt\_\_r'srtter\_ltayb'ratl\_oteyc\_o**h**aie','sports\_car',',a'sde,

'phaeton','police\_car','police\_cruiser','prowl\_car','race\_car','racer',

'station\_waggon','station\_wagon','stock\_car','subcompact','subcompact\_car',

'taxi','taxicab','tourer','touring\_car','two-seater','used-car','waggon',

'wagon']

通过访问上位词来操纵层次结构。有些词有多条路径，因为它们可以归类在多种分类 中 。carn.01 与entityn.01 之间有两条路径，因为 wheeled\_vehicle.n.01 可以同时被归类 为车辆和容器。

[Synset('motor\_vehicle.n.01')]

>>>motorcar.hypernyms()

>>>paths =motorcar.hypernym\_paths()

>>>len(paths)

2

>>>[synset.name for synset in paths[0]]

'art[tty **0**11',hysical\_entity.n.01','object.n.01','whole.n.02',

'instrumentality.n.03','container.n.01','wheeled\_vehicle.n.01', 'self-propelled\_vehicle.n.01','motor\_vehicle.n.01','car.n.01'] >>>[synset.name for synset in paths[1]]

['entity.n.01','physical\_entity.n.01','object.n.01','whole.n.02', 'artifact.n.01',

'instrumentality.n.03','conveyance.n.03','vehicle.n.01','wheeled\_ vehicle.n.01',

'self-propelled\_vehicle.n.01', 'motor\_vehicle.n.01','car.n.01']

我们可以用如下方式得到一个最笼统的上位(或根上位)同义词集。

>>>motorcar.root\_hypernyms()

[Synset('entity.n.01')]

**轮到你来：**

尝试 NLTK 中便捷的图形化 WordNet 浏览器：nltk.app.wordnet()。沿 着上位词与下位词之间的链接，探索 WordNet 的层次结构。

**更多的词汇关系**

上位词和下位词被称为**词汇关系，**因为它们是同义集之间的关系。这两者的关系为

上下定位“is-a” 层次。WordNet 网络另一个重要的定位方式是从条目到它们的部 件 (**部分)** 或到包含它们的东西(整体)。例如： 一棵树可以分成树干、树冠等部 **分，这些都是** **partmeronyms()。一棵树的实质是由心材和边材组成的，即** **substance\_**

meronyms()。树木的集合形成了一个森林，即member\_holonyms)。

>>>wn.synset('tree.n.01').part\_meronyms()

[Synset('burl.n.02'),Synset('crown.n.07'),Synset('stump.n.01'), Synset('trunk.n.01'),Synset('limb.n.02')]

>>>wn.synset('tree.n.01').substance\_meronyms()

[Synset('heartwood.n.01'),Synset('sapwood.n.01')]

>>>wn.synset('tree.n.01').member\_holonyms()

[Synset('forest.n.01')]

看看有多么复杂吧，比如单词mint, 有好几个密切相关的意思。mint.n.04 是mint.n.02 的一部分，同时也是组成 mint.n.05 的材料。

>>>for synset in wn.synsets('mint',wn.NOUN):

print synset.name +':',synset.definition

batch.n.02:(often followed by 'of')a large number or amount or extent mint.n.02:any north temperate plant of the genus Mentha with aromatic

leaves and small mauve flowers

mint.n.03:any member of the mint family of plants

mint.n.04:the leaves of a mint plant used fresh or candied mint.n.05:a candy that is flavored with a mint oil

mint.n.06:a plant where money is coined by authority of the government >>>wn.synset('mint.n.04').part\_holonyms()

[Synset('mint.n.02')]

>>>wn.synset('mint.n.04').substance\_holonyms()

[Synset('mint.n.05')]

动词之间也存在关系。例如：走路的动作包括抬脚的动作，所以走路**蕴涵**着抬脚。 一些动词有多个含义。

>>>wn.synset('walk.v.01').entailments()

[Synset('step.v.01')]

>>>wn.synset('eat.v.01').entailments()

[Synset('swallow.v.01'),Synset('chew.v.01')]

>>>wn.synset('tease.v.03').entailments()

[Synset('arouse.v.07'),Synset('disappoint.v.01')]

词条之间还存在一些词汇关系，如： **反义词。**

>>>wn.lemma('supply.n.02.supply').antonyms()

[Lemma('demand.n.02.demand')]

>>>wn.lemma('rush.v.01.rush').antonyms()

[Lemma('linger.v.04.linger')]

>>>wn.lemma('horizontal.a.01.horizontal').antonyms()

[Lemma('vertical.a.01.vertical'),Lemma('inclined.a.02.inclined')] >>>wn.lemma('staccato.r.01.staccato').antonyms()

[Lemma('legato.r.01.legato')]

使 用 dir() 查 看 词 汇 关 系 和 同 义 词 集 上 定 义 的 其 他 方 法 。 例 如 ： 尝 试

**dir(wn.synset(harmony.n.02'))。**

**语义相似度**

同义词集是由复杂的词汇关系网络所连接起来的。给定一个同义词集，可以遍历 WordNet 网络来查找相关含义的同义词集。知道哪些词是语义相关的，是对索引文 本集合有用的，当搜索一个一般性的用语时——例如车辆——就可以匹配包含特定 术语——例如豪华轿车——的文档。

每个同义词集都有一个或多个上位词路径连接到一个根上位词，如 entity.n.01。连 接到同一个根的两个同义词集可能有一些共同的上位词(见图2-8)。如果两个同义 词集共用一个特定的上位词——在上位词层次结构中处于较低层——它们一定有 密切的联系。

.sssseett**(**'le.n.01')

>>>minke =wn.synset('minke\_whale.n.01')

>>>tortoise =wn.synset('tortoise.n.01')

>>>novel =wn.synset('novel.n.01')

>>>right.lowest\_common\_hypernyms (minke)

[Synset('baleen\_whale.n.01')]

>>>right.lowest\_common\_hypernyms(orca)

[Synset('whale.n.02')]

>>>right.lowest\_common\_hypernyms(tortoise)

[Synset('vertebrate.n.01')]

>>>right.lowest\_common\_hypernyms (novel)

[Synset('entity.n.01')]

当然，鲸鱼是非常具体的(须鲸更是如此),脊椎动物是更具一般化，而实体完全是抽 象的。我们可以通过查找每个同义词集的深度来量化这个普遍性的概念。

>>>wn.synset('baleen\_whale.n.01').min\_depth()

14

>>>wn.synset('whale.n.02').min\_depth()





.synset('entity.n.01').min\_depth()

0

**WordNet** 同义词集定义了类似的方法来进行深入的观察。例如： path\_similarity 基 于上位词层次结构概念中相互关联的最短路径下，在0～1范围内的相似度(两者 之间没有路径就返回-1)。同义词集与自身比较将返回1。考虑以下的相似度：露 脊鲸与小须鲸、逆戟鲸、乌龟以及小说。虽然这些数字本身的意义并不大，但是从 海洋生物的语义空间转移到非生物时，数字是减少的。

>>>right.path\_similarity(minke)

0.25

>>>right.path\_similarity(orca)

0.16666666666666666

>>>right.path\_similarity(tortoise)

0.07 6923076923076927

>>>right.path\_similarity(novel)

0.04 3478260869565216

还有一些其他的相似性度量方法；你可以输入help(wn)获得更多的信 息 。NLTK 还包括 VerbNet, 一个连接到 WordNet 的层次结构动词词 典，可以通过 nltk.corpus.verbnet 来访问。

**2.6** **小结**

● 文本语料库是一个大型结构化文本的集合。NLTK 包含了许多语料库，如：布 朗语料库nltk.corpus.brown。

● 有些文本语料库是分类的，例如通过文体或者主题分类；有时候语料库的分类 会相互重叠。

· 条件频率分布是频率分布的集合，每个分布都有不同的条件。它们可以用于计 数内容或者文体中指定词的频率。

● 行数较多的Python 程序应该使用文本编辑器来输入，保存为.py 后缀的文件， 并使用 import 语句来调用。

· Python 函数允许一段特定的代码块与一个名字联系在一起，然后可以根据需要 多次重用这些代码。

● 一些被称为“方法”的函数与某个对象联系在起来，命名形式为：对象名加点 加方法名，例如： x.funct(y)或者word.isalpha()。

· 要想找到一些关于变量v 的信息，可以在 Pyhon 交互式解释器中输入 help(v) 来阅读这一类对象的帮助条目。

● WordNet 是一个面向语义的英语词典，由同义词的集合 或称为同义词集 (synsets)  ~~组~~成，并且组成一个网络。

● 默认情况下有些函数是不能使用的，必须使用Python 的 import语句来访问。

**2.7** **深入阅读**

本章的附加材料公布在 <http://www.nltk.org/>上，包括网络上免费提供的资源链接。 语料库方法总结请参阅语料库HOWTO( <http://www.nltk.org/howto>), 还有更广泛的 在线 API文档资料。

公开发行语料库的重要来源是语言数据联盟(LDC) 和欧洲语言资源局 (ELRA)。 它们提供数以百计的由几十种语言版本组成的已标注文本和语音语料库。非商业许 可证允许这些数据用于教学和科研。其中一些语料库也提供商业许可(但需要较高 的费用)。

这些语料库和许多其他语言资源使用 OLAC 元数据格式存档，可以通过 <http://www.language-archives.org/>上的 OLAC 主页搜索到。语料库列表(见http://gandalf. aksis.uib.no/corpora/sub.html)是一个讨论语料库内容的邮件列表，你可以通过搜索 列表档案或寄往列表中的地址找到相关资源。Ethnologue有着世界上最完整的语言 的清单，地址为 <http://www.ethnologue.com>/。7000种语言中只有几十种有大量适合 NLP 使用的数字资源。

本章涉及**语料库语言学**领域。在这一领域中，其他有用的书籍有 (Biber,Conrad, &Reppen,1998) 、(McEnery,2006) 、(Meyer,2002) 、(Sampson &McCarthy, 2005)和 ( Scott& Tribble,2006) 。 有关语言学中海量数据分析的深入阅读材料有

(Baayen,2008) 、(Gries,2009) 和 (Woods,Fletcher,&Hughes,1986)。

WordNet的最初描述出现在Fellbaum(1998) 中。虽然 WordNet 最初是为心理语言 学研究开发的，但它目前在自然语言处理和信息检索领域也被广泛使用。WordNets 正在开发许多其他语言的版本，在<http://www.globalwordnet.org/>中有记录。学习度 量WordNet 得相似性可以阅读 (Budanitsky&Hirst,2006)。

本章涉及的其他主题是语音和词汇语义学，读者可以参考(Jurafsky&Martin,2008) 的第7章和第20章。

**2.8** **练习**

1.O 创建变量phrase 包含一个词的链表。尝试本章描述的操作，包括加法、乘法、 索引、切片和排序。

2.O 使用语料库模块处理 austen-persuasion.txt。这本书中有多少词标识符?多少 词类型?

3.O 使用布朗语料库阅读器 nltk.corpus.brown.words()或网络文本语料库阅读器 nltk.corpus.webtext.words()来访问两个不同文体中的样例文本。

4.O 使用 state\_union语料库阅读器，访问《国情咨文报告》文本。计数每个文档 中出现的 men 、women 和people。随时间的推移这些词的用法有什么变化?

5.O 考查名词的整体部分关系。请记住，共有3种整体部分关系，所以需要使

用 member\_meronyms(),part\_meronyms),substance\_meronyms),member\_ holonyms(),part\_holonyms()及 substance\_holonyms)。

6.O 在比较词表的讨论中，创建一个对象叫做 translate, 通过它你可以使用德语和 意大利语词汇查找对应的英语词汇。这种方法可能会出现什么问题?你能提出一个 办法来避免这个问题吗?

7.O 根据Strunk 和 White 的《Elements of Style》, 词 however 在句子开头使用是 “in whatever way” 或 “to whatever extent” 的意思，而没有 “nevertheless” 的意思。正 确用法的例子： However you advise him,he will probably do as he thinks best.

(<http://www.bartleby.com/141/strunk3.html>)。使用词汇索引工具在各种文本中研究这 个词的实际用法。也可以看 LanguageLog 发布在 <http://itre.cis.upenn.edu/>~ myl/languagelog/archives/001913.html 上的 “Fossilized prejudices about'however'”。

8.①在名字语料库上定义一个条件频率分布，看看哪个首字母在男性名字中比在 女性名字中更常用(见图2-7)。

9.①挑选两个文本，研究它们之间在词汇、词汇丰富性、文体等方面的差异。你 能找出几个在这两个文本中词意相对不同的词吗?例如：在《白鲸记》与《理智与 情感》中的 monstrous。

10.①阅读 BBC新闻文章： “UK’s Vicky Pollards left behind’”<http://news.bbc.co.uk/> 1/hi/education/6173441.stm。文章给出了有关青少年语言的以下统计：“使用最多的 20个词，包括 yeah 、no 、but 和 like, 占所有词的大约三分之一”。对于大量文本 源来说，占所有词的三分之一的标识符有多少词类型?从这个统计中能得出什么结

论?更多相关信息请阅读 LanguageLog上的 <http://itre.cis.upenn.edu/>～myVlanguagelog/

archives/003993.html。

11.①调查模式分布表，寻找其他模式。试着依据自己对不同文体的印象派理解来 解释它们。你能找到其他封闭的，展现了不同文体之间显著差异的词汇归类吗?

12.OCMU 发音词典包含某些词的多种发音。它包含多少种不同的词汇?具有多 种发音的词汇在这个词典中的比例是多少?

13. ①没有下位词的名词在同义词集中所占的百分比是多少?你可以使用 wn.all\_synsets(n')得到所有名词的同义词集。

14.①定义函数 supergloss(s), 使用同义词集s 作为它的参数，返回字符串，包含 s 的定义和s 所有的上位词与下位词的定义的连接字符串。

15.①编写一段程序，找出所有在布朗语料库中出现至少3次的词。

16.①编写一段程序，生成如表1-1所示的词汇多样性得分表(例如：标识符/类型 的比例)。包括布朗语料库文体的全集 (nltk.corpus.brown.categories())。哪个文体 词汇多样性最低(每个类型的标识符数最多)?和你预测的结果相同吗?

17.①编写一个函数，找出文本中最常出现的50个词，停用词除外。

18.①编写一段程序，输出文本中50个最常见的双连词(相邻词对),忽略包含停 用词的双连词。

19.①编写一段程序，按文体创建词频表，以2.1 节给出的词频表为范例，选择 你自己的词汇，并尝试找出那些在文体中很突出或很缺乏的词汇。并讨论你的研 究结果。

20.①编写一个函数word\_freq(), 用一个词和布朗语料库中的一个部分名字作为参 数，计算这部分语料中词的频率。

21.①编写一段程序，利用CMU 发音词典来估算一个文本中的音节数。

22.①定义一个函数 hedge(text), 用于处理文本并产生一个在每三个词之间插入一 个词 like 的新版本。

23.● **齐夫定律：**f(w)是自由文本中词w 的频率。假设一个文本中的所有词都按照 它们的频率排名，频率最高的在最前面。齐夫定律指出一个词类型的频率与它的排 名成反比(即 f×r=k,k 是某个常数)。例如：最常见的第50个词类型出现的频率 应该是最常见的第150个词型出现频率的3倍。

a. 编写一个函数用于处理大文本，使用 pylab.plot 根据词排名画出词的频率。 你赞同齐夫定律吗? (提示：使用对数刻度。)所绘线的极端情况是怎 样的?

b. 随机生成文本，如：使用 random.choice("abcdefg"), 注意要包括空格字符。 事先需要输入random。使用字符串连接操作将字符累积成一个很长的字符 串。然后为这个字符串分词，生成前面的齐夫图，比较这两个图。此时你 如何看待齐夫定律?

24.●进一步修改例2-1 的文本生成程序，并完成下列任务。

a. 在一个词链表中存储n 个最相似的词，使用 random.choice()从链表中随机 选取一个词。(需要事先输入random)

b. 选择特定的文体，如：布朗语料库中的一部分，《创世纪》的翻译版，古腾 堡语料库中的文本或者网络文本。在此语料上建立模型，并产生随机文本。 尝试不同的起始字。文本的可理解性如何?讨论这种方法产生随机文本的 长处和短处。

c. 现在使用两种不同文体测试你的系统，尝试在混合文体中生成文本。并讨 论你的观察结果。

25.●定义函数 find\_language(), 用字符串作为其参数，返回包含此字符串的词汇 所属的语言列表。使用《世界人权宣言》(udhr) 的语料，将搜索限制在Latin-1 编 码的文件中。

26.●名词上位词层次的分枝因素是什么?也就是说，对于每一个具有下位词或上 位词层次中子层的名词同义词集，它们平均有几个下位词?使用 wn.all\_synsets(n') 获得所有名词同义词集。

27.●词的多义性是指它所有含义的个数。利用WordNet, 使用len(wn.synsets(dog',n') 可以判断名词 dog 有7种含义。计算 WordNet 中名词、动词、形容词和副词的平 均多义性。

28.●使用预定义的相似性度量给下面的每个词对的相似性打分。按相似性递减的 顺序排名。你的排名与这里给出的顺序有多接近?由Miller&Charles(1998) 实验 得出的顺序： car-automobile 、gem-jewel、journey-voyage 、boy-lad 、coast-shore、

asylum-madhouse 、magician-wizard 、midday-noon 、furnace-stove 、food-fruit、bird-cock、 bird-crane 、tool-implement 、brother-monk、lad-brother、crane-implement、journey-car、 monk-oracle 、cemetery-woodland 、food-rooster 、coast-hill 、forest-graveyard 、 shore-woodland 、monk-slave 、coast-forest 、lad-wizard 、chord-smile 、glass-magician、 rooster-voyage 、noon-string。



**第3章**

**处理原始文本**

文本的最重要来源无疑是网络。探索现成的文本集合，如我们在前面章节中看到的语 料库，是很方便的。然而，每个人都有自己的文本来源，需要学习如何访问它们。

本章的目的是要回答下列问题。

(1)怎样才能编写程序访问本地和网络上的文件，从而获得无限的语言材料?

(2)如何把文档分割成单独的单词和标点符号，并进行文本语料上分析?

(3)怎样编写程序产生格式化的输出，并把结果保存在文件中?

为了解决这些问题，本章将介绍 NLP 的重要概念，包括分词和词干提取。在此过 程中，巩固 Python 知识并且学习关于字符串、文件和正则表达式的知识。网络上 的文本都是 HTML 格式的，我们还将学习如何使用HTML。

重点：

从本章开始所有的交互式会话或程序都假设以下面的导入语句作为开始。 >>>from \_future\_import division

>>>import nltk,re,pprint

**3.1** **从网络和硬盘访问文本**

**电子书**

NLTK 语料库集合中存有古腾堡项目的一小部分样例文本。如果你对分析古腾堡项



目的其他文本感兴趣，可以在 <http://www.gutenberg.org/catalog/>上浏览25000 本免费在 线书籍的目录，获得 ASCⅡ 码文本文件的 URL。虽然90%的古腾堡项目的文本是 英语的，但是它还包括超过50种语言的材料，包括加泰罗尼亚语、中文、荷兰语、 芬兰语、法语、德语、意大利语、葡萄牙语和西班牙语(每种语言都有超过100个 文本)。

编号2554的文本是《罪与罚》的英文翻译，按照如下方式访问。

>>>from urllib import urlopen

>>>url="<http://www.gu> tenberg.org/files/2554/2554.txt"

>>>raw =urlopen (url).read()

>>>type(raw)

<type'str'>

>>>len(raw)

1176831

>>>raw[:75]

'The Project Gutenberg EBook of Crime and Punishment,by Fyodor Dostoevsky\r\n'

**进程** **read(将需要几秒钟来下载这本书。如果** **Internet 代理** **Python 无**

法正确检测出来，需要用下面的方法手动指定代理。

={'http':'<http://www.someproxy.com:3128>'} =urlopen(url,proxies=proxies).read()

>>>proxies >>>raw

变量 raw 包含有176831个字符的字符串。(type(raw) 可以视为一个字符串。)这本 书原始的内容，包括很多我们不感兴趣的细节，如空格、换行符和空行。请注意， 文件开头的\r 和\n, 这 Python用来显示特殊的回车和换行字符的方式(这个文件 一定是在 Windows 机器上创建的)。对于语言处理，要将字符串分解为词和标点 符号，正如我们在第1章中所学习的。这一步被称为分词，用于产生词汇和标点 符号的链表。

>>>tokens =nltk.word\_tokenize(raw)

>>>type(tokens)

<type 'list'>

>>>len(tokens)

255809

>>>tokens[:10]

['The','Project','Gutenberg','EBook','of','Crime','and','Punishment',

',','by']

请注意， NLTK 需要分词，但之前打开的 URL 读入字符串的任务都没有分词。如 果进一步在链表中创建NLTK 文本，便可进行如第1章所示的其他语言的处理，包 括常规的链表操作，例如切片。

>>>text =nltk.Text(tokens)

>>>type(text)

<type 'nltk.text.Text'>

>>>text[1020:1060]

['CHAPTER','I','On','an','exceptionally','hot','evening','early','in',

'July','a','young','man','came','out','of','the','garret','in',

'which','he','lodged','in','S','.','Place','and','walked','slowly',

',','as','though','in','hesitation',',','towards','K','.','bridge','.']

>>>text.collocations()

Katerina Ivanovna;Pulcheria Alexandrovna;Avdotya Romanovna;Pyotr Petrovitch;Project Gutenberg;Marfa Petrovna;Rodion Romanovitch;

Sofya Semyonovna;Nikodim Fomitch;did not;Hay Market;Andrey

Semyonovitch;old woman;Literary Archive;Dmitri Prokofitch;great

deal;United States;Praskovya Pavlovna;Porfiry Petrovitch;ear rings

请注意，古腾堡项目以排列形式出现。这是因为从古腾堡项目下载的文本都包含一 个首部，里面有文本的名称、作者、扫描和校对文本人的名字、许可证等信息。有 时这些信息出现在文件末尾页脚处。由于无法可靠地检测出文本内容的开始和结 束，因此在从原始文本中挑出内容之前，需要手工检查文件来发现标记内容开始和 结尾的特定字符串。

>>>raw.find("PART I")

5303

>>>raw.rfind("End of Project Gutenberg's Crime")

1157681

>>>raw=raw[5303:1157681]①

>>>raw.find("PARTI")

0

方法findO和rfindO(反向的 find) 用于获取字符串切片所需要的正确索引值°。利 用这个切片重新给 raw赋值，则raw 以“PARTI” 开始一直到(但不包括)标记内 容结尾的句子。

这是我们第一次接触到网络的实际内容：在网络上找到的文本可能含有不必要的内 容，并没有一个自动的方法来去除它。但只需要少量的额外工作，我们就可以提取 出我们需要的材料。

**处理** **HTML**

网络上的文本大部分是 HTML 文件形式。你可以使用网络浏览器将网页作为文本 保存为本地文件，然后按照后面关于文件小节描述的那样来访问它。不过，如果要 经常这样做，最简单的办法是直接利用 Python。第一步是像以前一样使用urlopen。 为了好玩，我们可以挑选被称为“Blondes to die out in 200 years”的 BBC 新闻故事， 一个都市传奇被 BBC 作为确立的科学事实流传下来。

>>>url="<http://news.bbc.co.uk/2/hi/health/2284783.stm>"

>>>html =urlopen(url).read()

>>>html[:60]

¹<!doctype html public"-//W3C//DTD HTML 4.0 Transitional//EN'

输入print html可以看到 HTML的全部内容，包括meta 元标签、影像地图、JavaScript、 表单和表格。

从 HTML 中提取文本是极其常见的任务，NLTK 提供了辅助函数 nltk.clean\_htmlO将 HTML字符串作为参数，返回原始文本。然后可以对原始文本进行分词，获得我们 熟悉的文本结构。

>>>raw =nltk.clean\_html(html)

>>>tokens =nltk.word\_tokenize(raw)

>>>tokens

['BBC','NEWS','I','Health','l','Blondes',"'",'to','die','out',..]

其中仍然含有不需要的内容，包括网站导航及有关报道等。通过一些尝试和出错你 可以找到内容索引的开始和结尾，并选择你感兴趣的标识符，按照前面讲的那样初 始化文本。

>>>tokens =tokens[96:399]

>>>text =nltk.Text(tokens)

>>>text.concordance('gene')

they say too few people now carry the gene for blondes to last beyond the next tw

t blonde hair is caused by a recessive gene .In order for a child to have blonde

to have blonde hair,it must have the gene on both sides of the family in the gra

there is a disadvantage of having that gene or by chance .They don 't disappear



ondes would disappear is if having the gene was a disadvantage and I do not think

**更多更复杂的有关处理** **HTML的内容，可以下载**[**http://www.crummy.com/**](http://www.crummy.com/) **software/BeautifulSoup/上的** **Beautiful Soup 软件包。**

**处理搜索引擎的结果**

网络可以被看作未经标注的巨大语料库。网络搜索引擎提供了一种有效的手段，搜 索大量文本作为有关语言学的例子。搜索引擎的主要优势是规模：当搜索一个这样 庞大的文件集时，会更有利于你找到感兴趣语言模式。而且，你可以使用非常具体 的模式，仅仅在较小的范围匹配一两个例子，但在网络上可能匹配成千上万的例子。 网络搜索引擎的第二个优势是非常容易使用。因此，它是一个非常方便的工具，可 以快速检查理论是否合理。具体见表3-1 的例子。

**表3-1** **搭配的谷歌命中次数：搭配的命中次数包括词absolutely 或** **definitely**,

**后面带有** **adore、love、like 或prefer 其中的一个(Liberman,in LanguageLog,2005)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Google命中次数** | **adore** | **love** | **like** | **prefer** |
| absolutely | 289000 | 905000 | 16200 | 644 |
| definitely | 1460 | 51000 | 158000 | 62600 |
| 比率 | 198:1 | 18:1 | 1:10 | 1:97 |

遗憾的是，搜索引擎有一些显著的缺点。首先，搜索方式允许的范围受到严格限制。 不同于本地驱动器中的语料库，你可以编写程序来搜索任意复杂的模式，搜索引擎 一般只允许搜索单个词或词串，有时也允许使用通配符。其次，搜索引擎得到的结 果不一致，并且在不同的时间或在不同的地理区域会给出截然不同的结果。当内容 在多个站点重复出现，搜索结果会增加。最后，搜索引擎返回结果中的标记可能会 不可预料地改变，基于模式的方法定位特定的内容将无法进行(通过使用搜索引擎 APIs 可以改善这个问题)。

**轮到你来：**

在网络中搜索 “the of”(引号内的内容)。基于庞大的数据，我们是否 可以得出“the of” 是英语中常用搭配的结论吗?

**处理** **RSS** **订阅**

博客圈是文本的重要来源，无论是正式的还是非正式的。在一个叫做 Universal Feed Parser的第三方Python库(可从<http://feedparser.org/>免费下载)的帮助下，我们可 以访问博客的内容，如下所示。

>>>import feedparser

>>>llog =feedparser.parse("<http://languagelog.ldc.upenn.edu/nll/?feed=> atom>")>>llog['feed']['title']

g(logtries)



>>>post.title

u"He's My BF"

>>>content =post.content[0].value

u¹<p>Today I was chatting with three of our visiting graduate students f'

>>>content[:70]

>>>nltk.word\_tokenize(nltk.html\_clean(content))

>>>nltk.word\_tokenize(nltk.clean\_html(llog.entries[2].content[0].value))

[u'Today',u'I',u'was',u'chatting',u'with',u'three',u'of',u'our',

u'visiting',

u'graduate',u'students',u'from',u'the',u'PRC',u'.',u'Thinking', u'that',u'I',

u'was',u'being',u'au',u'courant',u',',u'I',u'mentioned',u'the',

u'expression',

u'DUI4XIANG4',u'\u5c0d\u8c61',u'("',u'boy',u'/',u'girl',u'friend',

u'"',·..]

结果字符串中的前缀u 表示它们是Unicode 字符串(见3.3节)。随着工作的进一步 深入，我们可以编写程序创建一个博客帖子的小语料库，并以此作为 NLP 的工作 基础。

**读取本地文件**

为了读取本地文件，需要使用Python 内置的 open)函数，然后利用readO方法。假 设有一个文件 document.txt, 你可以利用如下方法加载它的内容。

>>>f =open('document.txt')

>>>raw =f.read()



**轮到你来：**

使用文本编辑器创建一个名为 document.txt 的文件，然后输入几行文 字，保存为纯文本。如果使用的是IDLE, 在“文件”菜单中选择“新 建窗口”命令，在新窗口中输入所需的文本，然后在 IDLE 弹出式对 话框中的文件夹内保存文件为 document.txt。然后在 Python 解释器中 使用 f=open(document.txt) 打开这个文件，并使用 print f.read(O检查其 内容。

当你尝试这样做时可能会产生各种各样的错误。如果解释器无法找到你的文件，你 会看到类似下面的错误。

>>>f =open('document.txt')

Traceback(most recent call last):

File"<pyshell#7>",line 1,in -toplevel-

f =open('document.txt')

IOError: [Errno 2] No such file or directory:'document.txt'

如果要检查正在打开文件是否在正确的目录中，使用IDLE“文件”菜单上的“打 开”命令，将显示 IDLE 当前目录下所有文件的清单。另一种方法是在 Python 中检查当前目录。

>>>os.listdir('.')

>>>import os

访问文本文件时可能遇到的另一个问题是换行符公约，这个公约因操作系统不同而 不同。内置的 open()函数的第二个参数用于控制如何打开文件：open(document.txt, 'rU') 。T意味着以只读方式打开文件(默认),'U 表示“通用”,提示我们忽略不同 的换行符公约。

假设已经打开了该文件，有几种方法可以阅读此文件。利用read) 方法创建了一个 包含整个文件内容的字符串。

>>>f.read()

'Time flies like an arrow.\nFruit flies like a banana.\n'

\n'字符是换行符，相当于按键盘上的Enter 键开始一个新行。

也可以使用 for 循环一次读文件中的一行。

>>>f =open('document.txt','rU')

>>>for line in f:

.. · print line.strip()

Time flies like an arrow.

Fruit flies like a banana.

使用 stripO方法删除输入行结尾的换行符。

NLTK 中的语料库文件也可以使用这些方法来访问。只需使用 nltk.data.findO来获取 语料库项目的文件名，然后就可以使用上述方式打开和阅读它。

>>>path =nltk.data.find('corpora/gutenberg/melville-moby\_dick.txt')

>>>raw =open(path,'rU').read()

**从** **PDF 、MS Word 及其他二进制格式中提取文本**

ASCⅡ 码文本和 HTML 文本是可读格式。文字常常以二进制格式出现，如 PDF 和 MSWord, 只能使用专门的软件打开。利用第三方函数库如 pypdf 和 pywin32可 访问这些格式。从多列文档中提取文本是相当具有挑战性的。 一次性转换几个文件， 会比较简单些，用一个合适的应用程序打开文件，以文本格式保存到本地驱动器， 然后以如下所述的方式访问它。如果该文档已经在网络上，你可以在Google 的搜 索框输入它的 URL。搜索结果通常包括这个文档的 HTML 版本的链接，你可以 将它保存为文本。

**捕获用户输入**

想要捕捉用户与程序交互时输入的文本，可调用 Python 函数 raw\_input()提示用户 输入数据。将用户输入保存到变量后，可以像其他字符串那样操纵它。

>>>s =raw\_input("Enter some text:")

Enter some text:On an exceptionally hot evening early in July >>>print "You typed",len(nltk.word\_tokenize(s)),"words."

You typed 8 words.

**NLP 的流程**

图3-1 总结了本节涵盖的内容，包括在第1章中学习的建立词汇表的过程。(其中 一个步骤，规范化将在3.6节中讨论。)

|  |  |  |
| --- | --- | --- |
| HTML  ASCII  Text  Vocab | html urlopen(url).read()  raw s nltk.clean\_html(html)  raw =raw[750:23506]  tokens =nltk.wordpunct\_tokenize(raw)  tokens =tokens[20:1834]  text s nltk.Text(tokens)  words =[w.lower()for w in text] vocab =sorted(set(words)) | 下载网页，如果需要  可以剥除HTML,整  理成所需要的内容  标记文本，选出其中  感兴趣的标记，创建  个NLTK 文本  标准化处理文字，创建  词汇表 |

**图3-1** **处理流程：我们打开一个** **URL** **代码读取里面** **HTML** **格式的内容，去除标记，并选择字**

**符的切片，然后分词，是否转换为** **nltk.Text对象是可选择的。我们也可以将所有词汇小写并提取** **词汇表**

在这条流程后面还有很多操作。要正确理解它，这样有助于明确其中提到的每个变 量的类型。使用type(x)可以找出任一Python 对象x 的类型，如 type(1)是<int>, 因 为1是一个整数。

当载入一个URL 或文件的内容时，或者当去掉 HTML 标记时，都是在处理字符 串，也就是 Python 的<str>数据类型(在3.2节，我们将学习更多有关字符串的 内容)。

>>>raw =open('document.txt').read()

>>>type(raw)

<type 'str'>

将一个字符串分词，会产生一个(词的)链表，这是Python 的<list>类型。规范化 和排序链表可产生其他链表。

>>>tokens =nltk.word\_tokenize(raw)

>>>type(tokens)

<type'list'>

olower()for w in tokens]

<type'list'>

>>>vocab =sorted(set(words))

>>>type(vocab)

<type 'list'>

对象的类型决定了它可以执行哪些操作。比如说可以追加元素到一个链表，但不能 追加元素到一个字符串。

>>>vocab.append('blog')

>>>raw.append('blog')

Traceback(most recent call last):

File "<stdin>",line AttributeError:'str'

1,in <module>

object has no attribute 'append'

同样的，我们可以连接字符串与字符串，以及链表和链表，但我们不能连接字符串 与链表。

knhn','paul','george','ringo']

>>>query='Who >>>beatles

>>>query +beatles

Traceback(most recent call last):

File"<stdin>",line 1,in <module>

TypeError:cannot concatenate 'str'and 'list' objects

在下一节中，我们将更加详细地探讨字符串，并进一步探索字符串和链表之间的 关系。

**3.2** **字符串：最底层的文本处理**

现在研究之前未介绍的基本数据类型。在前面的章节中，侧重于将文本当作词链表，并 没有细致地探讨词汇及它们是如何在编程语言中被处理的。通过使用NLTK 中的语料库 接口，我们可以忽略这些文本所在的文件。词的内容和文件的内容在编程语言中是由一 个叫做**字符串**的基本数据类型来表示的。在本节中，我们将详细探讨字符串，并展示字 符串与词汇、文本和文件之间的联系。

**字符串的基本操作**

使用单引号“或双引号②指定字符串，如下面的例子所示。如果一个字符串中包含一 个单引号，必须在单引号前加反斜杠让 Python 知道这是字符串中的单引号。或者 也可以将这个字符串放入双引号中②。否则，字符串内的单引号将被解释为字符串 结束标志， Python 解释器将产生语法错误。

>>>monty ='Monty Python’①

>>>monty

ronty Python's Flying Circus"②

>>> circus



>>>circus

ctoitslying Circus'④

Fct,ton's Flying Circus'

SyntaxError:invalid syntax

有时字符串跨好几行。Python 提供了多种方式表示它们。在下面的例子中， 一个包 含两个字符串的序列被连接为一个字符串。我们需要使用反斜杠①或者括号②,这样 解释器就知道第一行的表达式不完整了。

>>>couplet ="Shall I compare thee to a Summer's day?"\

"Thou are more lovely and more temperate:"①

>>>print couplet

Shall I compare thee to a Summer's day?Thou are more lovely and more tem

"And Summer's lease hath all too short a date:")②

>>>print couplet

Rough winds do shake the darling buds of May,And Summer's lease hath all too short a date:

不幸的是，这些方法并没有展现给我们十四行诗中两行之间的换行。为此，我们可 以使用如下所示的三重引号的字符串。

>>>couplet ="""Shall I compare thee to a Summer's day? ...Thou are more lovely and more temperate:"""

>>>print couplet

Shall I compare thee to a Summer's day?

oeol rese: darling buds of May,

...And Summer's lease hath all too short a date:'!!

>>>print couplet

Rough winds do shake the darling buds of May,

And Summer's lease hath all too short a date:

现在我们可以定义字符串，也可以尝试一些像上面简单的操作。首先，让我们来看 看+操作，被称为**连接”**。此操作产生一个新字符串，它是由两个原始字符串首尾相 连粘贴复制而成的。请注意，连接不会做一些比较聪明的事，例如在词汇之间插入 空格。我们甚至可以对字符串使用乘法”。



>>>'very'+'very'+'very’①

'veryveryvery'

>>>'very'\*3②

'veryveryvery'

**轮到你来：**

试运行下面的代码，然后利用你对字符串+和\*操作的理解，弄清楚它 是如何运作的。要小心区分字符串”,是一个空格符，而字符串",是

一1,2,3,4,5,6,7,6,5,4,3,2,1]

>>>b=[''\*2\*(7-i)+'very'\*i for i in a]

>>>for line in b:

print b

加法和乘法运算不仅仅适用于数字也适用于字符串。但是，注意不能对字符串使用 减法或除法。

>>>'very'-'y'

Traceback(most recent call last):

File "<stdin>",line 1,in <module>

TypeError:unsupported operand type(s)for -:'str'and 'str'

>>>'very'/2

Traceback(most recent call last):

File "<stdin>",line 1,in <module>

TypeError:unsupported operand type(s)for /:'str'and 'int'

这两种错误消息告诉我们：对数据类型的理解已产生了困惑。第一种情况说明减法 操作(即-)不能适用于对象类型 str (字符串),而第二种情况说明除法的两个操 作数不能分别为 str 和 int。

**输出字符串**

如果想看变量的内容或计算的结果，需要把变量的名称输入到解释器。还可以使用 print 语句来查看一个变量的内容。

>>>print monty

Monty Python

请注意这次是没有引号的。当通过输入变量的名字到解释器中来检查变量时，解释器 输出 Python中的变量值。因为它是一个字符串，结果被引用。然而，当我们告诉解释

器输出这个变量时，我们没有看到引号字符，因为字符串的内容里面没有引号。

print 语句可以以多种方式显示多行程序，如下所示。

>>>grail ='Holy Grail'

>>>print monty +grail

Monty PythonHoly Grail

>>>print monty,grail

rhoyly,"il the",grail

Monty Python and the Holy Grail

**访问单个字符**

如1.2节中的链表，字符串也可以被索引，从零开始。当索引一个字符串时，可得 到它的一个字符(或字母)。 一个单独的字符并没有什么特别，它只是一个长度为 1的字符串。

>>>monty[0]

'M'

>>>monty[3]

't'

>>>monty[5]

1 1

与链表一样，如果尝试访问一个超出字符串范围的索引时，便会产生错误。

>>>monty[20]

Traceback(most recent call last):

File "<stdin>",line 1,in ?

IndexError:string index out of range

同时，我们还可以使用字符串的负数索引，其中-1 是最后一个字符的索引°。正数和 负数的索引提供两种方式用来指示一个字符串中的任何位置。在这种情况下，当一 个字符串长度为12时，索引5和-7指示的是相同的字符(一个空格)。(请注意， 5=len(monty)-7)

>>>monty[-1] ①

'n'

>>>monty[5]

1 1

>>>monty[-7]

1 1

编写一个for 循环，遍历字符串中的字符。print 语句结尾加一个逗号，这是为了告 诉Python 不要在行尾输出换行符。

>>>sent ='colorless green ideas sleep >>>for char in sent:

… · print char,

c o 1 o r 1 e S S g r e e n i d e a S

furiously'

e e P

S 1

furiously

也可以计数单个字符。通过将所有字符小写来忽略大小写的区分，并过滤掉非字母 字符。

 rbpeur .eb-by\_dick.txt')

>>>fdist =nltk.FreqDist(ch.lower()for ch in raw if ch.isalpha())

'[f'''j'','r','l','d','u','m','c','w',

>>>fdist.keys()

这段代码以出现频率进行排序，最高的排在最前面，并按顺序显示英文字母(这是 一个相当复杂的过程，我们会在以后的内容中更仔细地解释)。可以用 fdist.plot) 可视化这个分布。 一个文本相关的字母频率特征可以用在文本语言自动识别中。

**访问子字符串**

子字符串是为进 一 步处理而从 一 个字符串中取出的任意连续片段。我们可以很容易 地访问子字符串，就像我们对链表进行切片一样(见图3-2)。例如：下面的代码访 问从索引6到索引10(但不包括)的子字符串。

>>>monty[6:10]

'Pyth

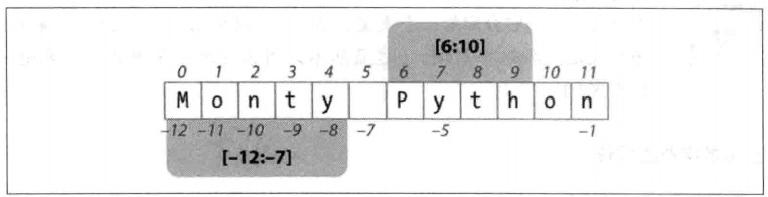


图3-2 字符串切片：伴随字符串 Monty Python 显示它的正数和负数索引引；两个子字符串都选 择使用“切片”符号。切片[m,n] 包含从位置m 到 n-1 中的字符



在这里，我们看到的字符是P' 、y' 、t 和"h', 它们分别对应于 monty[6]…monty[9] 而不是 monty[10] 。这是因为切片开始于第一个索引，但结束于最后一个索引的 前一个。

也可以使用负数索引切片——也是同样的基本规则，从第一个索引开始到最后一个 索引的前一个结束；在这里是在空格字符前结束。

>>>monty[-12:-7]

'Monty'

与链表切片一样，如果省略了第一个值，子字符串将从字符串的开头开始。

如果省略了第二个值，则子字符串直到字符串的结尾结束。

>>>monty[:5]

'Monty'

>>>monty[6:]

'Python'

使用 in 操作符测试一个字符串是否包含一个特定的子字符串，如下所示。

>>>phrase='And now for something completely different'

>>>if 'thing'in phrase:

.. · print 'found "thing"'

found "thing"

也可以使用 find()找到一个子字符串在字符串内的位置。

>>>monty.find('Python')

6

**轮到你来：**

造一句话，将它分配给一个变量，例如：sent='my sentence…'。编写 切片表达式抽取个别词。(这显然不是用来处理文本中的词的方便 的方式!)

**更多的字符串操作**

Python 对处理字符串的支持很全面。表3-2是一个总结，其中包括一些以前还没有 看到的操作。关于字符串的更多信息，可在 Python 提示符下输入help(str)。

les[2]

**表3-2** **有用的字符串方法：表1-4中字符串测试之外的字符串上的操作；**

**所有的方法都产生一个新的字符串或链表**

|  |  |
| --- | --- |
| **方** **法** | **功** **能** |
| s.find(t) | 字符串s中包含t的第一个索引(没找到返回-1) |
| s.rfind(t) | 字符串s中包含t的最后一个索引(没找到返回-1) |
| s.index(t) | 与s.find(t)功能类似，但没找到时引起ValueError |
| s.rindex(t) | 与s.rfind(t)功能类似，但没找到时引起ValueError |
| s,join(text) | 连接字符串s与text中的词汇 |
| s.split(t) | 在所有找到t的位置将s分割成链表(默认为空白符) |
| s.splitlines( | 将s按行分割成字符串链表 |
| s.lower() | 将字符串s小写 |
| s.upper() | 将字符串s大写 |
| s.titlecase() | 将字符串s首字母大写 |
| s.stripO | 返回一个没有首尾空白字符的s的复制 |
| s.replace(t,u) | 用u替换s中的t |

**链表与字符串的差异**

字符串和链表都是一种序列。可以通过索引抽取它们中的一部分，可以给它们切片， 也可以使用连接将它们合并在一起。但是，字符串和链表之间不能连接。

>>>query ='Who knows?'

>>>beatles=['John','Paul','George','Ringo']

'o'

>>>query[2]

>>>query[:2]

'Wh'

>>>beatles[:2]

['John','Paul']

+"I don't"

>>>query

"Who knows? I don't"

>>>beatles + 'Brian'

Traceback(most recent call File "<stdin>",line 1,in

last):

<module>

TypeError:can only concatenate list(not "str") to list

>>>beatles +['Brian'] ['John','Paul','George','Ringo','Brian']

当在一个 Python 程序中打开并读入一个文件，可得到一个对应整个文件内容的



字符串。如果使用 for 循环来处理这个字符串元素，可以挑选出来的只是单个的 字符——不选择粒度。相比之下，链表中的元素可以很大也可以很小。例如： 它们可能是段落、句子、短语、单词或字符。所以，链表的优势是可以灵活地 决定它包含的元素，相应的后续处理也变得灵活。因此，在一段 NLP 代码中可 能做的第一件事情就是将一个字符串分词放入一个字符串链表中(见3.7节)。 相反，当要将结果写入到一个文件或终端的时候，通常会将它们格式化为一个 字符串(见3.9节)。

链表与字符串的功能有很大不同。链表增加了可以改变其中的元素的能力。

>>>beatles[0]="John >>>del beatles[-1]

Lennon"

>>>beatles

['John Lennon','Paul', 'George']

尝试在字符串上将 query 的第0个字符修改为 'F', 得到结果如下。

>>>query[0]='F'

Traceback(most recent call last):

File "<stdin>",line 1,in ?

TypeError:object does not support item assignment

这是因为字符串是**不可变的：** 一旦创建了字符串，就无法改变它。然而，链表 是**可变的，** 其内容随时可以修改。链表支持修改原始值的操作，而不是产生一 个新的值。

轮到你来：

通过尝试本章结尾的练习，巩固有关字符串知识。

**3.3** **使用** **Unicode 进行文字处理**

经常需要应用程序处理不同的语言和不同的字符集。“纯文本”的概念是虚构的。 在讲英语的国家，可能使用的是ASCⅡ 码。在欧洲，可能使用的是扩展拉丁字符集， 包含有丹麦语和挪威语中的“g”, 匈牙利语中的“8”,西班牙和布列塔尼语中的“n”, 捷克语和斯洛伐克语中的 “n” 。在本节中，将概述如何使用 Unicode 处理使用非

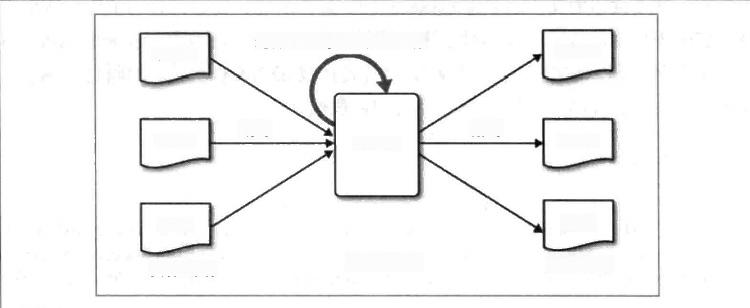
ASCⅡ字符集的文本。

**什么是** **Unicode**

Unicode 支持一百万种以上的字符。每个字符分配有一个编号，称为**编码点**。在 Python 中，编码点写作\uXXXX 的形式，其中XXXX 是四位十六进制形式数。

在一段程序中，可以像普通字符串那样操纵Unicode 字符串。然而，当Unicode 字 符被存储在文件里或在终端上显示时，必须被编码为字节流。由于一些编码(如 ASCⅡ 和 Latin-2) 中的每个编码点都使用单字节，所以它们只需支持Unicode 中的 一个小子集就足够一种语言使用了。其他的编码(如UTF-8) 使用多个字节，可以 表示全部的 Unicode 字符。

文件中的文本都是有特定编码的，所以需要一些机制来将文本翻译成 Unicode——翻译 成 Unicode 叫做**解码。**相对的，要将 Unicode 写入一个文件或终端，首先需要将 Unicode 转化为合适的编码——这种将 Unicode 转化为其他编码的过程叫做**编码，** 如图3-3所示。



|  |
| --- |
| 进行一些 Python 处理工作  GB2312  解码  Unicode  UTF-8  文件/终端 |

内存中程序

文件/终端

编码

Latin-2

Latin-2

GB2312

UTF-8

**图3-3** **Unicode 的解码和编码**

从 Unicode 的角度来看，字符是可以实现一个或多个字形的抽象的实体。只有字形 可以出现在屏幕上或被打印在纸上。字体是字符到字形的映射。

**从文件中提取已编码文本**

假设有一个小的文本文件，并且知道它是如何编码的。例如： polish-lat2.txt 是波兰语

的文本片段(来源于波兰语Wikipedia;可以在htp:/pl.wikipedia.org/wiki/Biblioteka\_Pruska 中看到)。此文件是Latin-2 编码的，也称为ISO-8859-2。nltk.data.find()函数为定位 文件。

>>>path =nltk.data.find('corpora/unicode\_samples/polish-lat2.txt')

Python的 codecs 模块提供了将编码数据读入为 Unicode字符串和将 Unicode字符串 以编码形式写出的函数。codecs.open()函数有一个 encoding参数来指定被读取或写 入的文件的编码。让我们导入codecs 模块，以 “latin2” 为 encoding的参数，调用 该参数以Unicode打开指定的波兰语文件。

>>>import codecs

>>>f =codecs.open(path,encoding='latin2')

关于 codecs 允许的 encoding 参数列表，可以在 <http://docs.python.org/lib/standard>- encodings.html中看到。请注意可以使用f=codecs.open(path,'w',encoding='utf-8') 在文 件中写入Unicode编码数据。

从文件对象f 读出的文本将以Unicode 返回。如前所述，为了在终端上查看这个文 本，我们需要使用合适的编码对它进行编码。Python 特定的编码 unicode\_escape 是 一个虚拟的编码，它把所有非 ASCⅡ 字符转换成\uXXXX 形式。编码点在 ASCⅡ 码0～127的范围以外但低于256的，使用两位数字的形式lxXX 表示。

>>for**li** nlie.strip()

print

line.encode('unicode\_escape')

Puska Biblioteka Pa\u0144stwowa.Jej dawne zbiory znane pod nazw\u0105 "Berlinka"to skarb kultury i sztuki niemieckiej.Przewiezione przez

Niemc\xf3w pod koniec II wojny \u015bwiatowej na Dolny \u015al\u0105sk,

zosta\u0142y

odnalezione po 1945 r.na terytorium Polski.Trafi\u0142y do Biblioteki Jagiello\u0144skiej w Krakowie,obejmuj\u0105 ponad 500 tys.zabytkowych archiwali\xf3w,m.in.manuskrypty Goethego,Mozarta,Beethovena,Bacha.

在输出的第一行中有一个以u 转义字符串开始的 Unicode 转义字符串，即\u0144。相 关的 Unicode 字符将会以字形h 显示在屏幕上。在前面例子的第三行中的xF3, 对 应字形为6,在128～255范围内。



在 Python 中， 一个 Unicode 字符串常量可以通过在字符串常量前面加一个u 也就 是u'hello'来指定。任意 Unicode字符通过在Unicode字符串常量内使用\uXXXX 转 义序列来定义。使用ordO查找字符的整数序数。

>>>ord('a')

97

97 的十六进制四位数是0061,因此可以使用相应的转义序列定义 Unicode 字符串 常量。

>>>a =u'\u0061

>>>a

u'a'

>>>printa

a

请注意， Python的 print语句假设 Unicode 字符的默认编码是 ASCⅡ码。然而，h 不 在ASCⅡ 码范围之内，除非指定编码否则不能被输出。在下面的例子中，指定 print 使用repr) 转化字符串， repr()输出UTF-8转义序列(以xXX 的形式),而不是试图 显示字形。

>>>nacute =u'\u0144'

>>>nacute

u'\u0144!

>>>nacute\_utf =nacute.encode('utf8')

>>>print repr(nacute\_utf)

'\xc5\x84'

如果操作系统和区域的设置支持 UTF-8 编码字符，输入 Python 命令： print nacute\_utf, 可在屏幕上看到h。

决定屏幕上显示字形的因素很多。如果确定编码正确但 Python代码仍 然未能显示出预期的字形，应检查系统上是否安装了所需的字体。

unicodedata 模块使我们可以检查Unicode 字符的属性。在下面的例子中，我们选择 超出ASCⅡ 范围的波兰语文本的第三行中的所有字符，输出它们的UTF-8 转义值， 然后是使用标准 Unicode约定的编码点整数(即以U+为前缀的十六进制数字),随 后是 Unicode 名称。

rt ut.readlines()

>>>line =lines[2]

>>>print line.encode('unicode\_escape')

Niemc\xf3w pod koniec II wojny \u015bwiatowej na Dolny \u015al\u0105sk, zosta\u0142y\n

>>>for c in line:

 if ord(c)>127:

· print'%rU+804x8s'8(c.encode('utf8'),ord(c),unicodedata.

name(c))

'\xc3\xb3'U+00f3 LATIN SMALL LETTER O WITH ACUTE

'\xc5\x9b'U+015b LATIN SMALL LETTER S WITH ACUTE

'\xc5\x9a'U+015a LATIN CAPITAL LETTER S WITH ACUTE

'\xc4\x85'U+0105 LATIN SMALL LETTER A WITH OGONEK

'\xc5\x82'U+0142 LATIN SMALL LETTER L WITH STROKE

如果在前面的例子中格式化字符串时用%s 替 换 %r ( 产 生reprO 值),并且系统支持 UTF-8 , 可看到类似下面的输出。

6U+00f3 LATIN SMALL LETTER O WITH ACUTE

SU+015b LATIN SMALL LETTER S WITH ACUTE

SU+015a LATIN CAPITAL LETTER S WITH ACUTE

aU+0105 LATIN SMALL LETTER A WITH OGONEK

王U+0142 LATIN SMALL LETTER L WITH STROKE

另外，根据系统的具体情况，可能需要用“latin2”替换示例中的编码“utf8”。

下面的例子展示的是 Python 字符串函数和 re 模块是如何接收 Unicode 字符串的。

>>>line.find(u'zosta\u0142y')

>>line =line.lower()

>>>print line.encode('unicode\_escape')

niemc\xf3w pod koniec ii wojny \u015bwiatowej na dolny \u015bl\u0105sk, zosta\u0142y\n

rrch(u'\u015b\w\*',line)

>>>m.group()

u'\u015bwiatowej'

**NLTK 分词器允许** **Unicode 字符串作为输入，并输出相应的** **Unicode 字符串。**

>>>nltk.word\_tokenize(line)

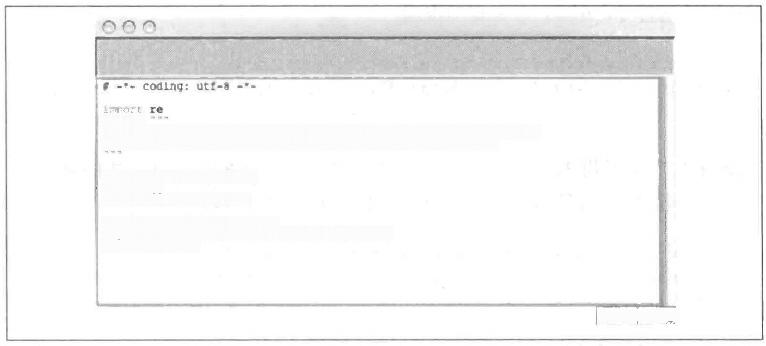
[u'niemc\xf3w',u'pod',u'koniec',u'ii',u'wojny',u'\u015bwiatowej',

u'na',u'dolny',u'\u015bl\u0105sk',u'zosta\u0142y']



**在** **Python** **中使用本地编码**

除了使用特定的本地编码字符，也可以在 Python 文件中使用自定义的字符串输入 及编辑标准方法。为了做到这一点，需要在文件的第一行或第二行中输入字符串： #-\*-coding:<coding>-\*-' 。 请注意，<coding> 是一个如'latin-1' 、'big5'和'utf-8'的字 符串(见图3-4)。图3-4展示了常规表达式中如何使用已编码字符串。



polish-utf8.py-/Users/ewan/svn/nltk/doc/images/polish-utf8.py

erwmlezioneprzez Nlemcow pod konlec II woiny &wlatowe1 na Dolny

Slask,zostaly odnalezlone Po 1945 r.na terytorlum Polgkt.

u sent.decodet'utf8')

u.lower()

ptint **u.encode(** 'utf8')

SACUT**E■re.compile(+** |S')

**replaced re.sub(Sncvit,** '[sacute]', **sent)**

prlat **replaced**

Ln:17 Col20

**图3-4** **Unicode 与IDLE:IDLE 编辑器中用UTF-8 编码字符串常量；这需要在** **IDLE 属性中设**

**置相应的字体；这里选择** **“Courier CE”**

**3.4** **使用正则表达式检测词组搭配**

许多语言处理任务都涉及模式匹配。例如：可以使用endswith(ed') 找到以 “ed” 结 尾的词。在表1-4 中可以看到各种“词测试”。正则表达式提出了一个更加强大和 灵活的方法描述感兴趣的字符模式。

介绍正则表达式的其他出版物有很多，它们围绕正则表达式的语法组 织，应用于搜索文本文件。这里不再赘述这些，只专注于在语言处理 的不同阶段如何使用正则表达式。像往常一样，将采用以问题为基础 的方式，只在解决实际问题需要时才介绍新特性。在讨论中，使用箭 头来表示正则表达式，就像这样：(patb)。

在 Python中使用正则表达式，需要使用importre 导入re 函数库，还需要用于搜索的

词汇链表。再次使用词汇语料库(2.4节),对它进行预处理消除某些名称。



>>>import re

>>>wordlist =[w for w in nltk.corpus.words.words('en')if w.islower()]

**使用基本的元字符**

使用正则表达式αed$》查找以ed 结尾的词汇。使用函数re.search(p,s) 检查字符串 s 中是否有模式 p。指定感兴趣的字符，然后使用美元符号，它是正则表达式中有 特殊用途的符号，用来匹配单词的末尾。

>>>[w for w in wordlist if re.search('eds',w)]

['abaissed', 'abandoned', 'abased','abashed','abatised', 'abed',

'aborted',...]

**通配符“** .”可以用来匹配任何单个字符。假设有一个8个字母组成的字谜， j 是第三个字母， t 是第六个字母。每个空白单元格用句点隔开。

>>>[w for w in wordlist if re.search('^..j..t..$',w)]

['abjectly', 'adjuster','dejected', 'dejectly', 'injector', 'majestic',...]

**轮到你来：**

插入符号“^”匹配字符串的开始，就像“$”符号匹配字符串的结尾。 如果我们不用这两个符号，刚才例子将得到什么样的结果?

最后，符号“?”表示前面的字符是可选的。因此《^e-?mail $》将匹配email 和e-mail。 我们可以使用 sum(1 for w in text if re.search(^e-?mail$',w)) 计数文本中这个词(任 一拼写形式)出现的总次数。

**范围与闭包**

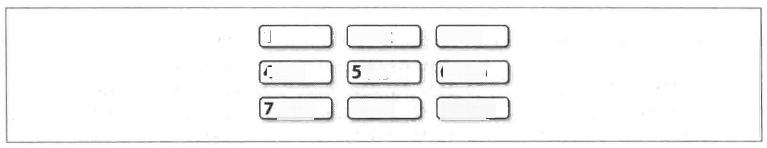
T9 系统用于在手机上输入文本(见图3-5)。两个或两个以上的词汇以相同的击键顺序 输入，这叫做**输入法联想提示。**例如： hole 和 golf 都是通过输入序列4653 得到的。 还有哪些其他词汇由相同的序列产生?这里使用正则表达式《^[ghi][mno][jlk][def]$》来 判断。

>>>[w for w in wordlist if re.search('^[ghi][mno][jlk][def]$',w)]

['gold','golf','hold','hole']



表达式的第一部分《^[ghi]》匹配以g 、h或i 开始的词。表达式的第二部分《[mno]》限制 了第二个字符是 m 、n 或 o 。第三部分和第四部分也有限制。4个字母要满足所有 这些限制。注意，方括号内的字符顺序不重要，所以可以写《^[hig][nom][ljk][fed]$》 来匹配同样的词汇。

1

3

2 ABC

DEF

MNO

4 GH

JKL

8 TUV

6

9

PQRS

WxYz

**图3-5** **T9:9** **个键上的文字**

**轮到你来：**

只用数字键盘的一部分搜索词汇叫“手指绕口令”。例如《^[ghijklmno]+$》, 或更为简洁的《^[g-o]+$》,匹配的是只使用中间行的4、5、6键的词汇， 《^[a-fj-o]+$》匹配的是使用右上角2、3、5、6键的词汇。“- ”和“+” 是什么意思?

进一步探索“+”符号。请注意，它适用于单个字母或括号内的字母集。

>>>chat\_words =sorted(set(w for w in nltk.corpus.nps\_chat.words())) >>>[w for w in chat\_words if re.search('^m+i+n+e+$',w)]

['miiiiiiiiiiiiinnnnnnnnnnneeeeeeeeee','miiiiiinnnnnnnnnneeeeeeee','mine', 'mmmmmmmmiiiiiiiiinnnnnnnnneeeeeeee']

>>>[w for w in chat\_words if re.search('^[ha]+S',w)]

['a','aaaaaaaaaaaaaaaaa','aaahhhh','ah','ahah','ahahah','ahh',

'ahhahahaha','ahhh','ahhhh','ahhhhhh','ahhhhhhhhhhhhhh','h','ha','haaa',

'hah', 'haha', 'hahaaa', 'hahah', 'hahaha', 'hahahaa','hahahah', 'hahahaha',...]

很显然，“+”表示的是“前面项目的一个或多个实例”,它可以是单独的字母如m, 可以是一个集合如[fed], 还可以是一个范围如[d-f] 。现在用“\*”替换“+”,表示 “前面的项目的零个或多个实例”。正则表达式《^m\*i\*n\*e\*$》匹配的是所有用 《^m+i+n+e+》找到的，并且包括其中一些字母不出现的词汇，例如： me 、min 和 mmmmm。请注意“+”和“\*”符号有时被称为 **Kleene 闭包，** 或者闭包。

当运算符“^”出现在方括号内的第一个字符位置时有其他的功能。例如： 《[^aeiouAEIOU]》匹配除元音字母之外的所有字母。可以搜索NPS 聊天语料库中完



全由非元音字母组成的词汇，使用《^[^aciouAEIOU]+$》查找诸如：);):)、grrr 、cyb3r 和 zzzzzzzz 这样的词。请注意其中包含的非字母字符。

下面是另外一些正则表达式的例子，用来寻找匹配特定模式的词汇标识符，这些例 子演示的是如何使用新的符号：\、{}、0和|。

>>>wsj=sorted(set(nltk.corpus.treebank.words()))

>>>[w for w in wsj if re.search('^[0-9]+\.[0-9]+S',w)]

['0.0085','0.05','0.1','0.16','0.2','0.25','0.28','0.3','0.4','0.5', '0.50','0.54','0.56','0.60','0.7','0.82','0.84','0.9','0.95','0.99', '1.01','1.1','1.125','1.14','1.1650','1.17','1.18','1.19','1.2',..] >>>[w for w in wsj if re.search('^[A-Z]+\$S',w)]

['CS','US$']

>>>[w for w in wsj if re.search('^[0-9]{4}$',w)]

['1614','1637','1787','1901','1903','1917','1925','1929','1933',…]

>>>[w for w in wsj if re.search('^[0-9]+-[a-z]{3,5}$',w)] ['10-day','10-lap','10-year','100-share','12-point','12-year',...]

>>>[w for win wsj if re.search('^[a-z](5,}-[a-z]{2,3}-[a-z]{,6}$',w)]

['black-and-white','bread-and-butter','father-in-law','machine-gun- toting',

'savings-and-loan']

>>>[w for w in wsj if re.search('(edling)$',w)]

['62%-owned','Absorbed','According','Adopting','Advanced',

'Advancing',...]

**轮到你来：**

研究前面的例子，在你继续阅读之前尝试弄清楚\、{}、()和|这些符号

**的功能。**

你可能已经知道反斜杠表示其后面的字母不再有特殊的含义而是按照字面的意思

匹配词中特定的字符。因此，虽然“.”有特殊含义，但“\.”只匹配一个句号。大 括号表达式，如{3,5},表示前面的项目重复指定的次数。管道字符表示从其左边 的内容和右边的内容中选择一个。圆括号表示一个操作符的范围，它们可以与管 道(或叫析取)符号一起使用，如：《w(ilelailoo)b), 匹配 wit 、wet 、wait 和 woot。 你可以省略这个例子里的最后一个表达式中的括号，使用《edling$》搜索看看会发生 什么,这对你的理解有帮助。

我们所看到的元字符总结在表3-3 中。

**表3-3** **正则表达式基本元字符，其中包括通配符、范围和闭包**

|  |  |
| --- | --- |
| **操** **作** **符** | **行** **为** |
|  | 通配符，匹配所有字符 |
| ^abc | 匹配以abc开始的字符串 |
| abc$ | 匹配以abc结尾的字符串 |
| [abc] | 匹配字符集合 |
| [A-Z0-9] | 匹配字符范围 |
| ed|ing|s | 匹配指定的字符串(析取) |
| 率 | 前面的项目零个或多个，如a\*、[a-z]\*(也叫Kleene闭包) |
| + | 前面的项目1个或多个，如a+、[a-z]+ |
| ? | 前面的项目零个或1个(即：可选)如：a?、[a-z]? |
| {n} | 重复n次，n为非负整数 |
| {n,} | 至少重复n次 |
| {,n} | 重复不多于n次 |
| {m,n} | 至少重复m次不多于n次 |
| a(b|c)+ | 括号表示操作符的范围 |

对 Python 解释器而言， 一个正则表达式与任何其他字符串没有不同。如果字符串 中包含一个反斜杠后面跟着一些特殊字符，Python解释器将会特殊处理它们。例如： “\b” 会被解释为一个退格符号。 一般情况下，当使用含有反斜杠的正则表达式时， 应该告诉解释器一定不要解释字符串里面的符号，而仅仅是将它直接传递给re 库 来处理。通过给字符串加一个前缀 “r” 来表明它是一个**原始字符串。**例如：原始 字符串r\band\b'包含两个“\b” 符号会被re 库解释为匹配词的边界而不是解释为退 格字符。如果能逐渐习惯使用r…来表示正则表达式——就像从现在开始做的那样 ——你将会避免这些解释上的歧义。

**3.5** **正则表达式的有益应用**

前面的例子都涉及使用 re.search(regexp,w) 匹配一些正则表达式 regexp 来搜索词 w 。除了检查正则表达式是否匹配单词外，还可以使用正则表达式从词汇中提取特 征或以特殊的方式来修改词。

**提取字符块**

通过re.findallO(“find all” 即找到所有)方法找出所有(无重叠的)匹配的指定正



则表达式。找出一个词中的所有元音，并计数。

>>>word='supercalifragilisticexpialidocious'

>>>re.findall(r'[aeiou]',word)

['u','e','a','i','a','i','i','i','e','i','a','i','o','i','o','u']

>>>len(re.findall(r'[aeiou]',word))

16

找出文本中两个或两个以上的元音序列，并确定它们的相对频率。

>>>wsj=sorted(set(nltk.corpus.treebank.words()))

>>>fd =nltk.FreqDist(vs forfoo sjfindall(r'[aeiou]{2,}',word))

>>>fd.items()

[('io',549),('ea',476),('ie',331),('ou',329),('ai',261),('ia',253),

''',..]

**轮到你来：**

在 W3C 日期时间格式中，日期表示为：2009-12-31。用正则表达式替 换下面 Python 代码中的“?”,将字符串'2009-12-31'转换为一个整数链

表[2009,12,31]。

[int(n)for n in re.findall(?,'2009-12-31¹)]

**在字符块上做更多事情**

一旦学会使用 re.findall()从词中提取字符块，就可以在这些字符块上做一些有趣的 事情，例如将它们粘贴在一起或用它们绘图。

英文文本是高度冗余的，忽略掉词内部的元音仍然可以轻松阅读，有些时候这很明 显。例如： declaration 变成 dclrtn,inalienable 变成 inlnble, 保留所有词首或词尾的 元音序列。在下一个例子中，正则表达式匹配词首元音序列，词尾元音序列和所有 的辅音；其他的被忽略。这3个阶段从左到右依次处理，如果词匹配了3个部分之 一，正则表达式后面的部分将被忽略。使用re.findall()提取所有匹配的词中的字符， 然后使用"join() 将它们连接在一起(见3.9节的连接操作)。

>>>regexp =r¹^[AEIOUaeiou]+I[AEIOUaeiou]+Sl[^AEIOUaeiou]' >>>def compress(word):

pieces =re.findall(regexp,word)

· return''.join(pieces)

… ·

>>>english\_udhr =nltk.corpus.udhr.words('English-Latin1')

>>>print nltk.tokenwrap(compress(w)for w in english\_udhr[:75])

Unvrsl Dclrtn of Hmn Rghts Prmble Whrs rcgntn of the inhrnt dgnty and

of the eql and inlnble rghts of all mmbrs of the hmn fmly is the fndtn

of frdm,jstce and pce in the wrld,Whrs dsrgrd and cntmpt fr hmn

and the advntof a wrld in whch hmn bngs shllenjy frdm of spchand

rghts hve rsltd in brbrs acts whch hve outrgd the cnscnce of mnknd ,

接下来，将正则表达式与条件频率分布结合起来。在这里，从罗托卡特语词汇中提 取所有辅音-元音序列，如 ka和 si。因为每部分都是成对的，它可以用来初始化条 件频率分布。然后将每对的频率列表。

>>>rotokas\_words =nltk.corpus,toolbox.words('rotokas.dic')

>>>cvs=[cv for w in rotokas\_words for cv in re.findall(r'[ptksvr][aeiou]',w)]

>>>cfd =nltk.ConditionalFreqDist(cvs)

>>>cfd.tabulate()

k p r s t

V

a

418

83

187

0

47

93

e

i

94

148

31

105

63

84

0

100

8

0

27

105

O

420

34

89

2

148

48

u

173

51

79

1

37

49

考查s 行和t 行，发现它们是部分“互补分布”的，这个证据表明它们不是这种语 言中的不同因素。从而可以从罗托卡特语字母表中去除 s,简单加入一个发音规则： 当字母t 跟在i 后面时发s 的音。(注意单独的条目 “su” 即 kasuari,“cassowary” 是从英语中借来的)。

如果想要检查表格中数字背后的词汇，需要找到包含给定辅音-元音对应的单词列 表。例如： cv\_index['su]所有包含 “su” 的词汇。下面的例子展示了如何能做到这 一点。

>>>cv\_word\_pairs =[(cv,w)for w in rotokas\_words

for cv in re.findall(r'[ptksvr][aeiou]',w)]

>>>cv\_index =nltk.Index(cv\_word\_pairs)

>>>cv\_index ['su']

['kasuari']

>>>cv\_index['po']

['kaapo','kaapopato','kaipori','kaiporipie','kaiporivira','kapo',

'kapoa',

'kapokao','kapokapo','kapokapo','kapokapoa','kapokapoa','kapokapora',

...]

依次处理每个词 w, 找出每个词对应匹配的正则表达式《[ptksvr][aeiou]》的所有子字 符串。对于词kasuari, 需找到 ka 、su 和 ri。从而，链表 cv\_word\_pairs 将包含('ka', 'kasuari') 、('su','kasuari') 和 ('ri','kasuari') 。 进一步使用nltk.IndexO将它们转换成 有用的索引。

**查找词干**

在使用网络搜索引擎时，通常不介意(甚至没有注意到)文档中的词汇与搜索条件的 后缀形式是否相同。查询“laptops”会找到含有“laptop”的文档，反之亦然。事实上， “laptop” 与 “laptops” 只是字典中的同一个词(或词条)的两种形式。对于一些语言 处理任务，需忽略词语结尾，而只处理词干。

抽出一个词的词干的方法有很多种。这里采用的是一种简单直观的方法，直接去掉 任何看起来像后缀的字符。

>>>def

stem(word):

for

suffixin['ing','ly','ed','ious','ies','ive','es','s',

'ment']:

· if word.endswith(suffix):

\* return word[:-len(suffix)]

..return word

最终使用NLTK 中内置的词干，而如何能够使用正则表达式实现这个任务呢?第 一步是建立所有后缀的连接，并把它放在括号内以限制这个连接的范围。

>>>re.findall(r'^.\*(ingllyledliouslieslivelesls|ment)s','processing')

['ing']

尽管正则表达式匹配整个单词，但是 re.findall()只给出后缀。这是因为括号有第二 个功能：选择要提取的子字符串。如果要使用括号来指定连接的范围，又不想选 择要输出字符串的，必须添加“?:”。下面是改进后的版本。

>>re.findall(r¹^.\*(?:ingllyledliousliesliveles|s|ment)S','processing')

['processing']

然而，实际上，是想将词分成词干和后缀。所以，应该只是用括号括起正则表达式 的如下两个部分。

>>>re.findall(r¹^(.\*)(ingllyledliousliesliveles|s|ment)$','processing') [('process', 'ing')]

这看起来很有用途，但仍然有一个问题。让我们来看看另外的词 “processes”。

>>>re.findall(r'^(.\*)(ingllyledliousliesliveles|s|ment)$','processes') [('processe','s')]

正则表达式错误地找到了后缀“-s”, 而不是后缀“-es” 。这表明另一个微妙之 处：“\*”操作符是“贪婪的”,所以表达式的“.\*”部分试图尽可能多地匹配输 入的字符串。如果使用“非贪婪”版本的“\*”操作符，写成“\*?”,就得到想 要的结果。

>>>re.findall(r'^(.\*?)(ingllyledliousliesliveles|s|ment)$','processes') [('process', 'es')]

还可以通过将第二个括号中的内容变成可选来得到空后缀。

>>>re.findall(r¹^(.\*?)(ingllyledliousliesliveles|s|ment)?S','language') [('language','')]

虽然这种方法有许多问题，(你能发现它们吗?)但我们仍将继续定义一个函数来 获取词干，并将它应用到整个文本。

… ·

·

…

.

psuf'^(=.\*r?e)ls|s|ment)?$'

>>>def stem(word):

return stem

 swords

...is no basis for a system of government.Supreme executive power derives from

...a mandate from the masses,not from some farcical aquatic ceremony."""

>>>tokens =nltk.word\_tokenize(raw)

>>>[stem(t)for t in tokens]

['DENNIS',':','Listen',',','strange','women','ly','in','pond',

'distribut','sword','i','no','basi','for','a','system','of','govern',

'the', 'mass',',', 'not','from','some','farcical','aquatic',

'.','Supreme','execut','power','deriv','from','a','mandate','from',

'ceremony', '.']



请注意正则表达式不仅将 “ponds” 的 “s” 删除，同时也将 “basis” 的 “is” 删除 了。它产生一些非词(未被确认、收录的词)如 distribut 与 deriv, 但这些都是在 一些应用中可接受的词干。

**搜索已分词文本**

可以使用一种特殊的正则表达式搜索一个文本中多个词(这里的文本是一个标识符 列表)。例如：“<a><man>” 找出文本中所有 “a man” 的实例。尖括号用于标记标 识符的边界，尖括号之间的所有空白都被忽略(这只对NLTK 中的 findallO方法处理 文本有效)。在下面的例子中，使用<.\*>①,令其匹配所有单个标识符，并置于括号 里，这样就只匹配词(例如：monied) 而不匹配短语(例如： amonied man)。第二个 例子找出以词“bro”为结尾的3个词组成的短语②。最后一个例子找出以字母“1” 开始的3个或更多词组成的序列。

oo**b**yy.findala.T>e(x(t)e<nr①ds('melville-moby\_dick.txt'))

>>>from nltk.corpus import gutenberg,nps\_chat

monied;nervous;dangerous;white;white;white;pious;queer;good;

mature;white;Cape;great;wise;wise;butterless;white;fiendish;

pale;furious;better;certain;complete;dismasted;younger;brave;

brave;brave;brave

>>>chat.findall(r"<.\*><.\*><bro>")②

>>>chat =nltk.Text(nps\_chat.words())

a;tl .twizted bro

lol lol lol; lmao lol lol; lol lol lol; la la la la la; la la la; la

la la;lovely lol lol love;lol lol lol.;la la la;la la la

**轮到你来：**

巩固对正则表达式模式与替换的理解，使用 nltk.re\_show(p,s), 它能 标注字符串s 中所有匹配模式p 的地方，以及nltk.app.nemo), 它能 提供一个探索正则表达式的图形界面。可以尝试本章结尾的正则表达 式的一些练习。

当研究的语言现象与特定词语相关时，建立搜索模式变得容易多了。在某些情况下， 一个小小的创意可能会花很大功夫。例如：在大型文本语料库中搜索“x and other ys” 形式的表达式时发现上位词(见2.5节)。

>>>from nltk.corpus import brown

>>>hobbies\_learned =nltk.Text (brown.words(categories=['hobbies','learned']))



>>>hobbies\_learned.findall(r"<\w\*><and><other><\w\*s>")

speed and other activities;water and other liquids;tomb and other landmarks;Statues and other monuments;pearls and other jewels;

charts and other items;roads and other features;figures and other objects;military and other areas;demands and other factors;

abstracts and other compilations;iron and other metals

只要有足够多的文本，将得到一整套有用的分类标准信息，而不需要任何手工劳动。 然而，搜索结果中通常会包含误报，即想要排除的情况。例如，结果 “demands and other factors” 表 明“demands” 是类型 “factor” 的一个实例，但是这句话实际上是 关于要求增加工资的。尽管如此，仍可以通过手工纠正这些搜索的结果来构建自己 的英语概念本体。

这种自动和人工处理相结合的方式是最常见的建造新语料库的方式。 更多内容将在第11章继续讲述。

搜索语料库也会有遗漏的问题，即漏掉了想要包含的情况。仅仅因为找不到任何搜 索模式的实例，就断定一些语言现象在语料库中不存在，是很冒险的。也许只是没 有足够仔细地思考合适的模式。

轮到你来：

查找模式 “asxas y” 的实例以发现实体及其属性信息。

**3.6** **规范化文本**

在前面的例子中，在处理文本词汇前经常要将文本转换为小写，即 set(w.lower()for w in text)。通过使用lower) 将文本规范化为小写，这样一来 “The” 与 “the” 的区别被忽 略。我们常常会进行更多的尝试，例如：去掉所有的词缀及提取词干的任务等。下一 步是确保结果形式是字典中确定的词，即词形归并的任务。首先，需要定义将在本节 中使用的数据。

>>>raw="""DENNIS:Listen,strange women lying in ponds distributing swords

...is no basis for a system of government.Supreme executive power derives from

...a mandate from the masses,not from some farcical aquatic ceremony."""

>>>tokens =nltk.word\_tokenize(raw)

**词干提取器**

NLTK 中包括了一些现成的词干提取器，如果需要使用词干提取器，应该优先使用 它们中的一个，而不是使用正则表达式制作自己的词干提取器，因为NLTK 中的词 干提取器能处理的不规则情况很广泛。Porter 和 Lancaster 词干提取器按照它们自 己的规则剥离词缀。下面的例子表明 Porter 词干提取器正确处理了词 lying (将它 映射为lie), 而 Lancaster 词干提取器并没有处理好。

>>>porter =nltk.PorterStemmer()

>>>lancaster =nltk.LancasterStemmer()

>>>[porter.stem(t)for t in tokens]

['DENNI',':','Listen',',','strang','women','lie','in','pond',

'distribut','sword','is','no','basi','for','a','system','of','govern', '.','Suprem','execut','power','deriv','from','a','mandat','from',

'the','mass',',','not','from','some','farcic','aquat','ceremoni','.']

>>>[lancaster.stem(t)for t in tokens]

['den',':','list',',','strange','wom','lying','in','pond','distribut',

'sword','is','no','bas','for','a','system','of','govern','.','suprem',

'execut','pow','der','from','a','mand','from','the','mass',',','not',

'from','som','farc','aqu','ceremony','.']

词干提取过程没有明确定义，通常选择最适合应用的词干提取器。如果要索引文本 或使搜索支持不同词汇形式的话， Porter 词干提取器是一个很好的选择(如例3-1 所示，它采用了面向对象编程技术，这超出了本书的范围，字符串格式化技术将在 3.9节讲述， enumerate()将在4.2节解释)。

例3-1 使用词干提取器索引文本

class IndexedText(object):

def self.\_\_it(setmer,text):

self.\_stemmer =stemmer

self.\_index =nltk.Index((self.\_stem(word),i)

for(i,word)in enumerate(text))

def key \_rds**(**rd,width=40):

ocr iitehl/ \_index[key]: # words of context

lcontext =' '.join(self.\_text[i-wc:i])

rcontext =''.join(self.\_text[i:i+wc])

ldisplay='8\*s' %(width,lcontext[-width:]) rdisplay='%-\*s'%(width,rcontext[:width])

print ldisplay,rdisplay



def \_stem(self,word):

return self.\_stemmer.stem(word).lower()

>>>porter =nltk.PorterStemmer()

>>>grail=nltk.corpus.webtext.words('grail.txt')

>>>text =IndexedText (porter,grail)

>>>text.concordance('lie')

r king!DENNIS:Listen,strange women lying in ponds distributing swords is no

beat a very brave retreat .ROBIN:All lies!MINSTREL:[singing ]Bravest

of

Nay.Nay.Come.Come.You may lie here .Oh,but you are wounded! doctors immediately!No,no,please!Lie down .[clap clap ]PIGLET:

Well

ere is much danger,for beyond the cave lies the Gorge of Eternal Peril, which

you.Oh...TIM:To the north there lies a cave --the cave of Caerbannog

h it and lived!Bones of full fifty men lie strewn about its lair .So,

braostop our fight 'til each one of you lies dead,and the Holy Grail

returns t

**词形归并**

**WordNet** 词形归并器删除词缀产生的词，都是它的字典中的词。这个额外的检查过 程使词形归并器比刚才提到的词干提取器速度要慢。请注意，它并没有处理“lying”, 而是将 “women” 转换为 “woman”。

>>>wnl =nltk.WordNetLemmatizer()

',',n'bgaes'','lsyyi'nd',

>>>[wnl.lemmatize(t)for t in tokens]

'government','.','Supreme','executive','power','derives','from','a',

'mandate','from','the','mass',',','not','from','some','farcical', 'aquatic','ceremony','.']

如果想编译一些文本词汇，或者想要一个有效词条(或中心词)列表， WordNet 词 形归并器是一个不错的选择。

另一个规范化任务涉及识别**非标准词，** 包括数字、缩写、日期及任何 此类标识符到一个特殊的词汇的映射。例如：每一个十进制数可以映 射到一个单独的标识符0.0上，每个首字母缩写可以映射为 AAA。这使 词汇量变小，提高了许多语言建模任务的准确性。

**3.7** **用正则表达式为文本分词**

分词是将字符串切割成可识别的构成语言数据的语言单元。虽然这是一项基础任务， 但一直拖延到现在为止才介绍，是因为许多语料库已经分过词了，还因为NLTK中 包括一些分词器。现在你已经熟悉了正则表达式，你可以学习如何使用它们来为文 本分词，并对此过程中有更多的掌控权。

**分词的简单方法**

文本分词是在空格符处分割文本最简单的方法。考虑以下摘自《爱丽丝梦游仙境》 中的文本。

>>>raw="""'When I'M a Duchess,'she said to herself,(not in a very hopeful tone

...though),'I won't have any pepper in my kitchen AT ALL.Soup does very

...well without--Maybe it's always pepper that makes people hot-tempered,'..."""

可以使用 raw.split()在空格符处分割原始文本。使用正则表达式能做同样的事情， 匹配字符串中的所有空格符①是不够的，因为这将导致分词结果包含“\n” 换行符。 同时需要匹配任何数量的空格符、制表符或换行符?。

>>>re.split(r'',raw)①

'in'"'When","I'M",'a',"Duchess,'",'she','said','to','herself,','(not',

'a','very','hopeful','tone\nthough),',"'I","won't",'have','any',

'pepper',

'in','my','kitchen','AT','ALL.','Soup','does','very\nwell',

'without--Maybe',

"it's",'always','pepper','that','makes','people',

"hot

'in'"'When","I'M",'a',"Duchess,'",'she','said','to','herself,','(not',

'a','very','hopeful','tone','though),',"'I","won't",'have','any',

'pepper',

'in','my','kitchen','AT','ALL.','Soup','does','very','well',

'without--Maybe',

"it's",'always','pepper','that','makes','people',

"hot-tempered,'..."]



正则表达式《[\t\n]+》匹配一个或多个空格、制表符(\t) 或换行符(\n)。其他 空白字符，如回车和换页符，也应该被包含。于是，将使用 re 库内置的缩写 “\s”, 表示匹配所有空白字符。前面的例子中第二条语句可以改写为 re.split(r^\s+',raw)。

**要点：**

记住在正则表达式前加字母 “r”, 告诉 Python解释器按照字面表示对 待字符串而不去处理正则表达式中包含的反斜杠字符。

在空格符处分割文本可得到如“(not” 和 “herself,” 这样的标识符。另一种方法是 使用Python提供给我们的字符类“\w” 匹配词中的字符，相当于[a-zA-Z0-9\_]。定 义这个类的补充部分“\W” 即所有字母、数字和下划线以外的字符。还可以在一 个简单的正则表达式中用\W来分割所有单词字符以外的输入。

>>>re.split(r'\W+',raw)

['','When','I','M','a','Duchess','she','said','to','herself',

'not', 'in',

'a','very','hopeful','tone','though','I','won','t','have','any',

'pepper',

'in','my','kitchen','AT','ALL','Soup','does','very','well','without',

,'s','always','pepper','that','makes','people','hot',

‘']

可以看到，在开始和结尾都有一个空字符串(要了解原因请尝试xx.split(x)) 。通过 re.findall(r\w+',raw) 使用模式匹配词汇而不是空白符号，得到相同的标识符，但没 有空字符串。现在，匹配词汇，并扩展正则表达式覆盖更广泛的范围。正则表达式 《\w+|\Slw\*》将首先尝试匹配词中字符的所有序列。如果没有找到匹配的，它会尝试 匹配后面跟着词中字符的任何非空白字符(“\S” 是“\s” 的补)。这意味着标点会 与跟在后面的字母(如's) 在一起，但两个或两个以上的标点字符序列会被分割。

>>>re.findall(r'\w+l\S\w\*',raw)

["'When",'I',"'M",'a','Duchess',',',"'",'she','said','to','herself',

' '

, ,

'(not','in','a','very','hopeful','tone','though',')',',',"'I",'won',

"'t",

'have','any','pepper','in','my','kitchen','AT','ALL','.','Soup','does',

'very','well','without','-','-Maybe','it',"'s",'always','pepper','that',

'makes','people','hot','-tempered',',',"'",'.','.','.']

扩展前面表达式中的 “w+”, 允许连字符和撇号：《\w+([-]\w+)\*》。这个表达式表 示“\w+” 后面跟零个或更多“[-]w+” 的实例，它会匹配hot-tempered 和 it's。( 这 个表达式中需要包含“?:”,原因前面已经讨论过。)还需要添加一个模式来匹配引 号字符使它们与它们包括的文字分开。

>>>print re.findall(r"\w+(?:[-']\w+)\*I'l[-.(]+1\S\w\*",raw)

["'",'When',"I'M",'a','Duchess',',',"'",'she!,'said','to',

'herself',',',

'(','not','in','a','very','hopeful','tone','though',')',',',"'",'I',

"won't",'have','any','pepper','in','my','kitchen','AT','ALL','.',

'Soup',

'does','very','well','without','--','Maybe',"it's",'always','pepper',

'that','makes','people','hot-tempered',',',"'",'...']

在这个例子中的表达式也包括《[-(]+》,使得双连字符、省略号和左括号被单独分词。

表3-4列出了本节中的正则表达式字符类符号，以及一些其他有用的符号。

**表3-4** **正则表达式符号**

|  |  |
| --- | --- |
| **符** **号** | **功** **能** |
| \b | 词边界(零宽度) |
| \d | 任一十进制数字(相当于[0-9]) |
| \D | 任何非数字字符(等价于[^0-9]) |
| \s | 任何空白字符(相当于[\t\n\r\flv]) |
| \S | 任何非空白字符(相当于[^\t\n\r\f\v]) |
| \w | 任何字母数字字符(相当于[a-zA-Z0-9]) |
| \W | 任何非字母数字字符(相当于[^a-zA-Z0-9\_]) |
| \t | 制表符 |
| \n | 换行符 |

**NLTK** **的正则表达式分词器**

函数 nltk.regexp\_tokenize() 与 re.findall) 类似(一直在使用它进行分词)。然而， nltk.regexp\_tokenize()分词效率更高，且避免了括号特殊处理的需要。为了增强可读 性，将正则表达式分几行写，每行添加一个注释。特别的“(?x)”“verbose 标志” 告诉Python 去掉嵌入的空白字符和注释。



ttern ='Tha.s.A.pprg to lw ve. ..'regexps

· ||\  .onaers,

· |\.\.\ #ellipsis

丨 [][·,;"'?():-\_`] #these are separate tokens

111

>>>nltk.regexp\_tokenize(text,pattern)

['That','U.S.A.','poster-print','costs','S12.40','...']

使用 verbose 标志时，可以不再使用''来匹配空格字符，而用“\s” 代替。 regexp\_tokenize()函数有一个可选的 gaps 参数。设置为 True 时，正则表达式指定标 识符间的距离，就像使用 re.splitO一样。

可以使用 set(tokens).difference(wordlist), 通过比较分词结果与一个词 表，然后报告任何没有在词表出现的标识符，来评估一个分词器。也 可以先将所有标记变成小写。

**分词的进一步问题**

现在，分词是一个比预期更为艰巨的任务。没有任何单一的解决方案能在所有领域 都行之有效，我们必须根据应用领域的需要决定哪些是标识符。

在开发分词器时，访问已经手工标注好的原始文本是有益的，这可以让你的分词器的 输出结果与高品质(或称“黄金标准”)的标注进行比较。NLTK 语料库集合包括宾州 树库的数据样本，包括《华尔街日报》原始文本 (nltk.corpus.treebank raw.raw)) 和分 好词的版本 (nltk.corpus.treebank.words))。

分词的最后一个问题是缩写的存在，如“didn't”。如果想分析一个句子的意思，可 将这种形式规范化为两个独立的形式： “did” 和 “n't” (不是not) 可能更加有用。 我们可以通过查表来完成这项工作。

**3.8** **分割**

本节将讨论更高级的概念，可能会使你在第一次阅读本章时想跳过本节。

分词是一个更普遍的分割问题实例。在本节中，我们将看到这个问题的另外两个实

'I will\ntell you.',

例，它们的使用与到目前为止已经在本书中学到的技术完全不同。

**断句**

在词级水平处理文本时通常假定能够将文本划分成单个句子。正如我们了解到的， 一些语料库可提供句子级别的访问。在下面的例子中，计算布朗语料库中每个句子 的平均词数。

>>>len(nltk.corpus.brown.words())/len(nltk.corpus.brown.sents())

20.25 0994070456922

在其他情况下，文本可能只是一个字符流。在将文本分词之前，需要将它分割成句 子 。NLTK 通过包含 Punkt 句子分割器 (Kiss &Strunk,2006) 简化了这些。这里 是将它作为一篇小说文本断句的例子。(请注意，如果在读到这篇文章时分割器内 部数据已经更新过，你将会看到不同的输出。)

>>>sent\_tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')

>>>text =nltk.corpus.gutenberg.raw('chesterton-thursday.txt')

>>>sents =sent\_tokenizer.tokenize(text)

>>>pprint.pprint(sents[171:181])

['"Nonsense!',

"said Gregory,who was very rational when anyone else\nattempted paradox.',

'"Why do all the clerks and navvies in the\nrailway trains look so sad and

tired,...',

'It is because they know that the train is going right.',

'It\nis because they know that whatever place they have taken a ticket\nfor that ..',

'It is because after they have\npassed Sloane Square they know that the next stat...',

'Oh,their wild rapture!',

'oh,\ntheir eyes like stars and their souls again in Eden,if the next\ nstation w...'

'"\n\n"It is you who are unpoetical,"replied the poet Syme.']

请注意，这个例子其实是一个单独的句子，用于报道Lucian Gregory 先生的演讲。 然而，引用的演讲包含几个句子，这些已经被分割成几个单独的字符串。这对于大 多数应用程序都是合理的行为。

断句是困难的，因为句号会被用来标记缩写而另一些句号同时标记缩写和句子结

束，就像发生在缩写如 “U.S.A.” 上的那样。断句的另一种方法见6.2节。

**分词**

对于一些书写系统，由于没有词边界的可视表示这一事实，文本分词变得更加困难。

例如：在中文中，3个字符的字符串：爱国人 (ai4“love”[verb],guo3“country”,

ren2“person”) 可以被分词为“爱国/人”,“country-loving person”,或者“爱/国人”,

“love country-person”。

类似的问题在口语语言处理中也会出现，听者必须将连续的语音流分割成单个的 词汇。当事先不认识这些词时，这个问题就演变成一个特别具有挑战性的版本。

语言学习者会面对这个问题，就像小孩听父母说话一样。

考虑下面的例子，单词的边界已被去除。

(1)a.doyouseethekitty

b.seethedoggy

c.doyoulikethekitty

d.likethedoggy

第一个挑战仅仅是表示这个问题：需要找到一种方法来分开文本内容与分词标志。 可以给每个字符标注一个布尔值来指示这个字符后面是否有一个分词标志(这个 想法将在第7 章“分块”中大量使用)。假设说话人会给语言学习者一个停顿， 这往往是对应一个延长的暂停。这里是一种表示方法，包括初始的分词和最终分 词目标。

>>>text="doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"

>>>seg1="000000OOOOOOOO010OOOOOOOO01OOOOOO0OOOO0O000100000000000" >>>seg2 ="0100100100100001001001000010100100010010000100010010000"

观察由0和1组成的分词表示字符串。它们比源文本短一个字符，因为长度为n 文 本可以在n-1 个地方被分割。例3-2中的函数 segment()演示了如何从这个表示回到 初始分词的文本。

例3-2.从分词表示字符串 seg1 和 seg2 中重建文本分词。seg1 和 seg2 表示假设的 一些儿童讲话的初始和最终分词。函数 segment()可以使用它们重现分词的文本。

def segment(text,segs):

words =[]

last =0

for i in seigs)):

sd(text[last:i+1])

words.append(text[last:])

return words

>>>text ="doyouseethekittyseethedoggydoyoulikethekittylikethedoggy" >>>seg1="000000OOOOOOOO0100OOOOOOO010OOOOOOOOOOO0000100000000000"

>>>seg2="0100100100100001001001000010100100010010000100010010000" >>>segment(text,seg1)

['doyouseethekitty','seethedoggy','doyoulikethekitty','likethedoggy']

o,g'','',y',d,'he','doggy','do','you',

现在分词的任务变成了一个搜索问题：找到将文本字符串正确分割成词汇的字位 串。假定学习者接收词，并将它们存储在一个内部词典中。给定一个合适的词典是 能够由词典中词的序列来重构源文本的。读过Brent &Cart-wright(1995) 之后， 可以定义一个**目标函数，**一个打分函数，将基于词典的大小和从词典中重构源文本 所需的信息量尽量优化它的值。在图3-6中说明了这些。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SEGMENTATION**  doyou See thekitty  see thedogg  doyou like thekitt  ike thedogc | **REPRESENTATION**   |  |  | | --- | --- | | LEXICON  1.doyou  2.see  3.like  4.thekitt  5.thedogg  6.y | DERIVATION  2 4 6  5 | | **OBJECTIVE**  LEXICON:  6+4+5+8+8+2=33  DERIVATION:  4+3+4+3=14  TOTAL:  33+14=47 |

**图3-6** **计算目标函数：给定一个假设的源文本的分词(左),推导出一个词典和推导表，它能**

**让源文本重构，然后合计每个词项(包括边界标志)与推导表的字符数，作为分词质量的得分；** **得分值越小表明分词越好**

实现这个目标函数是很简单的，如例3-3所示。

例3-3 计算存储词典和重构源文本的成本

def evaluate(text,segs):

words =segment(text,segs)

text\_size =len(words)

lexicon\_size =len(' '.join(list(set(words))))

return text\_size +lexicon\_size

>>>text="doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"

>>>seg1="000000OOOOOOOO010OOOOOOOO01OOOOOOOOOO0O0000100000000000"

>>>seg2 ="0100100100100001001001000010100100010010000100010010000" >>>seg3 ="0000100100000011001000000110000100010000001100010000001"

''','thedogg','y','doyou','like',

>>>evaluate(text,seg3)





63

最后一步是寻找最大化目标函数值0和1的模式，如例3-4所示。请注意，最好的分 词包括像 “thekitty” 这样的“词”,因为数据中没有足够的证据进一步分割这个词。

例3-4 使用模拟退火算法的非确定性搜索： 一开始仅搜索短语分词；随机扰动 0和1,它们与“温度”成比例；每次迭代温度都会降低，扰动边界会减少

from random import randint

def flip(segs,pos):

return segs[:pos]+str(1-int(segs[pos]))+segs[pos+1:]

def flip\_n(segs,n):

for i in range(n):

segs =flip(segs,randint(0,len(segs)-1))

return segs

def anneal(text,segs,iterations,cooling\_rate):

temperature =float(len(segs))

while temperature >0.5:

best\_segs,best =segs,evaluate(text,segs)

for i in range(iterations):

=evalpeund(temperature)))

if score <best:

best,best\_segs =score,guess

score,segs =best,best\_segs

temperature =temperature /cooling\_rate

print evaluate(text,segs),segment(text,segs)

print

return segs

>>>text ="doyouseethekittyseethedoggydoyoulikethekittylikethedoggy" >>>seg1="00000O0OO0OOOO010OOOOOOO00100OOOOOO0OOO0OO0100000000000"

>>>anneal(text,seg1,5000,1.2)

60 ['doyouseetheki','tty','see','thedoggy','doyouliketh','ekittylike', 'thedoggy']

58['doy','ouseetheki','ttysee','thedoggy','doy','o','ulikethekittylike', 'thedoggy']

56['doyou','seetheki','ttysee','thedoggy','doyou','liketh','ekittylike',

'thedoggy']

54 ['doyou','seethekit','tysee','thedoggy','doyou','likethekittylike', thedoggy']

53 ['doyou','seethekit','tysee','thedoggy','doyou','like','thekitty', 'like','thedoggy']

51['doyou','seethekittysee','thedoggy','doyou','like','thekitty','like', thedoggy']

42 ['doyou','see','thekitty','see','thedoggy','doyou','like','thekitty', 'like','thedoggy']

'0000100100000001001000000010000100010000000100010000000’

有了足够的数据，就可能以一个合理的准确度自动将文本分割成词汇。这种方法可 用于为那些词的边界没有任何视觉表示的书写系统分词。

**3.9** **格式化：从链表到字符串**

我们经常会编写程序以输出单独的数据项，例如语料库中满足一些复杂标准的特定 元素，或者单独的总数统计，例如词计数器或标注器的性能。更多的时候，我们编 写程序以产生结构化的结果，例如： 一个数字或语言形式的表格，或原始数据的格 式变换。当结果是语言时，应选择文字输出。然而当结果是数值时，最好选择图形 输出。在本节中，你将会学到各种表现程序的输出方式。

**从链表到字符串**

用于文本处理最简单的结构化对象是词链表。当需要把这些输出到显示器或文件 时，必须把这些词的链表转换成字符串。在Python中，使用join()方法，并指定作 为“胶水”使用的字符串。

>>>silly=['We','called','him','Tortoise','because','he','taught', 'us','.'>]>>''.join(silly)

'We called him Tortoise because he taught us .'

>>>';'.join(silly)

e>;c''aidn;mill;)ortoise;because;he;taught;us;.'

'WecalledhimTortoisebecausehetaughtus.'

''join(silly) 的意思是：取出 silly 中的所有项目，将它们连接成一个大的字符串，使 用''作为项目之间的间隔符，即join()是一种将字符串作为胶水的方法。(许多人感 到join()这种表示方法是违反直觉的。)join()方法只适用于一种字符串的链表——我 们一直把它叫做一个文本——一种在 Python 中享有某些特权的复杂类型。

**字符串与格式**

共有两种方式可显示对象的内容。

>>>word = 'cat'

>>>sentence ="""hello

rint wrd

cat

>>>print sentence

hello

world

>>>word

'cat'

>>> sentence

'hello\nworld'

print 命令产生的是 Python 尝试以可读的形式输出的一个对象的内容。第二种方 法——叫做变量提示——向我们显示可用于重新创建该对象的字符串。重要的是要 记住这些都仅仅是字符串，为了用户的方便而显示的。它们并不会给我们实际对象 内部表示的任何线索。

还有许多其他有用的方法以字符串的形式来展示对象。这可能是为了方便人们阅读， 或是需要将数据导入一个特定的能被外部程序使用的文件格式。

格式化输出通常包含变量和预先指定字符串的组合。例如：给定一个频率分布 fdist, 可以进行如下操作。

>>>fdist =nltk.FreqDist(['dog','cat','dog','cat','dog','snake',

'dog', >>>'c word in fdist:

print word,'->',fdist[word],';',

dog ->4;cat ->3; snake ->1;

除了不必要的空格符问题，包含交替出现变量和常量的表达式是难以阅读和维护的。 更好的解决方法是使用**字符串格式化表达式。**

>>>for word in fdist:

print'8s->8d;'8(word,fdist[word]),

dog->4;cat->3;snake->1;

要了解这里发生了什么事情，需要测试字符串格式化表达式。(这是探索新语法的常 用方法。)

>>>'%s->%d;’8('cat',3)

'cat->3;'

>>> '%s->%d;’%'cat'

Traceback(most recent call last):

File "<stdin>",line 1,in <module>

TypeError:not enough arguments for format string

特殊符号%s 和 %d 是字符串和整数(十进制数)的占位符。可以将这些嵌入在字符 串中，然后使用%操作符把它们组合起来。让我们进一步解开这段代码，以便更 仔细地观察它的行为。

>>>'8s->' 'cat'

'cat->'

>>>'8d' 3

·3'

>>>'I want a 8s right now'8 'coffee'

'I want a coffee right now'

可以有多个占位符，但%操作符后面必须指定数目完全相同的数值的元组。

>>>"%s wants a 8s 8s"g("Lee","sandwich","for lunch")

'Lee wants a sandwich for lunch'

还可以间接地提供占位符的值。下面是一个使用 for 循环的例子。

>>>template ='Lee wants a ss right now'

>>>menu =['sandwich','spam fritter','pancake']

>>>for snack in menu:

print template g snack

\*

Lee wants a sandwich right now

Lee wants a spam fritter right now

Lee wants a pancake right now

符号%s 和 %d 被称为**转换说明符**。它们以%字符开始，以一个转换字符如s (表示 字符串)或 d (十进制整数)结束。其中包含转换说明符的字符串被称为**格式字** **符串。**组合格式字符串和%操作符及值的元组，来创建一个完整的字符串格式化表 达式。

**排列**

到目前为止，格式化字符串可以在页面(或屏幕)上输出任意的宽度，如%s 和 %d。 也可以指定宽度，如%6s, 产生一个宽度为6的字符串。缺省情况是右对齐的”, 也可以包括一个减号使它左对齐②。如果事先不知道要显示的值应该有多宽时，可 以在格式化字符串中用\*替换宽度值，然后再指定一个变量。

>>>'86s’ 号 'dog′①

' dog'

>>>'8-6s’8'dog’②

'dog'

>>>width =6

>>>‘8-\*s’%(width,'dog')③

dog

其他控制字符用于十进制整数和浮点数。因为百分号%在格式化字符串中有特殊解 释，必须要在它前面加另一个%才能输出它。

>>>count,total =3205,9375

>>>"accuracy for sd words:82.4f%%"%(total,100 \*count /total) 'accuracy for 9375 words:34.1867%'

格式化字符串的一个重要用途是用于数据制表。回想一下，在2.1 节中从条件频率 分布中制表的数据。让我们自己来制表，实现对标题和列宽的完全控制，如例3-5 所示。注意语言处理工作与结果制表之间是明确分离的。

例3-5 布朗语料库不同部分的频率模型。

def tabulate(cfdist,words,categories):

print 'g-16s'%'Category',

for word in words: #column headings

print '%6s'8 word,

print

for category in categories:

print 'g-16s'8 category, #row heading

for word in words: #for each word

print '86d'8 cfdist[category][word], #print table cell

print #end the row

>>>from nltk.corpus import brown

>>>cfd =nltk.ConditionalFreqDist(

 (genre,word)

>scaco(e)rie\_ftmance',

'humor']

>>>modals =['can','could','may','might','must','will']

>>>tabulate(cfd,modals,genres)

Category can could may might must will

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| news | 93 | 86 | 66 | 38 | 50 | 389 |
| religion | 82 | 59 | 78 | 12 | 54 | 71 |
| hobbies | 268 | 58 | 131 | 22 | 83 | 264 |
| science\_fiction | 16 | 49 | 4 | 12 | 8 | 16 |
| romance | 74 | 193 | 11 | 51 | 45 | 43 |
| humor | 16 | 30 | 8 | 8 | 9 | 13 |

回顾在例3-1 列出的格式化字符串“%\*s”, 通过使用变量指定字段的宽度。

>>>'%\*s'%(15,"Monty Python")

'Monty Python'

使**用** **width=max(len(w)for w in words**)自动定制列的宽度，使其足够容纳所有的词。 要记住 print 语句结尾处的逗号要增加一个额外的空格，这样能够防止列标题相互 重叠。

**将结果写入文件**

我们已经学习了如何读取文本文件(见3.1 节)。将输出写入文件往往也很有用。

打开下面的代码可编写文件 output.txt, 将程序的输出保存到文件中。

dust\_filset(npceonr't'o')rds('english-kjv.txt'))

>>>for word in sorted(words):

output\_file.write(word +"\n")



said (4),

**轮到你来：**

在写入文件之前，为每个字符串追加\n 会有什么影响?如果使用的是 Windows机器，可以用 word+"rln" 代替。如果输入outputfile.write(word) 会发生什么?

当将非文本数据写入文件时，必须先将它转换为字符串。正如前面所讲的那样，可 以使用格式化字符串进行这一转换。关闭文件之前，需要把总词数写入文件。

>>>len(words)

2789

>>>str(len(words))

'2789'

>>>output\_file.write(str(len(words))+"\n")

>>>output\_file.close()

**注意!**

你应该避免文件名中包含空格字符，例如： output file.txt,避免使用除了 大小写区别外其他都相同的文件名，例如： Outputtxt与 output.TXT。

**文本换行**

当程序的输出是文档式的而不是表格式时，通常会有必要包装一下以便可以方 便地显示它。考虑下面的输出，它的行尾溢出了，且使用了一个复杂的 print 语句。

|  |
| --- |
| >>> saying =['After','all','is','said','and','done',,,  more','is','said','than','done",'.']  for word in saying:  print word,'(+str(len(word))+'),',  After r(5),all(3),is(2),said(4),and(3),done(4),,(1),more(4),is(2), |

可以在 Python 的 textwrap 模块下采取换行。为了最大程度的清晰，将一个步骤分 在一行。

>>>from textwrap import fill

>>>format ='&s(%d),'

>>>pieces=[format s(word,len(word))for word in saying]

>>>output=''.join(pieces)

>>>wrapped =fill(output)

>>>print wrapped

(4),is(2),said(4),than(4),done

After(5),all(3),is(2),said(4),and(3),done(4),,(1),more

(4),.(1),

请注意，在 more 与其下面的数字之间有一个换行符。如果希望避免这种情况，可 以重新定义格式化字符串，使它不包含空格(例如：“%s\_(%d),'), 然后不输出wrapped 的值而是输出wrapped.replace(\_,'')。

**3.10** **小结**

● 在本书中，我们将文本作为一个词链表。“原始文本”是一个潜在的长字符串， 其中包含文字和用于设置格式的空白字符，也是我们通常存储和可视化文本的 原料。

· 在 Python 中指定一个字符串使用单引号或双引号： Monty Python' 、"Monty Python"。

· 字符串中的字符是使用索引来访问的，索引从零计数： 'Monty Python'[0]的值 是M。求字符串的长度可以使用lenO。

● 子字符串使用切片符号访问： 'Monty Python'[1:5]的值是onty。如果省略起始索 引，子字符串从字符串的开始处开始。如果省略结尾索引，切片会一直到字符 串的结尾处结束。

● 字符串可以被分割成链表：: Monty Python'split()得到[Monty','Python']。链表可 以连接成字符串：V.join([Monty',Python]) 得到'Monty/Python'。

● 可以使用 text =open(f),read()从一个文件 f 读取文本。可以使用 text = urlopen(u).read)从一个URLu 读取文本。可以使用 for line in open(f) 遍历文本 文件的每一行。

· 在网上找到的文本可能包含不需要的内容(如页眉、页脚和标记),在进行任 何语言处理之前需要去除它们。

● 分词是将文本分割成基本单位或标记，例如词和标点符号等。基于空格符的分 词对于许多应用程序都是不够的，因为它会捆绑标点符号和词。NLTK 提供了 一个现成的分词器nltk.word\_tokenize()。

· 词形归并是一个过程，将一个词的各种形式(如： appeared,appears) 映射到

这个词标准的或引用的形式，也称为词位或词元(如： appear)。

● 正则表达式是用来指定模式的一种强大而灵活的方法。只要导入了re模块，就 可以使用re.findall()找到一个字符串中匹配一个模式的所有子字符串。

● 如果正则表达式字符串包含反斜杠，应该使用原始字符串与r前缀：r'regexp',告 诉Python不要预处理这个字符串。

● 如果某些字符前使用了反斜杠，例如：\n, 处理时会有特殊的含义(换行符)。 然而，当反斜杠用于正则表达式通配符和操作符时，如：\.,V,\$, 这些字符 失去其特殊的含义，只按字面表示匹配。

● 一个字符串格式化表达式template %arg\_tuple 包含一个格式字符串 template, 它由如%-6s 和%0.2d 这样的转换标识符组成。

**3.11** **深入阅读**

本章的课外材料发布在 <http://www.nltk.org/>, 包括网络上免费提供的资源的链接。 记得在 <http://docs.python.org/>查看 Python 的参考材料。(例如：此文档涵盖“通用 换行符支持”,解释了各种操作系统如何规定不同的换行符。)

有关使用 NLTK 处理词汇的更多例子请参阅 <http://www.nltk.org/howto> 上的分词、 词干提取及语料库 HOWTO文档。(Jurafsky&Martin,2008) 的第2章和第3章包 含有关正则表达式和形态学更高级的材料。关于Python 文本处理更广泛的讨论请参 阅 (Mertz,2003) 。 规范非标准词的信息请参阅 (Sproat et al.,2001)。

关于正则表达式的参考材料很多，无论是理论的还是实践的。在 Python 中使用正 则表达式的入门教程，请参阅Kuchling's Regular Expression HOWTO,http://www. amk.ca/python/howto/regex/。

关于使用正则表达式的全面而详细的手册，请参阅 (Friedl,2002), 其中涵盖包括 Python 在内大多数主要编程语言的语法。其他材料还包括(Jurafsky & Martin, 2008)的第2.1节， (Mertz,2003) 的第3章。

网上有许多关于 Unicode 的资源。以下是与处理 Unicode 的 Python 的工具有关的有 益的讨论。

· PEP-100 <http://www.python.org/dev/peps/pep-0100/>

· Jason Orendorff,Unicode for Programmers,<http://wwwjorendorff.com/articles/> unicode/

· A.M.Kuchling,Unicode HOWTO,<http://www.amk.ca/python/howto/unicode>

· Frederik Lundh,Python Unicode Objects,<http://effbot.org/zone/unicode-objects.htm>

· Joel Spolsky,The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets(No Excuses!), <http://wwwjoelonsoftware.com/articles/Unicode.html>

SIGHAN,ACL 中文语言处理特别兴趣小组 (<http://sighan.org/>), 重点关注中文文

本分词的问题。分割英文文本的方法依据 (Brent &Cartwright,1995); 这项工作

属于语言获取领域 (Niyogi,2006)。

搭配是**多词表达式的**一种特殊情况。 一个多词表达式是一个小短语，仅从它的词汇 不能预测它的意义和其他属性，例如： part-of-speech(Baldwin&Kim,2010)。

模拟退火是一种启发式算法，以求得大型离散的搜索空间内的函数的最佳值，同时 基于对金属冶炼中的退火模拟。该技术在许多人工智能文本中都有描述。

(Hearst,1992) 描述了使用如xand other ys的搜索模式发现文本中下位词的方法。

**3.12** **练习**

1.O 定义一个字符串s='colorless' 。 编写Python 语句将其变为 “colourless”, 只使

用切片和连接操作。

2.O 使用切片符号删除词汇形态上的结尾。例如：'dogs'[:-1]删除了dogs 的最后 一个字符，留下 dog。使用切片符号删除下面这些词的词缀(插入连字符指示词 缀的边界，在字符串中省略掉连字符): dish-es、run-ning 、nation-ality 、un-do、

pre-heat。

3.O 我们学过当索引超出字符串的末尾时，将产生 IndexError。构造一个超出字符 串前端的索引，这有可能吗?

4.O 可以为切片指定一个“步长”。下面的表达式每隔切片内一个字符返回一次： monty[6:11:2]。也可以反向进行： monty[10:5:-2] 。自己尝试一下，然后更换不同的 步长。

5.O 如果让解释器处理 monty[::-1] 会发生什么?解释为什么这是一个合理的 结果。

6.O 说明以下的正则表达式匹配的字符串类。

a.[a-zA-Z]+

b.[A-Z][a-z]\*

c.p[aeiou]{,2}t

d.\d+(\.\d+)?

e.([^aeiou][aeiou][^aeiou])\*

f.\w+l[^\w\s]+

使用 nltk.re\_show()测试你的答案。

7.O 编写正则表达式匹配下面字符串类。

a. 一个单独的限定符(假设只有 a 、an 和 the 为限定符)。

b. 整数加法和乘法的算术表达式，如：2\*3+8。

8.O 编写一个工具函数以URL 为参数，返回删除所有的HTML 标记的URL 的内容。 使用urllib.urlopen访问的URL的内容，例如：raw\_contents=urllib.urlopen(http://www.nltk. org/).read)

9.O 将一些文字保存到文件 corpus.txt。定义一个函数load(f)以要读取的文件名为 其唯一参数，返回包含文件中文本的字符串。

a. 使用 nltk.regexp\_tokenize)创建一个分词器分割这个文本中的各种标点符号。

使用一个多行的正则表达式，行内要有注释，使用 verbose 标志(?x)。

[('The',3),('dog',3),('gave',4),('John',4),('the',3),('newspaper',9)]

b. 使用 nltk.regexp\_tokenizeO创建一个分词器，分割以下几种表达式：货币金额；

日期；个人和组织的名称。

10.O 将下面的循环改写为链表推导。

esult =[] =['The','dog', 'gave','John','the','newspaper']

>>>for word in sent:

word\_len =(word,len(word))

.. result.append(word\_len)

>>>result

11.O 定义一个字符串raw 包含你自己选择的句子。现在，分裂raw 的一些字符以 外的空间，例如：‘s’。

12.O 编写一个 for 循环输出一个字符串的字符，每行一个。

13.O 不带参数的 split 与以''作为参数的 split, 即 sent.splitO与 sent.split('') 的区别 是什么?当被分割的字符串包含制表符、连续的空格或一个制表符与空格的序列会 发生什么? ( 在IDLE 中使用\t'来输入制表符。)

14.O 创建一个变量 words, 包含一个词链表。实验 words.sort()和 sorted(words) 。 它们有什么区别?

15.O 通过在Python 提示符输入以下表达式，探索字符串和整数的区别："3"\*7 和3\*7。尝试使用 int("3")和 str(3)进行字符串和整数之间的转换。

16.O 之前，我们要求你用文本编辑器创建一个名为 test.py 的文件，只包含一 行 monty='Monty Python'。如果你还没有这样做(或无法找到该文件),去吧!现 在就做。接下来，打开一个新的 Python 解释器，并在提示符下输入表达式 monty。 你会从解释器得到一个错误。现在，请尝试以下代码(注意你要丢弃文件名中 的.py)。

>>>from test import msg

>>>msg

这一次，Python 应该返回一个值。你也可以尝试import test, 在这种情况下， Python

应该能够处理提示符处的表达式test.monty。

17.O 格式化字符串%6s 与%-6s 用来显示长度大于6个字符的字符串时，会发生 什么?

18.①阅读语料库中的一些文字，为它们分词，输出其中出现的所有 wh-类型 词的列表。(英语中的 wh-类型词被用在疑问句、关系从句和感叹句： who 、 which 、what等。)按顺序输出它们。在这个列表中含有因大小写或标点符号而 重复的词吗?

19.①创建一个文件，包含词汇和(任意指定)频率，其中每行包含一个词、 一个 空格和一个正整数，如：fuzzy 53。使用open(filename).readlines()将文件读入 Python 链表。接下来，使用 split) 将每一行分成两个字段，并使用int()将其中的数字转换 为整数。结果要是链表形式：[[fuzzy',53],…] 。

20.①编写代码以访问喜爱的网页，并从中提取一些文字。例如，访问一个天气网 站，提取你所在的城市今天的最高温度预报。

21.①编写一个函数 unknown(), 以 URL 为参数，返回一个出现在网页上的未知 词链表。为了做到这一点，请提取所有由小写字母组成的子字符串(使用 re.findall)), 并去除所有在Words语料库中出现的项目(nltk.corpus.words) 。尝试 手动分类这些词，并讨论你的发现。

22.①使用上面建议的正则表达式处理网址<http://news.bbc.co.uk/>, 检查处理结果。 你会看到那里仍然有相当数量的非文本数据，特别是JavaScrijpt 命令。你可能还会 发现句子分割没有被妥善保留。定义更深入的正则表达式，改善此网页文本的提取。

23.①你能编写出一个正则表达式以下面的方式来分词吗?将词 don't 分为 do 和 nt? 解释为什么这个正则表达式无法正常工作：《n't|w+》。

24.0尝试编写代码将文本转换成hAck3r, 使用正则表达式和替换，其中e→3,i→ 1, o→0,1→ |,s→5,.→5w33t! ,ate→8 。 在转换之前将文本规范化为小写。可以自行 添加更多的替换。现在尝试将s 映射到两个不同的值：词开头的s 映射为$,词内部的 s 映射为5。

25.OPig Latin 是英语文本的一种简单的变换。文本中每个词按如下方式变换：将 出现在词首的所有辅音(或辅音群)移到词尾，然后添加 ay, 例如：string→ingstray, idle→idleay (见 <http://en.wikipedia.org/wiki/Pig>\_Latin)。

a. 编写一个函数转换一个词为Pig Latin。

b. 编写代码转换文本而不是单个的词。

c. 进一步扩展它，保留大写字母，将qu 保持在一起(例如：这样quiet 会变 成ietquay), 并检测y 是作为一个辅音(如：yellow) 还是元音(如： style)。

26.①下载一种包含元音和谐的语言(如匈牙利语)的一些文本，提取词汇的元音 序列，并创建一个元音二元语法表。

27. OPython 的 random 模块包括函数 choice(), 它从一个序列中随机选择一个项 目。例如： choice("aehh ")会产生4种可能字符中的一个，字母h 的几率是其他字 母的两倍。编写一个表达式产生器，从字符串"aehh" 产生500个随机选择的字母的 序列，并将这个表达式写入函数"join()调用中，将它们连接成一个长字符串。你得 到的结果应该看起来像失去控制的喷嚏或狂笑： he haha ee heheeh eha。使用splitO 和join()再次规范化这个字符串中的空格。

28.①考虑下面的摘自MedLine语料库的句子中的数字表达式： The corresponding free cortisol fractions in these sera were 4.53 +/-0.15%and 8.16+/-0.23%,respectively. 我们应该说数字表达式4.53+/-0.15%是3个词吗?或者我们应该说它是一个单独 的复合词?或者我们应该说它实际上是9个词，因为它读作 “four point five three,plus or minus fifteen percent”?或者我们应该说这不是一个“真正的”词，因 为它不会出现在任何词典中?讨论这些不同的可能性。你能想出产生这些答案中至 少两个以上可能性的应用领域吗?

29.①可读性测量用于为一个文本的阅读难度打分，给语言学习者挑选适当难度的 文本。在一个给定的文本中，定义μw为每个词的平均字母数，μ₂为每个句子的平 均词数。文本自动可读性指数(Automated Readability Index,缩写 ARI) 被定义为： **4.71μw+0.5μ₃-21.4 3。**计算布朗语料库各部分的ARI 得分，包括f部分(popular lore)

和 j(learned) 。 利 用 nltk.corpus.brown.wordsO 产 生 一 个 词 汇 序 列 ，

nltk.corpus.brown.sents()产生一个句子的序列的事实。

30. ①使用 Porter 词干提取器规范化一些已标注的文本叫做为每个词提取词干。用 Lancaster 词干提取器做同样的事情，看看你是否能观察到一些差别。

31.①定义变量 saying 包含链表[After,'all,is,'said,and,'done',,,'more,'is,'said,

'than','done,'.] 。 使用 for 循环处理这个链表，并将结果存储在一个新的链表 lengths 中。提示：以使用 lengths =[], 分配一个空链表给 lengths 为开始。然后每次循环 中用 append) 添加另一个长度值到链表中。

32.①定义一个变量 silly 包含字符串： newly formed bland ideas are inexpressible in an infuriating way'。(这碰巧是合法的解释，讲英语和西班牙语的双语者可以理解乔 姆斯基著名的无意义短语： colorless green ideas sleep furiously, 来自维基百科)。现 在编写代码执行以下任务。

a. 分割 silly 为一个字符串链表， 一个词一个字符串，使用Python 的 split) 操 作，并保存到叫做 bland的变量中。

b. 提 取 silly 中每个词的第二个字母，将它们连接成一个字符串，得到 'eoldrnnnna'。

c. 使用join() 将 bland 中的词组合到一个单独的字符串。确保结果字符串中的 词以空格隔开。

d. 按字母顺序输出 silly 中的词，每行一个。

33.OindexO 函数可用于查找序列中的项目。例如：'inexpressible.index('e') 表示字 母e 的第一个位置的索引值。

a. 当查找子字符串如： 'inexpressible.index(re') 时会发生什么?

b. 定义一个变量 words 包含一个词链表。现在使用 words.index()来查找一个 单独的词的位置。

c. 定义一个练习32中的变量 silly。使用 index()函数结合链表切片，建立一个 包括 silly 中 in 之前(但不包括in) 的所有词的链表 phrase。

34.①编写代码，将国家的形容词转换为它们对应的名词形式，如将 Canadian 和

Australian 转换为 Canada 和 Australia (见<http://en.wikipedia.org/wiki/List>\_of\_adjectival\_ forms\_of place\_names)。

35.①阅读 LanguageLog中关于短语的 as best as p can 和 as best p can 形式的帖子， 其 中p是一个代名词。在语料库和3.5节中描述的搜索已标注的文本的 findallO方法的 基础下，调查这一现象。帖子附在 http:/itre.cis.upenn.edu/～ myl/languagelog/ archives/002733.html 中。

36.①研究《创世记》的lolcat版本(使用 nltk.corpus.genesis.words(lolcat.txt)可以访 问),和将文本转换为 <http://www.lolcatbible.com/index.php?title-How> to speak\_lolcat. 处的lolspeak 的规则。定义正则表达式将英文词转换成相应的 lolspeak 词。

37.①使用 help(re.sub)并参照本章的深入阅读，阅读有关 re.sub()函数使用正则表 达式进行字符串替换的内容。使用 re.sub 编写代码从一个 HTML 文件中删除 HTML标记，规范化空格。

38.●分词有趣的挑战是已经被分割的跨行的词。例如：如果long-term被分割， 将得到字符串long-lnterm。

a. 编写一个正则表达式，识别连字符连接的跨行处的词汇。这个表达式需要 包含\n 字符。

b. 使用 re.sub()从这些词中删除\n字符。

c. 如何确定 一 旦换行符被删除后不应该保留连字符的词汇，如： encyclo-lnpedia'?

39.●阅读维基百科有关 Soundex 条目。使用 Python 实现这个算法。

40.●获取两个或多个文体的原始文本，计算它们各自在前面关于阅读难度的练习 中描述的阅读难度得分。例如： 比较 ABC 农村新闻和 ABC 科学新闻 (nltk.corpus.abc) 。使用Punkt处理句子分割。

41.●将下面的嵌套循环重写为嵌套链表推导。

>>>words



=['attribution','confabulation','elocution',

'sequoia','tenacious','unidirectional']

>>>vsequences =set()

>>>for ls in =w[]ords:

\*

· · for char in word:

· if char in 'aeiou':

vowels.append(char)

... vsequences.add(''.join(vowels))

d'eiuio','euoia','oauaio','uiieioa']

42.●使用 WordNet 为一个文本集合创建语义索引。扩展例3.1 中的一致性搜索程 序，使用它的第一个同义词集偏移索引每个词，例如： wn.synsets(dog)[0].offset( 或 者使用上位词层次中的一些祖先的偏移，这是可选的)。

43.●在多语言语料库如世界人权宣言语料库(nltk.corpus.udhr), 和 NLTK 的频率 分布和关系排序的功能 (nltk.FreqDist,nltk.spearman\_correlation) 的基础下，开发 一个系统，猜测未知文本。为简单起见，使用单一的字符编码和几种语言。

44.●编写一个程序处理文本，发现某个词以一种新的意义被使用的情况。对于每 个词计算其所有同义词集与其上下文的所有同义词集之间的 WordNet 相似性。(请 注意，这是一个粗略的办法，要做得好是很困难的，属开放性研究问题。)

45. ●阅读关于规范化非标准词的文章(Sproat et al.,2001),实现一个类似的文字 规范系统。



**第4章**

**编写结构化程序**

现在，对 Python 编程语言处理自然语言的知识已经有了体会。不过，如果你是 Python 或者编程新手，你仍然要努力对付 Python, 并尚未感觉到能完全掌握它。 在这一章中，我们将解决以下问题。

(1)怎么能写出结构良好、可读的程序，使你和其他人能够很容易地重用它?

(2)基本结构块，如循环、函数及赋值，是如何执行的?

(3)Python 编程的陷阱有哪些，你将如何避免它们?

在本章，你将巩固基本的编程结构，以一种自然和简洁的方式了解更多关于使用 Python 语言特征的内容，并学习一些有用的自然语言数据可视化技术。如前所述， 本章包含许多例子和练习(和以前一样， 一些练习会引入新材料)。编程新手应仔 细做完它们，并在需要时查询其他编程介绍；有经验的程序员可以快速浏览本章。

在这本书的其他章节中，为了讲解NLP 的需要，我们收集了一些编程的概念。在这里， 我们使用一个更传统的方法，使材料更紧密地与编程语言的结构联系在一起。这里不 会完整地讲述编程语言，我们只关注对NLP 最重要的语言结构和习惯用法。

**4.1** **回到基础**

**赋值**

赋值似乎是最基本的编程概念，不值得单独讨论。不过，也有一些令人吃惊的微妙

之处。思考下面的代码片段。

>>>foo ='Monty'

>>>bar =foo ①

>>>foo ='Python’②

>>>bar

'Monty'

这个结果与预期的完全一样。当在代码中写入 bar =foo°时， foo 的值(字符串 'Monty') 被赋值给 bar。也就是说， bar 是foo 的一个副本，所以当第②行用一个新 的字符串Python'覆盖foo 时 ，bar 的值不会受到影响。

然而，赋值语句并不总是以这种方式复制副本。赋值总是一个表达式值的复制，但 值并不总是你希望的那样。特别是结构化对象的“值”,例如一个链表，实际上是 一个对象的引用。在下面的例子中，①将 foo 的引用分配给新的变量 bar 。现在， 修改 foo 内容②时，bar 的内容也会发生改变。

>>>foo =['Monty','Python']

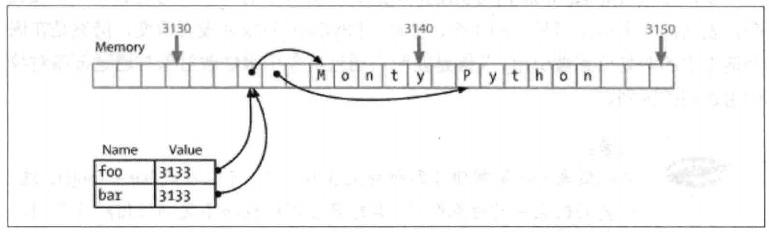
>>>bar =foo ①

>>>foo[1]='Bodkin'②

>>>bar

['Monty','Bodkin']

bar=foo° 行并不会复制变量的内容，只有它的“引用对象”。要了解这里发生了什 么事，我们需要知道链表是如何存储在计算机内存的。在图4.1中，我们看到链表 foo 是存储在位置3133处(其本身就是一系列包含字符串指向其他位置的指针)的 一个对象的引用。当赋值 bar=foo 时，仅仅是3133位置处的引用被复制。这种行 为可延伸到语言的其他方面，如参数传递(见4.4节)。



**图4-1** **链表赋值与计算机内存：两个链表对象foo** **和bar引用计算机内存中的相同的位置；更**

**新foo 将会修改bar, 反之亦然**



**创建一个空链表变量** **empty, 然后在下一行使用它三次。**

>>>empty =[]

>>>nested =[empty,empty,empty]

>>>nested

[[], [], []]

>>>nested[1].append('Python')

>>>nested

[['Python'],['Python'],['Python']]

观察发现，改变链表中嵌套链表内的一个项目，它们就全改变了。这是因为3个元 素中的每一个实际上都只是被内存中的同一链表所引用。

**轮到你来：**

用乘法创建一个链表的链表： n ested =[[]] \*3。 现在修改链表中的一 个元素，观察到所有的元素都改变了。使用 Python的 id(函数找出任 一对象的数字标识符，并验证id(nested[0]),id(nested[1])与 id(nested[2]) 是一样的。

现在请注意，当分配新值给链表中的某个元素时，它并不会传送给其他元素。

>>>nested =[[]]\*3

>>>nested[1].append('Python')

>>>nested[1]=['Monty']

>>>nested

[['Python'],['Monty'],['Python']]

一开始用含有3个引用的链表，每个引用指向一个空链表对象。然后，通过给它追 加Python'来修改这个对象，结果生成了3个指向同一个链表对象[Python']的链表。 下一步，使用新对象[Monty] 的引用来覆盖3个对象中的一个。这最后一步修改嵌 套链表内的3个引用对象中的1个。然而， [Python']对象并没有改变，仍然是在嵌

套链表中的两处位置被引用。关键是要明白通过对象引来修改对象与通过覆盖对象 引用之间的区别。

**重要：**

要从链表foo 复制项目到新的链表bar 中，可以写成 bar=foo[:] 。这 会复制链表中的对象引用。若想只复制结构而不复制任何对象引用， 可以使用copy.deepcopyO 。

>>>snake\_nest

**等式**

Python 提供了两种方法来检查一对项目是否相同。is 操作符测试对象的标识符。可 以用它来验证早先对对象的观察结果。首先，创建一个链表，其中包含同一对象的 多个副本，证明它们不仅对于==完全相同，而且它们是同一个对象：

>>>size=5

>>>python =['Python']

>>>snake\_nest =[python] \*size

>>>snake\_nest[0]==snake\_nest[1]==snake\_nest[2]==snake\_nest[3]== snake\_nest[4]

True

>>>snake\_nest[0] is snake\_nest[l]is snake\_nest[2]is snake\_nest[3]is snake\_nest[4]

True

现在，将一个新的 python 放入嵌套中。可以很容易地表明这些对象不完全相同。

odom.choice(range(size))

>>>snake\_nest[position]=['Python']

[['Python'],['Python'],['Python'],['Python'],['Python']]

>>>snake\_nest[0]==snake\_nest[1]==snake\_nest[2]==snake\_nest[3]== snake\_nest[4]

True

>>>snake\_nest[0]is snake\_nest[1]is snake\_nest[2]is snake\_nest[3]is snake\_nest[4]

False

你可以再做几对测试，发现哪个位置包含闯入者(不同的数字),函数 idO使检测 变得更加容易。

>>> [id(snake)for snake in snake\_nest]

[513528,533168,513528,513528,513528]

这表明链表中的第二个项目有一个独特的标识符。如果你尝试运行这段代码，希望 看到结果链表中的不同数字，如果“闯入者”(不同的数字)在不同的位置，请不 要奇怪。

有两种等式可能看上去有些奇怪。然而，这真的只是类型与标识符式的区别，与自

然语言相似，只是在这里用一种编程语言中呈现出来。

**条件语句**

在 if 语句的条件部分， 一个非空字符串或链表被判定为真，而一个空字符串或链表 被判定为假。

>>>mixed=['cat','',['dog'],[]]

>>>for element in mixed:

.

if

…

·

element:

print element

c.t

['dog']

也就是说，不必在条件中写入： iflen(element)>0:。

使用 if…elif与在一行中使用两个 if 语句有什么区别呢?考虑以下情况。

>>>animals =['cat','dog']

>>>if 'artin1animals:

……

… elif 'dog'in animals:

print 2

… ·

1

因为表达式中 if 子句条件满足，Python就不会比较elif子句，所有程序永远不会输 出2。相反，如果用 if 替换elif, 那么程序会输出1和2。所以elif 子句比单独的if 子句能提供更多潜在的信息；当它被判定为真时，不仅表明条件满足而且前面的 if 子句的条件不满足。

all) 函数和 any(函数可以应用到一个链表(或其他序列),以检查是否全部或任一 项目满足一些条件。

>>>sent=['No','good','fish','goes','anywhere','without','a',

'porpoise','.']

>>>all(len(w)>4 for w in sent)

False

>>>any(len(w)>4 for w in sent)

True



**4.2** **序列**

到目前为止，我们已经学习到了两种序列对象：字符串和链表。还有另一种序列称 **为元组**。元组由逗号操作符“构造，而且通常用括号括起来。实际上，我们已经在 前面的章节中介绍过它们，有时也被称为“配对”,因为总是有两个成分。然而， 元组的成分可以是任意的。与链表和字符串一样，元组可以被索引°和切片，并有 长度°。

>>>t='walk','fem',3①

>>>t



'walk'

>>>t[1:]③



**注意!**

元组使用逗号操作符来构造。括号是Python 语法的一般功能，是用 于分组的。可以通过添加一个尾随的逗号，像这样： 'snark', 来定义 一个包含单个元素'snark'的元组。空元组是在特殊的情况下，使用空 括号()定义。

直接比较字符串、链表和元组，在各个类型上进行索引、切片和长度操作。

 ='I=n'r'','spectroroute']

>>>raw[2],text[3],pair[1]

('t','the','turned')

>>>raw[-3:],text[-3:],pair[-3:]

>>>len(raw),len(text),len(pair)

('ute',['off','the','spectroroute'],(6,'turned'))

(29,5,2)

请注意在此代码示例中，在一行代码中计算多个值，中间用逗号分隔。这些用逗号 分隔的表达式其实就是元组——如果没有歧义，Python 允许忽略元组周围的括号。 当输出一个元组时，括号始终显示。以这种方式使用元组，我们将这些项目暗中聚

集在一起。



**轮到你来：**

定义一个集合，如：使用 set(text),当把它转换成一个链表或遍历其 成员时，看看会发生什么。

**序列类型上的操作**

我们可以用各种有用的方式遍历序列s 中的项目，如表4-1所示。

**表4-1** **遍历序列的各种方式**

|  |  |
| --- | --- |
| **Python表达式** | **评** **论** |
| for item in s | 遍历s中的元素 |
| for item in sorted(s) | 按顺序遍历s中的元素 |
| for item in set(s) | 遍历s中无重复的元素 |
| for item in reversed(s) | 按逆序遍历s中的元素 |
| for item in set(s).difference(t) | 遍历在集合s中且不在集合t的元素 |
| for item in random.shuffle(s) | 按随机顺序遍历s中的元素 |

表格4-1所示的序列功能可以多种方式相互结合，例如：要获得无重复的逆序排列 的 s 的元素，可以使用reversed(sorted(set(s)))。

可以在这些序列类型之间相互转换。例如：tuple(s)将任何种类的序列转换成元组， list(s)将任何种类的序列转换成链表。还可以使用join(函数将字符串链表转换成单 独的字符串，例如：'?join(words)。

其他一些对象，如 FreqDist, 也可以转换成序列(使用list()) 且支持迭代。

>>>raw ='Red lorry,yellow lorry,red lorry,yellow lorry.'

>>>text =nltk.word\_tokenize(raw)

>>>fdist =nltk.FreqDist(text)

>>>list(fdist)

>>>for

['lorry',',',

·



432111

key in fdist:

'yellow','.',

print fdist[key],

'Red', 'red']

在接下来的例子中，使用元组重新安排链表中的内容。(可以省略括号，因为逗号

**比赋值的优先级更高。)**

>>>words =['I','turned','off','the','spectroroute'] >>>words[2],words[3],words[4]=words[3],words[4],words[2]

>>>words

['I','turned','the','spectroroute','off']

这是移动链表内部项目的一种常用且可读的方式。在上述任务中它相当于下面的传 统方式：不使用元组(注意这种方法需要一个临时变量tmp)。

>>>tmp =words[2]

>>>words[2]=words[3]

>>>words[3] =words[4]

>>>words[4]=tmp

正如我们所看到的， Python 包含序列处理函数，如 sorted()和 reversed(), 它们可 以重新排列序列中的项目。还包含修改序列结构的函数以很方便地处理语言。 因此， zip) 取出两个或两个以上的序列中的项目，将它们“压缩”打包成单个 的配对链表。给定一个序列 s,enumerate(s) 返回一个包含索引及索引处所在项 目的配对。

>>>words =['I','turned','off','the','spectroroute']

>>>tags =['noun','verb','prep','det','noun']

>>>zip(words,tags)

('the','det'),('spectroroute','noun')]

[('I','noun'),('turned','verb'),('off','prep'),

>>>list(enumerate(words))

[(0,'I'),(1,'turned'),(2,'off'),(3,'the'),(4,'spectroroute')]

对于一些 NLP 的任务，有必要将一个序列分割成两个或两个以上的部分。例如： 我们可以用90%的数据来“训练”一个系统，并用剩余10%的数据进行测试。要 做到这一点，我们需要确定分割数据的位置°,然后在这个位置分割序列②。

=ir=nanltkatg,\*est.(sxt)ta①t=.dwtexaotradcut],text[cut:]②



9

可以验证在此过程中的原始数据没有丢失，也没有复制③。也可以验证两块大小的

比例是我们预期的④。

合并不同类型的序列

综 合 3 种类型的序列及链表推导式，将一个字符串中的词按它们的长度进行 排序。

>>>words ='I turned off the spectroroute'.split()①

>>>wordlens =[(len(word),word)for word in words]②

>>>wordlens.sort()③

>>>''.join(w for(\_,w)in wordlens)④

'I off the turned spectroroute'

上述代码段中每一行都分别包含一个显著的特征。 一个简单的字符串实际上是 一个已定义了方法的对象，如split)°。使用链表推导式建立一个元组的链表°, 其中每个元组由一个数字(词长)和这个词组成，例如：(3,'the') 。使用 sort) 方法°就地排序链表。最后，丢弃长度信息，并将这些词连接回一个字符串④。 (下划线只是一个普通的 Python 变量，按照惯例，下划线表示不会使用到其值 的变量。)

上面的代码说明了这些序列类型的重要区别，现在开始谈论这些序列类型的共性。 首先，字符串出现在开头和结尾，这是很典型的，程序先读取文本，再显示输出。 链表和元组在中间，但使用的目的不同。链表是典型的具有相同类型对象的序列， 它的长度是任意的。我们经常使用链表保存词序列。相反，元组通常是不同类型的 对象的集合，长度固定。我们经常使用元组来保存记录：一些与实体相关的不同字 段的集合。你需要一些时间来习惯使用链表与元组之间的区别，所以列举了下面的 例子。

>>>lexicon =[



.. ]

('the','det',['Di:','D@']),

('off','prep',['Qf','o:f'])

在这里，用一个链表表示词典，因为它是一个单一类型对象的集合——词汇条 目——没有预定的长度。个别条目被表示为元组，因为它是一个具有不同解释的对 象的集合，例如：正确的拼写形式、词性、发音(以SAMPA 计算机可读的音标表 示，见<http://www.phon.ucl.ac.uk/home/sampa/>)。请注意，这些发音都是用链表存储 的。(为什么呢?)



决定何时使用元组还是链表的一个好办法是看一个项目的内容是否取决 与它的位置。例如： 一个已标注的词标识符由两个具有不同解释的字符 串组成，将第一项解释为词标识符，第二项为标注。因此，我们使用这 样的元组：(grail,noun); 一个形式为(noun’,'grail) 的元组将是无意义的， 因为这将使词 noun 被标注为grail。相反，文本中的元素都是标识符，位 置并不重要。因此，我们使用这样的链表： [venetian’,blind] 。一个形式 为[blind',v enetian']的链表也同样有效。词的语言学意义可能会有所不 同，但作为标识符的链表项的解释是不变的。

链表和元组之间在使用上的区别已经讲过了。然而，还有一个更加基本的区别：在 Python 中，列表是可变的，而元组是不可变的。换句话说，列表可以被修改，而元 组不可以。下面是修改链表的操作。

')turned',

>>>del lexicon[0]

'VBD', ['t3:nd','t3'nd'])

**轮到你来：**

使用 lexicon=tuple(lexicon) 将词典转换为元组。然后尝试上述操作， 确认它们都不能运用在元组上。

**产生器表达式**

大多数时候都在使用列表推导式，因为用它处理文本结构紧凑且可读性好。下面的 例子是：分词和规范一个文本。

>>>text ='''"When I use a word,"Humpty Dumpty said in rather a scornful tone, ..."it means just what I choose it to mean -neither more nor less."' >>>[w.lower()for w in nltk.word\_tokenize(text)]

['"','when','i','use','a','word',',','"','humpty','dumpty',

'said',...]

假设现在要进一步处理这些词。可以将上面的表达式插入到一些其他函数的调用中°, 并省略方括号②。

>>>max([w.lower()for w in nltk.word\_tokenize(text)])①

>>>max(w.lower()for w in nltk.word\_tokenize(text))②

'word!

'word'

第二行使用了**产生器表达式，**这不仅仅是因为标记方便：在许多语言处理的案例中， 产生器表达式会更高效。在①中，链表对象的存储空间必须在 max(的值被计算之 前分配。如果文本非常大，计算速度会很慢。在②中，数据流向调用它的函数。由 于调用的函数只是要找到最大值——按字典顺序排在最后的词——它可以处理数 据流，而无需存储任何超过迄今为止的最大值。

**4.3** **风格的问题**

编程是一门艺术般的科学。无可争议的程序设计的“圣经”是由 Donald Knuth 编 写的2500多页的《计算机程序设计艺术》。目前已有许多关于编程艺术的书籍，它 们认为人类，不只是电脑，必须阅读和理解程序。在这里，我们挑选了一些编程风 格的问题，对代码的可读性，包括代码布局、程序与声明的风格、使用循环变量等 都有重要的影响。

**Python** **代码风格**

编写程序时，需要进行许多微妙的选择：名称、间距、注释等。当你在看别人编写 的代码时，风格上不必要的差异使人难以理解。因此，Python 语言的设计者出版了 Python 代码风格指南： <http://www.python.org/dev/peps/pep-0008/>。风格指南中提出 的基本值是一致的，目的是最大限度地提高代码的可读性。我们在这里简要回顾一 下书中一些主要建议，并请读者阅读完整的指南，里面有对实例的详细讨论。

代码布局中每个缩进级别应使用4个空格。要确保在文件中编写 Python 代码时， 避免使用 tab 缩进，因为很可能由于不同文本编辑器的不同解释而产生混乱。每行 应少于80个字符长，如果必要的话，可以在圆括号、方括号或花括号内换行，这 样 Python 就能够探测到该行与下一行是连续的了，如下面的例子所示。

>>>cv\_word\_pairs =[(cv,w)for w in rotokas\_words

for cv in re.findall('[ptksvr][aeiou]',w)]

>>>cfd =nltk.ConditionalFreqDist(

… (genre,word)

· for genre in brown.categories()

 for word in brown.words(categories=genre))

>>>ha\_words



=['aaahhhh','ah','ahah','ahahah','ahh','ahhahahaha', 'ahhh','ahhhh','ahhhhhh','ahhhhhhhhhhhhhh','ha',

'haaa','hah','haha','hahaaa','hahah','hahaha']



**如果需要在圆括号、方括号或大括号外换行，通常可以添加额外的括号，也可以在** **行尾需要换行的地方添加一个反斜杠。**

>>>if(b(lsy[2les)>4 and u(]nydllables[2])==la[3]==syllables[1][3]):

. · process(syllables)

>>>if lseynll**l**l[e2s])[>2in iosdyl**l**[l2e]3]yll\ables[1][3]:

process(syllables)

**键入空格来代替制表符很快就会成为一件苦差事。许多程序编辑器对**

Python提供内在支持，能够自动缩进代码，加亮任何语法错误(包括 **缩进错误)。了解** **Python 编辑器列表，请见** **http:/wiki.python.org/moin**/ **PythonEditors。**

**过程风格与声明风格**

我们刚才已经学习了可以用不同的方式执行相同的任务，其中蕴含着对执行效率的 影响。另一个影响程序开发的因素是编程风格。思考下面的程序，计算布朗语料库 中词的平均长度。

>>>tokens =nltk.corpus.brown.words(categories='news')

l 

>>>for token in tokens:

. · anltln(token)

>>>print total /count

4.27 65382469

在这段程序中，使用变量count 跟踪遇到的标识符的数量，而 total 储存所有词的长 度的总和。这是一个低级别的风格，与机器代码(即计算机的 CPU 所执行的基本 操作)相差不远。两个变量就像CPU 的两个寄存器，积累许多中间环节产生的值， 以及直到最后也毫无意义的值。我们能看到，这段程序是以过程风格编写的， 一步 一步教机器如何运行。现在，考虑下面的程序来计算同样的对象。

>>>total =sum(len(t)for t in tokens)

>>>print total /len(tokens)

4.27 65382469

第一行使用生成器表达式累加标示符的长度，第二行像前面一样计算平均值。每行

代码执行一个完整的、有意义的工作，能以高级别的属性方式来理解，如： “total 是标识符长度的总和”。实施细节交给 Python 解释器处理。第二段程序使用内置函 数，在一个更抽象的层面构成程序，生成的代码可读性更好。 一起看下面的极端 的例子。

>>>word\_list =[]

>>>len\_word\_list =len(word\_list)

>>>i=0

>>>while i<len(tokens):



··

·

·

j=0

while j<len\_word\_list and word\_list[j]<tokens[i]:

j+=1

if j==0 or tokens[i]!=word\_list[j]:

word\_list.insert(j,tokens[i])

len\_word\_list +=1

i+=1

等效的声明版本使用熟悉的内置函数，可以立即了解代码的目的。

>>>word\_list =sorted(set(tokens))

另一种情况，如果要每行输出一个计数值，那么循环计数器是必不可少的。然而， 我们可以使用 enumerate()处理序列s, 为 s 中每个项目产生一个(i,s[i]) 形式的元组， 以(C,s[0]) 开始。下面枚举频率分布的值，捕获变量rank 和 word 中的整数-字符串 对。按照产生排序项列表时的需要，输出 rank+1, 使计数从1开始。

>>>fd =nltk.FreqDist(nltk.corpus.brown.words())

>>>cumulative =0.0

>>>for rank, word in enumerate(fd):

cumulative +=fd[word]\*100 /fd.N()

· print "83d 86.2f888s"8(rank+1,cumulative,word)

· if cumulative >25:

. . break

1 5.40%the

2 10.42%,

314.67%.

4 17.78%of

5 20.198 and

6 22.40%to

7 24.29% a

8 25.97%in

到目前为止，使用循环变量存储最大值或最小值是很有用的方法。让我们用这种方

**法找出文本中最长的词。**

gest ='' =nltk.corpus.gutenberg.words('milton-paradise.txt')

>>>for word in text:

if len(word)>len(longest):

· longest =word

>>>longest

'unextinguishable'

然而，有一个更加有效的解决方案是使用两个链表推导式。

>>>maxlen =max(len(word)for word in text)

>>> [word for word in text if len(word)=-maxlen]

['unextinguishable','transubstantiate','inextinguishable',

'incomprehensible']

请注意，第一个解决方案找到的是第一个长度最长的词，而第二种方案找到的是所 有最长的词(通常这才是我们想要的)。虽然这两个解决方案之间存在理论效率的 差异，主要是在内存中读取数据； 一旦数据准备好，第二阶段处理数据可以瞬间高 效地完成。我们还需要平衡程序与程序员之间的效率关系。快速但含义模糊的解决 方案将会更难以理解和维护。

**计数器的一些合理用途**

在某些情况下，仍然要在链表推导式中使用循环变量。例如：需要使用循环变量来 提取链表中的连续重叠的n-grams。

>>>sent =['The','dog','gave','John','the','newspaper']

>>>n =3

>>>[sent[i:i+n]for i in range(len(sent)-n+1)]

[['The','dog','gave'],

['dog','gave','John'],

['gave','John','the'],

['John', 'the', 'newspaper']]

确保循环变量范围的正确是相当困难的。因为这是 NLP 中的常见操作， NLTK 提 供了支持函数 bigrams(text) 、trigrams(text)和ngrams(text,n)。

下面例子是如何使用循环变量构建多维结构。例如：建立一个m 行n 列的数组，其

**中每个元素是一个集合，我们可以使用嵌套的链表推导式。**

>>>m,n =3,7

>>>array =[[set()for i in range(n)] for j in range(m)] >>>array[2][5].add('Alice')

>>>pprint.pprint(array)

[[set([]),set([]),set([]),set([]),set([]),set([]),set([])],

[set([]),set([]),set([]),set([]),set([]),set([]),set([])],

[set([]),set([]),set([]),set([]),set([]),set(['Alice']),set([])]]

观察发现循环变量i 和j 在产生对象过程中没有用到；它们只是为了确保 for 语句 语法正确。这种用法的另一个例子是，表达式[very'for i in range(3)]产生一个包含 3个very'实例的链表，但没有整数。

请注意，这种做法不适用于乘法，原因与我们之前所讨论的对象复制有关。

>>>array =[[set()]\*n]\*m

>>>array[2][5].add(7)

>>>pprint.pprint(array)

[[set([7]),set([7]),set([7]),set([7]),set([7]),set([7]),set([7])],

[set([7]),set([7]),set([7]),set([7]),set([7]),set([7]),set([7])],

[set([7]),set([7]),set([7]),set([7]),set([7]),set([7]),set([7])]]

迭代是一个重要的编程概念。采取其他语言中的习惯用法是很有用的。然而，正如 我们已经看到的，Python提供一些优雅和可读性高的程序。

**4.4** **函数：结构化编程的基础**

函数提供了程序代码打包和重用的有效方法，在2.3节中介绍过。例如：假设要从 HTML文件中读取文本。包括以下几个步骤：打开文件，将它读入，规范化空白符 号，剥离 HTML标记。我们可以将这些步骤集中到一个函数，并给它取一个名字， 如get\_text(), 如例4.2所示。

例4-1 从文件中读取文本

import re

def get\_text(file):

"""Read text from a file,normalizing whitespace and stripping HTML markup."""

text =open(file).read()

t =re.subt'ext)

return text

现在，任何时候要从 HTML 文件得到某文本，只要用文件的名字作为唯一的参数 调用 get\_text()即可。返回字符串，并指定变量，例如：contents=get\_text("test.html")。 再次进行该操作时，只需要调用这个函数即可。

使用函数可以为程序节约空间。更重要的是：为函数选择名称可以提高程序可读性。 在上面的例子中，任何时候需要从文件读取某文本，都不必弄乱这4 行代码，只 需要调用 get\_text()即可。这种命名方式能提供一些“语义解释”——可以帮助读 者理解程序的“意思”。

请注意，上例中的函数定义包含字符串。函数定义内的第一个字符串被称为 **docstring。**它不仅帮助阅读代码的人记录函数，而且还向从文件加载这段代码的程 序员提供更便捷的访问方式。

>>>help(get\_text)

Help on function get\_text:

get\_text(file)

Read text from a file,normalizing whitespace

and stripping HTML markup.

我们已知，函数有助于提高工作的可重用性和可读性。它还益于提高程序的可 靠性。当重用已开发和测试过的代码时，可以更放心，因为它已经正确处理过 各种案例。还可以避免忘记一些重要步骤，或引入 bug 的风险。调用函数程序 也提高了可靠性。该程序的作者可以处理一个较短的程序，其组成成分也更透 明地运行。

简而言之，顾名思义，函数可以捕捉功能。它是一个代码段并提有一个有意义的名 字，同时执行明确的任务。函数使我们可以不去考虑细节，而去考虑一个更大的方 面，如何高效地编程。

本节的其余部分将仔细地分析函数，探索它的机制，讨论如何使程序更容易阅读的 方法。

**函数的输入和输出**

使用函数的参数传递信息给函数，这些参数是用括号括起的变量和常量的列表，在 函数定义中跟在函数名称之后。下面是一个完整的例子。

>>>def repeat(msg,num): ①

return ''.join([msg]\*num)

>>>monty ='Monty Python'

>>>repeat(monty,3)②

'Monty Python Monty Python Monty Python'

首先定义函数的两个参数： msg 和 num°。然后调用函数，并传递给它两个参数： monty 和32;这些参数填补了参数提供的“占位符”,给函数体中出现的 msg 和 num 赋值。

在下面的例子中不需要有任何参数。

>>>def monty():

return "Monty Python"

>>>monty()

'Monty Python'

函数通常会通过return 语句将其结果返回给调用它的程序。对于调用程序，看上去 函数调用好像已被函数结果所替代。

>>>repeat(monty(),3)

'Monty Python Monty Python Monty Python'

>>>repeat('Monty Python',3)

'Monty Python Monty Python Monty Python'

Python 函数并不是一定需要有 return 语句。有些函数在运行的同时会附带输出 结果、修改文件或者更新参数的内容。(这种函数在其他一些编程语言中被称为 “过程”)。

考虑以下3 个排序功能。第三个是危险的，因为程序员在调用它时可能意识不到它 已经修改了它的输入。 一般情况下，函数应该修改参数的内容 (my\_sort1)) 或返 回一个值 (my\_sort2)), 而不是两个都做 (my\_sort30)。

>>>def my\_sort1(mylist):#good:modifies its argument,no return value

mylist.sort()

>>>def my\_sort2(mylist):#good:doesn't touch its argument,returns value

>>>def

...



return sorted(mylist)

my\_sort3(mylist):#bad:modifies mylist.sort()

return mylist

its argument and also returns it

**参数传递**

在4.1 节中，已经介绍过赋值操作。 一个结构化对象的值是该对象的引用，函数也是 一 样的。Python 把函数的参数解释成其值(这被称为**按值传递)**。在下面的代码中， set\_upO 有两个参数，都在函数内部被修改。将空字符串分配给 w, 将空链表分配 给 p 。调用该函数后，w 没有变，而p 却改变了。

>>>def set\_up(word,properties):

· word ='lolcat'

..  .ap=p5end('noun')

>>> w=

>>>p=[

>>>set\_up(w,p)

>>>w

11

>>>F

['noun']

请注意， w 没有被函数改变。当调用 set\_up(w,p) 时， w (空字符串)的值被分配到 一个新的变量 word。在函数内部word 值被修改。然而，这种变化并没有传递给 w。 这个参数传递过程与下面的赋值序列是一样的。

>>> W = t

d =w='lolcat'

>>> W

让我们来看看链表p 上发生了什么。当调用 set\_up(w,p) 时 ，p 的值( 一 个空列 表的引用)被分配到一个新的局部变量 properties, 所以现在这两个变量都引用 到相同的内存位置上。函数修改了 properties,正如我们所看到的，这种变化也 反映在p 值上。函数也分配给 properties 一个新的值(数字5),但这并不能修改 该内存位置上的内容，而是创建了一个新的局部变量。这种做法同下列赋值序 列相似。



prorties =p

aend['noun']

>>>p

['noun']

因此，要理解 Python 按值传递参数，只要了解它是如何赋值的就足够了。记住， 可以使用idO函数和 is 操作符来检验每个语句执行之后对对象标识符的理解。

**变量的作用域**

函数定义为变量创建了一个新的局部的范围。当在函数体内部分配一个新的变量 时，这个名字只在该函数内部被定义。函数体外或者在其他函数体内，该名称是不 可见的。这意味着你可以放心地选择变量名而不必担心它与其他函数定义中使用的 名称冲突。

当在一个函数体内部使用一个现有的名字时， Python 解释器先尝试按照函数本地的 名字来解释。如果没有发现，解释器将检查它是否是一个模块内的全局名称。如果 还没有成功，最后，解释器会检查它是否是 Python 内置的名字。这就是所谓的名 称解析的**LGB 规则：** 本 地 (local), 全局 (global), 然后内置 (built-in)。

**注意!**

函数可以使用 global 声明创建一个新的全局变量。然而，这种做法 应尽可能避免。在函数内部定义全局变量会导致上下文具有依赖性 而且限制函数的可移植性(或重用性)。 一般来说，应该使用参数作 为函数的输入，返回值作为函数的输出。

**参数类型检查**

编写程序时， Python 不会强迫我们声明变量的类型，这使得我们可以定义参数类型 灵活的函数。例如：希望一个标注只是一个词序列，而不管这个序列被表示为链表、 元组或是迭代器(一种新的序列类型，我们将在下面讨论)。

然而，我们常常想编写一些能被他人利用的程序，并希望以一种保守的风格，当函 数没有被正确调用时提供有益的警告。下面的 tag(函数假设其参数将始终是一个字 符串。

>>>def tag(word):

·

if word return

else:

return

in

'det'

'noun'

['a','the','all']:

·

>>>tag('the')

'det'

>>>tag('knight')

'noun'

>>>tag(["'Tis",'but','a','scratch']) ①

'noun'

该函数对参数'the'和'knight'返回合理的值，传递给它一个链表°,看看会发生什 么——没有反应，虽然它返回的结果显然是不正确的。可以进一步采取一些步骤来 确保 tag(O函数的参数 word 是一个字符串。 一种简单的做法是使用 if not type(word) is str 检查参数的类型，如果 word 不是一个字符串，简单地返回Python特殊的空值： None。这是一个小小的改善，因为该函数能够检查参数类型，并试图对错误的输入 返回一个“特殊的”诊断结果。然而，这也是危险的，因为调用程序可能不会检测 None 是故意设定的“特殊”值，这种诊断的返回值可能被传递到程序的其他部分从 而产生不可预测的后果。如果这个词是一个 Unicode 字符串，那么这种方法也会失 败。因为它的类型是unicode 而不是str。这里有一个更好的解决方案，和Python 的 basestring 的类型一起使用 assert 语句，就能生成 unicode 和str类型。

>>>def tag(word):

assert isinstance(word,basestring),"argument to tag()must be

a string"

\*

·

if word

return else:

in ['a','the','all']:

'det'

return 'noun'

如果 assert 语句失败，将产生一个不可忽视的错误而停止程序执行。此外，该错误信 息是容易理解的。在程序中添加断点能帮助你找到逻辑错误，是一种**防御性编程。** 一 个更基础的方法是使用 docstrings 为每个函数记录参数，正如本节后面描述的那样。

**功能分解**

结构良好的程序通常都广泛使用函数。当一个程序代码块增长到超过10～20行， 如果将代码分成一个或多个函数，并且每一个都有明确的目的，这将对提升可读性

有很大的帮助。这类似于好文章被划分成段，每一段话表示一个主要思想。

函数提供了一种重要的抽象概念，将多个动作组合成一个单一的复杂的动作，并关 联一个名称。(与为单一的复杂的动作 fetch 组合动作 go 和bring back作比较。)当 使用函数时，可以用更高的抽象水平编写主程序，使其结构更透明，如下所示。

>>>data =load\_corpus()

>>>results =analyze(data)

>>>present(results)

适当使用函数能提高程序的可读性和可维护性。另外，重新实现一个函数使使用更 高效的代码替换函数体成为一种可能，而不需要关心程序的其余部分。

思考例4-2中freq\_words 函数。对作为参数传递输入的频率分布进行更新，并输出 前n 个最频繁的词的链表。

例4-2 用来计算高频词的拙劣函数

def freq\_words(url,freqdist,n):

text =nltk.clean\_url(url)

for word in nltk.word\_tokenize(text):

freqdist.inc(word.lower())

>>>dist=./)/w.archives.gov/national-archives-experience"\

>>

>>>freq\_words(constitution,fd,20)

['the', 'of', 'charters', 'bill','constitution', 'declaration','impact','freedom','-','making',

'rights',',',

'independence']

这个函数存在几个问题。该函数有两个副作用：修改了第二个参数的内容，而且输 出了已计算过的结果。如果在函数内部初始化 FreqDist()对象(在它被填入的同一个 地方),并且去掉选择集而将结果显示给调用程序的话，函数会更容易理解且重用性 更高。在例4-3中，**重构此**函数，并仅提供一个url参数来简化其接口。

**例4-3** **用来计算高频词的函数**

def freq\_words(url):

freqdist =nltk.FreqDist()

text =nltk.clean\_url(url)

for word in nltk.word\_tokenize(text):

freqdist.inc(word.lower())

return freqdist

>>>fd =freq\_words(constitution)

>>>print fd.keys()[:20]

['the','of','charters','bill','constitution','rights',',',

'declaration','impact','freedom','-','making','independence']

请注意，目前已经将 freq\_words 的工作简化，利用3行代码即可完成。

>>>words =nltk.word\_tokenize(nltk.clean\_url(constitution))

>>>fd=nltk.FreqDist(word.lower()for word in words)

>>>fd.keys()[:20]

['the','of','charters','bill','constitution','rights',',',

'declaration','impact','freedom','-','making','independence']

**文档说明函数**

如果已经熟练掌握如何把程序分解成函数，那么就很容易地使用通俗易懂的语言来 描述每个函数的目的，并且在函数定义的顶部 docstring 中提供这些描述。这条语 句不是解释函数是如何实现的。实际上，不需要改变这条语句的前提下，可使用不 同的方法来重新实现这个函数。

对于最简单的函数， 一个单行的docstring 通常就足够了(见例4-1)。但需要提供一个在 一行中包含一个完整句子的三重引号字符串。对于重要的函数，需要在第一行提供一句 话总结，因为很多的 docstring 处理工具会索引这个字符串。它后面应该有一个空行，然 后是更详细的功能说明(见 <http://www.python.org/dev/peps/pep-0257/>可获得更多 docstring 约定的相关信息)。

docstring 中可以包括 **doctest 块，**用来说明函数的用法和预期的输出。这些都可以使用 Python的 docutils 模块自动检测。docstring 中应当记录函数的每个参数的类型和返回 类型。至少可以用纯文本来做这些。请注意，NLTK 中使用 “epytext” 标记语言来记 录参数。这种格式可以自动转换成结构化丰富的API 文档(见<http://www.nltk.org/>), 并包含某些“字段”的特殊处理，例如：@param 允许清楚地记录函数的输入和输 出。例4-4演示了一个完整的 docstring。

例4-4 演示一个完整的 docstring, 包括总结、详细的解释、doctest 例子及特定参 数、类型、返回值类型和异常的 epytext 标记。

{0<i<=len(test)}such that C{test[i]==reference[i]}.

def accuracy(reference,test):

n n作

Calculate the fraction of test items that equal the corresponding reference

items.

Given a list of reference values and a corresponding list of test values, return the fraction of corresponding values that are equal.

In particular,return the fraction of indexes

>>>accuracy(['ADJ','N', 'V','N'],['N', 'N','V','ADJ']) 0.5

@param reference:An ordered list of reference values.

@type reference:C{list}

aparam test:A list of values to compare against the corresponding

@type test:C{list}

reference values.

**@**yi:CV{aftrror:If C{reference}and C{length}do not have the

same length.

n n R

if len(reference) !=len(test):

num\_reaError("Lists must have the same length.")

for ifxn=u=miyn\_:co(rence,test):

return float(num\_correct)/len(reference)

**4.5** **更多关于函数**

本节将介绍更高级的特性。

**作为参数的函数**

到目前为止，我们传递给函数的参数一直都是简单的对象，如字符串或链表等结构 化对象。Python 也允许将一个函数作为另一个函数的参数。现在，我们可以抽象此 操作，对相同数据进行不同操作。正如下面的例子，我们可以将内置函数len()或用 户定义的函数 last\_letter()作为另一个函数的参数。

>>>sent

>>>def

=['Take','care','of','the','sense',',','and','the',

'sounds','will','take','care','of','themselves','.']

extract\_property(prop):

return [prop(word)for word in sent]

>>>extract\_property(len)

>[4>,,d2e,f3s,t1,l,tt3e,ro,d,:,2,10,1]

return word[-1]

'>e>'x'p't'(\_'dtte''e','s','l','e','e','f','s','.']

对象 len 和 last\_letter 可以像链表和字典那样被传递。请注意，只有在调用该函数 时，才在函数名后使用括号；当只是将函数作为一个对象时，括号被省略。

Python 提供了多种方式来定义函数使其可作为其他函数的参数，即所谓的 **lambda 表达式。**在很多地方没有必要使用上述的last\_letter()函数，因此没有必要给它一个 名字。因此可以等价地这样写：

['e','e','f','e','e',',','d','e','s','l','e','e','f','s','.']

>>>extract\_property(lambda w:w[-1])

下一个例子将演示传递一个函数给 sortedO函数。当用唯一的参数(需要排序的链 表)调用后者时，它使用的是内置的比较函数 cmp(。 然而，可以提供自己的排序 函数，例如：按长度递减排序。

>>>sorted(sent)

[',','.','Take','and','care','care','of','of','sense','sounds',

'take', 'the', 'the', 'themselves', 'will']

>>>sorted(sent,cmp)

[',','.','Take','and','care','care','of','of','sense','sounds',

'take','the','the','themselves','will']

>>>sorted(sent,lambda x,y:cmp(len(y),len(x)))

['themselves','sounds','sense','Take','care','will','take','care',

'the','and','the','of','of',',','.']

**累计函数**

这些函数从初始化存储数据开始，迭代和处理输入的数据，最后返回最终的对象(一 个大的结构或汇总的结果)。标准的方式是先初始化空链表，累计材料，然后返回 这个链表，如例4-5所示的函数 search10。

例4-5 累计输出到一个链表

def sl(sstring,words):

for word in words:

if substring in word:



result.append(word)

return result

def search2(substring,words):

for word in words:

yield word

if substring in word:

print "search1:n

for item in searchl('zz',nltk.corpus.brown.words()):

print item

print "search2:"

for item in search2('zz',nltk.corpus.brown.words()):

print item

函数 search2()是一个生成器。第一次调用此函数时，它会运行到 yield 语句然后停 下来。调用程序获得第一个词，并进行任何必要的处理。 一旦调用程序对另一个词 做好准备，函数会从之前停下来的地方继续执行，直到再次遇到 yield 语句。这种 方法通常更有效，因为函数只产生调用程序需要的数据，并不需要分配额外的内存 来存储输出(参见前面关于生成器表达式的讨论)。

下面是一个更复杂的生成器例子，用以产生一个词链表的所有排列。为了强制 permutations()函数产生所有它的输出，将它包装在list)调用中。

permutations(seq):

>>>def

·

if len(seq)<=1:

yield seq

· else:

· for perm in permutations(seq[1:]):

for i in range(len(perm)+1):

· yield perm[:i]+seq[0:1]+perm[i:]

>>

[['police','fish','buffalo'], ['fish','police','buffalo'], ['fish','buffalo','police'],['police','buffalo','fish'],

['buffalo','police','fish'],['buffalo','fish','police']]

permutations 函数使用了一种技术叫递归，将在4.7节讨论。产生一组 词的排列对于创建数据以测试其中的语法(见第8章)十分有用。

**高阶函数**

Python 提供了一些具有函数式编程语言标准特征的高阶函数，如： Haskell。我们

将与使用链表推导式相对应的表达式一起，在这里演示它们。

从定义一个函数 is\_content\_word()开始，检查一个词是否来自一个开放的实词 类。将此函数作为 filter()的第一个参数，此参数把这个函数应用到每个项目中， 这些项目都包含在函数第二个参数的序列中，而且只在该函数返回 True 时保持 不变。

>>>def is\_content\_word(word):

return word.lower()not in ['a','of','the','and','will',',','.']

>>>sent =['Take','care','of','the','sense', ',','and','the',

'sounds','will','take','care','of','themselves','.']

>>>filter(is\_content\_word,sent)

['Take','care','sense','sounds','take','care','themselves']

>>>[w for w in sent if is\_content\_word(w)]

['Take','care','sense','sounds','take','care','themselves']

另一个高阶函数是 mapO, 将一个函数运用到一个序列中的每一项。它是我们在本节 前面看到的函数 extract propertyO的一个通用版本。下面是找出布朗语料库新闻部分中 句子平均长度的一个简单方法，后面紧跟着的是使用链表推导式计算的等效版本。

>>>lengths =map(len,nltk.corpus.brown.sents(categories='news')) >>>sum(lengths)/len(lengths)

21.75 08111616

>>>lengths =[len(w)for w in nltk.corpus.brown.sents(categories='news')]]

>>>sum(lengths)/len(lengths)

21.75 08111616

在上面的例子中，我们指定了一个用户定义的函数is\_content\_word) 和一个内置函 数len()。还可以提供一个 lambda 表达式。下面是两个等效的例子，计数每个词中 的元音的数量。

>>>map(lambda w:len(filter(lambda c:c.lower()in "aeiou",w)),sent)

[2,2,1,1,2,0,1,1,2,1,2,2,1,3,0]

>>>[len([c for c in w if c.lower()in "aeiou"])for w in sent] [2,2,1,1,2,0,1,1,2,1,2,2,1,3,0]

以链表推导为基础的解决方案通常比基于高阶函数的解决方案可读性更好，本书更 青睐于使用前者。

**参数的命名**

当有很多参数时，很容易混淆顺序。可以通过名字引用参数，甚至可以给它们分配

**默认值以供调用程序在没有提供该参数时使用。现在参数可以按任意顺序指定，也**

可以省略。

>>>def repeat(msg=¹<empty>',num=1):

return msg \*num

>>>repeat(num=3)

¹<empty><empty><empty>'

>>>repeat(msg='Alice')

'Alice'

>>>repeat(num=5,msg='Alice')

'AliceAliceAliceAliceAlice'

这些被称为**关键字参数。**如果混合使用这两种参数，就必须确保未命名的参数在命 名的参数前面。必须是这样，因为未命名参数是根据位置来定义的。还可以定义一 个函数，接受任意数量的未命名和命名参数，并通过一个就地的参数链表\*args 和 一个就地的关键字参数字典\*\*kwargs 来访问它们。

>>>def

..

generic(\*args,\*\*kwargs):

print args

print kwargs

>>>generic(1,"African (1, 'African swallow') {'monty': 'python'}

swallow",monty="python")

当\*args 作为函数参数时，它实际上对应函数所有的未命名参数。下面是 Python 语 法在这方面的另一个演示例子，考虑参数数目可变的函数 zipO是如何运行的。我们 将使用变量名\*song 来表示名字\*args 而无特别的含义。

>>>song =[['four','calling','birds'],

**[**'he,nh']l,e','doves']]

>>>zip(song[0],song[1],song[2])

[('four','three','two'),('calling','French','turtle'),('birds','hens', 'doves')]

>>>zip(\*song)

[('four','three','two'),('calling','French','turtle'),('birds','hens', 'doves')]

应明白这个例子中输入的\*song 仅仅是一个方便的记号，相当于输入 song[0]、

song[1] 、song[2].

下面是另一个在函数的定义中使用关键字参数的例子，有3种等效的方法来调用



这个函数。

>>def textfree),,)num=10):

·· · st =iet)in tokens if len(t)>=min)

return fregdist.keys()[:num]

>>>fw=freq\_words('ch01.rst',4,10)

>>>fw=freq\_words('ch01.rst',min=4,num=10)

>>>fw =freq\_words('ch01.rst',num=10,min=4)

已命名参数的另一个作用是允许选择性使用参数。因此，可以在要使用默认值的地方 省略任何参数：freq\_words(ch01.rst,min=4),freq\_words(ch01.rst,4)。可选参数的另一 个常见用途是作为标志使用。这里是同一个函数的修订版本，如果设置了 verbose 标志，则函数将会报告其进展情况。

>>>def freqwords(file,min=1,num=10,verbose=False):

\*





\*



\*



freqdist =FreqDist()

xt le).ring",file

if trace:print "Read in 8d characters"%len(file)

for ord w

einoqdist.N()%100 ==0:print"."

if trace:print

return freqdist.keys()[:num]

**注意!**

注意不要使用可变对象作为参数的默认值。这个函数的一系列调用 将使用同一个对象，有时会出现离奇的结果，就像我们稍后会在关

于调试的讨论中看到的那样。

**4.6** **程序开发**

编程是一种技能，是需要长期使用各种编程语言且做过多种项目后才能学到的技 能。关键的高层次能力是算法设计及其在结构化编程中的实现。关键的低层次的能 力包括熟悉语言的语法结构，以及排除程序故障(是指程序不能表现出预期的行为) 的各种诊断方法。

本节将描述程序模块的内部结构，以及如何组织多模块程序。然后描述程序

开发过程中出现的各种问题，如何解决这些问题，更好的是如何从一开始就 避免它们。

**Python 模块的结构**

程序模块的目的是把逻辑上相关的定义和函数结合在一起，以方便重用和抽象化。 Python 模块只是一些单独的.py文件。例如：如果在处理一种特定的语料格式，读 取和写入这种格式的函数可以放在一起。这两种格式所使用的常量，如字段分隔符 或EXTN =".inf"文件扩展名，都可以共享。如果要更新格式，只有一个文件需要 改变。同样的，模块可以包含用于创建和操纵某种特定数据结构的代码，如语法树； 或执行特定的处理任务，如绘制语料统计图表。

开始编写 Python 模块时，有一些例子来模拟是有益的。可以使用变 量 file 来定 位系统中任一NLTK 模块的代码。

>>>nltk.metrics.distance.\_file\_

'/usr/lib/python2.5/site-packages/nltk/metrics/distance.pyc'

模块将返回到已编译.pyc 文件的位置上，在你自己的机器上看到的位置可能不同。 需要打开对应的.py 源文件，它和.pyc 文件放在同一目录下。另外，还可以在网站 上查看该模块的最新版本： <http://code.google.com/p/nltk/source/browse/trunk/nltk/> nltk/metrics/distance.py,

与其他NLTK 的模块一样，distance.py 以一组注释行开始，包括模块标题和作者信息。 (由于代码会被发布，还将包括代码可用的URL、版权声明和许可信息。)接下来是模 块级的docstring,三重引号的多行字符串，当有人输入help(nltk.metrics.distance)时，所 输出的关于模块的信息也包括在内。

#Natural Language Toolkit:Distance Metrics

#

#Copyright(C) 2001-2009 NLTK Project

#Author:Edward Loper <edloperegradient.cis.upenn.edu>

# Steven Bird <sb@csse.unimelb.edu.au>

# Tom Lippincott <tom@cs.columbia.edu>

#URL:<<http://www.nltk.org/>>

#For license information,see LICENSE.TXT

#

1 H



Distance Metrics.

Compute the distance between two items(usually strings).

As metrics,they must satisfy the following three requirements:

1.d(a,a)=0

2.d(a,b) >=0

3.d(a,c)<=d(a,b)+d(b,c)

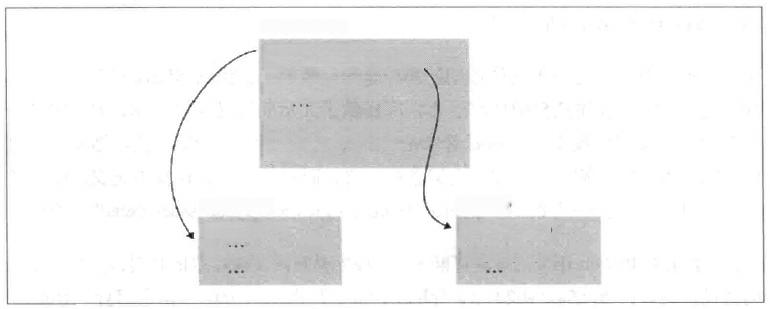
n ##

在这之后是模块所需要的所有导入语句，然后是所有全局变量，接着是组成模块主 要部分的一系列函数的定义。有些模块定义“类”,这是面向对象编程的主要部分， 这超出了这本书的范围。(大多数NLTK 的模块还包括 demo()函数，用来演示模块 的使用方法。)

模块的一些变量和函数仅用于模块内部。它们的名字应该以下划线开头， 如\_helper(),用于隐藏其名称。如果另一个模块使用习惯用法： from module import\* 来导入，这些名称将不会被导入。可以使用像这样的一 个特殊的内置变量； all =Tedit\_distance',jaccard\_distance], 来选择 性地列出一个模块的外部可访问列表。

**多模块程序**

一些程序汇集多种任务，例如：从语料库加载数据，对数据进行一些分析，然后将其可 视化。我们已经拥有了稳定的模块用来加载数据和实现数据可视化。下面的例子表示的 是编码分析任务，并从现有的模块中调用一些函数，如图4-2所示。



my\_program py

from inf\_corpus import load

from display import plot

def analyze(…):

data z load('hsi.inf')

result ·anslyze(data)

plot(result)

inf\_corpus py dsplay py

def load(…): SIZE ·640,480

def dump(…): def plot(…):

图4-2 一个多模块程序的结构：主程序 my\_program.py 从其他两个模块导入函数；重要的分 析任务在主程序本地进行，而一般的载入和可视化任务被分离开以便可以重用和抽象

通过将工作分成几个模块和使用 import 语句访问别处定义的函数，可以保持各个 模块的简单性，并易于维护。这种做法也将导致越来越多的模块集合，从而建立复 杂的涉及模块间层次结构的系统。设计这样的系统是一个复杂的软件工程任务，这 超出了本书的范围。

**误差源头**

当程序不按预期运作时，你若能信手拈来各种解决办法，这就说明了你已精通编程。 一些细节上的失误，如放错位置的符号，可能导致程序的行为异常。我们把这些叫 做“bugs”, 因为与它们所导致的损害相比起来，它们是很小的。它们不知不觉地 潜入我们的代码，只有在很久以后，当我们在一些新的数据上运行程序时才会发现 它们的存在。有时，修复了一个错误又会暴露出另一个，于是我们有一个鲜明的印 象 ：bug 在移动。我们唯一的安慰是 bugs 是自发的而不是程序员的错误。

实话说，调试代码是很难的，因为有那么多的方式会故障的出现。我们对输入数据、 算法甚至编程语言的理解可能是错误的。让我们分别来看看每种情况的例子。

首先，输入的数据可能包含一些意想不到的字符。例如： WordNet 的同义词集名称 的形式是 tree.n.01, 用逗号分割成3个独立的部分。最初NLTK的 WordNet 模块使 用split(.) 分解这些名称。然而，当有人试图寻找词 PhD 时，这种方法就不能用了， 它的同义集名称是 ph.d.n.01, 包含4个逗号而不是预期的2个。解决的办法是使用 rsplit(!,2) 利用最右边的逗号，最多分割两次，留下字符串ph.d.不变。虽然在模块发 布之前已经测试过，但就在几个星期前有人检测到这个问题(见 <http://code.google>. com/p/nltk/issues/detail?id=297)。

第二，提供的函数可能不会像预期的那样运作。例如：在测试NLTK 中的 WordNet 接口时， 一名作者注意到即使底层数据库提供了大量的反义词的信息，但是同义词 集都没有定义任何反义词。这看着像是 WordNet 接口中的一个错误，结果却是对 WordNet 本身的误解：反义词是在词条中定义，而不是在同义词集中定义。唯一的 “bug”是对接口的一个误解(见<http://code.google.com/p/nltk/issues/detail?id=98>)。

第三，我们对Python语义的理解可能出错。很容易对两个操作符的相对范围作出错误 的假设。例如："%s.%s.%02d”%"ph.d.","n",1 产生一个运行错误 TypeError:not enough arguments for format string。这是因为百分号操作符优先级高于逗号运算符。

>>>fnd\_ed'uolg','teh','lolcat','sitted','on','teh','mat'],3)①

解决办法是添加括号以强制限定所需的范围。看看另一个例子，假设我们定义一个 函数来收集一个文本中给定长度的所有标识符。该函数有文本和词长作为参数， 还有一个额外的参数，允许指定结果的初始值作为参数。

>>>def find\_words(text,wordlength,result=[]):

for word in text:

 if len(word)==wordlength:

.  result.append(word)

['omg','teh','teh','mat']

>>>find\_words(['omg','teh','lolcat','sitted','on','teh','mat'],2,

['ur'])②

>['ur>',>fi']\_words(['omg','teh','lolcat','sitted','on','teh','mat'],3)③

['omg','teh','teh','mat','omg','teh','teh','mat']

第一次调用find\_wordsO“得到所有预期的3个字母的词。第二次，我们为 result 指定 一个初始值： 一个单元素链表[ur], 如预期的一样，结果中存在这个词连，同时文本 中的其他双字母的词也出现在结果中。再次使用①中相同的参数调用 find\_wordsO°, 但却得到了不同的结果!每次在不使用第三个参数的前提下调用 find\_words(),结果都 只会延长前次调用的结果，而不是以在函数定义中所指定的空结果链表开始。程序 的行为并不如预期，因为我们错误地认为在函数被调用时会创建默认值。然而，它 只创建了一次，就是在 Python 解释器加载这个函数时。每当没有给函数提供明确 的值时，就会使用这个链表对象。

**调试技术**

由于大多数代码错误是因为程序员做了错误的假设，当检测到bug 后要做的第一件 事是检查你的假设。向程序添加 print 语句定位问题，显示重要的变量值，并显示 程序的进展程度。

如果程序产生一个“异常”的运行错误，解释器会输出**堆栈跟踪，** 精确定位错误发 生时程序执行的位置。如果程序取决于输入数据，尽量将它减少到能产生错误的最 小程度。

一旦你已经将问题定位在一个特定的函数或一行代码，你需要弄清楚是什么出了错 误。使用交互式命令行重现错误发生时的情况往往很有帮助。定义一些变量，然后

复制粘贴可能出错的代码行到会话中，看看会发生什么。通过阅读一些文档和测试 与之前代码功能相同的其他代码的示例，来检查你对代码的理解。尝试将你的代码 解释给别人听，也许她会发现出错的地方。

Python 提供了一个**调试器，**它允许你监视程序的执行，指定程序暂停运行的行 号(即**断点**),逐步调试代码段和检查变量的值。可以用如下方式在代码中调用 调试器。

>>>import pdb

>>>import mymodule

>>>pdb.run('mymodule.myfunction()')

结果会给出一个提示 (Pdb), 可以在那里输入指令给调试器。输入 help 来查看命 令的完整列表。输入 step (或只输入s) 将执行当前行然后停止。如果当前行调用 一个函数，它将进入这个函数并停止在第一行。类似的输入 next (或只输入 n),

但它会在当前函数中的下一行停止执行。break( 或b) 命令可用于创建或列出断点。 输入 continue (或 c) 会继续执行直到遇到下一个断点。输入变量的名称可以检查 任何变量值。

可以使用 Python 调试器来查找 find\_words()函数的问题。请记住问题是在第二次调 用函数时产生的。 一开始在调用该函数°时不使用调试器，使用可能的最小输入。 第二次使用调试器调用它②。

>>>find\_words(['cat'],3)①

>>>import pdb

>>>pdb.run("find\_words(['dog'],3)")②

['cat']

><string>(1)<module>()

(Pdb)step

--Call--

><stdin>(1)find\_words()

(Pdb)args

text =['dog']

wordlength =3

result =['cat']

在这里，只输入了两个命令到调试器： step 将我们带入函数体内部，args 显示它的 参数的值。我们立即就可以看到 result 有一个初始值[cat], 而不是如预期的空链表。 调试器帮助我们定位问题，促使我们检查对 Python 函数的理解。

**防御性编程**

为了避免一些调试的痛苦，养成防御性的编程习惯是有益的。与其写20行程序然 后测试它，还不如自下而上地打造一些明确可以运作的小程序片段。每次将这些程 序片段组合成更大的单位时，都要仔细地看它是否能如预期的那样运作。考虑在代 码中添加 assert 语句，指定变量的属性，例如： assert(isinstance(text,list)) 。 如果text 的值在代码被用在一些较大环境中时变为了一个字符串，将产生 AssertionError,

此时你将立即得到此问题的通知。

一旦你觉得发现了错误，从一个假设的视角来查看你的解决方案。在重新运行该程 序之前尝试预测修正错误后的影响。如果bug 不能被修正，不要盲目修改代码并希 望它会奇迹般地重新开始运作。相反，每一次修改都要尝试阐明错误是什么及为什 么这样修改会解决这个问题的假设。如果这个问题没有解决就撤消这次修改。

开发程序时，扩展其功能，并修复所有bug, 维护一套测试用例是有益的。这被称为 **回归测试，**因为它是用来检测代码“回归”的地方——修改代码后造成某个意想不 到的副作用的地方，是以前能运行的但现在失效了的地方。Python 以 doctest 模块的 形式提供了一个简单的回归测试框架。这个模块搜索文本块的一段代码或文档文件， 这类似于交互式Python 会话，你已经在这本书中看到了很多次。它执行找到 Python 命令，测试其输出是否与原始文件中所提供的输出匹配。每当有不匹配时，它会报 告预期值和实际值。有关详情，请查询在 <http://docs.python.org/library/doctest.html> 上的 doctest 文档。除了回归测试它的值，doctest 模块有助于确保你的软件文档与 代码保持同步。

也许最重要的防御性编程策略是要清楚地表述你的代码，选择有意义的变量和函数 名，并通过将代码分解成拥有良好文档接口的函数和模块来尽可能地简化代码。

**4.7** **算法设计**

本节将介绍更高级的概念。

解决算法问题的一个重要部分是为手头的问题选择或改造一个合适的算法。有时会有几 种选择，能否选择最好的一个取决于每个算法随数据增长的运行能力。关于这个话题的 书很多，我们只介绍其中的一些关键概念并详细介绍在自然语言处理中最普遍的做法。

最有名的策略被称为**分而治之。**解决一个大小为n 的问题通过将其分成两个大小 为 n/2 的问题，再解决这些问题，把它们的结果组合成原问题的结果。例如：假 设有一堆卡片，每张卡片上写了一个词。通过将它分成两半并分别给另外两个人 来排序(他们又可以做同样的事情)。然后，得到两个排好序的卡片堆，将它们 合并成一个排序堆就容易得多。参见图4-3 以了解这个过程。

|  |  |  |
| --- | --- | --- |
| SPLIT  SPL  SPLIT  MERGE  MERGE  MERGE | rsh cat dag lion nch at dog lion nsh cat dog lion hsh cat dog lion ct fsh dog lion cat dog ish llion  ant bird car dog | bird pig rat ant  bird pg rat ant  bird pig rat ant  bird Pig rat ant  bird pig ant rat  ant bird pig rat  fsh lion pig rat |

**图4-3** **通过分而治之排序：对一个数组排序，将其分成两半并对每一半进行排序(递归);将** **每个排好序的一半合并成一个完整的链表(再次递归);这个算法被称为“归并排序”**

另一个例子是在词典中查找一个词。打开在书本中部附近的某个地方，比较指定的 词与当前页面上的词。如果它在词典中的前面，就在词典的前一半重复上面的过程； 如果它在后面，就使用词典的后一半。这种搜索方法被称为二分查找，因为它的每 一步都将问题分成两半。

算法设计的另一种方法是，将问题转化为一个已熟知解决方法的另外一个问题来解 决它。例如：为了检测链表中的重复项，可以预排序这个链表，然后通过一次扫描 检查是否有相邻的两个元素是相同的。

**递归**

上面的关于排序和搜索的例子有一个醒目的特征：解决一个大小为 n 的问题，可 以将其分成两半，然后处理一个或多个大小为 n/2 的问题。实现这种方法的常见 方式是使用递归。定义一个函数 f, 用于简化问题，并调用自身来解决一个或多个 同样的问题。然后把它们的结果组合成为原问题的解答。

例如：假设有n 个词，要计算出它们结合在一起组成词序列的方式有多少种。

如果只有一个词 (n=1), 那么只用一种方式组成一个序列。如果有两个词，就 有两种方式将它们组成一个序列。3 个词就有6 种可能性。 一般的， n 个词有 n×n-1×…×2×1种方式(即n 的阶乘)。可以将这些编写成如下代码。

>>>def factoriall(n):

· .result =1

for i in range(n):

.. · . result \*=(i+1)

…… return result

此外，也可以使用一种递归算法来解决这个问题，该算法基于以下观察：假设有办 法为n-1个不同的词构建所有的排列。然后对于每个这样的排列，有n 个地方可以 插入一个新词：开始处、结尾处或任意两个词之间空隙(有 n-2 个)。因此，简单 地将n-1 个词的解决方案数乘以n 的值。我们还需要**基础案例，**也就是说，如果有 一个词，就只有一个顺序。可以将这些编写成如下代码。

>>>def factorial2(n):

 if n ==1:

return 1

\* · else:

return n \*factorial2(n-1)

使用这两种算法解决同样的问题。 一个使用迭代，而另一个使用递归。可以用递归 处理深层嵌套的对象，例如： WordNet 的上位词层次。计数以给定同义词集s 为根 的上位词层次的大小。找到s 的每个下位词的大小，然后将它们加到一起(加1表 示同义词集本身)。下面的函数 size1)进行如上的操作；注意，在函数体中也包括 sizel)的递归调用。

>>>def sizel(s):

return 1+sum(sizel(child)for child in s.hyponyms())

也可以为这个问题设计一种迭代解决方案来处理层的层次结构。第一层是同义词集 本身°,然后是同义词集所有的下位词，之后是所有下位词的下位词。每次循环通 过查找上一层的所有下位词来计算下一层。它也存档了到目前为止遇到的同义词 集的总数目②

>>>def

·

la2)[:s]①

total =0

·

·

whilyl for h in c.hyponyms()]③

return total

迭代解决方案不仅代码更长而且更难理解。它迫使我们以程序的方式式思考问题， 并跟踪 layer 和total 随时间发生的变化。这两种解决方案给出了相同的结果。下面 的列子中将使用 import 语句的新的形式，把 wordnet缩写为 wn。

>>>from nltk.corpus import wordnet as wn

>>>dog =wn.synset('dog.n.01')

>>>sizel(dog)

190

>>>size2(dog)

190

作为递归的最后一个例子，让我们用它来构建一个深嵌套的对象。**字母查找树** (letter trie) 是一种可以用来索引词汇的数据结构， 一次一个字母。(根据词retrieval 而得名)。例如：如果 trie 包含一个字母树，那么 trie[c] 是一个较小的查找树且包 含了所有以c 开头的词。例4-6演示了使用Python字典(见5.3节)构建查找树的 递归过程。为了插入词chien (法语中狗的意思),我们将c 分类，递归地在 trie[c] 的子查找树中插入hien。当我们存储了预期值后(本例中是词chien), 递归将一直 持续到词中没有剩余的字母为止。

例4-6 构建一个字母查找树： 一个递归函数建立一个嵌套的字典结构，每一级嵌 套包含给定前缀的所有单词，而子查找树含有所有可能的后续词

def insert(trie,key,value):

if kefst,rest =key[0],key[1:]

if firsiteftn]r:

insert(trie[first],rest,value)

else:

trie['value']=value

>>>trie =nltk.defaultdict(dict)

>>>insert(trie,'chat','cat')

>>>insert(trie,'chien', 'dog')

>>>insert(trie,'chair','flesh')

>>>insert(trie,'chic', 'stylish')

>>>trie =dict(trie) #for nicer printing



>>>trie['c']['h']['a']['t']['value']

pprint.pprint(trie)

{'c':{'h':{'a':{'t':{'value':'cat'}},

{'i':{'r':{'value':'flesh'}}},

'i':('e':{'n':{'value':'dog'}}}

{'c':{'value':'stylish'}}}}}

**注意!**

尽管递归编程结构简单，但它是有代价的。每次函数调用时， 一些 状态信息需要推入堆栈，这样一旦函数结束了，也可以从它离开的 地方继续执行。出于这个原因，迭代的解决方案往往比递归解决方 案更高效。

**空间与时间的权衡**

通过建立一个辅助的数据结构，可以显著地加快程序的执行，例如一个索引。例 4-7实现对一个简单的电影评论语料库的全文检索系统。通过索引文档集合，进行 了更快速的查找。

**例4-7** **一个简单的全文检索系统**

def s):=open(file).read()

contents =re.sub(r'<.\*?>','',contents)

contents =re.sub('\s+','',contents)

return contents

def snippet(doc,term):#buggy

text =''\*30 +raw(doc)+''\*30

pos =text.index(term)

return text[pos-30:pos+30]

irli =lBtius.evxi.\_"reviews.abspaths()

idx =nltk.Index((w,f)for f in files for w in raw(f).split())

query =!

while query !="quit":

query =raw\_input("query>")

if query in idx:

for doc in idx[query]:

print snippet(doc,query)

else:

print "Not found"

这是一个微妙空间与时间折中的例子，涉及到使用整数标识符来替换语料库的标识 符。为语料库创建一个词汇表，其中每个词仅被存储一次，然后转化这个链表以便 能通过查找任意词来识别它的标识符。每个文档都进行预处理，以便使词链表变 成整数链表。现在所有的语言模型都可以使用整数。例4-8中针对一个已标注的 语料库给出示范。

例4-8 预处理已标注的语料库数据，将所有的词和标注转换成整数

def preprocess(tagged\_corpus):

words =set()

tags =set()

for sent in tagged\_corpus:

for word, tag in sent:

words.add(word)

tags.add(tag)

wm =dict((w,i)for(i,w) in enumerate(words))

tm =dict((t,i)for(i,t)in enumerate(tags))

return[[(wm[w],tm[t])for(w,t)in sent]for sent in tagged\_corpus]

另一个空间时间权衡的例子是维护词汇表。如果需要处理一段输入的文本来检查 所有的词是否在现有的词汇表中，词汇表应存储为一个集合，而不是一个链表。 集合中的元素会自动索引，所以测试一个大集合的成员将远远快于测试相应的链 表的成员。

可以使用timeit 模块检测这种说法。Timer 类有两个参数： 一个是多次执行的声明， 一个是只在开始时执行一次的设置代码。我们将分别使用整数的链表°和整数的集合② 来模拟10万个项目的词汇表。测试声明将产生随机项，它有50%的机会出现在词 汇表中③。

>>>from timeit import Timer

>>>vocab\_size >>>setup\_list >>>setup\_set >>>statement

>>>print

2.78 092288971 >>>print

=100000

="import random;vocab =range(8d)"%vocab\_size①

="import random;vocab =set(range(8d))"%vocab\_size② ="random.randint(0,8d)in vocab"8 vocab\_size \*2③

Timer(statement,setup\_list).timeit(1000)

Timer(statement,setup\_set).timeit(1000)

0.00 37260055542

执行1000次链表成员资格测试总共需要2.8秒，而在集合上的等效测试仅需0.0037

秒，也就是说快了3个数量级!

**动态规划**

动态规划 (Dynamic programming) 是一种在自然语言处理中被广泛使用的算法设 计方法。“programming” 一词的用法与你期望的可能不同，是规划或调度的意思。 动态规划用于解决包含多个重叠子问题的问题。不是反复计算这些子问题，而是简 单地将它们的计算结果存储在一个查找表中。在本节的余下部分，我们将介绍动态 规划，但在一个相当不同的背景：句法分析。

Pingala 是一名生活在大约公元前5世纪的印度作家，作品有名为《Chandas Shastra》 的梵文韵律专著。Virahanka大约在公元6世纪期间延续了这项工作，研究短音节和 长音节组合产生一个长度为n 的旋律组合数。短音节，标记为S, 占一个长度单位； 而长音节，标记为L, 占2个长度单位。Pingala发现，例如：有5种方式构造一个 长度为4的旋律： V₄={LL,SSL,SLS,LSS,SSSS}。 请看，我们可以将V₄ 分成两 个子集，以L 开始的子集和以S 开始的子集，如(1)所示。

(1)V₄=

LL,LSS

i.e.L prefixed to each item of V₂={L,SS}

SSL,SLS, SSSS

i.e.S prefixed to each item of V₃={SL,LS,SSS}

有了这个观察结果，我们可以编写一个称为 virahanka10的递归函数来计算这些旋 律，如例4-9所示。请注意，要计算 V₄, 我们先要计算 V₃和 V₂。但要计算 V₃, 我们先要计算V₂ 和V₁ 。在(2)中描述了这种**调用结构。**

例4-9 4种方法计算梵文旋律：迭代；自底向上的动态规划；自上而下的动态规 划；内置默记法

def virahankal(n):

if n ==0:

return [""]

elif n ==1:

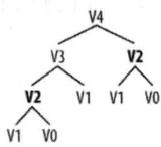
return ["S"]

else=["S"+prosody for prosody in virahankal(n-1)]

1=["L"+prosody for prosody in virahankal(n-2)] return s+1

def virahanka2(n):

lookup =[[""],["S"]]



for = S" for prosody in lookup[i+1]]

1=["L"+prosody for prosody in lookup[i]]

lookup.append(s +1)

return lookup[n]

def virahanka3(n,lookup={0:[""],1:["S"]}):

if nsS: for prosody in virahanka3(n- 1)]

1=["L"+prosody for prosody in virahanka3(n-2)] lookup[n]=s +1

return lookup[n]

from nltk import memoize

@memoize

def i(n):

return [""]

elif n ==1:

return ["S"]

else:

s =["S"+prosody for prosody in virahanka4(n-1)] 1=["L"+prosody for prosody in virahanka4(n-2)] return s +1

>>>virahankal(4)

['SSSS','SSL','SLS','LSS','LL']

>>>virahanka2(4)

['SSSS','SSL','SLS','LSS','LL']

>>>virahanka3(4)

['SSSs','SSL','SLS','LSS','LL']

>>>virahanka4(4)

['SSSS','SSL','SLS','LSS','LL']

(2)

正如你可以看到，V₂ 计算了两次。这看上去可能并不像是一个重大的问题，但事 实证明，当n 变大时使用这种递归技术显得很浪费。为了计算 V₂o, 我们将计算 V₂共4181次；为了计算V₄0我们将计算 V₂共63245986次!更好的选择是将 V₂ 的值存储在一个表中，当我们需要它时查这个表， V₃ 等其他值也是同样处理。函 数 virahanka2O实现了使用动态规划方法来解决这个问题。它的工作原理是将此问

题的所有较小的实例的计算结果填充到一个表格(叫做lookup), 一旦得到了我们 感兴趣的值就立即停止。此时，我们读出值，并返回它。最重要的是，每个子问题 只计算了一次。

请注意，virahanka2(所采取的办法是在解决较大问题之前先解决较小的问题。因此， 这被称为自下而上的动态规划。遗憾的是，对于某些应用它还是相当浪费资源的， 因为它计算的一些子问题在解决主问题时可能并不需要。采用自上而下的方法进行 动态规划可避免这种计算的浪费，如例4-9 中函数 virahanka3O 所示。不同于自下 而上的方法，这种方法是递归的。通过检查是否先前已存储了结果，它避免了 virahanka1) 的巨大浪费。如果没有存储，就递归地计算结果，并将结果存储在表中。 最后一步返回存储的结果。最后一种方法， invirahanka40, 使用 Python 的“装饰 器”称为默记法 (memoize), 它会完成 virahanka3) 所做的繁琐工作而不会搞乱程 序。这种“默记”过程中会存储每次函数调用时的结果及使用到的参数。如果随后 的函数调用了同样的参数，它会返回存储的结果，而不是重新计算。(这方面的 Python 语法超出了本书的讨论范围。)

这里我们只简要介绍了动态规划。在8.4节中，我们将继续学习。

**4.8** **Python** **库的样例**

Python 有数以百计的第三方库，有扩展 Python 功能的专门软件包。NLTK 就是一 个这样的库。要全面了解 Python 编程能力，你应该熟悉几个其他的库。它们中的 大多数都需要在你的计算机上进行手动安装。

**Matplotlib 绘图工具**

Python 拥有一些有用的可视化语言数据的库。Matplotlib 包支持 MATLAB 风格接 口的复杂绘图函数，详情参见 <http://matplotlib.sourceforge.net/>。

到目前为止，我们一直专注文字介绍和使用格式化输出语句并以列的形式得到输 出。以图形的形式显示数值数据往往是非常有用的，因为这更容易检测到模式的形 式。例如：在例3-5中，我们看到一个数字的表格，显示按类别划分的布朗语料库 中的特殊情态动词的频率。例4-10 中的程序以图形格式展示同样的信息。输出显 示在图4-4中(一个图形显示的彩色图)。

**例4-10** **布朗语料库中不同部分的情态动词频率**

colors ='rgbcmyk'#red,green,blue,cyan,magenta,yellow,black

def bar\_chart(categories,words,counts):

"Plot a bar chart showing counts for each word by category"

dport=.arange(len(words))

width=1/(len(categories)+1)

bar\_groups =[]

for a(i)d:th,counts[categories[c]],width,

color=colors[c len(colors)])

bar\_groups.append(bars)

ceknsd(h,or) bar\_groups],categories,loc='upper left')

pylab.ylabel('Frequency')

pylab.title('Frequency of Six Modal Verbs by Genre')

pylab.show()

>>>genres >>>modals

>>>cfdist

·

·

… ·

·

=['news','religion','hobbies','government','adventure']

=nltk.ConditionalFreqDiscan','could','may','might','must','will']

(genre,word)

for genre in genres

for word in nltk.corpus.brown.words(categories=genre)

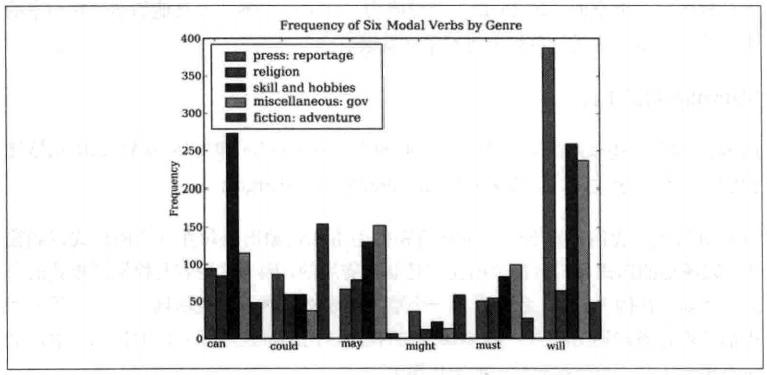
if word in modals)

>>>counts ={}

>>>for s edist[genre][word]for

>>>bar\_chart(genres,modals,counts)

word in modals]



**图4-4** **条形图显示布朗语料库中不同部分的情态动词频率：这个可视化图形由例4-10中的程序产生**

从该柱状图可以看出 may 和 must 有几乎相同的相对频率。could 和 might 也 一样。

也可以动态地产生这些数据的可视化图形。例如： 一个使用表单输入的网页可以允 许访问者指定搜索参数，提交表单，然后生成一个动态生成的可视化图形。要做到 这一点，我们必须为 matplotlib 指定Agg 后台，它是一个产生栅格(像素)图像的 库°。下一步，我们像以前一样使用相同的PyLab 方法，但不是用pylab.show)把结 果显示在图形终端，而是使用 pylab.savefig(把它保存到一个文件里②。我们指定文 件名和 dpi, 然后输出并指示网页浏览器来加载该文件的 HTML 标记。



>>>pylab.savefig('modals.png')②

>>>print 'Content-Type:text/html'

>>>print

>>>print ¹<html><body>'

>>>print¹<img src="modals.png"/>'

>>>print‘</body></html>'

**NetworkX**

NetworkX包定义和操作由节点和边组成的结构(称为图)。它可以从https://networkx. lanl.gov/得到。NetworkX 可以和Matplotlib 结合使用来可视化如 WordNet 的网络结 构(语义网络，在2.5节介绍过)。例4-11 中的程序初始化一个空的图，然后遍 历 WordNet 上位词层次并为图添加边°。请注意，遍历是递归的②,使用在4.7 节讨论的编程技术。结果如图4-5所示。

**例4-11** **使用** **NetworkX 和** **Matplotlib 库**

import networkx as nx

import matplotlib

from nltk.corpus import wordnet as wn

def e.shortest\_path\_distance(start)

for child in node.hyponyms():

graph.add\_edge(node.name,child.name)①

traverse(graph,start,child)②

def hyponym\_graph(start):

G =nx.Graph()③

G.depth ={}

s(G,start,start)

def graph\_draw(graph):

nx.draapshiiezg[,graph.degree(n)for n in graph],

tdhe\_\_l F[garl.depth[n]for n in graph],

matplotlib.pyplot.show()

>>>dog =wn.synset('dog.n.01')

>>>graph =hyponym\_graph(dog)

>>>graph\_draw(graph)

|  |
| --- |
|  |

**图4-5** **使用NetworkX 和** **Matplotlib 可视化数据：显示了WordNet的部分上位词层次，开始于** **dog.n.01(中间最黑的节点);节点的大小对应子节点的数目，颜色对应节点到dog.n.01的距离；**

**此可视化图形由例4-11** **中的程序产生**

**CSV**

语言分析工作往往涉及数据统计表，包括有关词项的信息、试验研究的参与者名单 或从语料库提取的语言特征。这里有一个 CSV(C omma-separated values, 逗号分 隔型取值格式)格式的简单词典片段。

sleep,sli:p,v.i,a condition of body and mind ...

sebyto lip and setting down each foot .

我们可以使用 Python的 CSV 库读写这种格式存储的文件。例如：可以打开一个叫 做 lexicon.csv的 CSV 文件”,并遍历它的行②。



>>>for row in csv.reader(input\_file):②

…… print row

['sleep','sli:p','v.i','a condition

['walk','wo:k','v.intr','progress

of

by

body

lifting

and

and

mind

setting

.']

down

each

foot ['wa ','weik','intrans','cease to sleep']

每一行是一个字符串列表。如果字段包含有数值数据，它们将作为字符串出现，所 以都必须使用 int()或 float()转换。

**NumPy**

NumPy 包(基本的数值运算包)为 Python 中的数值处理提供了大量的支持。 NumPy 有一个多维数组对象，它很容易被初始化和访问。

enumpy=i( y,0,0],[1,1,1],[2,2,2]],

· [[3,3,3],[4,4,4], [5,5,5]],

.. [[6,6,6],[7,7,7],[8,8,8]]])

>>>cube[1,1,1]

4

>>>cube[2].transpose()

array([[6,7,8],

[6,7,8],

[6,7,8]])

>>>cube[2,1:]

array([[7,7,7],

[8,8,8]])

**NumPy** 包括线性代数函数。在这里我们进行矩阵的奇异值分解，这是在**潜在语义** **分析中**使用的操作，它能帮助识别一个文档集合中的隐含概念。

>>>from numpy import linalg

>>>a=array([[4,0],[3,-5]])

>>>u,s,vt =linalg.svd(a)

>>> u

array([[-0.4472136,-0.89442719],

[-0.89442719,0.4472136 ]])

>>>s

array([6.32455532,3.16227766])

>>>vt

array([[-0.70710678,0.70710678],

[-0.70710678,-0.70710678]])

在NumPy数组中广泛使用了NLTK的聚类包nltk.cluster,支持包括k-means聚类、高斯 EM聚类、组平均凝聚聚类及聚类分析图。有关详细信息，请输入help(nltk.cluster)。

**其他** **Python 库**

还有许多其他的 Python 库，你可以使用 <http://pypi.python.org/>处的 Python 包索引 找到它们。许多库提供了外部软件接口，例如：关系数据库(如： mysql-python) 和大数据集合(如 PyLucene)。还有很多其他库能访问各种文件格式，如 PDF、

MSWord和 XML(pypdf、pywin32 、xml.etree) 、RSS 订阅(如 feedparser) 及电子 邮箱(如 imaplib 、email)。

**4.9** **小结**

● 使用对象引用进行 Python 赋值和参数传递，例如：如果 a 是一个链表，我们 分配b=a, 然后任何a 上的操作都将修改 b, 反之亦然。

· is 测试是否两个对象是相同的内部对象，而==测试是否两个对象是相等的。两 者的区别和标识符与类型的区别相似。

● 字符串、链表和元组是不同类型的序列对象，支持常见的操作如：索引、切片、 len) 、sorted() 和使用in 的成员测试。

● 可以通过打开文件将文本写入到一个文件： ofile=open(output.txt,w), 然后 加入内容到文件：ofile.write("Monty Python"),最后关闭文件 ofile.close)。

● 声明式的编程风格通常会使代码更简洁更可读；手动递增循环变量通常是不必 要的。枚举一个序列，使用 enumerate()。

● 函数是一个重要的编程抽象化过程，需要理解的关键概念有：参数传递、变量 的范围和 docstrings.

● 函数作为一个命名空间：函数内部定义的名称在该函数外不可见，除非这些名 称被宣布为是全局的。

● 模块允许将材料与本地的文件逻辑的关联起来。 一个模块作为一个命名空间： 在一个模块中定义的名称，如变量和函数，在其他模块中不可见，除非这些名

称被导入。

● 动态规划是一种在 NLP 中广泛使用的算法设计技术，它存储以前的计算结果， 以避免不必要的重复计算。

**4.10** **深入阅读**

本章已经触及编程中的许多主题， 一些是 Python 特有的， 一些是相当普遍的。我 们只是触及了表面，更多有关这些主题的内容，可以访问 <http://www.nltk.org/>对本 章进行深入的阅读。

Python 网站提供了大量文档。理解内置的函数和标准类型是很重要的，在 <http://docs.python.org/library/functions.html> 和 <http://docs.python.org/ibrary/stdtypes.html> 处 有描述。我们已经学习了产生器及其对提高效率的重要性。关于迭代器，请参考 <http://docs.python.org/library/itertools.html>。查询你喜欢的 Python 书籍中这些主题 的更多信息。还有一个使用Python 进行多媒体处理的优秀的资源，即 (Guzdial, 2005),其中包括声音文件。

使用在线 Python 文档时，要注意你安装的版本可能与你正在阅读的文档的版本不 同。可以使用 import sys;sys.version 轻松地查询你现有的版本。特定版本的文档在 <http://www.python.org/doc/versions/>处。

算法设计是计算机科学中一个丰富的领域。可以查阅 (Harel,2004),(Levitin,

2004)和 (Knuth,2006) 。(Hunt&Thomas,2000) 和 (McConnell,2004) 为软

件开发实践提供了有益的指导。

**4.11** **练习**

1.O 使用 Python 的帮助功能，找出更多关于序列对象的内容。在解释器中输入 help(str)、help(list)和 help(tuple)。将产生每种类型所支持函数的完整列表。 一些 函数名字有些特别，两侧有下划线；正如帮助文档显示的那样，每个这样的函 数对应于一些较为熟悉的东西。例如 x.getitem\_\_(y) 仅仅是以长篇大论的方式 使用 x[y]。

2.O 确定3个能同时在元组和链表上都可以执行的操作。确定3个不能在元组上 执行的链表操作。命名一段使用链表替代元组后 Python产生错误的代码。

3.O 了解如何创建一个由单个项目组成的元组。至少使用两种方法。

4.O 创建链表 words =[is',NLP','fun',"?] 。 使用一系列赋值语句(如 words[1]= words[2]) 和临时变量tmp 将这个链表转换为链表[NLP','is,'fun',!!] 。 现在，使用 元组赋值做相同的转换。

5.O 通过输入help(cmp) 阅读关于内置的比较函数 cmp 的内容。它与比较运算符在 行为上有何不同?

6.〇创建n-grams 的滑动窗口的方法在下面两种极端情况下是否正确： n=1 和 n= len(sent)?

7.O 指出当空字符串和空链表出现在 if 从句的条件部分时，它们的判断结果是 False。在这种情况下，它们被说成是出现在一个布尔上下文中。尝试各种不同的布 尔上下文中的非布尔表达式，看它们被判断为 True 或 False。

8.O 使用不等号比较字符串，如： 'Monty'<'Python' 。 当'Z'<'a '时会发生什么?尝 试具有共同前缀的字符串对，如：'Monty'<'Montague' 。 阅读有关“字典排序”的 内容以便了解这里发生的详情。尝试比较结构化对象，如：(Monty’,1)<(Monty',2)。

这与预期一样吗?

9.O 编写代码删除字符串开头和结尾处的空白，规范词之间的空格并使其成为一 个单独的空格字符。

a. 使用 split()和 join()。

b. 使用正则表达式替换。

10.O 编写一个程序按长度对词排序。定义一个辅助函数 cmp\_len, 它在词长上使 用 cmp 比较函数。

11.①创建一个词链表并将其存储在变量sent1。现在赋值sent2=sent1。修改 sent1 中的一个项目，验证 sent2 改变了。

a. 现在尝试同样的练习，但使用 sent2=sent1[:] 替换赋值语句。再次修改 sent1 看看sent2 会发生什么。并解释之。

b. 现在定义text1 为一个字符串链表的链表(例如：表示由多个句子组成的文本)。 现在赋值 text2=text1[:], 分配一个新值给其中一个词，例如：text1[1][1]= 'Monty'。检查这对 text2 做了什么。并解释之。

c. 导入Python 的 deepcopy()函数(即 from copy import deepcopy), 查询其文 档，并验证它生成的任一对象的新副本。

12.①使用链表乘法初始化n×m 的空字符串链表的链表，例如： word\_table=[["]\* n]\*m 。 当设置其中一个值时会发生什么事，例如： word\_table[1][2]="hello"? 解 释为什么会出现这种情况。现在写出一个表达式，使用 range()构造一个链表，表明 它没有这个问题。

13.①编写代码初始化一个名为 word\_vowels 的二维数组的集合，处理一个词链表， 把每个词添加到 word\_vowels[1][V], 其中1是词的长度， v 是它包含的元音的数量。

14.①编写一个函数 novel10(text), 输出所有第一次出现在文本最后10%的词。

15.①编写一个程序将一个句子表示成一个单独的字符串，分割并计数其中的词。 让它输出每一个词及其频率，按字母顺序排列，每行一个。

16.O 阅读有关 Gematria 的内容，它是一种方法，用来分配数字给词汇，在具有 相同数字的词之间构造映射以发现文本隐藏的含义。(<http://en.wikipedia.org/wiki/> Gematria,<http://essenes.net/gemcal.htm>)。

a. 编写一个函数 gematria(), 根据 letter\_vals 中的字母值，累加某个词的字母 的数值。

>>>letter\_vals={'a':1,'b':2,'c':3,'d':4,'e':5,'f':80,'g':3,'h':8,

...'i':10,'j':10,'k':20,'l':30,'m':40,'n':50,'o':70,'p':80,'q':100,

..'r':200,'s':300,'t':400,'u':6,'v':6,'w':800,'x':60,'y':10,'z':7)

b. 处理一个语料库(如 nltk.corpus.state\_union) 对每个文档计数字母数值为 666的词有多少个。

c. 编写一个函数 decode) 来处理文本，为了发现文本的“隐藏的含义”,随机替 换掉与Gematria等值的词，。

17.①编写一个函数 shorten(text,n)处理文本，省略文本中前n 个最频繁出现的词。 它的可读性会如何?

18.①编写代码输出词汇的索引，允许别人根据其含义查找词汇(或它们的发音； 或者词汇条目中包含的任何属性)。

19.①编写一个链表推导来对 WordNet 中与给定同义词集接近的同义词集链表进 行排序。例如：给定同义词集minke\_whale.n.01、orca.n.01、novel.n.01 和 tortoise.n.01, 按照它们与 right\_whale.n.01 的 path\_distance()对它们进行排序。

20.①编写一个函数处理一个词链表(含重复项),并返回一个按照频率递减排序 的词链表(没有重复项)。例如：如果输入链表中包含词 table 的10个实例， chair 的9个实例，那么在输出链表中 table 会出现在 chair 前面。

21.①编写一个函数以一个文本和一个词汇表作为它的参数，返回在文本中出现但 不在词汇表中的一组词。两个参数都可以表示为字符串链表。你能使用 set.difference()在一行代码中实现这些吗?

22. ①从 Python 的标准库的 operator 模块导入 itemgetter) 函数(即 from operator import itemgetter)。 创建一个包含几个词的链表。现在尝试调用： sorted(words, key=itemgetter(1))和 sorted(words,key=itemgetter(-1))。解释 itemgetter()正在做什么。

23.①编写一个递归函数 lookup(trie,key)在查找树中查找一个关键字，返回找到的 值。然后扩展这个函数成：当一个词由其前缀唯一确定时返回这个词(例如： vanguard 是唯一的以vang-开头的词，所以lookup(trie,vang) 应该返回与 lookup(trie, 'vanguard')相同的内容)。

24.①阅读“关键字联接”的相关内容((Scott& Tribble,2006) 的第5章)。使 用 NetworkX 包，从 NLTK 的莎士比亚语料库中提取关键字，并画出关键字联接 网络。

25.①阅读有关字符串编辑距离和 Levenshtein算法的内容。尝试 nltk.edit\_dist()提

供的实例。这用的是动态规划的何种方式?它使用的是自下而上或自上而下的方法 吗? (也可参阅 <http://norvig.com/spell-correct.html>)

26.OCat alan 数出现在组合数学的许多应用中，包括解析树的计数(8.6 节)。该 级数可以定义如下： Co=1, Cn+1=Zo.n(C;Cn-i)。

a. 编写一个递归函数计算第n 个 Catalan数 Cn。

b. 现在编写另一个函数使用动态规划做这个计算。

c. 使用 timeit 模块比较当n 增加时这些函数的性能。

27.●重现有关著作权鉴定的 (Zhao &Zobel.2007) 中的一些结果。

28.●研究性别特异词汇选择，看你是否可以重现像这 <http://www.clintoneast.com/> articles/words.php 所示的部分结果。

29.●编写一个递归函数，按字母顺序排列输出一个查找树。

chair: 'flesh'

---t: 'cat'

--ic:'stylish'

---en: 'dog'

30.●在查找树数据结构的基础上，编写一个递归函数处理文本，定位在每个词的 独特点，并丢弃每个词的其余部分。这样压缩了多少?产生的文本可读性如何?

31.●以一个单独的长字符串的形式，获取一些原始文本。使用 Python 的 textwrap 模 块将它分割成多行。现在，编写代码实现在词之间添加额外的空格，以调整输出。 每一行必须具有相同的宽度，每行的空格均匀分布。不能出现以空白开始或结束 的行。

32.●开发一个简单的挖掘总结工具，输出一个文档中所包含的总词频最高的句子。 使用 FreqDist) 计数词频，并使用 sum 累加每个句子中词的频率。按照句子的得分 排序。最后，按文档顺序输出n 个最高得分的句子。仔细检查你的程序设计，尤其 是关于这种双重的排序方法。确保程序写得尽可能明了。

33.●阅读下面的关于形容词的语义倾向文章。使用NetworkX 包可视化一个形容

词的网络，其中的边表示相同和不同的语义倾向(见 htp:/www.aclweb.org/anthology/ P97-P1023)。

34.●设计一个算法找出一个文档集合中“统计学上不可能的短语”(见 <http://www.amazon.com/gp/search-inside/sipshelp.html>)。

35.●编写一个程序实现发现一种nxn 的四方联词的蛮力算法：纵横字谜，它的第

n 行的词与第 n 列的词相同。讨论见 <http://itre.cis.upenn.edu/>～myllanguagelog/

archives/002679.html。



**第5章**

**分类和标注词汇**

我们早在小学就学过名词、动词、形容词和副词之间的差异。这些“词类”不是文 法家闲的没事发明的，而是对许多语言处理任务都有帮助的分类。正如我们将看到 的，这些分类源于对文本中词分布的简单分析。本章的目标就是要回答下列问题。

(1)什么是词汇分类，在自然语言处理中它们如何使用?

(2)对于存储词汇和它们的分类来说什么是好的Python 数据结构?

(3)如何自动标注文本中每个词汇的词类?

在这个过程中，我们将介绍NLP 的一些基本技术，包括序列标注、N-gram 模型、 回退和评估。这些技术在许多方面都很有用，标注为我们提供了介绍它们的简 单上下文。我们还将理解到标注为何是典型的处于NLP 流水线中继分词之后的 第二个步骤。

将词汇按它们的词性(parts-of-speech,POS)分类并相应地对它们进行标注，这个 过程叫作**词性标注** (part-of-speech tagging,POS tagging) 或干脆简称标注。词性也 **称为词类或词汇范畴**。用于特定任务标记的集合被称为一个标记集。我们在本章的 重点是利用标记和自动标注文本。

**5.1** **使用词性标注器**

词性标注器 (part-of-speech tagger 或 POS tagger) 处理一个词序列，为每个词附加



一个词性标记(不要忘记 import nltk)。

k.stk\_.nize("And now for something completely different")

[('And','CC¹),('now','RB'),('for','IN'),('something','NN'),

('completely','RB'),('different','JJ')]

在这里我们看到 and 是 CC, 即并列连词； now 和 completely 是 RB, 或者副词；for 是IN, 即介词；something 是NN, 即名词； different 是 JJ, 即形容词。

NLTK 中提供了每个标记的文档，可以使用标记来查询，如： nltk.help.upenn\_tagset(RB)), 或正则表达式，如： nltk.help.upenn\_brown tagset(NN. \*)。 一 些语料库有标记集文档的 README 文件；见 nltk.name.readme(), 用语料库的名称替换 name 。

让我们来看看下面的例子，包括一些同形同音异义词。

>>>text=nltk.word\_tokenize("They refuse to permit us to obtain the refuse per

[('They','PRP'),('refuse','VBP'),('to','TO'),('permit','VB'),('us',

'PRP'),

('to','TO'),('obtain','VB'),('the','DT'),('refuse','NN'),('permit',

'NN')]

请注意 refuse 和 permit 都以一般现在时动词 (VBP) 和名词 (NN) 形式出现。例 如：refUSE 是一个动词，意为“拒绝”,而 REFuse 是一个名词，意思是“垃圾” (即它们不是同音同义词)。因此，我们需要知道正在使用哪一个词以便能正确读课 文。(出于这个原因，文本转语音系统通常要进行词性标注。)

**轮到你来：**

很多词，如 ski 和 race, 可以作为名词或动词而发音没有区别。你能 想出其他的吗?提示：想想一个常见的东西，尝试把词 to放到它前面， 看它是否也是一个动词；或者想想一个动作，尝试把the放在它前面， 看它是否也是一个名词。现在用这个词的两种用途造句，在这句话上 运行POS 标注器。

词汇类别，如“名词”,以及词性标记，如 NN, 看上去似乎有些用途，但在细 节上将使许多读者感到晦涩。你可能想知道为什么要引进这种额外的信息吗?这

>>>tagged\_token [0]

些类别中很多都源于对文本中词语分布的浅层分析。考虑下面的分析，涉及 woman (名词),bought (动词),over (介词)和 the (限定词)。text.similar() 方 法为词w 找出所有上下文 wlww2, 然后找出所有出现在相同上下文中的词w', 即 w1w'w2。

>>>text=nltk.Text(word.lower()for word in nltk.corpus.brown.words())

>>>text.similar('woman')

Building word-context index...

man time day year car moment world family house country child boy

state job way war girl place room word

>>>text.similar('bought')

made said put done seen had found left given heard brought got been

txe**t**iomll'vi'elt that

in on to of and for with from at by that into as up out down through

is all about

>>>text.similar('the')

a his this their its her an that our any all one these my in your no some other and

搜索 woman 找到的是名词；搜索 bought 找到的大部分是动词；搜索 over 一般会找 到介词；搜索 the 找到某些限定词。 一个标注器能够正确识别句子上下文中这些词 的标记。例如： The woman bought over $150,000 worth of clothes。

一个标注器也可以对未知词的认识过程建模。例如：我们可以根据词根 scrobble 猜测 scrobbling 可能是一个动词，并有可能发生在 he was scrobbling 这样的上下 文中。

**5.2** **标注语料库**

**表示已标注的标识符**

按照 NLTK 的约定，已标注的标识符使用一个由标识符和标记组成的元组来表示。 我们可以使用函数 str2tuple()表示一个已标注的标识符的标准字符串创建一个如下 的特殊元组。

>>>tagged\_token =nltk.tag.str2tuple('fly/NN')

>>>tagged\_token

('fly','NN')

'fly'

>>>tagged\_token[1]

'NN'

我们可以直接从一个字符串构造一个已标注的标识符的链表。第一步是对字符串分 词以便能访问单独的词/标记字符串，然后将每一个转换成一个元组(使用 str2tuple))。

>>>sent=

..The/AT grand/JJ jury/NN commented/VBD on/IN a/AT number/NN of/IN

.other/AP topics/NNS,/,AMONG/IN them/PPO the/AT Atlanta/NP and/cc

…Fulton/NP-tl County/NN-tl purchasing/VBG departments/NNS which/WDT it/PPS ...said/VBD``/``ARE/BER well/QL operated/VBN and/Cc follow/VB generally/RB

…accepted/VBN practices/NNS which/WDT inure/VB to/IN the/AT best/JJT ..interest/NN of/IN both/ABX governments/NNS''/''./.



>>>[nltk.tag.str2tuple(t)for t in sent.split()]

[('The','AT'),('grand','JJ'),('jury','NN'),('commented','VBD'),

('on','IN'),('a','AT'),('number','NN'),...('.','.')]

**读取已标注的语料库**

**NLTK** 中包括的若干语料库已标注了词性。如果利用文本编辑器打开布朗语料库的 文件就能看到下面的例子。

The/at Fulton/np-tl County/nn-tl Grand/jj-tl Jury/nn-tl said/vbd Friday/nr- an/at inves-tigation/nn of/in Atlanta's/nps recent/jj primary/nn election/nn produced/vbd /no/atevidence/nn''/''that/cs any/dti irregularities/nns took/vbd place/nn ./.

其他语料库使用各种格式存储词性标记。NLTK 中的语料库阅读器提供了统一 的接口，以至于不必理会不同的文件格式。与刚才上面提取并显示的文件不同， 布朗语料库的语料库阅读器按如下所示的方式表示数据。注意：部分词性标记 已转换为大写的。自从布朗语料库发布以来，这已成为标准的做法。

>>>nltk.corpus.brown.tagged\_words()

[('The','AT'),('Fulton','NP-TL'),('County','NN-TL'),...]

>>>nltk.corpus.brown.tagged\_words(simplify\_tags=True)

[('The','DET'),('Fulton','N'),('County','N'),...]

只要语料库包含已标注的文本， NLTK 的语料库接口都将有一个 tagged\_words0方

**法。下面是举例，再次使用布朗语料库所示的输出格式。**

>>>print nltk.corpus.nps\_chat.tagged\_words()

[('now','RB'),('im','PRP'),('left','VBD'),.]

>>>nltk.corpus.conl12000.tagged\_words()

[('Confidence','NN'),('in','IN'),('the','DT'),..]

>>>nltk.corpus.treebank.tagged\_words()

..]

[('Pierre','NNP'),('Vinken','NNP'),(',',','),

并非所有的语料库都采用同一组标记。查看前面提到的标记集的帮助函数和 readmeO 方法中的文档。最初，我们想避免这些标记集的复杂化，所以使用内置映射到简化 的标记集。

>>>nltk.corpus.brown.tagged\_words(simplify\_tags=True)

'),t')ged\_words(simplify\_tags=True)

[('Pierre','NP'),('Vinken','NP'),(',',','),...]

NLTK 中还有其他几种语言的已标注语料库，包括中文、印地语、葡萄牙语、西班 牙语、荷兰语和加泰罗尼亚语。这些通常含有非 ASCⅡ 文本，当输出较大的结构如 列表时，Python 总是以十六进制显示。

>>>nltk.corpus.sinica\_treebank.tagged\_words()

[('\xe4\xb8\x80','Neu¹),('\xe5\x8f\x8b\xe6\x83\x85',‘Nad¹), ..]

>>>nltk.corpus.indian.tagged\_words()

[('\xe0\xa6\xae\xe0\xa6\xb9\xe0\xa6\xbf\xe0\xa6\xb7\xe0\xa7\x87\xe0\xa 6\xb0', 'NN'),

('\xe0\xa6\xb8\xe0\xa6\xa8\xe0\xa7\x8d\xe0\xa6\xa4\xe0\xa6\xbe\xe0\xa6\ xa8','NN'),

...]

>>>nltk.corpus.mac\_morpho.tagged\_words()

[('Jersei','N'),('atinge','V'),('m\xe9dia','N'),...]

>>>nltk.corpus.con112002.tagged\_words()

k),('I'sFpa\_c').tagged\_words()

[('El','da0ms0'),('Tribunal\_Suprem','np0000o'),…]

如果你的环境设置正确，并有适合的编辑器和字体，你应该能够以人可读的方式显

示单个字符串。例如：图5-1显示的是使用 nltk.corpus.indian 访问的数据。

如果语料库也被分割成句子，则利用tagged\_sents() 方法将已标注的词划分成句子， 而不是将它们表示成一个大链表。这对开发自动标注器是有益的，因为它们在句子 链表上进行训练和测试，而不是词链表。

|  |  |
| --- | --- |
| Bangla:市平咐可后/'NN'布TB/'NN'可T呵T8/'NNP'I/'CC'T/'NNP’?/None ,4/'J]’?/None 、研ts/'NN'、可/'J]'本./'NN'x/'NN'可/'VM'/'SYM' Hindi:rr6 H/'NNP'不t/'PREP' x\*/'JJ'H t/'NN'a /'NNPC' FT/'NNP  t/'PREP' /'VFM'NETWTZ/'NN'书/'PREP'X)HY/'NN'市/'PREP'偷可/'PREP¹,T/'NNP'  TKT/'PREP'TTT/'NVB'不t/'VFM' /'VAUX'TTFT/'NN'FT/'PREP' /'NN'  \*HTX/'NN'不I/'PREP’ f/'NN'不1/'PREP' /'NN'市/'PREP’不TW/'PREP'  rH/'JVB'不x/'VFM'x1/'VAUX' /'VAUX'1/'PUNC'  Marathi:ymm/'JJ’俞硬TRH/'NN’可团两可/'NNPC’可府/'NNP'πfI/'PRP'?/None | |
| w^t/'NN’RTTF^t/'NN' /'NN'x\*?/None /'NN'  Telugu:ou 'NN'oo/'PREP'SoyS/'VJJ'S/'NN's/'PREP' | t/'VM'./'SYM'  dr/'NN' |

**图5-1** **4种印度语言的词性标注数据：孟加拉语、印地语、马拉地语和泰卢固语**

**简化的词性标记集**

已标注的语料库使用许多不同的标记集公约来标注词汇。为了帮助我们开始，我们 将学习简化的标记集(表5-1)。

**表5-1** **简化的标记集**

|  |  |  |
| --- | --- | --- |
| **标** **记** | **含** **义** | **例** **子** |
| ADJ | 形容词 | new,good,high,special,big,local |
| ADV | 动词 | really,already,still,early,now |
| CNJ | 连词 | and,or,but,if,while,although |
| DET | 限定词 | the,a,some,most,every,no |
| EX | 存在量词 | there,there's |
| FW | 外来词 | dolce,ersatz,esprit,quo,maitre |
| MOD | 情态动词 | will,can,would,may,must,should |
| N | 名词 | year,home,costs,time,education |
| NP | 专有名词 | Alison,Africa,April,Washington |
| NUM | 数词 | twenty-four,fourth,1991,14:24 |
| PRO | 代词 | he,their,her,its,my,I,us |
| P | 介词 | on,of,at,with,by,into,under |
| TO | 词to | to |
| UH | 感叹词 | ah,bang,ha,whee,hmpf,oops |
| V | 动词 | is,has,get,do,make,see,run |
| VD | 过去式 | said,took,told,made,asked |
| VG | 现在分词 | making,going,playing,working |
| VN | 过去分词 | given,taken,begun,sung |
| WH | W h 限 定 词 | who,which,when,what,where,how |



**让我们来看看这些标记中哪些是布朗语料库的新闻类中最常见的。**

>>>from nltk.corpus import brown

>>>brown\_news\_tagged =brown.tagged\_words(categories='news',simplify\_tags= True)

>>>tag\_fd =nltk.FreqDist(tag for(word, tag) in brown\_news\_tagged)

>>>tag\_fd.keys()

['N','P','DET','NP','V','ADJ',',','.','CNJ','PRO','ADV','VD',...]

**轮到你来：**

使用tag\_fd.plot(cumulative=True)为上面显示的频率分布绘图。上述列

**表中的前五个被标记的词的百分比是多少?**

我们可以使用这些标记进行强有力的搜索，结合图形化的 POS 一致性工具 nltk.app.concordance()。用它来寻找任一词和POS 标记的组合，如：NNNN、hit/VD、 hit/VN 和 the ADJ man。

**名词**

名词一般指的是人、地点、事情和概念，例如女人、苏格兰、图书、情报。名词可

能出现在限定词和形容词之后，可以是动词的主语或宾语，如表5-2所示。

**表5-2** **名词的句法模式**

|  |  |  |
| --- | --- | --- |
| **词** | **限** **定** **词** **之** **后** | **动词的主语** |
| woman | the woman who I saw yesterday … | the woman sat dowr |
| Scotland | the Scotland I remember as a child … | Scotland has five million people |
| book | the book I bought yesterday … | this book recounts the colonization of Australia |
| intelligence | the intelligence displayed by the child … | Mary's intelligence impressed her teachers |

简化的名词标记对普通名词是N, 如书；对专有名词是NP, 如苏格兰。

让我们检查一些已标注的文本，看看哪些词类出现在名词前，频率最高的在最前面。 首先，构建一个双连词链表，它的成员是它们自己的词-标记对，例如：((The’, 'DET),(Fulton',NP)) 和 ((Fulton',NP'),(County',NI)) 。 然后，构建双连词标 记部分的FreqDist。

>>>word\_tag\_pairs =nltk.bigrams(brown\_news\_tagged) >>>list(nltk.FreqDist(a[1]for (a,b)in word\_tag\_pairs

if b[1]=='N'))

['DET','ADJ','N','P','NP','NUM','V','PRO','CNJ','.',',','VG',

'VN' ]

这证实了我们的断言：名词出现在限定词和形容词之后，包括数字形容词(数词， 标注为NUM )。

**动词**

动词是用来描述事件和行动的词，例如： fall 和 eat, 如表5-3所示。在一个句子中， 动词通常表示涉及一个或多个名词短语所指示物的关系。

**表5-3** **动词的句法模式**

|  |  |  |
| --- | --- | --- |
| **词** | **例** **子** | **修饰符与修饰语(斜体字)** |
| fall | Rome fell | Dot com stocks suddenly fell like a stone |
| eat | Mice eat cheese | John ate the pizza with gusto |

新闻文本中最常见的动词是什么?让我们按频率排序所有动词。

>>>wo rd\_tag\_fd =nltk.FreqDist(wsj)

>>>wsj=nltk.corpus.treebank.tagged\_words(simplify\_tags=True)

>>>[word+"/"+tag for (word,tag)in word\_tag\_fdif tag.startswith('V')] ['is/V','said/VD','was/VD','are/V','be/V','has/V','have/V','says/V',

'were/VD','had/VD','been/VN',"'s/V",'do/V','say/V','make/V','did/VD',

'rose/VD','does/V','expected/VN','buy/V','take/V','get/V','sell/V',

'help/V','added/VD','including/VG','according/VG','made/VN','pay/V',..]

请注意，频率分布中计算的项目是词-标记对。由于词汇和标记是成对的，我们可 以把词作为条件，标记作为事件，使用条件-事件对的链表初始化一个条件频率分 布。这样我们可以看到一个给定词的标记的频率顺序列表。

>>>cfdl =nltk.ConditionalFreqDist(wsj)

>>>cfd1['yield'].keys()

['V','N']

>>>cfd1['cut'].keys()

['V','VD','N','VN']

可以颠倒配对的顺序，以标记作为条件，词汇作为事件。现在我们可以看到对于给

定标记的可能词。

>>>cfd2 =nltk.ConditionalFreqDist((tag,word)for(word,tag)in wsj)

>>>cfd2['VN'].keys()



['been','expected','made','compared','based','priced','used','sold',

'named','designed','held','fined','taken','paid','traded','said',..]

要弄清 VD (过去式)和 VN (过去分词)之间的区别，让我们找到可以同是 VD 和 VN 的词汇，看看它们周围的文字的情况。

>>>[w for w in cfdl.conditions()if 'VD'in cfdl[w]and 'VN'in cfd1[w]]

['Asked','accelerated','accepted','accused','acquired','added',

'ado.index(('kicked', 'vD'))

>>>wsj[idx1-4:idx1+1]

('''e,'Pd')'prog']'),('trades','N'),('swiftly','ADV'),

>>>idx2 =wsj.index(('kicked','vN'))

>>>wsj[idx2-4:idx2+1]

[('head','N'),('of','P'),('state','N'),('has','V'),('kicked','VN')]

在这种情况下，我们看到过去分词kicked 前面是助动词 have。这是普遍真实的吗?

**轮到你来：**

通过 cfd2['VN'].keys()指定一个过去分词的链表，尝试收集所有处在链 表项目前面的词-标记对。

**形容词和副词**

另外两个重要的词类是**形容词和副词。**形容词修饰名词，可以作为修饰符(如：the large pizza 中的 large) 或谓语(如：the pizza is large)。英语形容词可以有内部结构 (如：the falling stocks 中的 fall+ing)。副词修饰动词，指定时间、方式、地点或动词 描述的事件发展方向(如： the stocks fell quickly 中的quickly)。副词也可以修饰形容 词(如： Mary's teacher was really nice 中的really)。

英语中还有几个封闭的词类，如介词、冠词(也常称为**限定词，**如： the、a)、情 **态动词** (如： should 、may) 、人称代词(如： she 、they) 。每个词典和语法对这些 词的分类都不同。

**轮到你来：**

如果你对这些词性中的一些还不是确定，使用 nltk.app.concordance) 学习它们，或看 YouTube 上《School-house Rock!》语法视频，或查询 本书第5.9节。

**未简化的标记**

让我们找出每个名词类型中最频繁的名词。例5-1 中的程序找出了所有以NN 开始 的标记，并为每个标记提供了几个示例词汇。名词有的变化形式，最重要的含有$ 的名词所有格，含有S 的复数名词(因为复数名词通常以s 结尾),以及含有P 的 专有名词。此外，大多数的标记都有后缀修饰符： -NC 表示引用， -HL表示标题中 的词， -TL 表示标题(布朗标记的特征)。

**例5-1** **找出最频繁的名词标记的程序**

def findtags(tag\_prefix,tagged\_text):

cfd=nltk.ConditionalFreqDist((tag,word)for(word,tag)in tagged\_text if tag.startswith(tag\_prefix))

return dict((tag,cfd[tag].keys()[:5])for tag in cfd.conditions()) >>>tagdict =findtags('NN',nltk.corpus.brown.tagged\_words(categories='news')) >>>for tag in sorted(tagdict):

 print tag,tagdict[tag]

N ['year','time','state','week','man']

NN§["year's","world's","state's","nation's","company's"]

NNS-HL["Golf's","Navy's"]

NNS-TL["President's","University's","League's","Gallery's","Army's"] NN-HL['cut','Salary','condition','Question','business']

NN-NC['eva','ova','aya']

NN-TL['President','House','State','University','City']

NN-TL-HL['Fort','City','Commissioner','Grove','House']

NNS ['years','members','people','sales','men']

NNS§["children's","women's","men's","janitors'","taxpayers'"] NNSS-HL ["Dealers'","Idols'"]

NNSS-TL["Women's","States'","Giants'","Officers'","Bombers'"] NNS-HL['years','idols','Creations','thanks','centers']

NNS-TL['States','Nations','Masters','Rules','Communists']

NNS-TL-HL['Nations']

在本章后续部分创建词性标注器时，我们将使用这些未简化的标记。

**探索已标注的语料库**

简要地回顾我们在前面的章节中学习的语料库，接下来我们要研究 POS 标记。 假设我们正在研究词 often, 想看看它是如何在文本中使用的。我们可以试着观察

跟在 often 后面的词汇。

>>>brown\_learned\_text =brown.words(categories='learned')

>>>sorted(set(b for(a,b)in nltk.ibigrams(brown\_learned\_text)if a ==

'often'))

**[',','.',** 'accomplished','analytically','appear','apt','associated',

'assuming',

'became','become','been','began','call','called','carefully','chose', ...]

然后，使用**tagged\_words(**)方法查看跟随词的词性标记可能更有指导性。

>>>brown\_lrnd\_tagged =brown.tagged words(categories='learned',simplify\_ tags=True)

>>>tags =[b[1]for(a,b)in nltk.ibigrams(brown\_lrnd\_tagged)if a[0]== 'often']

>>>fd =nltk.FreqDist(tags)

>>>fd.tabulate()

VN V VD DET ADJ ADV P CNJ , TO VG WH VBZ

1512 8 5 5 4 4 3 3 1 1 1 1 1

请注意 often 后面最高频率的词性是动词。名词从来没有在这个位置出现(在这个 特定的语料中)。

接下来，在较大范围的上下文中找出涉及特定标记和词序列的词(在这种情况 下，“<Verb> 到<Verb>”) 。 在例5-2中，考虑句子中的每个三词窗口°,检查它 们是否符合我们的标准”。如果标记匹配，将输出对应的词③。

例5-2 使用 POS 标记寻找三词短语

from nltk.corpus import brown def process(sentence):

for(wl,tl),(w2,t2),(w3,t3)in

if(t1.startswith('V')and

print w1,w2,w3③

nltk.trigrams(sentence):①

t2 =='TO'and t3.startswith('V')):②

>>>for

tagged\_sent in

process(tagged\_sent)

brown.tagged\_sents():

… ·

·

.

combined to achieve

continue to place

serve to protect

wanted to wait

allowed to place

expected to become



最后，让我们看看与它们的标记关系高度模糊不清的词。要了解为什么要标注这样 的词，是因为它们各自的上下文可以帮助我们弄清楚标记之间的区别。

>>>brown\_news\_tagged =brown.tagged\_words (categories='news',simplify\_

tags=True) >>>data

>>>for

… ·

=nltk.ConditionalFreqDist((word.lower(),tag)

for(word,tag)in brown\_news\_tagged) word in data.conditions():

if len(data[word])>3:

·

..



tags =data[word].keys()

print word,''.join(tags)



best ADJ ADV NP V

better ADJ ADVVDET

close ADV ADJ V N

cut V N VN VD

even ADV DET ADJ V

grant NP N V-

hit VVD VNN

lay ADJ V NP VD

left VD ADJ N VN

like CNJ V ADJ P -

near P ADV ADJ DET

open ADJ VN ADV

past N ADJ DET P

present ADJ ADV V N

read V VN VD NP

right ADJ NDET ADV

second NUMADV DET N

set VN V VD N -

that CNJ V WH DET

**轮到你来：**

打开 POS 一致性工具 **nltk.app.concordance(O**和加载完整的布朗语料 库(简化标记集)。现在挑选一些上面代码例子末尾处列出的词，看 看词的标记如何与词的上下文相关。例如：搜索 near会看到所有混合 在一起的形式，搜索 near/ADJ会看到它作为形容词使用，near N会看 到只是名词跟在后面的情况，等等。

**5.3** **使用** **Python** **字典映射词及其属性**

正如我们已经看到，已标注词(word,tag) 的形式表示词和词性标记的关联。 一

旦开始进行词性标注，需要创建一个将标记分配给词的程序，这是在给定上下文 中最可能出现的标记。我们可以认为这个过程是从词到标记的**映射**。在 Python 中存储映射最自然的方式是使用所谓的**字典**数据类型(在其他的编程语言又称为关 **联数组或哈希数组)**。在本节中，我们学习字典是如何表示包括词性在内的各种不 同语言信息的。

**索引链表** **VS** **字典**

我们已经看到，文本在 Python 中被视为词链表。链表的一个重要的属性是可 以通过给出其索引来“查找”特定项目，例如： text1[100] 。 请注意我们如何 指定数字，然后取回词。我们可以把链表认为是一个简单的表格，如图5-2 所示。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 0 | Call | | 1 | me | | 2 | Ishmael | | 3 |  | |

**图5-2** **链表查找：在整数索引的基础上，访问Python 链表的内容**

将上述情况与频率分布进行对比，在1.3节我们先指定词然后取回数字，如： fdist['monstrous'], 表示的是给定的词在文本中出现的次数。用词查询类似于使用一 本字典。其他一些例子如图5-3所示。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Domain Name Resolution Word Frequency Table  Phone List   |  |  | | --- | --- | | Alex | x154 | | Dana | x642 | | Kim | x911 | | Les | x120 | | Sandy | x124 |  |  |  | | --- | --- | | computational | 25 | | language | 196 | | linguistics | 17 | | natural | 56 | | processing | 57 |  |  |  | | --- | --- | | aclweb.org | <128.231.23.4> | | amazon.Com | <12.118.92.43> | | google.com | <28.31.23.124> | | python.org | <18.21.3.144> | | sourceforge.net | <51.98.23.53> | | |

**图5-3** **字典查询：使用一个关键字，如某人的名字、一个域名或一个英文单词，访问一个字典** **的条目；映射(map)、哈希表(hashmap)、哈希** **(hash)、关联数组(associative array)是** **字典的其他名字**

在电话簿中，我们用名字查找条目然后得到电话号码。在浏览器中输入一个域名， 计算机将查找到一个 IP 地址。 一个词频表可以查找出一个词在一个文本集合中的

使用频率。在所有这些情况中，都是从名称映射到数字，而不是其他如链表那样的 方式。在一般情况下，我们希望能够在任意类型的信息之间映射。表5-4列出了各 种语言学对象及它们的映射。

**表5-4** **语言学对象从键到值的映射**

|  |  |  |
| --- | --- | --- |
| **语言学对象** | **映** **射** **来** **自** | **映** **射** **到** |
| 文档索引 | 词 | 页面列表(找到词的地方) |
| 同义词 | 词意 | 同义词列表 |
| 词典 | 中心词 | 词条项(词性、意思定义、词源) |
| 比较单词列表 | 注释术语 | 同源词(词列表，每种语言一个) |
| 词形分析 | 表面形式 | 形态学分析(词素组件列表) |

大多数情况下，都是从一个“词”映射到一些结构化对象。例如： 一个文档索引从 一个词(可以表示为一个字符串)映射到页面列表(表示为一个整数列表)。本节 中，我们将学习如何在Python 中表示这些映射。

**Python 字典**

Python 提供了一个字典数据类型，可用来做任意类型之间的映射。它更像是一个传 统的字典，以一种高效的方式来查找事物。然而，正如我们从表5-4看到的，它具 有更广泛的用途。

为了说明这一点，定义 **pos** 为空字典，然后给它添加4个项目，指定一些词的词性。 使用熟悉的方括号将条目添加到字典。

>>>pos ={}

>>>pos

{>}>>pos['colorless']='ADJ'①

>>>pos

{'colorless':'ADJ'}

as']o'ADV'

>>>pos②

{'furiously':'ADV','ideas':'N','colorless':'ADJ','sleep':'V'}

例子中，①说的是colorless 的词性是形容词，或者更具体地说，在字典 pos 中，键



'colorless'被指定了值!ADJ 。 当检查pos 的值②时，我们得到的是一个键-值对集合。 一旦我们在字典中确认使用这样的方式，就可以使用按键检索值。

>>>pos['ideas']

'N'

>>>pos['colorless']

'ADJ'

当然，也可能会无意中使用一个尚未分配值的键。

>>>pos['green']

Traceback(most recent call last):

File"<stdin>",line 1,in ?

KeyError:'green'

这就提出了一个重要的问题，与链表和字符串中使用len() 算出哪些整数是合法索引 不同，如何算出一个字典的合法键?如果字典不是太大，我们可以简单地通过评估 变量pos 检查它的内容。正如在前面②行中所看到的，它为我们提供了键-值对。 请注意它们的顺序与最初放入它们的顺序不同，这是因为字典不是序列而是映射 (见图5-3),它的键并不按固有的顺序排列。

另外，要找到键，我们可以将字典转换成一个链表°或在需要使用链表的地方使用

字典，如作为 sortedO的参数°或用在 for 循环中。

>>>list(pos)①

['ideas', 'furiously','colorless','sleep']

esds('p,'②iously', 'ideas','sleep']

>>>[w for w in pos if w.endswith('s')]③

['colorless','ideas']

当输入 list(pos)时，你看到的可能会与这里显示的顺序不同。如果你 想看到有序的键，只需要对它们进行排序。

同时，与使用for 循环遍历字典中的所有键一样，我们可以使用for 循环输出链表。

>>>for word in sorted(pos):

print word +":",pos[word]

■

. .

colorless:ADJ

furiously:ADV

sleep:V

ideas:N

最后，利用字典方法keysO 、values()和items()可以访问作为单独链表的键、值及 键-值对。我们甚至可以按它们的第一个元素排序元组(如果第一个元素相同，就 使用它们的第二个元素)。

>>>pos.keys()

['colorless','furiously','sleep','ideas']

>>>pos.values()

['ADJ','ADV','V','N']

>>>pos.items()

[('colorless','ADJ'),('furiously','ADV'),('sleep','V'),('ideas','N')]

>>>for key,val in sorted(pos.items()):①

print key +":",val



colorless:ADJ

furiously:ADV

ideas:N

sleep:V

要确保在字典中查找某词时， 一个键只得到一个值。现在假设我们试图用字典来存 储既是动词又是名词的词 sleep。

>>>pos['sleep']='V'

>>>pos['sleep']



>>>pos['sleep']

'N'

一开始，pos[sleep]给的值是V'。但是，它立即被一个新值N'覆盖了。换句话说， 字典中只能有一个'sleep'条目。然而，有一个方法可以在该项目中存储多个值：使 用链表值，例如：pos['sleep']=[N','V] 。实际上，这就是我们在2.4节中看到的 CM U 发音字典，它为一个词存储多个发音。

**定义字典**

可以使用键-值对格式创建字典。共有两种方式，我们通常使用第一种。

>>>pos={'colorless':'ADJ','ideas':'N','sleep':'V','furiously':'ADV'}

>>>pos =dict(colorless='ADJ',ideas='N',sleep='V',furiously='ADV')



请注意：字典的键必须是不可改变的类型，如字符串和元组。如果我们尝试使用可 变键定义字典会得到 TypeError。

>>>pos={['ideas','blogs','adventures']:'N')

Traceback(most recent call last):

File "<stdin>",line 1,in <module>

TypeError:list objects are unhashable

**默认字典**

如果我们试图访问一个不在字典中的键，会得到错误。然而，如果一个字典能为这 个新键自动创建一个条目并给它一个默认值，如0或者一个空链表，这将是有用的。 自Python2.5 以来，出现了一种特殊的称为 defaultdict的字典。(考虑到有读者使用 Python2.4,NLTK 提供 了nltk.defaultdict。) 为了使用它，我们必须提供一个参数， 用来创建默认值，如： int 、float 、str、list 、dict 、tuple。

>>>frequency =nltk.defaultdict(int)

>>>frequency['colorless']=4

>>>frequency['ideas']

 =nltk.defaultdict(list)

>>>pos['sleep']=['N','V']

>>>pos['ideas']

[]

这些默认值实际的功能是将其他对象转换为指定类型的函数(例如： int("2") 、list("2"))。当它们被调用的时候没有参数，也就是说， int)、 list(), 分别返回0和[]。

前面的例子中指定字典项的默认值为一个特定的数据类型的默认值。然而，也可 以指定任何我们喜欢的默认值，只要提供可以无参数的被调用产生所需值的函数的 名字。观察下面词性的例子，创建任一条目的默认值是N'的字典°。当我们访问 一个不存在的条目②时，它会自动添加到字典⑤。

['(lambda:'N')①

>>>pos['blog']②

'N'

>>>pos.items()

[('blog',‘N'),('colorless','ADJ')]③



这个例子使用的是lambda 表达式，在4.4节介绍过。这个 lambda 表 达式没有指定参数，所以要用不带参数的括号调用它。因此，下面的 f 和 g 的定义是等价的。

=lambda:

>>>f

'N'

>>>f()

'N'

g():

>>>def

..return

'N!

>>>g()

'N'

下面来学习默认字典如何被应用在较大规模的语言处理任务中。许多语言处理任 务，包括标注，费很大力气来正确处理文本中只出现过一次的词。当有固定的词 汇并且不会有新词出现时，可能处理效果会更好。可以在默认字典下预处理文本， 并使用特殊的“超出词汇表”标识符， UNK(out of vocabulary)替换低频词汇。(你 能不参照下面的例子想出处理的方法吗?)

我们需要创建一个默认字典，映射其中的每个词为它们的替换词。最频繁的n 个词 将被映射回自身。其他的被映射到UNK。

>>>alice =nltk.corpus.gutenberg.words('carroll-alice.txt') >>>vocab =nltk.FreqDist(alice)

ping sttvc0t](lambda:'UNK')

>>>for v in v1000:

mapping[v]=v

>>>alice2 =[mapping[v]for v in alice]

>>>alice2[:100]

['UNK','Alice',"'",'s','Adventures','in','Wonderland','by','UNK','UNK', 'UNK','UNK','CHAPTER','I','.','UNK','the','Rabbit','-','UNK','Alice',

'was','beginning','to','get','very','tired','of','sitting','by','her',

'sister','on','the','bank',',','and','of','having','nothing','to','do',

':','once','or','twice','she','had','UNK','into','the','book','her', 'sister','was','UNK',',','but','it','had','no','pictures','or','UNK',

'in','it',',',"'",'and','what','is','the','use','of','a','book',",'",

'thought','Alice',"'",'without','pictures','or','conversation',"?'",..]

>>>len(set(alice2))

1001

**递增地更新字典**

可以使用字典计数出现的次数，模拟如图1-3所示的计数词汇的方法。首先初始化一

个空的 defaultdict, 然后处理文本中每个词性标记。如果以前没有见过该标记，就默认 计数为零。每遇到一个标记，就使用+=运算符递增它的计数(见例5-3)。

**例5-3** **递增地更新字典，按值排序**

>>>counts =nltk.defaultdict(int)

>>>from nltk.corpus import brown

t,s[=i1n brown.tagged\_words(categories='news'):

\*

·

>>>counts['N']

22226

>>>list(counts)

['FW','DET','WH',"''",'VBZ','VB+PPO',"'",']','ADJ','PRO','\*','-',..]

>>>from operator import itemgetter

>>>sorted(counts.items(),key=itemgetter(1),reverse=True)

[('N',22226),('P',10845),('DET',10648),('NP',8336),('V',7313),..]

>>>[t for t,c in sorted(counts.items(),key=itemgetter(1),reverse=True)] ['N','P','DET','NP','V','ADJ',',','.','CNJ','PRO','ADV','VD',...]

例5-3的列表演示了字典按值排序的习惯用法，按频率递减顺序显示词汇。sortedO 的第一个参数是要排序的项目，它是由一个 POS 标记和一个频率组成的元组链 表。第二个参数使用函数 itemgetter()指定排序键。 一般情况下， itemgetter(n) 返 回一个函数，这个函数可以在一些其他序列对象上被调用以获得该序列的第 n 个元素。

>>>pair =('NP',8336)

>>>pair[1]

8336

>>>itemgetter(1)(pair)

8336

sorted) 最后一个参数的指定项目应以相反的顺序返回，即频率值递减。

在例5-3的开头还有另一个有用的习惯用法，初始化 defaultdict, 然后使用 for 循环 来更新其值。下面是该用法的举例。

>>>my\_dictionary =nltk.defaultdict(function to create default value) >>>for item in sequence:

my\_dictionary[item\_key]is updated with information about item



**下面是这种模式的另一个实例，通过最后两个字母索引词汇。**

>>>last\_letters =nltk.defaultdict(list)

>>>words =nltk.corpus.words.words('en')

>>>for word in words:

· lkaesyt\_l[k-].append(word)

>>>last\_letters['ly']

['abactinally','abandonedly','abasedly', 'abashedly', 'abashlessly', 'abbreviately',

'abdominally','abhorrently','abidingly','abiogenetically','abiologically', ... 

['blazy','bleezy','blowzy','boozy','breezy','bronzy','buzzy','Chazy',

...]

下面的例子中使用相同的模式创建颠倒顺序的词字典。(尝试运行第3行，以便能 弄清楚为什么这个程序能运行。)

>>>anagrams =nltk.defaultdict(list)

>>>for w '.orted(word))

anagrams[key].append(word)

['entrail','latrine','ratline','reliant','retinal','trenail']

>>>anagrams['aeilnrt']

由于积累这类词是一种常用的任务，因为NLTK 以nltk.Index()形式提供了一种创建 defaultdict(list)更方便的方式。

>>>anagrams =nltk.Index((''.join(sorted(w)),w)for w

>>>anagrams['aeilnrt']

['entrail','latrine','ratline','reliant','retinal','trenail']

in words)

nltk.Index 是额外支持初始化的 defaultdict(list)。 类似的， nltk. FreqDist 本质上是额外支持初始化的 defaultdict(int) (附带排序和 绘图方法)。

**复杂的键和值**

我们可以使用具有复杂的键和值的默认字典。研究一个词的可能标记范围，并对定

**词本身和前一个词进行标记。在下例中，我们将看到这些信息如何被** **POS 标注器使用。**

>>>pos =nltk.defaultdict(lambda:nltk.defaultdict(int))

>>>brown\_news\_tagged =brown.tagged\_words(categories='news',simplify\_ tags=True)

>>>for((w1,t1),(w2,t2))in nltk.ibigrams(brown\_news\_tagged):①

pos[(t1,w2)][t2]+=1②



defaultdict(<type'int'>, {'ADV':3,'ADJ':9,'N':3})

这个例子使用的字典，它的条目的默认值也是一个字典(其默认值是int(),即0)。 请注意例子中如何遍历已标注语料库的双连词，即每次遍历都处理词-标记对°。 每次循环次时，更新字典 pos 中的条目(t1,w2)、 标记和它后面的词°。当在 pos 中找到项目时，必须指定一个复合键，然后得到一个字典对象。POS 标注器可以 使用这些信息来决定词 right, 当前面是限定词时，应标注为ADJ。

**颠倒字典**

字典支持高级查找，可获得任意键的值。如果 d 是一个字典，k 是一个键，输入d[K],

可立即获得值。如果给定一个值查找对应的键，要慢一些和麻烦一些。

>>>counts =nltk.defaultdict(int)

>>>for word in nltk.corpus.gutenberg.words('milton-paradise.txt'): counts[word]+=1

 ['brought','Him','virtue','Against','There','thine','King','mortal',

'every','been']

如果要经常进行“反向查找”,可建立一个映射值到键的字典。在任意两个键都不 具有相同值的情况下，只要得到字典中的所有键-值对，并创建新的值-键对字典即 可。下面的例子演示了另一种用键-值对初始化字典 pos 的方式。

>>>pos={'colorless':'ADJ','ideas':'N','sleep':'V','furiously':'ADV'} >>>pos2 =dict((value,key)for(key,value)in pos.items())



首先将词性字典做得更实用些，使用字典的 update()方法在pos 中加入一些词，创

建多个键具有相同值的情况。这样一来，刚才介绍的反向查找技术就将不再起作用 (为什么?)。取而代之的是使用append()为每个词性积累词，如下例所示。

>>>pos.update({'cats':'N','scratch':'V','peacefully':'ADV','old':

'AD>pos2 =nltk.defaultdict(list)

>>>for key,value in pos.items():

· pos2[value].append(key)

·

>>>pos2['ADV']

['peacefully','furiously']

现在，我们已经颠倒了字典pos, 可以查任意词性并且找到所有具有此词性的词。 还可以使用 NLTK 中的索引支持进行相同的操作，如下例所示。

>>>pos2 =nltk.Index((value,key)for(key,value)in pos.items())

>>>pos2['ADV']

['peacefully','furiously']

表5-5给出 Python 字典方法的总结。

**表5-5** **Python字典方法：常用的方法与字典相关习惯用法的总结**

|  |  |
| --- | --- |
| **示** **例** | **说** **明** |
| d={} | 创建一个空的字典，并将分配给d |
| d[key]=value | 分配一个值给一个给定的字典键 |
| d.keys() | 字典的键的链表 |
| list(d) | 字典的键的链表 |
| sorted(d | 字典的键，排序 |
| key in d | 测试一个特定的键是否在字典中 |
| for key in d | 遍历字典的键 |
| d.values() | 字典中的值的链表 |
| dict([(k1,v1),(k2,v2),…]) | 从一个键-值对链表创建一个字典 |
| d1.update(d2) | 添加d2中所有项目到d1 |
| defaultdict(int) | 一个默认值为0的字典 |

**5.4** **自动标注**

本章的剩余部分将探讨以不同的方式给文本自动添加词性标记。词的标记依赖

于这个词和它在句子中的上下文。出于这个原因，我们将在处理(已标注)句 子层面上的数据而不是词汇层面的。从加载将要使用的数据开始。

nn\_lpdus\_sergged\_sents(categories='news')

>>>brown\_sents =brown.sents(categories='news')

**默认标注器**

最简单的标注器是为每个标识符分配同样的标记。这似乎是一个相当普通的方 法，但为标注器的性能建立了一个重要的标准。为了得到最好的效果，我们用最 有可能的标记标注每个词。通过下例找出哪个标记是最有可能的(现在使用未简 化标记集):

>>>tags =[tag for(word,tag)in brown.tagged\_words(categories='news')] >>>nltk.FreqDist(tags).max()

'NN'

现在我们可以创建一个将所有词都标注成NN 的标注器。

>>>raw='I do not like green eggs and ham,I do not like them Sam I am!' >>>tokens =nltk.word\_tokenize(raw)

>>>default\_tagger =nltk.DefaultTagger('NN')

>>>default\_tagger.tag(tokens)

[('I','NN¹),('do','NN'),('not','NN'),('like','NN'),('green','NN'),

('eggs','NN'),('and','NN'),('ham','NN'),(',','NN'),('I','NN'),

('do','NN¹),('not','NN'),('like','NN'),('them','NN'),('Sam','NN'),

('I','NN'), ('am','NN'),('!', 'NN')]

不出所料，这种方法的表现相当不好。在一个典型的语料库中，它只正确标注了八 分之一的标识符，如下例所示。

>>>default\_tagger.evaluate(brown\_tagged\_sents)

0.13089484257215028

默认的标注器给每个单独的词分配标记，即使是之前从未遇到过的词。碰巧的是， 在处理几千词的英文文本时发现，大多数新词都将是名词。这意味着，默认标注器 可以帮助我们提高语言处理系统的稳定性。



**正则表达式标注器**

正则表达式标注器基于匹配模式分配标记给标识符。例如： 一般情况下认为任一以 ed 结尾的词都是动词过去分词，任一以s 结尾的词都是名词所有格。下例中可以用 正则表达式的列表来表示这些。

>>>patterns =[

(r'.\*ingS','VBG'),

|  |  |
| --- | --- |
| …  ·  .  . .  … ·  . ·  ...] | (r'.\*edS', 'VBD'), (r'.\*esS','VBZ'),  (r'.\*oulds','MD'), (r'.\*\'sS','NNS'),  (r'.\*sS','NNS'),  (r¹^-?[0-9]+(.[0-9]+)?$','CD'), (r'.\*', 'NN') |

#gerunds

#simple past

#3rd singular present

#modals

#possessive nouns

#plural nouns

#cardinal numbers

#nouns(default)

请注意，这些是按顺序处理的，第一个匹配上的会被使用。现在建立一个标注器， 并用它来标记句子。完成这步会有约五分之一是正确的。

>>>regexp\_tagger =nltk.RegexpTagger(patterns)

g,\_(ga'('o)w,(()'relative','NN'),('handful',

'NN'),

('of','NN'), ('such','NN'),('reports', 'NNS'), ('was','NNS'), ('received','VBD'),

("'",'NN'),(',','NN'),('the','NN'),('jury','NN'),('said','NN'),

(',','NN'),

('“','NN'),('considering','VBG'),('the','NN'),('widespread',

'NN').rexp\_tagger.evaluate(brown\_tagged\_sents)

0.20326391789486245

最终的正则表达式《.\*》是全面的，可标注所有词为名词。除了作为正则表达式 标注器的一部分重新指定这个，这与默认标注器是等效的(只是效率低得多)。 有没有办法将这个标注器和默认标注器结合在一起呢?在下面的内容中，我们 将介绍这一点。

**轮到你来：**

看看你能不能想出一些模式，提高上面所示的正则表达式标注器的性 能。(请注意：6.1节介绍了能够部分自动化这类工作的方法。)

**查询标注器**

很多高频词没有NN 标记。找出100个最频繁的词，存储它们最有可能的标记。 然后我们可以使用这个信息作为“查找标注器”(NLTK UnigramTagger) 的模型， 如下例。

>>>fd =nltk.FreqDist(brown.words(categories='news'))

>>>cfd =nltk.ConditionalFreqDist(brown.tagged\_words(categories='news'))

tl\_yfr\_((wax())for word in most\_freq\_words)

>>>baseline\_tagger =nltk.UnigramTagger(model=likely\_tags)

>>>baseline\_tagger.evaluate(brown\_tagged\_sents)

0.45578495136941344

现在可以确定的是仅仅知道100个最频繁的词的标记就能正确标注很大一部分标 识符(事实上是近一半)。让我们来看看它在一些未标注的输入文本上是如何运 行的。

>>>sent =brown.sents(categories='news')[3]

tlayg'','AT'),('relative',None),

('handful',None),('of','IN'),('such',None),('reports',None),

('was','BEDZ'),('received',None),("'","''"),(',',','),

('the','AT'),('jury',None),('said','VBD'),(',',','),

(' ','`'),('considering',None),('the','AT'),('widespread',None), ('interest',None), ('in','IN'),('the','AT'),('election',None),

'er',NoI,C'),('the','AT'),('size',None),

('of','IN'),('this','DT'),('city',None),("'","''"),('.','.')]

许多词都被分配了None 标签，因为它们不在100个最频繁的词之中。在这些情 况下，我们想分配默认标记NN。换句话说，我们要先使用查找表，如果它不能 指定标记就使用默认标注器，这个过程叫做回退 (见5.5节)。如下例所示，通 过指定一个标注器作为另一个标注器的参数，从而可以完成该操作。现在查找 标注器将只存储名词以外的词-标记对，只要它不能给词分配标记，它将会调用 默认标注器。

>>>baseline\_tagger =nltk.UnigramTagger(model=likely\_tags,

backoff=nltk.DefaultTagger('NN'))

把所有这些放在一起，编写程序来创建和评估具有一定范围的查找标注器(如

Performance

Lookup Tagger Performance with Varying Model Size

例5-4)。

**例5-4 查找标注器的性能，使用不同大小的模型**

def lt =pdei,[oorax())for word in wordlist)

baseline\_tagger=nltk.UnigramTagger(model=lt,backoff=nltk.Default Tagger('NN'))

return baseline\_tagger.evaluate(brown.tagged\_sents(categories='news'))

def display():

import pylab

words\_by\_freq=list(nltk.FreqDist(brown.words(categories='news')))

cfd =nltk.ConditionalFreqDist(brown.tagged\_words(categories='news')) sizes=2 \*\*pylab.arange(15)

perfs =[performance(cfd,words\_by\_freq[:size])for size in sizes]

pylab.plot(sizes,perfs,'-bo')

pylab.title('Lookup Tagger Performance with Varying Model Size') pylab.xlabel('Model Size')

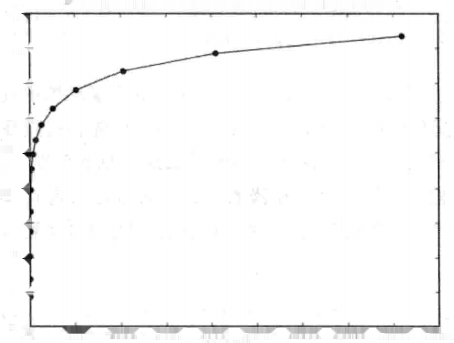
pylab.ylabel('Performance')

pylab.show()

>>>display()

观察图5-4,随着模型规模的增长，最初性能增加迅速，最终达到一个稳定水平，

这时模型规模大量增加，但性能的提高幅度却很小。(该例子使用的是 pylab 绘图 软件包，在4.8节讨论过)



1.0

0.9

0.8|

0.7|

0.6

0.5

0.4

0.3

0.2|

0.10

Model

2000 4000 6000

12000 14000 16000

10000

Size

18000

8000

**图5-4** **查找标注器**



**评估**

在前面的例子中，你会注意到对准确性得分的强调。事实上，对工具的性能评估是 NLP的一个中心主题。回想图1-5的处理流程； 一个模块输出中的任何错误在下游 模块都被无限地放大了。

对比专家分配的标记评估标注器的性能。由于通常很难获得专业和公正的判断， 所以使用**黄金标准**测试数据。这是一个手动标注并作为自动系统评估标准而被 接受的语料库。当给定词猜测的标记与黄金标准标记相同，标注器被视为是正 确的。

当然，设计和实施原始黄金标准标注的是人，更深入地分析可能会产生黄金标准中 的错误，或者会生成修正的标记集和更复杂的指导方针。然而，黄金标准就目前有 关的自动标注器的评估而言被定义成是“正确的”。

**轮到你来：**

开发已标注语料库是一个重大的任务。为确保高品质的标注，除了数据， 它会产生复杂的工具、文档和实践。标记集和其他编码方案不可避免地 依赖于一些理论主张，不是所有的理论主张都被共享。然而，语料库的 创作者往往竭尽全力使他们的工作尽可能中立，以最大限度地提高其工 作的有效性。我们将在第11章讨论创建语料库的问题。

**5.5** **N-gram 标注**

**一元标注器** **(Unigram** **Tagging)**

一元标注器利用一种简单的统计算法，对每个标识符分配最有可能的标记。例如： 它将分配标记J 给词 frequent, 因为frequent 用作形容词(例如： a frequent word) 的情况比用作动词(例如： I frequent this cafe) 更常见。 一元标注器的行为与查找 标注器相似(见5.4节),建立一元标注器的技术，称为**训练。**在下面的代码例子 中，“训练”一个一元标注器，用它来标注一个句子，然后进行评估。

nn\_lpdus\_sergged\_sents(categories='news')

>>>brown\_sents =brown.sents(categories='news')

>>>unigram\_tagger =nltk.UnigramTagger(brown\_tagged\_sents)

>>>unigram\_tagger.tag(brown\_sents[2007])

**)**,,**(**'N')()''A),(T'a''),('type','NN'),

(',',','),('being','BEG'),('on','IN'),('the','AT'),('ground','NN'),

('floor','NN'),('so','QL'),('that','CS'),('entrance','NN'),('is','BEZ'),

('direct', 'JJ'),('.','.')]

>>>unigram\_tagger.evaluate(brown\_tagged\_sents)

0.9349006503968017

通过在初始化标注器时指定已标注的句子数据作为参数来训练一元标注器。训练过 程中涉及检查每个词的标记，将所有词的最可能标记存储在一个字典里面，这个字 典存储在标注器内部。

**分离训练和测试数据**

现在，可在一些数据上训练标注器，必须注意不要在相同的数据上测试，如在前 面的例子中的那样。如果一个标注器只是记忆它的训练数据，而不试图建立一般 的模型，测试结果会更好，但在标注新文本时不起作用。相反，我们应该分割数据， 90%为训练数据，其余10%为测试数据。

>>>size =int(len(brown\_tagged\_sents)\*0.9)

>>>size

>train\_sents =brown\_tagged\_sents[:size]

>>>test\_sents =brown\_tagged\_sents[size:]

>>>unigram\_tagger =nltk.UnigramTagger(train\_sents)

>>>unigram\_tagger.evaluate(test\_sents)

0.81202033290142528

虽然得分更糟糕了，但是对这种标注器是无用的情况(如：它在前所未见的文本上 的性能)有了更好的了解。

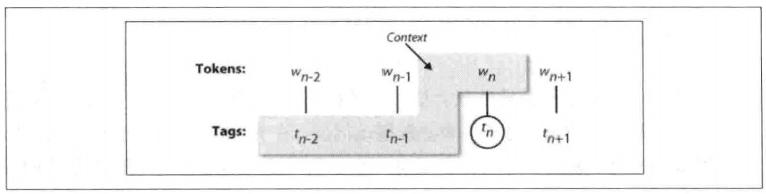
**一般的** **N-gram 的标注**

当基于 **unigrams** 处理语言处理任务时，可使用上下文中的项目。标注时，只考虑 当前的标识符，而不考虑其他上下文。给定一个模型，最好是为每个词标注其先验 的最可能的标记。这意味着将使用相同的标记标注词，如 wind, 不论它出现的上 下文是 the wind 还是 to wind。

**n-gram** 标注器是 unigram 标注器的一般化，它的上下文是当前词和它前面 n-1



个标识符的词性标记，如图5-5所示。要选择的是圆圈里的tn, 以及被标灰色阴 影的上下文。在图5-5所示的 n-gram标注器的例子中，令n=3, 也就是说，考 虑当前词的前两个词的标记。 n-gram 标注器将挑选在给定上下文中最有可能的 标记。



**图5-5** **标注器上下文**

1-gram 标注器是一元标注器(unigram tagger) 的另一个名称：即用 于标注上下文是标识符本身的标识符。2-gram 标注器也称为二元标 注器 (bigram taggers ),3-gram 标注器也称为三元标注器 (trigram taggers).

NgramTagger 类使用一个已标注的训练语料库来确定每个上下文中哪个词性标记 最有可能。下面的列子中，我们看到 n-gram 标注器的一个特殊情况，即 bigram 标 注器。首先，训练它，然后用它来标注未标注的句子。

>>>bigram\_tagger =nltk.BigramTagger(train\_sents)

>>>bigram\_tagger.tag(brown\_sents[2007])

[('Various', 'JJ'),('of','IN'), ('the','AT'),('apartments','NNS'),

('are','BER'),('of','IN'),('the','AT'),('terrace','NN'),

('type','NN'),(',',','),('being','BEG'),('on','IN'),('the','AT'),

('ground','NN'), ('floor','NN'),('so','Cs'), ('that','Cs'),

('entrance','NN'),('is','BEZ'),('direct','JJ'),('.','.')]

>>>unseen\_sent =brown\_sents[4203]

>>>bigram\_tagger.tag(unseen\_sent)

[('The','AT'),('population','NN'),('of','IN'),('the','AT'),('Congo','NP'),

('is','BEZ'),('13.5',None),('million',None),(',',None),('divided',None),

('into',None),('at',None),('least',None),('seven',None),('major',None),

('`',None),('culture',None),('clusters',None),("''",None),('and',None),

('innumerable',None), ('tribes',None), ('speaking',None), ('400',None), ('separate',None),('dialects',None),('.',None)]

请注意，bigram 标注器能够标注训练中它看到过的句子中的所有词，但对一个没见



过的句子却不行。只要遇到一个新词(如13.5)就无法给它分配标记。它不能标注 下面的词(如： million), 即使是在训练过程中看到过的，因为在训练过程中从来 没有见过它前面有None 标记的词。因此，标注器也无法标注句子的其余部分。它 的整体准确度得分非常低。

>>>bigram\_tagger.evaluate(test\_sents)

0.10276088906608193

当n 越大时，上下文的特异性就会增加，要标注的数据中包含训练数据中不存在 的上下文的几率也增大。这被称为数据稀疏问题，在NLP 中是相当普遍的。因此， 研究结果的精度和覆盖范围之间需要有一个权衡(这与信息检索中的精度/召回权 衡有关)。

**注意!**

N-gram 标注器不应考虑跨越句子边界的上下文。因此，NLTK 的标 注器被设计用于句子链表， 一个句子是一个词链表。在一个句子的 开始，tn-1和前面的标记被设置为None。

**组合标注器**

解决精度和覆盖范围之间权衡的一个办法是尽可能地使用更精确的算法，但却在很 多时候却逊于覆盖范围更广的算法。例如：可以按如下方式组合 bigram 标注器、 unigram 标注器和一个默认标注器。

● 尝试使用bigram 标注器标注标识符。

· 如果 bigram 标注器无法找到标记，尝试unigram 标注器。

● 如果 unigram 标注器也无法找到标记，使用默认标注器。

大多数 NLTK 标注器允许指定回退标注器。回退标注器自身可能也有回退标 注器。

>>>t0 =nltk.DefaultTagger('NN')

>>>t1 =nltk.UnigramTagger(train\_sents,backoff=t0)

>>>t2 =nltk.BigramTagger(train\_sents,backoff=t1)

>>>t2.evaluate(test\_sents)

0.84491179108940495



**轮到你来：**

通过定义一个名为t3 的 TrigramTagger, 扩展前面的例子，它是t2 的 回退标注器。

请注意，在标注器初始化时要指定回退标注器，从而训练时才能利用回退标注器。 于是，如果在上下文中 bigram 标注器将分配与它的 unigram 回退标注器一样的 标记，那么 bigram 标注器丢弃训练的实例。这样可以保持尽可能小的 bigram 标 注器模型。可以进一步确定的是标注器需要保存上下文的多个实例。例如： nltk.BigramTagger(sents,cutoff=2, backoff=t1) 将丢弃那些只出现一次或两次的 上下文。

**标注生词**

标注生词的方法是回退到正则表达式标注器或默认标注器。这些都无法利用上下 文。因此，如果标注器遇到词 blog, 但训练过程中没有看到过，它会分配相同的标 记，不论这个词出现的上下文是 the blog 还是to blog。我们怎样才能更好地处理这 些生词或**词汇表以外**的项目?

基于上下文标注生词的方法是限制标注器的词汇表为最频繁的 n 个词，使用5.3 节中的方法替代其他每个词为特殊的词 UNK。 训练时， unigram 标注器可能会将 UNK 标注名词。然而， n-gram 标注器会检测其他标记的上下文。例如：如果前面 的词是to (标注为TO), 那么UNK 可能会被标注为一个动词。

**存储标注器**

在大语料库中训练标注器可能需要花费大量的时间。没有必要重复训练标注器， 可将一个训练好的标注器保存到文件为以后重复使用。将标注器 t2保存到文件 t2.pk1。

>>>from cPickle import dump

>>>output =open('t2.pkl','wb')

>>>dump(t2,output,-1)

>>>output.close()

现在，可以在单独的 Python 进程中载入之前保存的标注器。

>>>from cPickle import load

>>>input =open('t2.pkl','rb')

>>>tagger =load(input)

>>>input.close()

**现在，检查它是否可以用来标注。**

rni"

>>>tokens =text.split()

>>>tagger.tag(tokens)

[('The','AT'),("board's",'NNS'),('action','NN'),('shows','NNS'),

('what','WDT'),('free','JJ'),('enterprise','NN'),('is','BEZ'),

('up','RP'),('against','IN'),('in','IN'),('our','PP$'),('complex','JJ'),

('maze','NN'),('of','IN'),('regulatory','NN'),('laws','NNS'),('.','.')]

**性能限制**

n-gram标注**器性能的上限是什么?参考** **trigram标注器。它遇到多少词性歧**义的情 况?我们可以根据经验解决这个问题。

>>>cfd =nltk.ConditionalFreqDist(

((x[1],y[1],z[0]),z[1])

· for sent in brown\_tagged\_sents

· for x,y,z in nltk.trigrams(sent))

>>>ambiguous\_contexts =[c for c in cfd.conditions()if len(cfd[c])>1] >>>sum(cfd[c].N()for c in ambiguous\_contexts)/cfd.N()

0.049297702068029296

因此，1/20 的 trigrams 是有歧义的。给定当前单词及其前两个标记，根据训练数据， 在5%的情况中，可能有一个以上的标记合理地分配给当前词。假设我们总是挑选在 这种含糊不清的上下文中挑选最有可能的标记，可以得出trigram 标注器性能的下界。

调查标注器性能的另一种方法是研究它的错误。有些标记可能会比其他的标记更难 分配，因此需要专门对这些数据进行预处理或后处理。 一种便捷查看标注错误的 方法是**混淆矩阵。**它利用图表表示期望的标记(黄金标准)与实际由标注器产生的 标记。

>>>test\_tags =[tag for sent in brown.sents(categories='editorial')

'ediad\_tags =[tag fwor)ag)in brown.ta(dsords(categories=

>>>print nltk.ConfusionMatrix(gold, test)

基于以上的分析，决定修改标记集。标记之间很难做出区分的部分可以被丢弃，因

为它在一些较大的处理任务的上下文中并不重要。

分析标注器性能界限的另一种方式是人工标注者经过商讨得到的。

一般情况下，标注过程会消除区别，例如：当所有的人称代词被标注为 PRP 时， 词的特性通常会失去。与此同时，标注过程引入了新的区别从而去除了含糊之处， 例如：deal 标注为VB 或 NN。这种消除某些区别并引入新的区别的特点是标注的 重要的特征，有利于分类和预测。当引入一个标记集的更细的划分时，在 n-gram 标注器决定什么样的标记分配给一个特定的词时，可以获得关于剩余上下文的更详 细的信息。然而，标注器同时也将需要做更多的工作来划分当前的标识符，只是因 为有更多可供选择的标记。相反，使用较少的区别(如简化的标记集),标注器有 关上下文的信息会减少，为当前标识符分类的选择范围也较小。

我们已经看到， 训练数据中的歧义可产生标注器性能的上限。有时更多的上 下文能解决这些歧义。然而，在其他情况下，如 (Abney,1996) 中指出的， 只有参考语法或现实世界的知识才能解决歧义。尽管存在缺陷，但词性标注 在利用统计方法进行自然语言处理的发展过程中起到了核心作用。20世纪90 年代初，统计标注器就可以在不利用较深的语言学知识的前提下解决一小部 分语言理解问题，即词性消歧。这个想法能再推进吗?在第7章中，我们将 继续介绍。

**跨句子边界标注**

n-gram 标注器使用最近的标记作为当前词选择标记的指导。当标记句子的第一个词 时 ，trigram 标注器将使用前面两个标识符的词性标记，通常是前面句子的最后一 个词和句子结尾的标点符号。然而，前一句结尾的词的类别与下一句的开头通常没 有关系。

为了应对这种情况，可以使用已标注句子的链表来训练、运行和评估标注器，如例 5-5 所示。

例5-5 句子层面的N-gram 标注。

brown\_tagged\_sents =brown.tagged\_sents(categories='news') brown\_sents =brown.sents(categories='news')

size =int(len(brown\_tagged\_sents)\*0.9)

train\_sents =brown\_tagged\_sents[:size]

test\_sents =brown\_tagged\_sents[size:]

t0

t1

t2

=nltk.DefaultTagger('NN')

=nltk.UnigramTagger(train\_sents,backoff=t0)

=nltk.BigramTagger(train\_sents,backoff=tl)

>>>t2.evaluate(test\_sents)

0.84491179108940495

**5.6** **基于转换的标注**

n-gram标注器存在的一个潜在的问题是n-gram 表的大小(或语言模型)。如果将各 种语言技术的标注器部署在移动计算设备上，在模型大小和标注器性能之间取得平 衡是很重要的。使用回退标注器的 n-gram 标注器可以存储 trigram 和 bigram 表， 这是一项巨大的稀疏阵列，可能有数亿条条目。

第二个问题是关于上下文的。n-gram 标注器从前面的上下文中获得的唯一信息是标 记，即使词本身可能是一个有用的信息源。n-gram 模型中以上下文中的词的其他特 征为条件是不切实际的。在本节中，我们利用 Brill 标注，它是一种归纳标注方法， 性能好，使用的模型仅有 n-gram 标注器的很小一部分。

Brill 标注是一种基于转换的学习，以它的发明者命名。 一般的想法很简单：猜想每 个词的标记，然后返回和修复错误的。在这种方式中，Brill标注器陆续将一个不良 标注的文本转换成一个好的。与 n-gram 标注一样，需要监督整个过程，因为我们 需要已标注的训练数据来评估标注器的猜测是否是一个错误。然而，不同于 n-gram 标注的是，它不计数观察结果，只编制一个转换修正规则链表。

Brill 标注的过程通常是类比绘画来解释的。假设要画一棵树，包括大树枝、树 枝、小枝、叶子和天蓝色背景。不是先画树然后在空白处画蓝色，而是先将整 个画布画成蓝色，然后通过在蓝色背景上上色“修正”树的部分。以同样的方 式，先统一画出褐色的树干然后再用更精细的刷子描绘细节。Brill 标注使用了 同样的想法：以大笔画开始，然后修复细节， 一点点细致地进行改变。让我们 看看下面的例子。

(1)The President said he will ask Congress to increase grants to states for vocational

rehabilitation.(总统表示：他将要求国会给各州增加拨款用于职业康复。)

研究两个规则的运作： (a) 当前面的词是TO 时，替换NN 为 VB;(b) 当下一个 标记是 NNS 时，替换TO 为 IN 。表5-6说明了这一过程，首先使用 unigram 标注 器标注，然后运用规则修正错误。

**表5-6** **Brill 标注的步骤**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Phrase** | **to** | **increase** | **grants** | **to** | **states** | **for** | **vocational** | **rehabilitation** |
| Unigram | TO | NN | NNS | TO | NNS | IN | JJ | NN |
| Rulel |  | VB |  |  |  |  |  |  |
| Rule2 |  |  |  | IN |  |  |  |  |
| Output | TO | VB | NNS | IN | NNS | IN | JJ | NN |
| Gold | TO | VB | NNS | IN | NNS | IN | JJ | NN |

在此表中，可看到以上两个规则。所有这些规则由以下形式的模板产生：“在上下 文C 中，替换T1 为 T2。” 典型的上下文是之前或之后的词或标记，或者是两到三 个词范围内出现的特定标记。在其训练阶段， T1 、T2 和 C 的标注器猜测值创造出 数以千计的候选规则。每一条规则都根据其净收益打分：修正不正确标记的数目减 去错误修改正确标记的数目。

Brill 标注器的另一个特性：规则是语言学可解释的。与采用潜在的巨大的 n-gram 表的 n-gram 标注器相比，我们并不能从直接观察这样的一个表中学到多少东西， 而 Brill 标注器可以。例5-6演示了NLTK 中的 Brill 标注器。

例5-6 Brill 标注器演示：标注器有一个 “X→Y 如果前面的词是Z” 的形式的 模板集合。这些模板中的变量是创建“规则”的特定词和标记的实例。得分规 则是纠正错误例子的数目减去误报的数目。除了训练标注器，演示还显示了剩 余的错误。

>>>nltk.tag.brill.demo()

Training Brill tagger on 80 sentences..

Finding initial useful rules..

Found 6555 useful rules.

B

S F r O

c i o t R

o X k h u

Score =Fixed -Broken

Fixed =num tags changed incorrect ->correct Broken =num tags changed correct ->incorrect

e

d

r

e

12

13

9

8

9

8

6

5

8

8

6

5

5

5

Other =num tags changed incorrect ->incorrect e

e

n

e

r

1 4 NN ->VB if the tag of the preceding word is'TO'

1 23 NN ->VBD if the tag of the following word is'DT!

0 9 1!NN ->VBD if the tag of the preceding word is 'NNS'

3 16 I NN ->NNP if the tag of words i-2...i-1 is'-NONE-'

3 6 I NN ->NNP if the tag of the following word is'NNP'

1 0 NN ->NNP if the text of words i-2...i-1 is 'like'

0 3 I NN ->VBN if the text of the following word is '\*-1'

>>>print(open("errors.out").read())

left context l word/test->gold l right context

,in/IN the/DT guests/NNS '/POS honor/NN,/,the/DT NN,/,the/DT speedway/NN DT speedway/NN hauled/VBD dway/NN hauled/VBD out/RP hauled/VBD out/RP four/CD P four/CD drivers/NNS ,/, NNS and/CC even/RB the/DT

ter/IN the/DT race/NN,/, s/NNS drooled/VBD like/IN olboys/NNS over/IN the/DT

|  |  |
| --- | --- |
| Then/NN->RB  '/VBD->POS  speedway/JJ->NN | l,/,in/IN the/DT guests/N I honor/NN,/,the/DT speed  I hauled/VBD out/RP four/CD |

hauled/NN->VBD I out/RP four/CD drivers/NN

out/NNP->RP l four/CD drivers/NNS,/,c

four/NNP->CD I drivers/NNS,/,crews/NNS drivers/NNP->NNS 1,/,crews/NNS and/CC even

|  |  |
| --- | --- |
| Crews/NN->NNS  official/NNP->JJ After/VBD->IN  Fortune/IN->NNP  |schoolboys/NNP->NNS cars/NN->NNS | l and/CC even/RB the/DT off l Indianapolis/NNP 500/CD a l the/DT race/NN,/,Fortun 1500/CD executives/NNS dro I over/IN the/DT cars/NNS a I and/CC drivers/NNS ./. |

**5.7** **如何确定一个词的分类**

我们已经详细研究了词类，现在转向一个更基本的问题：如何决定一个词属于哪一 类?首先应该考虑什么?在一般情况下，语言学家使用形态学、句法和语义线索确 定一个词的类别。

**形态学线索**

一个词的内部结构有助于为这个词分类。举例来说： -ness 是一个后缀，与形容词 结合产生名词，如 happy→happiness,ill→illness 。所以，如果遇到一个以-ness 结 尾的词，很可能是一个名词。同样的， -ment 是与一些动词结合产生名词的后缀， 如 govern→government 和 establish→establishment。

英语动词也可以是形态复杂的。例如： 一个动词的现在分词以-ing 结尾，表示正在

进行的还没有结束的行动(如： falling 、eating) 。-ing 后缀也出现在从动词派生的 名词中，如： the falling of the leaves ( 这被称为**动名词**)。

**句法线索**

另一个线索是一个词可能出现的典型的上下文语境。例如：假设已经确定了名词 类，那么,英语形容词的句法标准是它可以出现在一个名词前，或紧跟在词be 或 very 后。根据这些测试，near 应该被归类为形容词。

(2)a.the near window

b.The end is(very)near.

**语义线索**

最后一个线索是一个词的意思。例如：名词众所周知的一个定义是根据语义的：“一 个人、地方或事物的名称。”现代语言学中，词类的语义标准受到置疑，主要是因 为它们很难规范化。然而，语义标准巩固了我们对许多词类的判断，使我们能够在 不熟悉的语言中很好地猜测词的分类。例如：如果知道荷兰语词 verjaardag 的意思 与英语词birthday 相同，那么就可以猜测verjaardag 在荷兰语中是一个名词。然而， 不同的是：虽然zij is vandaag jarig与it's her birthday today 是一样的，但是词jarig 在荷兰语中实际上是形容词，与英语并不完全相同。

**新词**

所有的语言都会产生新的词汇。最近添加到牛津英语词典中的单词列表包括 cyberslacker、fatoush 、blamestorm 、SARS 、cantopop 、bupkis 、noughties 、muggle 和 robata。请注意，所有这些新词都是名词，名词被称为**开放类。**相反，介词被 认为是**封闭类。**也就是说，只有有限的词属于这个类别(例如： above、along、 at 、below 、beside 、between 、during 、for 、from 、in 、near 、on 、outside 、over、 past 、through 、towards、under、up、with),词类成员随着时间的推移会逐渐发 生改变。

**词性标记集中的形态学**

普通标记集经常会“捕捉”一些构词信息，即一种词借助句法角色获得的形态标记 信息。例如：思考下面句子中词go 的不同语法形式的选集。

(3)a.Go away!

b.He sometimes goes to the cafe.

c.All the cakes have gone.

d.We went on the excursion.

go 、goes 、gone 和 went 是词 go 在形态学上的区别。思考形式 goes 。它出现在 受限制的语法环境中，需要第三人称单数的主语。因此，下面的句子是不合语 法的。

(4)a.\*They sometimes goes to the cafe.

b.\*I sometimes goes to the cafe.

相比之下， gone 是过去分词形式，需要出现在 have 后面(在这个上下文中不能被 goes 替代),不能作为一个从句的主要动词出现。

(5)a.\*All the cakes have goes.

b.\*He sometimes gone to the cafe.

上面讨论的4种不同语法形式的标记集都被标注为VB 。虽然这对于一般的使用已 经足够了，但更精细的标记集中包含更多有关这些形式的信息，可以帮助其他处理 器检测标记序列的模式。布朗标记集中包括这些区别，如表5-7所示。

**表5-7** **布朗标记集的一些构词区别**

|  |  |  |
| --- | --- | --- |
| **形** **式** | **类** **别** | **标** **记** |
| go | 基本 | VB |
| goes | 第三人称单数 | VBZ |
| gone | 过去分词 | VBN |
| going | 动名词 | VBG |
| went | 一般过去时 | VBD |

除了这组动词标记，动词to be的各种形式也有特殊的标记： be/BE 、being/BEG 、 am/BEM、are/BER、is/BEZ、been/BEN、were/BED 和 was/BEDZ (加上额外的动 词否定形式的标记)。总的来说，这种动词细粒度标记意味着使用此标记集的自动 标注器能有效开展有限数量的形态分析。

大多数词性标注集都使用相同的基本类别，如名词、动词、形容词和介词。然而，

标记集之间的区别不仅在于它们如何细致地将词分类，更在于它们如何界定其类 别。例如： is 在一个标记集中可能会被简单地标注为动词，而在另一个标记集中被 标注为lexeme be的形式(如在布朗语料库中)。这种标记集的变化是不可避免的， 因为词性标记以不同的方式应用于不同的任务。换句话说，没有一个“正确的方式” 来分配标记，只能根据目标不同而产生的或多或少有用的方法。

**5.8** **小结**

● 词可以组成类，如名词、动词、形容词及副词。这些类称为词汇范畴或者词性。 词性被分配短标签或者标记，如NN 和 VB。

● 给文本中的词自动分配词性的过程称为词性标注、POS 标注或标注。

● 自动标注是 NLP 流程中的重要环节，适用于各种情况，包括预测先前未见过 的词的行为、分析语料库中词的使用及文本到语音转换系统。

· 一些语言学语料库，如布朗语料库，已经完成了词性标注。

· 有多种标注方法，如默认标注器、正则表达式标注器、unigram 标注器、n-gram 标注器。这些都可以结合一种叫做回退的技术一起使用。

● 标注器可以使用已标注语料库进行训练和评估。

● 回退是一个组合模型的方法：当一个较专业的模型(如 bigram 标注器)不能 为给定内容分配标记时，可回退到一个较一般的模型。(如 unigram 标注器)

● 词性标注是 NLP 中一个重要的、早期的序列分类任务：利用局部上下文语境 中的词和标记，对序列中任意一点的分类决策。

· 字典用来映射任意类型之间的信息，如字符串和数字： freq['cat]=12 。 使

用大括号创建字典： pos={},pos={furiously':'adv,ideas':'n','colorless': adj'}。

● N-gram标注器可以定义较大数值的n, 但是当n 大于3时，常常会面临数据稀 疏问题。即使使用大量的训练数据，看到的也只是上下文的一小部分。

· 基于转换的标注包括学习一系列“改变标记s 为标记t 在上下文c 中”形式的 修复规则，每个规则会修复错误，也可能引入(较小的)错误。

**5.9** **深入阅读**

本章的额外材料发布在 <http://www.nltk.org/>上，包括网上免费提供的资源链接。关 于使用 NLTK 标注的更多的例子，请查看 <http://www.nltk.org/howto> 上的标注 HOWTO 。Jurafsky &Martin(2008) 的 第4章和第5章包含更多有关n-grams 和词 性标注的高级材料。其他标注方法涉及到机器学习方法(见第6章)。在第7章 中，我们将学习分块标注的泛化，为连续的词序列分配单独的标记。

关于标记集的文档，请参阅 nltk.help.upenn\_tagset() 和 nltk.help.brown\_tagset()。词 汇 范畴在很多语言学教科书中都有介绍，包括本书第1章中所列的书籍。

还有许多其他类型的标注。可以按语音合成器的指令标注，表示应强调哪些词；可 以按词意数标注，表示使用词的哪个意思；也可以按形态特征标注。各类别的标记 例子如下所示。由于篇幅所限，我们只展示一个词的标记。还要注意的是前两个例 子使用的是XML 风格的标签，尖括号中的元素分布在已标注词的两侧。

*Speech Synthesis Markup Language (W3CSSML)*

That is a <emphasis>big</emphasis>car!

*SemCor:Brown Corpus tagged with WordNet senses*

Space in any <wf pos="NN"lemma="form"wnsn="4">form</wf>is completely measured by the three dimensions.(Wordnet form/nn sense 4:"shape,form, config-uration,contour,conformation")

***Morphological tagging,from the Turin University Italian Treebank***

SING) porto turistico dell'Albania .

E'italiano,come progetto e realizzazione ,il primo(PRIMO ADJ ORDIN M

请注意，标注也可以在更高层次上进行。下面是一个对话行为标注的例子，来自 NLTK中的 NPS 聊天语料库 (Forsyth &Martell,2007)。每段对话按照它的交际功 能分类。

|  |  |  |
| --- | --- | --- |
| Statement | User117 Dude...,I wanted some of that |  |
| YnQuestion | User120 m I missing something? |  |
| Bye | User117 I'm gonna go fix food,I'll be | back later. |
| System | User122 JOIN |  |
| System | User2 slaps User122 around a bit with | a large trout. |
| Statement | User12118/m pm me if u tryin to chat |  |

**5.10** **练习**

1.O 网上搜索 “spoof newspaper headlines”, 找到： British Left Waffles on Falkland Islands 和 Juvenile Court to Try Shooting Defendant。手工标注，看看利用词性标记 是否可以消除歧义。

2.O 找个同伴，轮流挑选一个既是名词又是动词的词(如： contest)。让对方预测哪一 个可能是布朗语料库中频率最高的。检查对方的预测，并打分。

3.O 分词和标注下面的句子：They wind back the clock,while we chase after the wind 。句中包含哪些不同的发音和词类?

4.O 回顾表5.4 中的映射。讨论你能想到的有关映射的其他的例子。它们从什么 类型的信息映射到什么类型的信息?

5.O 在交互模式下使用Python 解释器，实验本章中字典的例子。创建一个字典 d, 添加一些条目。如果尝试访问一个不存在的条目，如 d[xyz], 会发生什么?

6.O 尝试从字典d 删除一个元素，使用语法 del d[abc']。检查被删除的项目。

7.O 创建两个字典，d1 和 d2, 为每个添加一些条目。现在发出命令 d1.update(d2)。 会发生什么?它可能是有什么用?

8.O 创建一个字典e, 代表你选择的词的单独的词汇条目。定义键如：headword、 part-of-speech 、sense 和 example, 分配给它们适当的值。

9 . ○验证go 和 went 在分布上的限制，也就是说，它们不能自由地如第5.7节(3)中演 示的那样在上下文中互换。

10.O 训练一个 unigram 标注器，在一些新的文本上运行。观察没有分配到标记的 词。为什么没有分配到标记?

11.O 了解词缀标注器(输入help(nltk.AffixTagger))。训练一个词缀标注器，在一 些新的文本上运行。设置不同的词缀长度和最小词长。并讨论结果。

12.O 训练一个没有回退标注器的bigram 标注器，在一些训练数据上运行。然后， 在一些新的数据运行它。标注器的性能会发生什么改變?为什么呢?

13.O 我们可以使用字典指定由一个格式化字符串替换的值。阅读关于格式化字符 串的Python 库文档 (<http://docs.python.org/lib/typesseq-strings.html>), 使用这种方法 以两种不同的格式显示今天的日期。

14.①使用 sortedO和 set()获得布朗语料库使用的标记排序链表，删除重复。

15.①编写程序处理布朗语料库，找到以下问题的答案。

a. 哪些名词常以它们复数形式而不是它们的單数形式出现?(只考虑常规的复 数形式， -s 后缀形式的)。

b. 哪个词的不同标记数目最多?它们是什么,又代表什么?

c. 按频率递减的顺序列出标记。前20个最频繁的标记代表什么?

d. 名词后面最常见的是哪些标记?这些标记代表什么?

16.①探索有关查找标注器的以下问题。

a. 回退标注器被省略时，控制各种模型类型的标注器性能会发生什么?

b. 观察图5-4 的曲线；为查找标注器推荐一个能够平衡内存和性能的适当规 模。你能想出在什么情况下应该尽量减少内存使用，什么情况下性能最大 化而不必考虑内存使用?

17.①查找标注器的性能上限是什么?假设其表的大小没有限制。(提示：编写程 序算出分配有最可能标记的词的标识符的平均百分比。)

18.①生成已标注数据的一些统计数据，回答下列问题。

a. 总是被分配相同词性的词类的比例是多少?

b. 多少词是有歧义的，即从某种意义上说，是至少和两个标记一起出现的词?

c. 布朗语料库中有歧义的词的标识符的百分比是多少?

19.①利用 evaluate)方法算出在文本上运行的标注器的精度。例如：如果已标注 文本是[(the',DT),(dog',NN)], 标注器产生的输出是[(the',NN),(dog,NN'], 那 么得分为0.5。让我们尝试找出评价方法是如何工作的。

a. 标注器t 将词汇链表作为输入，产生的已标注词链表作为输出。 t.evaluate) 只以正确标注的文本作为唯一的参数。执行标注之前必须对输入做些 什么?

b. 一旦标注器创建了新标注的文本， evaluate) 方法将如何比较它与原来标注 的文本?计算准确性得分。

c. 现在，检查源代码来看看这个方法是如何实现的。检查nltk.tag.api. file 找到源代码的位置，使用编辑器打开文件(一定要使用文件 api.py, 而不 是编译过的二进制文件api.pyc)。

20.①编写代码，搜索布朗语料库，根据标记查找特定的词和短语，回答下列问题。

a. 产生一个标注为 MD的，不同词按字母顺序排序的列表。

b. 识别可能是复数名词或第三人称单数动词的词(如 deals 、flies)。

c. 识别三词介词短语，形式为IN +DET+NN (如 in the lab)。

d. 男性与女性代词的比例是多少?

21.①在表3-1中我们看到动词adore 、love 、like 和prefer 及前面的限定符如 really 的频率计数的表格。探讨这4个动词前出现的所有限定符(布朗标记QL)。

22.①定义可以用来作为生词的回退标注器的 regexp\_tagger。这个标注器只检查基 数词。通过特定的前缀或后缀字符串进行测试，它能够猜测其他标记。例如：我们 可以标注所有-s 结尾的词为复数名词。定义一个正则表达式标注器(使用 RegexpTagger()), 测试单词拼写的其他至少5个模式。(使用内联文档解释规则。)

23.①考虑上一练习中开发的正则表达式标注器，使用它的 accuracyO方法评估标注 器，想办法提高其性能。讨论你找到的结果。客观的评估是如何帮助开发过程的?

24.①数据稀疏问题有多严重?测试n-gram 标注器当n 从1增加到6时的性能。 将准确性得分制成表格。估计这些标注器需要的训练数据，假设词汇量大小为10⁵ 而标记集的大小为10²。

25.①获取另一种语言的一些已标注数据，在其上测试和评估各种标注器。如果这 种语言是形态复杂的，或者含有词类的任何字形线索(如：capitalization),可以考 虑为它开发一个正则表达式标注器(排在 unigram标注器之后，默认标注器之前)。 对比其他在英文数据上运行的标注器， tagger(s)的准确性如何?讨论将这些方法运 用到这种语言时遇到的问题。

26.①例5-4中的曲线显示了查找标注器的性能随模型的变化而增加。当训练数据 量变化时，绘制unigram 标注器的性能曲线。

27.①检查5.5节中 bigram 标注器t2 的混淆矩阵，确定一套或多套标记。定义字 典进行映射，在简化的数据上评估标注器。

28.①使用简化的标记集测试标注器(或制作一个你自己的，丢弃每个标记名中除 第一个字母外所有的字母)。这种标注器需要做的区分少，它获得的信息也更少。 讨论结果。

29.①回顾 bigram标注器在训练过程中遇到生词，且标注句子的其余部分为None 的例子。可能发生 bigram 标注器只处理了句子的一部分就失效的情况吗?即使句 子中没有包含生词(即使句子在训练过程中使用过)。在什么情况下会出现这种情 况呢?你可以编写一段程序，并找到一些这方面的例子吗?

30.①预处理布朗新闻数据，替换低频词为 UNK, 但留下标记不变。在这些数据 上训练和评估 bigram标注器。这样做有用吗?unigram 标注器和默认标注器的贡献 是什么?

31.①修改例5-4中的程序，将 pylab.plot()替换为 pylab.semilogx(),在 x 轴上使用 对数刻度。关于结果图形的形状，你注意到了什么?梯度告诉你什么呢?

32.①阅读 Brill 标注器演示函数的文档，使用 help(nltk.tag.brill.demo)。通过设置

不同的参数值测试这个标注器。是否有任何训练时间(语料库大小)和性能之间的 权衡?

33.①编写代码构建一个集合字典的字典。用它来存储一套可以跟在具有给定POS 标 记的给定词后面的POS 标记，例如： word;→tag→tag+1。

34. ●布朗语料库中有264个不同的词，共有3种可能的标签。

a. 打印一个表格， 一列是整数1…10,另一列是语料库中有1…10个不同标 记的不同词的数目。

b. 对含有不同标记数量最多的词，在语料库中将包含该词的句子输出，每个 可能的标记一个。

35.●编写程序，按照词must 后面的词的标记为它的上下文分类。这样可以区分 must 的“必须”和“应该”两种词意在用法上的区别吗?

36.●创建正则表达式标注器、各种 unigram 及n-gram 标注器，包括回退，在布 朗语料库上训练它们。

a. 创建这些标注器的3种不同组合。测试每个组合标注器的准确性。哪种组 合效果最好?

b. 尝试改变训练语料的规模。它是如何影响你的结果的?

37.●标注生词的方法是要考虑这个词的字母(使用 RegexpTagger)), 或完全忽 略这个词，将它标注为一个名词(使用nltk.DefaultTagger())。这些方法对于有新词 却不是名词的文本效果不大。思考句子： I like to blog on Kim's blog。如果blog 是 一个新词，那么查看前面的标记 (TO 和 NP$) 可能会有所帮助，即我们需要一个 对前面的标记敏感的默认标注器。

a. 创建一种新的unigram标注器，查看前一个词的标记，而忽略当前词。(做 到这一点的最好办法是修改 UnigramTagger()的源代码，需要Python 中的面 向对象编程的知识。)

b. 将这个标注器加入到回退标注器序列(包括普通的 trigram 和 bigram 标注

器),放在常用默认标注器的前面。

c. 评价这个新的unigram标注器的贡献。

38. ●思考5-5节中的代码，它确定了 trigram 标注器的准确性上限。回顾 Abney 的关于精确标注的不可能性的讨论 (Abney,2006) 。 解释为什么正确标注这些例子 需要获取词和标记以外的其他种类的信息。你如何看待这个问题的?

39.●使用nltk.probability 中的一些估计技术，例如 Lidstone 或 Laplace 估计，开 发一种统计标注器，它在训练中没有遇到而测试中遇到的上下文中表现优于 n-gram 回退标注器。

40.●检查 Brill 标注器创建的诊断文件 rules.out 和 errors.out。通过访问源代码 (<http://www.nltk.org/code>) 获得演示代码，创建你自己版本的Brill 标注器。并根据 你从检查rules.out 中了解到的，删除一些规则模板。增加一些新的规则模板，这些 模板使用那些有助于纠正在 errors.out 看到的错误的上下文。

41.●开发一个n-gram回退标注器，允许在标注器初始化时指定 “anti-n-grams”, 如：["the","the"] 。anti-n-grams 被分配给数字0,用来防止n-gram 回退(如避免估 计P(the|the) 而只是P(the))。

42. ●使用布朗语料库开发标注器时，调查3种不同的方式来定义训练和测试数据 之间的分割： genre(category) 、source(fileid)和句子。比较它们的相对性能，并讨论 哪种方法最合理。(你可能要使用n-交叉验证，将在6.3 节中进行讨论，以提高评 估的准确性。)

43. ●开发你自己的NgramTagger, 从NLTK 中的类继承，封装本章中所述的缩减 已标注的训练和测试数据词汇表的方法。确保 unigram 和默认回退标注器能够获得 全部词汇。



**第6章**

**学习分类文本**

模式识别是自然语言处理的一个核心部分。以-ed 结尾的词往往是过去时态动词 (见第5章)。频繁使用will 暗示着这是新闻文本(见第3章)。这些可观察到的模 式——词的结构和词频——恰好与特定方面的含义相关联，如：时态和主题。但我 们怎么知道从哪里开始寻找，哪一方面的形式与哪一方面的含义相关联?

本章的目标是要回答下列问题。

(1)怎样才能识别出语言数据中明显用于分类的特征?

(2)怎样才能构建用于自动执行语言处理任务的语言模型?

(3)从这些模型中我们可以学到哪些关于语言的知识?

在本章中，我们将研究一些重要的机器学习技术，包括决策树、朴素贝叶斯分类器 和最大熵分类。我们将忽略这些技术中的数学和统计的知识，集中关注如何及何时 使用它们(更多的技术背景知识见6.9节)。在研究这些方法之前，我们首先需要 知道这个主题的广泛范围。

**6.1** **监督式分类**

**分类**是为给定的输入选择正确**的类标签。**在基本的分类任务中，每个输入被认 为是与所有其他输入隔离的，并且标签集是预先定义的。下面是分类任务的一 些例子。

● 判断一封电子邮件是否是垃圾邮件。

· 从一个固定的主题领域列表中，如“体育”、“技术”和“政治”,来判断新闻 报道的主题是什么。

· 判断给定词 bank 的意思是指河的坡岸、金融机构、向一边倾斜的动作还是在 金融机构里的存储行为。

基本的分类任务有许多有趣的变化。例如：在多样分类中，每个实例可以分配多个 标签；在开放性分类中，标签集是事先没有定义的；在序列分类中，将输入链表作 为整体分类。

建立在训练语料(包含了每个输入的正确标签)基础之上的分类，叫做**监督式分类**。 监督式分类的框架图如图6-1所示。

|  |  |  |
| --- | --- | --- |
| (a)训练  标签  机器学习算法  特征提取器  特征  输入  (b)预测 | | |
| 特征提取器  特征  输入 | 分类模型 | 标签 |

**图6-1** **监督式分类。(a)** **在训练过程中，特征提取器将每一个输入值转换为特征集。这些特征** **集捕捉到每个输入中应被用于分类的基本信息，我们将在下一节中讨论该部分内容。特征集与标** **签的配对放入到机器学习算法中，生成模型。(b)** **在预测过程中，相同的特征提取器被用来将未** **见过的输入转换为特征集。之后，这些特征集被放入模型来产生预测标签**

在本节的剩余部分，我们将着眼于分类器如何解决各种各样的任务。我们讨论 的目的不是要全面性了解，而是旨在在文本分类器的帮助下给出执行任务的代 表性的例子。

**性别鉴定**

在2.4节中我们看到，男性和女性的名字有各自鲜明的特点。以a、e和 i结尾的姓 名很可能是女性姓名，而以k 、o 、r 和 s 结尾的姓名很可能是男性姓名。建立一个



分类器从而更精确地模拟这些差异。

创建分类器的第一步是决定什么样的输入特征是相关的，以及如何为这些特征编 码。在下面的例子中，从寻找给定名称的最后一个字母开始。下面的**特征提取器函** 数建立了一个字典，其中包含给定名称的相关信息。

>>>def gender\_features(word):

return {'last\_letter':word[-1])

>>>gender\_features('Shrek')

{'last\_letter': 'k'}

这个函数返回的字典被称为**特征集，**能把特征名称映射到它们的值。特征名称是区 分大小写的字符串，通常提供一个简短的、可读的特征描述。特征值是简单类型的 值，如布尔、数字和字符串。

大多数分类方法要求特征使用简单的类型进行编码，如布尔类型、数 字和字符串。要注意这个特征是简单类型，但并不一定意味着该特征 的值易于表达或计算；的确，它可以用非常复杂的和有益的值作为特 征，如：第2级监督分类器的输出。

定义一个特征提取器，同时准备一些例子和与其对应的类标签。

>>>from nltk.corpus import names

>>>import random

>>>names =([(name,'male')for name in names.words('male.txt')]+

[(name, 'female') for name in names.words('female.txt')]) >>>random.shuffle(names)

接下来，使用特征提取器处理 names 数据，并把特征集的结果链表划分为训练集和 **测试集。**训练集用于训练新的“朴素贝叶斯”分类器。

nres\_seettest\_=(egte[(,eset**]**

>>>classifier =nltk.NaiveBayesClassifier.train(train\_set)

在本章的后面，我们将学习更多关于朴素贝叶斯分类器的内容。现在，测试一些没 有出现在训练数据中的名字。

>>>classifier.classify(gender\_features('Neo'))

'male!



>>>classifier.classify(gender\_features('Trinity'))

'female'

从结果看到，《黑客帝国》中这些角色的名字被正确分类出来。尽管这部科幻电影 的背景是在2199年，但它仍然符合我们对有关名字和性别的预期。我们可以利用 大量未见过的数据系统地评估这个分类器。

>>>print nltk.classify.accuracy(classifier, test\_set)

0.758

最后，检查分类器，确定哪些特征对于区分名字的性别是最有效的。

>>>classifier.show\_most\_informative\_features(5)

last\_letter ='a' last\_letter ='k'

Most Informative Features

**l**tt\_\_ll w'p

last\_letter ='f'

female : male

=

=

=

=

male : female

male :female

male:female

male:female

=

38.3:1.0

31.4:1.0

15.3 :1.0

10.6 :1.0

10.6:1.0

此列表显示训练集中以a 结尾的名字中女性是男性的38倍，而以k 结尾的名字中 男性是女性的31倍。这些比率称为似然比，可以用于比较不同特征-结果关系。

**轮到你来：**

修改gender\_features()函数，为分类器提供名称的长度、它的第一个字 母及其他任何看起来可能有用的特征。再用这些新特征训练分类器， 并测试其准确性。

在处理大型语料库时，构建包含所有实例特征的单独链表会占用大量的内存。在这 种情况下，使用函数 nltk.classify.apply\_features, 返回一个像链表但不会在内存存 储所有特征集的对象。

>>>from nltk.classify import apply\_features

>>>train\_set =apply\_features(gender\_features,names[500:]) >>>test\_set =apply\_features(gender\_features, names[:500])

**选择正确的特征**

为一个学习方法选择相关的特征，并决定如何编码它们，这对学习方法在提取一个 好的模型方面可以产生巨大的影响。建立分类器的工作之一是找出哪些特征可能是

相关的，以及如何表示它们。虽然使用简单且明显的特征集往往可以得到像样的 性能，但是使用精心构建的基于对当前任务透彻理解上的特征，通常会达到更好 的效果。

一般地，特征提取是在反复试验和错误的过程中建立的，哪些信息与问题相关， 是通过直觉来引导的。它通常以“激进现实主义”的方法开始，包括你能想到的 所有特征，然后检查哪些特征是实际有用的。在例6-1 中对名字性别特征采取了 这种做法。

例6-1 特征提取器过拟合性别特征。这个特征提取器返回的特征集中包括大量的 指定特征，从而导致相对较小的名字语料库产生了过拟合

def name[0].lower()

features["lastletter"]=name[-1].lower()

for re"cf8gj"eqrr]sunvower().count(letter)

features["has(8s)"%letter]=(letter in name.lower())

return features

>>>gender\_features2('John')

{'count(j)':1, 'has(d)':False, 'count(b)':0, ..}

然而，所用的给定的学习算法的特征数目是有限的——如果你提供太多的特 征，那么该算法将高度依赖训练数据的特性而对一般化的新例子不起作用。 这个问题被称为**过拟合，**在小型训练集上运行时尤其会出现这种问题。例如： 使用例6-1 中所示的特征提取器训练朴素贝叶斯分类器，将会过拟合这个相对 较小的训练集，导致系统精度会比只考虑每个名字最后一个字母的分类器低 约 1 % 。

>>>featuresets =[(gender\_features2(n),g)for(n,g)in names] >>>train\_set,test\_set =featuresets[500:],featuresets[:500] >>>classifier =nltk.NaiveBayesClassifier.train(train\_set)

>>>print nltk.classify.accuracy(classifier,test\_set)

0.748

选定初始特征集， 一种能有效完善特征集的方法称为**错误分析。**首先，选择开 **发集，**其中包含用于创建模型的语料数据。然后将这种开发集分为训练集和开 **发测试集。**

>>>train\_names =names[1500:]

>>>devtest\_names =names[500:1500]

>>>test\_names =names[:500]

很重要的一点是训练集用于训练模型，开发测试集用于执行错误分析，测试集用于 系统的最终评估。利用单独的开发测试集而不是测试集进行错误分析，理由在后面 会介绍。图6-2所示为将语料数据划分成不同的子集。

|  |  |  |
| --- | --- | --- |
| evelopment Set | Corpus  Dev-Test Set | Test Set |

图6-2 用于训练有监督分类器的语料数据组织图。语料数据分为两类：开发集和测试集。开发 集通常被又分为训练集和开发测试集

将语料分为适当的数据集，然后使用训练集来训练模型°,之后在开发测试集上 运行②。

>>>train\_set =【(gender\_features(n),g) for(n,g) in train\_names]

>>>devtest\_set =[(gender\_features(n),g) for(n,g)in devtest\_names]

>>>test\_set =[(gender\_features(n),g)for(n,g)in test\_names]

>>>classifier =nltk.NaiveBayesClassifier.train(train\_set)①

>>>print nltk.classify.accuracy(classifier,devtest\_set)②

0.765

使用开发测试集可以生成分类器在预测名字性别时出现的错误列表。

>>>errors =[]

>>>for(sivsts\_ider\_features(name))

if guess !=tag:

errors.append((tag,guess,name))

然后，检查个别错误案例，在案例中该模型预测了错误的标签，尝试确定加入什么 样的额外信息才能够使其作出正确的决定(或者是现有的哪部分信息导致其做出了 错误的决定)。然后相应地调整特征集。下例中已建立的名字分类器在开发测试语 料上产生约100个错误。

>>>for(tag,guess,name)in sorted(errors):#doctest:+ELLIPSIS+NORMALIZE WHITESPACE print 'correct=8-8s guess=8-8s name=8-30s'%

(tag,guess,name)

correct=female guess=male

correct=female guess=male correct=female guess=male

correctale guess=female

correct=male guess=female



correct=male guess=female

name=Cindelyn

name=Katheryn

name=Kathryn

name=Aldrich

name=Mitch

name=Rich

浏览这个错误列表，它明确指出某些多字母后缀也可以指示名字性别。例如：以 yn 结尾的名字大多以女性为主，尽管事实上，以n 结尾的名字往往是男性；以ch 结尾的名字通常是男性，尽管以h 结尾的名字倾向于是女性。因此，调整特征提取 器使其包含两个字母后缀的特征。

>>>def gender\_features(word):

return {'suffix1':word[-1:],

'suffix2':word[-2:]}

**使用新的特征提取器重建分类器，我们看到测试数据集上的性能提高了近3个百分**

点(从76.5%到78.2%)。

>>>train\_set =[(gender\_features(n),g)for(n,g)in train\_names]

>>>devtest\_set =[(gender\_features(n),g)for(n,g) in devtest\_names]

>>>classifier =nltk.NaiveBayesClassifier.train(train\_set) >>>print nltk.classify.accuracy(classifier,devtest\_set) 0.782

这个错误分析过程可以不断重复，检查由于新改进的分类器导致某些错误产生的模 式，每一次错误分析过程会被重复，我们应该选择一个不同的开发测试/训练分割， 以确保该分类器不会反映开发测试集的特质。

但是， 一旦使用了开发测试集帮助我们开发模型，在关于这个模型在新数据会表现 有多好的问题上，它将无法给我们一个准确的结果!因此， 一定要保持测试集分离、 未使用过，直到我们的模型开发完毕。在这一点上，我们可以使用测试集评估模型



在新的输入值上执行效果如何。

**文档分类**

在2.1 节中，我们看到了几个有关语料库的例子，其中文档已经按类别标记。使用 这些语料库，建立分类器，自动为新文档添加适当的类别标签。首先，构造已标记 相应类别的文档清单。对于下面的例子，选择电影评论语料库，将每个评论归类为 正面或负面。

>>>from nltk.corpus import movie\_reviews

>>>documents =[(list(movie\_reviews.words(fileid)),category)

. · for category in movie\_reviews.categories()

for fileid in movie\_reviews.fileids(category)]

>>>random.shuffle(documents)

接下来，为文档定义特征提取器，这样分类器就会知道应注意哪些方面的数据(见 例6-2)。对于文档主题识别，可以为每个词定义一个特性以表示该文档是否包含这 个词。为了限制分类器需要处理的特征数目，构建整个语料库中前2000个最频繁 词的链表°。然后，定义一个特征提取器”,简单地检查这些词是否在一个给定的文 档中。

例6-2 文档分类的特征提取器，其特征表示每个词是否在一个给定的文档中

all\_words =nltk.FreqDist(w.lower()for w word\_features =all\_words.keys()[:2000] def document\_features(document):

in movie\_reviews.words())

t\_s =set(document)

for word in word\_features:

features['contains(8s)'gword]=(word in document\_words) return features

>>>print document\_features(movie\_reviews.words('pos/cv957\_8737.txt')) {'contains(waste)':False,'contains(lot)':False,..}

在③中，我们计算文档的所有词的集合，而不仅仅检查单词是否在文 本当中，因为检查一个词是否在一个集合中比检查它是否在一个链表

**中出现要快得多(见4.7节).**

现在，定义特征提取器，用它来训练分类器，并为新的电影评论加标签(见例6-3)。

为了检查生成的分类器可靠性如何，可在测试集上计算其准确性”。同时，我们还

可以**使用** **show\_most\_informative\_features()来找出哪些特征是分类器发现的并且是** 最有信息量的?。

**例6-3** **训练和测试分类器以进行文档分类**

featuresets =[(document\_features(d),c)for(d,c)in documents]

train\_set,test\_set =featuresets[100:],featuresets[:100]

classifier =nltk.NaiveBayesClassifier.train(train\_set)

>>>print nltk.classify.accuracy(classifier,test\_set)①

0.81

>>>classifier.show\_most\_informative\_features(5)②

Most Informative Features

|  |  |  |  |
| --- | --- | --- | --- |
| contains(outstanding)=True | pos:neg | 二 | 11.1 :1.0 |
| contains(seagal)=True | neg :pos | = | 7.7 :1.0 |
| contains(wonderfully)=True | pos : neg | = | 6.8 :1.0 |
| contains(damon)=True | pos :neg | = | 5.9 :1.0 |
| contains(wasted)=True | neg :pos | = | 5.8 :1.0 |

显然在这个语料库中，提到 Seagal 的评论中负面的大约是正面的8倍，而提到 Damon 的评论中正面的大约是负面的6倍。

**词性标注**

在第5章中，我们建立了一个正则表达式标注器，通过查找词内部的组成，为 词选择词性标记。然而，这个正则表达式标注器是手工标注的。作为替代，我 们可以训练一个分类器来算出哪个后缀最有信息量。首先，让我们找出最常见 的后缀。

>>>from nltk.corpus import brown

>>>suffix\_fdist =nltk.FreqDist()

>>>for word in brown.words():

… · word =word.lower()

… · suffix\_fdist.inc(word[-1:])

 suffix\_fdist.inc(word[-2:])

>>>commsuofnf\_ixsf\_d)keys()[:100]

>>>print common\_suffixes

['e',',','.','s','d','t','he','n','a','of','the',

'y','r','to','in','f','o','ed','nd','is','on','l',

'g','and','ng','er','as','ing','h','at','es','or',

're','it','“','an',"'",'m',';','i','ly','ion',..]

接下来，定义一个特征提取器函数，用来检查给定单词的后缀。

>>>def word):

for suffix in common\_suffixes:

features['endswith(%s)'8suffix]=word.lower().endswith(suffix)

return features

特征提取函数的行为就像有色玻璃一样，强调数据中的某些属性(颜色),并使 其无法看到其他属性。分类器在决定如何标记输入时，将完全依赖它们所强调 的属性。在这种情况下，分类器将只基于给定词拥有的(如果有)常见后缀来 作决定。

现在，定义特征提取器，可以用它来训练新的“决策树”的分类器(将在6.4节 讨论)。

re(ltrs\*()1,) for(n,g)in tagged\_words]

>>>tagged\_words =brown.tagged\_words(categories='news')

>>>train\_set,test\_set =featuresets[size:],featuresets[:size] >>>classifier =nltk.DecisionTreeClassifier.train(train\_set)

>>>nltk.classify.accuracy(classifier,test\_set)

0.62705121829935351

>>>classifier.classify(pos\_features('cats'))

'NNS'

**决策树模型的优点是容易解释。我们甚至可以指示** **NLTK 将它们以伪代码形式**

输出。

>>>print classifier.pseudocode(depth=4)

if endswith(,)==True:return ','

if

if if

endswith(,)==False:

en:return 'AT'

if endswith(s)==True:

if endswith(is)==True:return 'BEZ'

if endswith(is)==False: return 'VBZ'

ififendst:return '.'

if endswith(.)==False:return 'NN'

分类器首先检查某个词是否以逗号结尾——如果是，它会得到一个特别的标记 “,”。接下来，分类器检查词是否以“the” 结尾，这种情况它几乎肯定是一个限 定词。这个“后缀”已被决策树使用是因为词 the 太常见了。分类器继续检查词

是否以s 结尾，如果是，那么它极有可能得到动词标记 VBZ(除非它是词 is, 特殊标记为 BEZ), 如果不是，那么它往往是名词(除非它是标点符号“.”)。 实际的分类器包含进一步嵌套的 if-then 语句，参数 depth=4 只显示决定树的顶 端部分。

**探索上下文语境**

通过增加特征提取函数，可以修改词性标注器以利用各种词内部的其他特征，例如： 词长、所包含的音节数或者前缀。然而，只要特征提取器仅仅关注目标词，我们就 没法添加特征，这些依赖于词所出现的上下文语境。然而语境特征往往提供关于正 确标记的强大线索——例如：标注词fly 时，如果知道它前面的词是 “a”, 能够确 定它是名词，而不是动词。

为了应用基于词的上下文这个特征，必须修改定义特征提取器的模式。我们不 是只传递已标注的词，而是传递整个(未标注的)句子，以及目标词的索引。 例6-4演示了这种方法，使用依赖上下文的特征提取器来定义一个词性标记分 类器。

例6-4 一个词性分类器，它的特征检测器检查一个词的上下文以便决定应该分配 哪个词性标记。特别的，前面的词也作为特征

def pos\_features(sentence,i):“

features "sun[-][-1:],

if i ==0: "suffix(3)":sentence[i][-3:]}

features["prev-word"]="<START>"

else:

features["prev-word"] =sentence[i-1]

return features

>>>pos\_features(brown.sents()[0],8)

 =[brown.tagged\_sents(categories='news')

{'suffix(3)':'ion','prev-word':'an','suffix(2)':'on','suffix(1)':'n'}

>>>for tagged sent in tagged sents:

\_

untagged\_sent

\_ =nltk.tag.untag(tagged\_sent)

\* ··· for i,(word,tag)in enumerate(tagged\_sent):

· featuresets.append(

(pos\_features(untagged\_sent,i),tag))

>>>size =int(len(featuresets)\*0.1)

>>>train\_set,test\_set =featuresets[size:],featuresets[:size] >>>classifier =nltk.NaiveBayesClassifier.train(train\_set)

>>>nltk.classify.accuracy(classifier,test\_set)

0.78915962207856782

很显然，利用上下文特征能提高词性标注器的性能。例如：如果一个词紧跟在词large 或 gubernatorial 后面，分类器判定其极可能是名词。然而，它无法研究一般的情况： 如果一个词跟在形容词后面，可能是名词，因为它无法获得前面这个词的词性标记。 在一般情况下，简单的分类器总是将每一个输入与所有其他输入独立对待。在许多 情况下，这是有用的。例如：关于名字是否倾向于男性或女性可以通过一个事件一 个事件地具体分析来做出判定。然而，很多情况下，如词性标注，这时关注的是如 何解决分类问题。

**序列分类**

为了获取相关分类任务之间的依赖关系，我们可以使用**联合分类器**模型，为一些相 关的输入选择适当的标签。在词性标注的例子中，可以使用各种不同的**序列分类器** 模型为给定的句子中的所有词选择词性标签。

一种称为**连续分类或贪婪序列分类**的序列分类器策略，为第一个输入找到最有可 能的类标签，然后在此基础上找到下一个输入的最佳的标签。这个过程可以不断 重复直到所有的输入都被贴上标签。如5.5 节中的双字母标注器，首先为句子的 第一个词选择词性标记，然后基于词本身和前面词的预测标记，为每个随后的词 选择标记。

例6-5演示了这一过程。首先，扩展特征提取函数使其具有参数 history, 其中 包括已经为句子预测的标记链表°。history 中的每个标记对应 sentence 中的一个 词。但是请注意，history 将只包含已经归类的词的标记，也就是目标词左侧的 词。因此，虽然有可能查看目标词右边词的某些特征，但查看这些词的标记是 不可能的(因为还未生成它们)。

定义特征提取器，继续建立序列分类器°。在训练中，使用已标注的标记为征提 取器提供适当的历史信息，但标注新的句子时，基于标注器本身的输出来产生历 史信息。

**例6-5 使用连续分类器进行词性标注**

def pos\_features(sentence,i,history):①

features={"1**(**3"**)**:**[**1-**]**,}

if es["prev-word"]="<START>"

features["prev-tag"]="<START>"

else:features["prev-word"]=sentence[i-l]

features["prev-tag"]=history[i-l]

return features

class ConsecutivePosTagger(nltk.TaggerI):②

def int\_s(,[]train\_sents):

for tagged\_sent in train\_sents:

t =nltk.tag.untag(tagged\_sent)

for fea,,tts\_fe\_set,i,history)

train\_set.append((featureset,tag))

history.append(tag)

self.classifier =nltk.NaiveBayesClassifier.train(train\_set)

def htgo(lfntence):

for lenfyistory)

history.append(tag)

return zip(sentence,history)

>>>tagged\_sents =brown.tagged\_sents(categories='news')

>>>size =int(len(tagged\_sents)\*0.1)

>>>train\_sents,test\_sents =tagged\_sents[size:],tagged\_sents[:size] >>>tagger =ConsecutivePosTagger(train\_sents)

>>>print tagger.evaluate(test\_sents)

0.79796012981

**其他序列分类方法**

这种方法的缺点是一旦做出决定便无法更改。例如：如果决定将一个词标注为名词， 但后来发现应该是动词，那也没有办法修复我们的错误了。解决这个问题的方法是 采取转型策略。转型联合分类的工作原理是为输入的标签创建一个初始值，然后反

复提炼该值，尝试修复相关输入之间的不一致。在5.6节描述过的 Brill 标注器，是 使用这种策略的好例子。

另一种方案是为词性标记所有可能的序列打分，选择总得分最高的序列。隐马尔可 夫模型就采取了这种方法。**隐马尔可夫模型**类似于连续分类器，不光考虑输入也考 虑已预测标记的历史。然而，不是简单地找出一个给定词的单个最好标签，而是为 标记产生一个概率分布。然后将这些概率结合起来计算标记序列的概率得分，最后 选择最高概率的标记序列。不过，可能的标签序列数量相当大。给定拥有30 个标签 的标记集，大约有600万亿(30l) 种方式来标记一个10个词的句子。为了避免单 独考虑所有这些可能的序列，隐马尔可夫模型要求特征提取器只考虑最近的标记(或 最近的n 个标记，其中n 是相当小的)。由于这种限制，它可以使用动态规划(见 4.7节)来有效地找出最有可能的标记序列。特别是，对每个连续的词索引i, 当前 的及以前的每个可能的标记都将计算得分。这种基础的方法被两个更先进的模型所 采用，它们被称为**最大熵马尔可夫模型和线性链条件随机场模型；** 但为标记序列打 分用的是不同的算法。

**6.2** **监督式分类的举例**

**句子分割**

句子分割可以看作是标点符号的分类任务：每当遇到可能会结束句子的符号时，如 句号或问号，必须决定它是否终止了当前句子。

第一步是获得一些已被分割成句子的数据，将它转换成一种适合提取特征的形式。

>>>sents =nltk.corpus.treebank\_raw.sents()

>>>tokens =[]

>>>boundaries =set()

>>>offset =0

>>>for sent in nltk.corpus.treebank\_raw.sents():

. 

 boundaries.add(offset-1)

在这里， tokens 是单独句子标识符的合并链表， boundaries 是一个包含所有句子- 边界标识符索引的集合。下一步，我们需要指定用于决定标点是否表示句子边界的 数据特征。

>>>def punct\_features(tokens,i):

return {'next-word-capitalized':tokens[i+1][0].isupper(), · 'prevword':tokens[i-1].lower(),

· ruenvc-tw':toorkdeniss e-char':len(tokens[i- 1])==1}

基于这一特征提取器，我们可以通过选择所有的标点符号创建一个加标签的特征集 链表，然后标注它们是否是边界标识符。

>>>featuresets =[(punct\_features(tokens,i),(i in boundaries))

. · for i in range(1,len(tokens)-1)

 if tokens[i]in'.?!']

**使用这些特征集，我们可以训练和评估一个标点符号分类器。**

\_=\_(fseeatturets[size:],featuresets[:size]

>>>classifier =nltk.NaiveBayesClassifier.train(train\_set)

>>>nltk.classify.accuracy(classifier,test\_set)

0.97419354838709682

使用这种分类器断句，我们只需检查每个标点符号，看它是否是边界标识符，在边界

**标识符处分割词链表。例6-6中的列表显示了这一点是如何做到的。**

**例6-6基于分类的断句器**

def segment\_sentences(words):

anr**t**s ]

for i,word in words:

if word in'.?!'and classifier.classify(punct\_features(words,

i))==True:

if start <la(o

sents.append(words[start:i+1])

sents.append(words[start:])

return sents

**识别对话行为类型**

处理对话时，要将对话看作说话者执行的动作。对于表述行为的陈述句来说，这种 解释最为简单。例如： I forgive you或 I bet you can't climb that hill。但是问候、问 题、回答、断言和说明都可以被认为是基于语言的行动类型。识别对话中隐含言语

下**的对话行为**是理解谈话的重要步骤。

在2.1节中展示过的NPS 聊天语料库，包括超过10000个来自即时消息会话的帖子。 这些帖子都已经被贴上了15种对话行为类型中的某一种标签，例如：“陈述”、“情 感”、“ynQuestion” 、“Continuer”。因此，我们可以利用这些数据建立一个分类器， 用来识别新的即时消息帖子的对话行为类型。首先是提取基本的消息数据。调用 xml\_posts()得到一个数据结构，以表示每个帖子的XML 注释。

>>>posts =nltk.corpus.nps\_chat.xml\_posts()[:10000]

然后，定义一个简单的特征提取器，用于检查帖子包含什么词。

>>>def fea\_{ct\_features(post):

· for r['iprodstwer()]=True

return features

最后，通过把特征提取器应用到每个帖子中(使用 post.get('class')获取该帖子的对 话行为类型)以构造训练和测试数据，并创建一个新的分类器。

>>>featuresets=[(dialogue\_act\_features(post.text),post.get('class'))

>>>size =int(leeetsp)]1)

>>>train\_set,test\_set =featuresets[size:],featuresets[:size] >>>classifier =nltk.NaiveBayesClassifier.train(train\_set)

>>>print nltk.classify.accuracy(classifier,test\_set)

0.66

**识别文字蕴涵**

识别文字蕴涵 **(Recognizing textual entailment,RTE)** 是判断文本T内的一个给定片段是 否继承着另一个叫做“假设”的文本(已经在1.5节讨论过)。迄今为止，已经有4个RTE 挑战赛，在那里共享的开发和测试数据会提供给参赛队伍。下面是挑战赛3开发数据集 中的文本/假设对的两个例子。标签True表示保留了蕴涵， False 表示没保留蕴涵。

Challenge 3,Pair 34(True)

T:Parviz Davudi was representing Iran at a meeting of the Shanghai Co-operation Organisation(SCO),the fledgling association that binds Russia,China and four

former Soviet republics of central Asia together to fight terrorism.

**H:** China is a member of SCO.

Challenge 3,Pair 81 (False)

T:According to NC Articles of Organization,the members of LLC company are H.Nelson Beavers,III,H.Chester Beavers and Jennie Beavers Stewart.

H:Jennie Beavers Stewart is a share-holder of Carolina Analytical Laboratory.

应当强调的是，文字和假设之间的关系并不一定是逻辑蕴涵，而是人们是否会得出 结论：文本提供的合理证据证明假设是真实的。

我们可以把RTE 当作一个分类任务，尝试为每一对预测真/假标签。虽然这项任务 的成功做法似乎看上去需要涉及语法分析、语义和知识的组合，但是 RTE 的许多 早期使用粗浅的分析(基于文字和假设之间的在词级别的相似性)进行的尝试也取 得了相当不错的结果。在理想情况下，希望如果有蕴涵，那么假设所表示的所有信 息也应该在文本中表示。相反，如果假设中有的资料文本中没有，那么就没有蕴涵。

在RTE 特征探测器(例6-7)中，我们让词(即词类型)作为信息的代理，计数词 重叠的程度和假设中有而文本中没有的词的程度(由 hyp\_extra()方法获取)。不是 所有的词都是同样重要的——提到的命名实体，如人、组织和地方的名称，可能会 更为重要，这促使我们分别为 words 和 nes (命名实体)提取不同的信息。此外， 一些高频虚词作为“停用词”被过滤掉。

例6-7 “认识文字蕴涵”的特征提取器。RTEFeatureExtractor 类建立了一个在 文本和假设中都有的并已经除去了一些停用词后的词汇包，然后计算重叠性和 差异性

def rte\_features(rtepair):

={} =nltk.RTEFeatureExtractor(rtepair)

features['word\_overlap']=len(extractor.overlap('word'))

features['word\_hyp\_extra'] =len(extractor.hyp\_extra('word')) features['ne\_overlap']=len(extractor.overlap('ne'))

features['ne\_hyp\_extra']=len(extractor.hyp\_extra('ne')) return features

为了说明这些特征的内容，检查前面显示的文本/假设对34的一些属性。

>>>rtepair =nltk.corpus.rte.pairs(['rte3\_dev.xml'])[33] >>>extractor =nltk.RTEFeatureExtractor(rtepair)

>>>print extractor.text\_words

set(['Russia','Organisation','Shanghai','Asia','four','at',

'operation','SCo',...])

>>>print extractor.hyp\_words

set(['member', 'sco','China'])

>>>print extractor.overlap('word')

set([])

>>>print extractor.overlap('ne')

set(['SCo', 'China'])

>>>print extractor.hyp\_extra('word')

set(['member'])

这些特征表明假设中所有重要的词都包含在文本中，因此有某些证据能支持标记 为True、

nltk.classify.rte\_classify 模块使用这些方法在合并的 RTE 测试数据上取得了58%以 上的准确率。这个数字并不是很令人满意的，还需要大量的努力及更多的语言学处 理，才能达到更好的结果。

**扩展到大型数据集**

Python 提供了一个良好的环境可以进行基本的文本处理和特征提取。然而，它在处 理机器学习方法所需要的密集数值型计算的速度无法达到像C 语言那么快。因此， 如果尝试在大型数据集使用纯Python 的机器学习(如nltk.NaiveBayesClassifier), 你会发现学习算法会花费大量的时间和内存。

如果你打算用大量训练数据或大量特征来训练分类器，我们建议你探索NLTK与外 部机器学习包的接口技术。只要这些软件包安装完成，NLTK 就可以调用它们(通 过系统调用)来训练分类模型，而且明显比纯 Python 的分类完成得快。请看NLTK 网站所推荐的 NLTK 支持的机器学习包列表。

**6.3** **评估**

为了判断一个分类模型是否准确地捕捉了一种模式，我们必须评估该模型。评估的 结果决定着模型的可靠度及如何使用它。评估也是一个有效的工具，用于指导将来 如何改进这个模型。

**测试集**

大多数评估技术通过将模型在**测试集(或评估集)** 中为输入生成的标签与输入 的正确标签相比较，为模型打分。该测试集通常具有与训练集相同的格式。然 而， 一定要记住测试集与训练语料是截然不同的。如果简单地重复将训练集作 为测试集，那么将导致一个只记住了输入而没有学会如何推广到新的例子的模 型得到高分。

建立测试集时，往往是可用于测试的数据量和可用于训练的数据量之间的权衡。对于 具有少量平衡的标签和多样化测试集的分类任务，只要100个评估实例就可以进行评 估。但是，如果分类任务拥有大量的标签或包含了非常罕见的标签，那么就要保证出 现次数最少的标签至少出现50次。此外，如果测试集包含许多密切相关的实例——例 如：来自单独文档中的实例——那么测试集的大小应增加到能确保这种多样性的缺 乏不会扭曲评估结果。当有大量已标注数据可用时，只使用整体数据的10%进行 评估常会出现安全问题。

选择测试集时另一个需要考虑的情况是测试集与开发集中的实例相似程度。这两个 数据集越相似，就越无法将评估结果推广到其他数据集。例如：考虑词性标注任务。 极端情况下，我们可以通过反映单一文体(如新闻)的数据源随机分配的句子，来 创建训练集和测试集。

>>>import random

>>>from nltk.corpus import brown

>>>tagged\_sents =list(brown.tagged\_sents(categories='news')) >>>random.shuffle(tagged\_sents)

>>>size =int(len(tagged\_sents)\*0.1)

>>>train\_set,test\_set =tagged\_sents[size:],tagged\_sents[:size]

在这种情况下，测试集和训练集将是非常相似的。训练集和测试集均取自同一文体， 所以我们不能确信评估结果可以推广到其他文体。更糟糕的是，因为调用 random.shuffle(),测试集中包含来自训练中使用过的文档的句子。如果文档中有相 容的模式(也就是说，如果给定的词与特定词性标记一起频繁出现),那么这种差 异将同时体现在开发集和测试集之中。更好的做法是确保训练集和测试集是来自不 同的文件。

>>>file\_ids =brown.fileids(categories='news')

>>>size =int(len(file\_ids) \*0.1)

>>>train\_set

>>>test\_set

=brown.tagged\_sents(file\_ids[size:])

=brown.tagged\_sents(file\_ids[:size])

如果我们要进行更令人信服的评估，可以从与训练集文档联系较少的文档中获取测

试集。

>>>train\_set =brown.tagged\_sents(categories='news')

>>>test\_set =brown.tagged\_sents(categories='fiction')

如果在此测试集上建立了一个性能更好的分类器，那么就可以认为它在训练数据以

外也能得到很好的推广。

**准确度**

用于评估分类的最简单度量是**准确度，**测量测试集上分类器正确标注的输入比例。 例如：名字性别分类器，在包含80个名字的测试集上预测正确的名字有60个，它 的准确度为60/80=75%。nltk.classify.accuracyO 函数会在给定的测试集上计算分类 器模型的准确度。

>>>classifier =nltk.NaiveBayesClassifier.train(train\_set)

>>>print 'Accuracy:84.2f'8 nltk.classify.accuracy(classifier,test\_set)

0.75

解释一个分类器的准确度时，需要把测试集中单个类标签的频率考虑进去。例如： 能判断出词 bank 每次出现时的正确词意的分类器。如果我们在金融新闻专线文本 上评估此分类器，那么我们会发现，20次里面有19次是“金融机构”的意思。在 这种情况下，95%的准确度也难以令人信服，因为模型总是能返回“金融机构”的 意思。然而，如果我们在一个更加平衡的语料库上评估分类器，那里最频繁的词意 只占40%,那么95%的准确度得分将是一个非常好的结果。(在11.2节测量相互标 注器一致性程度时也会有类似的问题。)

**精确度和召回率**

准确度分数可能会产生的误导是在“搜索”任务中，如：信息检索，我们试图找出 与特定任务有关的文档。由于不相关的文档的数量远远多于相关文档的数量，将每 一个文档都标记为无关的模型，其准确度分数将非常接近100%。

因此，基于以下4个类别中每一项目的数量使用不同的测量集搜索任务，如图6-3 所示的。

● **真阳性**是相关项目中正确识别为相关的。

· **真阴性**是不相关项目中正确识别为不相关的。

● **假阳性** ( 或**I** **型错误)** 是不相关项目中错误识别为相关的。

● **假阴性**( 或**Ⅱ型错误)** 是相关项目中错误识别为不相关的。

系统输出： 小硬盘的文档

需要的信息： 相关的文档

**图6-3** **真与假的阳性和阴性**

|  |  |
| --- | --- |
| 真  积极的  相关，恢复的  X  假  积极的  相关，未恢复的 | X  假  积极的  不相关，恢复的  真  积极的  不相关，未恢复的 |

文档集合

给定以下4个数字，并定义以下指标。

● **精确度** (Precision), 表示发现的项目中有多少是相关的， TP/(TP+FP)。

● **召回率** **(Recall),** 表示相关的项目中发现了多少， TP/(TP+FN) 。

● **F-度量值**(F-Measure) ( 或F-得 分 ，F-Score),组合精确度和召回率为一个单 独的得分，被定义为精确度和召回率的调和平均数(2×Precision×Recall)/Precision+ Recall)。

**混淆矩阵**

当处理有3个或更多标签的分类任务时，可基于模型错误类型来细分模型的错误。混 **淆矩阵**是一个表，其中每个元素[i,j] 表示正确的标签i 被预测为标签j 的次数。因此，

**对角线元素(即** **cells[i,i]) 表示正确预测的标签，非对角线项目表示错误。在下面的例** **子中，我们为5.4节中开发的unigram 标注器生成一个混淆矩阵。**

>>>def tag\_list(tagged\_sents):

return [tag for sent in tagged\_sents for(word,tag)in sent]

>>>def apply\_tagger(tagger,corpus):

.… … return [tagger.tag(nltk.tag.untag(sent))for sent in corpus]

>>>gold =tag\_list(brown.tagged\_sents(categories='editorial'))

>>>test =tag\_list(apply\_tagger(t2,

brown.tagged\_sents(categories='editorial')))

>>>cm =nltk.ConfusionMatrix(gold,test)

N N

I

N

A

T

J

J

N

N

S

V

B

N

P

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NN | <11.88> 0.0% | 0.2名 | 0.0% | 0.3% | 0.0% |
| IN | 0.08 <9.0% |  | 0.0% |  |  |
| AT  JJ | 1.6% | <8.68  <4.0%>  <4.8%> |  | 0.0% | 0.0名 |
| NS | 1.5% |  | <3.2%> | <4.4%> | 0.0号 |
| B | 0.9名 | 0.0名 |  | 2.48> |  |
| NP | 1.0% | 0.0% |  |  | <1.98> |

(row =reference;col =test)

这个混淆矩阵显示了常见的错误，包括NN 替换为了 JJ(1.6% 的词),NN 替换为 了NNS(1.5% 的词)。注意点(.)表示值为0的单元格，对角线元素——对应正确 的分类——用尖括号标记。

**交叉验证**

为了评估模型，我们必须为测试集保留一部分已标注的数据。正如我们已经提到， 如果测试集太小了，我们的评价可能不准确。然而，测试集设置较大通常意味着 训练集将设置得较小，如果已标注数据的数量有限，这样设置会对性能会产生重 大影响。

这个问题的解决方案之一是在不同的测试集上执行多重评估，然后组合这些评估的得 分，这种技术被称为**交叉验证。**我们将原始语料细分为 N 个子集称**为折叠**(folds) 。 对于每一个折叠，我们使用除这个折叠中的数据外的其他所有数据来训练模型，然后 在这个折叠上测试模型。即使个别的折叠可能因太小了而不能为其自身给出准确的

评价分数，但是综合评估得分是基于大量的数据得到的，因此是相当可靠的。

第二点也同样重要，采用交叉验证的优势是，可以研究不同的训练集上性能变化有 多大。如果能从所有 N 个训练集得到非常相似的分数，那么认为得分的准确性是 可信的。另一方面，如果 N 个训练集上分数有很大不同，那么,我们应该对评估 得分的准确性持怀疑态度。

**6.4** **决策树**

接下来的三节中，我们将仔细研究可用于自动生成分类模型的3种机器学习方法： 决策树、朴素贝叶斯分类器和最大熵分类器。正如我们所看到的，可以把这些学习 方法看作黑盒子，直接训练模型，然后使用它们进行预测而不需要理解它们是如何 工作的。但是，仔细研究这些学习方法是如何基于一个训练集上的数据来选择模型 的，我们能学到很多的知识。了解这些方法可以帮助我们选择相应的特征，尤其是 这些特征该如何编码。理解生成的模型可以让我们更好地提取信息：哪些特征包含 最大的信息量及那些特征之间如何相互关联。

**决策树是**一个为输入值选择标签的简单流程图。这个流程图由检查特征值**的决策节** 点和分配标签的**叶节点**组成。为输入值选择标签，我们以流程图的初始决策节点 (称为其**根节点)**开始。此节点包含一个条件，检查其中某个输入值的特征，基于 该特征的值选择一个分支。沿着这个描述输入值的分支，每到达一个新的决策节 点，就会产生关于输入值特征的新条件。我们再继续沿着每个节点的条件选择出 的分支，直到到达为输入值提供标签的叶节点。图6-4所示为名字性别任务的决 策树模型。

|  |  |  |
| --- | --- | --- |
|  | astletter=vowel?  firstletter=k? lastletter=o?  astletter=I? astietter=t? count(f)=2? ength=3?  M F F M F M M F |  |

**图6-4** **名字性别任务的决策树模型。请注意，树图是按照习惯“颠倒”画出的，根在上面，叶** **子在下面**

一 旦我们有了决策树，就可以直接用它为新的输入值分配标签。问题是我们 如何能够建立决策树为给定的训练集建模。在研究建立决策树的学习算法之 前，先思考一个简单的任务：为语料库选择最好的“决策树桩 ”。**决策树桩**是 只有 一个节点的决策树，基于单个特征来决定如何为输入分类。每个可能的 特征值包含一个叶子，并为特征输入指定的类标签。为了建立决策树桩，首 先必须决定哪些特征应该使用。最简单的方法是为每个可能的特征建立一个 决策树桩，看哪一个在训练数据上得到的准确度最高，其他的替代方案我们 将在下面讨论。 一 旦选择了一个特征，就可以基于在训练集中所选例子最频 繁的标签，通过分配 一个标签给每个叶子，建立决策树桩(即在例子中所选 特征都有那个值)。

给出选择决策树桩的算法，生长出较大的决策树的算法就变得简单了。首先，选择 分类任务的整体最佳的决策树桩。然后，在训练集上检查每个叶子的准确度。没有 达到足够准确度的叶片会被新的决策树桩替换，新决策树桩是在训练语料的子集上 训练的，这些训练语库都是根据到叶子的路径来选择的。例如：我们可以通过替换 最左边的叶子为新的决策树桩来生成如图6-4所示的决策树，这个新的决策树桩是 在训练集的子集中训练出来的，训练集的名字指定不以 k 开始或以一个元音或1 结尾。

**熵和信息增益**

正如之前提到的，可通过几种方法来确定决策树桩最有信息量的特征。 一种流行的 方法，被称为**信息增益，**当利用给定的特征分割输入值时，信息增益能衡量输入值 的有序程度。要衡量原始输入值集合如何无序，要计算它们的标签的墒，如果输入 值的标签非常不同，墒就高；如果输入值的标签都相同，墒就低。特别是，熵被定 义为每个标签的概率乘以该标签的 log 概率的总和。

(1)H=Z/∈labelsP(1)×log₂P(1).

例如：图6-5显示了在名字性别预测任务中标签的墒是如何由男性名字与女性名字 的比例所决定的。请注意，如果大多数输入值具有相同的标签(例如，如果P(male) 接近0或接近1),那么墒很低。特别的，低频率的标签对墒不会有多大帮助(因 为P(1)很小),高频率的标签对熵也没有多大帮助(因为log2P(1)很小)。另一方面， 如果输入值的标签变化很多，那么有很多“中等”频率的标签，它们的P(I)和 log2P(1)

**都不小，所以墒很高。例6-8演示了如何计算标签链表的墒。**

|  |
| --- |
|  |

**图6-5** **在名字性别预测任务中标签的墒，在给定集合中，作为名字是男性的百分比函数**

**例6-8** **计算标签链表的墒**

import math

def l[bes) in nltk.FreqDist(labels)]

return -sum([p \*math.log(p,2)for p in probs])

>>>print entropy(['male','male','male','male'])

0.0

>>>print entropy(['male','female','male','male'])

0.811278124459

>>>print

1.0

>>>print

0.81 1278124459 >>>print

0.0

entropy(['female','male','female','male'])

entropy(['female','female','male','female'])

entropy(['female','female','female','female'])

一旦计算出原始输入值的标签集的墒，就可以判断应用决策树桩之后标签会变得多 么有序。可以计算每个决策树桩的叶子的熵，并利用这些叶子熵值的平均值(加权 每片叶子的样本数量)。信息增益等于原来的熵减去新减少的熵。信息增益越高， 将输入值分为相关组的决策树桩性能就越好，于是可以通过选择具有最高信息增益 的决策树桩来建立决策树。

决策树的另一个考虑因素是效率。前面描述的选择决策树桩的简单算法必须为每一个 可能的特征构建候选决策树桩，并且这个过程必须在构造决策树的每个节点上不断重 复。通过存储和重用先前评估的例子的信息开发出一些算法以减少训练时间。

决策树具有一些有用的性质。首先，它们简单明了，容易理解。决策树顶部附近尤 其如此，通过学习算法就可以找到非常有用的特征。决策树特别适合有很多层次分 类区别的情况。例如：决策树可以非常有效地捕捉进化树。

然而，决策树也有一些缺点。其中是，由于决策树的每个分支会划分训练数据，在 训练树的低节点中，可用的训练数据量可能会变得非常小。因此，这些较低的决策 节点可能会**过拟合**训练集，学习模式反映的是训练集的特质而不是问题底层重要的 语言学模式。对于这个问题的解决方案是当训练数据量变得太小时停止分裂节点。

另一种方案是长出一个完整的决策树，但随后进行剪枝剪去在开发测试集上不能提 高性能的决策节点。

决策树的第二个缺点是，它们强迫特征按照特定的顺序进行检查，即便特征可能是 相对独立的。例如：按主题分类文档(如体育、汽车或谋杀之谜)时，特征如 hasword(football), 极可能表示一个特定标签，无论其他的特征值是什么。由于决 定树顶部附近的空间有限，大部分这些特征将需要在树中的许多不同的分支中重 复。因为越往树的下方，分支的数量成指数倍增长，重复量可能非常大。

相关的问题是决策树不善于利用对正确的标签具有较弱预测能力的特征。由于这些 特征的渐进式改进相对较小，它们往往出现在决策树非常低的地方。但是到决策树 学习者减少到远远不够用到这些特征的时候，也不具备足够的训练数据来可靠地确 定它们应该有什么样的影响。如果我们能够换过来，在整个训练集中看看这些特征 的影响，那么我们也许能够做出一些关于它们是如何影响标签选择的结论。

决策树需要按特定的顺序检查特征的事实，这限制了它们利用相对独立特征的能 力。下面将讨论的朴素贝叶斯分类方法打破了这一限制，允许所有特征“并行”地 起作用。

**6.5** **朴素贝叶斯分类器**

**在朴素贝叶斯**分类器中，每个特征都能得到发言权，以确定哪个标签应该被分配到 给定的输入值中。为输入值选择标签，朴素贝叶斯分类器以计算每个标签的**先验概** 率开始，它通过检查训练集上的每个标签频率来确定。之后，每个特征的贡献与它 的先验概率结合，可得到每个标签的似然估计。似然估计最高的标签会分配给输入 值。图6-6说明了这一过程。

footbalr

suods

mystery

suods

mystery

suods

mystery

murder

automotive

murder

automotive

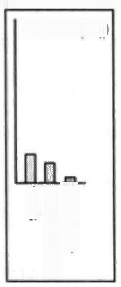
murder

automotive

sods

murder mystery

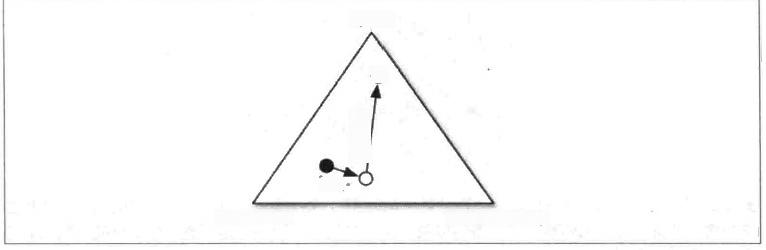
automotive



**Label** **Likelihoods**

P(label,f,\_f)

■



Sports

Q

dark

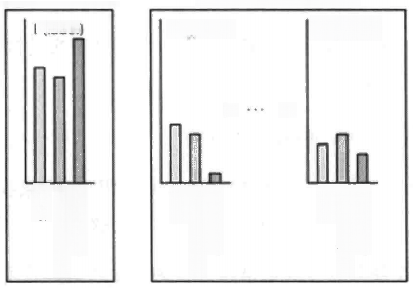
Automotive

Murder Mystery



**图6-6** **由朴素贝叶斯分类器所使用，为文档选择主题程序的抽象图解。在训练语料中，大多数文档是** **有关汽车的，所以分类器从接近“汽车”的标签的点上开始。但它会考虑每个特征的影响。在这个例子** **中，输入文档中包含的词dark,它是“谋杀之谜”的一个较弱的指标，也包含词football, 它是体育文** **档的一个有力指标。每个特征都作出贡献之后，分类器将检查哪个标签最接近，并将该标签分配给输入**

个别特征通过“投票反对”那些不经常出现的特征标签来对整体决策作出自己的贡 献。特别是，每个标签的似然得分因与输入值具有此特征标签的概率相乘而减小。 例如：如果词run 在12%的体育文档中出现，在10%的谋杀之谜的文档中出现，在 2%的汽车文档中出现，那么体育标签的似然得分将乘以0.12,谋杀之谜标签的似 然得分将乘以0.1,汽车标签的似然得分将乘以0.02。整体效果是：谋杀之谜标签 的得分会稍微比体育标签的得分减少得多一些，而汽车标签相对于其他两个标签会 显著减少。这个过程如图6-7和图6-8所示。



Prior Probabilities **Feature** **Contributions**

P(label P(f,llabel) P(f,label)

X

X

X

图6-7 计算朴素贝叶斯标签的似然得分。基于每个标签出现在训练数据中的频率，朴素贝叶斯 以计算每个标签的先验概率为开端。然后每个特征都用于计算每个标签的似然估计，通过用输入 值中有该特征的标签概率与之相乘可得到。似然得分结果可以认为是从给定的标签和特征集的训 练集中随机选取出来的值的概率估计，假设所有特征概率都是独立的



|  |  |
| --- | --- |
| label] | |
|  | f, |

**图6-8** **贝叶斯网络图演示朴素贝叶斯分类器假定的生成过程。要生成一个标记的输入，模型首** **先为输入选择标签，然后基于该标签生成每个输入的特征。对于给定标签，每个特征都被认为是** **完全独立于所有其他特征的**

**潜在概率模型**

理解朴素贝叶斯分类器的另一种方式是它为输入选择最有可能的标签，基于下面的 假设：每个输入值是通过首先为该输入值选择类标签，然后产生每个特征的方式产 生的，每个特征与其他特征完全独立。当然，这种假设是不现实的，特征彼此之间 往往高度依赖。我们将在本节结尾讨论这个假设的一些后果。这个简化的假设，称 **为朴素贝叶斯假设**( **或独立性假设**),它可以更容易地组合不同特征的贡献，因为 我们不必担心它们之间会相互影响。

基于这个假设，我们可以计算表达式P(label|features), 即一个具有特定标签(假定 具有特定的特征集)的输入概率。要为新的输入选择标签，我们可以简单地选择能 使P(Ifeatures) 最大的标签1。

首先，我们注意到P(label|features)等于具有特定标签和特定特征集的输入的概率除 以具有特定特征集的输入的概率。

(2)P(label|features)=P(features,label)/P(features)

接下来，我们注意到P(features)对每个标签选择都是相同的。因此，如果我们只是对寻 找最有可能的标签感兴趣，只需计算 P(features,label)就够了，我们称之为似然标签。

如果我们想生成每个标签的概率估计，而不是只选择最有可能的标

签，那么计算P(features)的最简单的方法是仅仅计算P(features,label) 在所有标签上的总和。

(3)P(features)=Zlabelelabels P(features,label)

似然标签可以展开为标签的概率乘以给定标签特征的概率。

(4)P(features,label)=P(label)×P(features|label)

此外，由于特征都是独立的(给定标签),我们可以分离每个独立特征的概率。

(5)P(features,label)=P(label)×IIre featuresP(f}label)

这正是我们前面讨论的用于计算标签可能性的方程式： P(label) 是一个给定标签的 先验概率，每个P(fYlabel)是单个特征对标签可能性的贡献。

**零计数和平滑**

计算 P(flabel) 最简单的方法，特征f 对标签 label 的标签似然性的贡献，是取得具 有给定特征和给定标签的训练实例百分比。

(6)P(flabel)=count(f,label)/count(label)

然而，当训练集中存在从来没有和给定标签一起出现的特征时，这种简单的方法会产生 一个问题。在这种情况下，P(flabel) 计算值将是0,这将导致给定标签的标签可能性为0。 从而，输入将永远不会被分配给这个标签，不管其他特征有多么适合这个标签。

这里的基本问题与计算P(f)label)有关，是对于给定标签，输入将具有特征的概率。 特别的，如果我们在训练集中没有看到特征/标签组合出现，并不意味着该组合不 会出现。例如：我们可能不会看到任何谋杀之迷中包含词 football 的文档，但我们 不希望作出结论认为这种文档是完全不可能存在的。

虽然当 count(f;label)相对高时，count(f,label)/count(label) 是估计 P(flabel) 的一个好方 法，当 count(f)变小时这个估计就变得不那么可靠。因此，建立朴素贝叶斯模型时， 通常采用更复杂的技术，称为平滑技术，用于计算P(flabel), 在给定标签的特征的 概率的基础上。例如：给定标签的特征概率的**期望似然估计**基本上会给每个 count(f,label)值增加0.5,**Heldout** 估计使用heldout 语料库计算特征频率与特征概率 之间的关系。nltk.probability 模块提供了多种平滑技术支持。

**非二元特征**

这里假设每个特征都是二元的，即每个输入要么有这个特征要么没有。标签值特征

(例如：颜色特征，可能有红色、绿色、蓝色、白色或橙色)可通过用二元特征， 如“颜色是红色”,替换它们，将它们转换为二元特征。数字特征可以通过**装箱转** 换为二元特征，即用特征，如“4<X<6”, 来替换它们。

另一种方法是使用回归方法来模拟数字特征的概率。例如：如果假设高度特征具有 贝尔曲线分布，那么就可以通过找到每个标签输入的高度均值和方差来估算 P(height|label)。在这种情况下， P(f=v|label) 将不会是一个固定值，会依据v 的值来 变化。

**独立的朴素性**

朴素贝叶斯分类器被称为“naive(天真、朴素)”的原因是它不切实际地假设所有 特征相互独立(给定标签)。特别的，几乎所有现实世界的问题含有的特征都会不 同程度地彼此依赖。如果要避免任何依赖其他特征的特征，那将很难构建良好的能 为机器学习算法提供所需信息的功能集。

如果忽略了独立性假设，使用特征不独立的朴素贝叶斯分类器会发生什么?产生的 问题是分类器“双重计数”高度相关的特征的影响，将分类器推向更接近给定的标 签而非合理的标签。

来看看这种情况怎么出现的，考虑包含两个相同特征fl和 f2 的名字性别分类器。 换句话说，f2 是 fl 的精确副本，不包含任何新的信息。当分类器考虑输入，在决 定选择哪一个标签时，它会同时包含fl 和 f2 的贡献。因此，这两个特征的信息内 容比重将会比它们应得的还要多。

当然，我们通常不会建立包含两个相同特征的朴素贝叶斯分类器。不过，我们会建 立包含相互依赖特征的分类器。例如：特征 ends-with(a) 和 ends-with(vowel) 是彼此 依赖，因为如果一个输入值有第一个特征，那么它也必有第二个特征。对于这些特 征，重复的信息可能会被训练集赋予比合理情况更多的比重。

**双重计数的原因**

双重计数问题出现的原因是在训练过程中特征的贡献被分开计算，但当使用分类器 为新输入选择标签时，这些特征的贡献被组合在一起了。因此，解决方案是考虑在 训练中特征的贡献之间可能的相互作用。然后，就可以使用这些相互作用来调整独

立特征所作出的贡献。

为了使结果更精确，我们可以重写用于计算标签似然性的方程，分离出每个功能 (或标签)所作出的贡献。

(7)P(features,label)=w[label]×IIre features W[f,label]

在 这 里 ，w[label] 是一个给定标签的“初始分数”,w[f,label] 是给定特征对一个标签 的似然性所作的贡献。我们称这些值 w[label]和 w[f,label]为模型的参数或权重。使 用朴素贝叶斯算法，我们单独设置这些参数。

(8)w[label]=P(label)

(9)w[f,label]=P(f)label)

然而，在下一节中，我们将学习另一个分类器，它在选择这些参数的值时会考虑它 们之间可能的相互作用。

6.6 最大熵分类器

最大熵分类器使用的模型与朴素贝叶斯分类器使用的非常相似。没有使用概率设置 模型的参数，而是使用搜索技术找出一组能最大限度地提高分类器性能的参数。特 别的，它查出能使训练语料的整体似然性最大的参数组。其定义如下。

(10)P(features)=Zx∈corpus P(label(x)|features(x))

其中P(label|features), 即特征为 features 的输入且类标签为 label 的概率，被定义为：

(11)P(label|features)=P(label,features)/Zlabel P(label,features)

由于在相关特征的影响之间潜在的复杂的相互作用，没有办法直接计算能最大限度 地提高训练集可能性的模型参数。因此，最大熵分类器采用迭代优化技术选择模型 参数，该技术利用随机值初始化模型的参数，然后反复优化这些参数，使它们更接 近最优解。这些迭代优化技术保证每次参数优化都会使它们更接近最佳值，但不一 定能提供方法来确定是否已经达到最佳值。由于最大熵分类器使用迭代优化技术选 择参数，花费的时间很长。当训练集的大小、特征的数目及标签的数目都很大时尤 其如此。



一些迭代优化技术比别的快得多。当训练最大熵模型时，应避免使用 广义迭代缩放(Generalized Iterative Scaling,GIS) 或改进的迭代缩放 (Improved Iterative Scaling,IIS), 这两者都比共轭梯度 (Conjugate Gradient,CG) 和 BFGS 优化方法慢很多。

**最大熵模型**

最大熵分类器模型是朴素贝叶斯分类器模型的泛化模型。像朴素贝叶斯模型一样， 最大熵分类器通过将适合于输入值和标签的参数乘在一起，为给定的输入值计算 每个标签的可能性。朴素贝叶斯分类器模型为每个标签定义一个参数，指定其先 验概率，为每个(特征，标签)对定义一个参数，为标签的似然性指定其独立特 征的贡献。

相比之下，最大熵分类器模型让用户来判断什么样的标签和特征组合应该得到自己 的参数。特别的，它可以使用一个单独的参数将一个特征与一个以上的标签关联起 来；或者将一个以上的特征与一个给定的标签关联起来。这有时会允许模型“概括” 相关的标签或特征之间的一些差异。

每个接收它自身参数的标签和特征的组合被称**为联合特征。**请注意，**联合特征**是加 标签值的属性，而(简单)特征是未加标签值的属性。

描述和讨论最大熵模型的文字中，术语“特征 features” 往往是指联合 特征；术语“上下文contexts”是指我们一直说的(简单)特征。

通常情况下，用来构建最大熵模型的联合特征完全反映了朴素贝叶斯模型所使用的联 合特征。特别的，每个标签定义的联合特征对应于 w[label], 每个(简单)特征和标 签组合定义的联合特征对应于w[f,label]。给定一个最大熵模型的联合特征，分配到给 定输入标签的得分仅仅是适用于该输入和标签的联合特征与参数之间的简单乘积。

(12)P(input,label)=ljoim-features(input,abel)W[joint-feature]

**熵的最大化**

进行最大熵分类的步骤是建立一个模型，捕捉单独的联合特征的频率，而不必作任

何无根据的假设。举一个例子将有助于说明这一原则。

假设从10个可能词意列表(标签从A-J) 中为一个给定的词找出正确词意。首先， 我们没有被告知其他任何关于词或词意的信息。我们可以为10种词意选择的概率 分布很多，例如：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** |
| (i) | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| (ii) | 5% | 15% | 0% | 30% | 0% | 8% | 12% | 0% | 6% | 24% |
| (iii) | 0% | 100% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |

虽然这些分布都可能是正确的，但我们最可能会选择的是分布 (i), 因为没有任何 更多的信息，也没有理由相信任何词的词意比其他的更有可能。另一方面，分布(ii) 及 (iii) 反映的假设并不被我们已知的信息支持。

直觉上分布 (i) 比其他的更“公平”,解决方法是引用熵的概念。在决策树的 讨论中，我们描述了熵作为衡量一套标签是如何“无序”的。特别的，如果是 一个单独的标签则熵较低，但如果标签的分布比较均匀则熵较高。在上述例子 中，我们选择了分布 (i) 因为它的标签概率分布均匀——换句话说，因为它 的熵较高。 一般情况下，**最大熵原理**是指在已知的分布下，我们会选择熵最高 的分布。

接下来，假设被告知词意A 出现的次数占55%。还有许多分布适合于这条新信息， 例如：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | **B** | **C** | **D** | E | F | G | H | I | J |
| (iv) | 55% | 45% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| (v) | 55% | 5% | 5% | 5% | 5% | 5% | 5% | 5% | 5% | 5% |
| (vi) | 55% | 3% | 1% | 2% | 9% | 5% | 0% | 25% | 0% | 0% |

但是，我们可能会选择最无根据的假设分布——在这种情况下，分布 (v)。

最后，假设被告知词up 出现在附近上下文中的次数占10%,当它出现在这个上下 文中时有80%的可能使用词意 A 或 C。在这种情况下，我们很难手工找到合适的 分布；然而，可以验证下面看起来适当的分布。

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A | **B** | C | **D** | E | F | G | **H** | I | J |
| (vii) | +up | 5.1% | 0.25% | 2.9% | 0.25% | 0.25% | 0.25% | 0.25% | 0.25% | 0.25% | 0.25% |
|  | -up | 49.9% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% | 4.46% |

特别的，这个分布与我们所知道的一致：如果我们将A 列的概率加起来，是55%, 如果我们将第1行的概率加起来，是10%;如果我们将+up 行词意A 和 C 的概率 加起来，是8%(或+up 情况的80%)。此外，其余的概率好像是“均匀分布”的。

纵观这个例子，我们将自己限制在已知的分布上。其中，我们选择最高熵的分布。 这也正是最大熵分类器的作用。特别的，对于每个联合特征，最大熵模型计算该特 征的“经验频率”——即它出现在训练集中的频率。然后，搜索能使熵最大的分布， 同时预测每个联合特征正确的频率。

**生成式分类器对比条件式分类器**

朴素贝叶斯分类器和最大熵分类器之间的一个重要差异是使用它们回答问题的类 型。朴素贝叶斯分类器是一个**生成式**分类器的例子，建立一个模型预测 P(input , label), 即(input,label)对的联合概率。因此，生成式模型可以用来回答下列问题。

(1)一个给定输入的最可能的标签是什么?

(2)对于一个给定输入， 一个给定标签有多大可能性?

(3)最有可能的输入值是多少?

(4)一个给定输入值的可能性有多大?

(5)一个给定输入具有一个给定标签的可能性有多大?

(6)对于一个可能有两个值的输入(但我们不知道是哪个),最可能的标签是什么?

另一方面，最大熵分类器是**条件式**分类器的一个例子。条件式分类器建立模型预测 P(labellinput)——一个给定输入值的标签概率。因此，条件式模型仍然可以被用来 回答问题1和2。然而，条件式模型不能用来回答剩下的问题3～6。

一般情况下，生成式模型确实比条件式模型强大，因为可以从联合概率P(input, label) 计算出条件概率 P(labellinput), 但反过来不行。然而，这种额外的功能

是要付出代价的。由于该模型更强大，它也有更多的“自由参数”需要学习的。 而训练集的大小是固定的。因此，使用更强大的模型时，可用来训练每个参数 值的数据也更少，使其难以找到最佳参数值。结果是这个生成式模型回答问题 1和2的效果可能不如条件式模型好，因为条件式模型可以集中精力在这两个 问题上。然而，如果我们确实需要回答问题3～6,那么我们别无选择，只能 使用生成式模型。

生成式模型与条件式模型之间的差别类似与一张地形图和一张地平线的图片之间 的区别。虽然地形图可用于回答的问题更广泛，但制作一张精确的地形图也明显比 制作一张精确的地平线图片更加困难。

**6.7** **为语言模式建模**

分类器可以帮助我们理解自然语言中存在的语言模式，允许我们建立明确的**模型以** 捕捉这些模式。通常情况下，这些模型使用监督式分类技术，但也可以建立分析型 激励模型。无论哪种方式，这些明确的模型有两个重要目的：帮助我们了解语言模 式，可以被用来预测新的语言数据。

明确的模型可以让我们洞察语言模式，这些模式在很大程度上取决于使用哪种模 型。 一些模型，如决策树，相对透明，直接给我们提供信息：哪些因素是决策中重 要的，哪些因素是彼此相关的。另一些模型，如多级神经网络，比较不透明。虽然 有可能通过研究洞察它们，但通常需要大量的工作。

但是，所有明确的模型都可以预测新的**未见过**的语言数据，这些并未包括在用 来构建模型的语料库中。对这些预测进行评价可评估模型的准确性。 一旦模型 被认为足够准确，它就可以被用来自动预测新的语言数据信息。这些预测模型 可以组合成系统，执行很多有用的语言处理任务，例如：文档分类、自动翻译、 问答系统。

**模型告诉我们什么?**

我们需要理解从自动构建的模型中学到的语言的什么。处理语言模型时的重要考虑 因素是描述性模型与解释性模型之间的区别。描述性模型捕获数据中的模式，但 它们并不提供任何有关数据包含这些模式的原因。例如：我们在表3.1中看到的，

同义词 absolutely 和 definitely 是不能互换的：我们说absolutely adore 而不是 definitely adore,definitely prefer 而不是 absolutely prefer。与此相反，解释性模型试图捕捉造 成语言模式的属性和关系。例如：我们可能会引入一个抽象概念“极性形容词”即 具有极端意义的形容词，并对一些形容词进行分类，如：adore 和 detest 是相反的 两极。解释性模式将包含的约束为：absolutely 只能与极性形容词结合， definitely 只能与非极性形容词结合。总之，描述性模型提供数据内相关性的信息，而解释性 模型是进一步假设因果关系。

大多数从语料库自动构建的模型是描述性模型。换句话说，它们可以告诉我们哪些 特征与给定的模式或结构有关，但它们不一定能告诉我们这些特征和模式之间如何 关联。如果我们的目标是理解语言模式，那么就可以使用哪些特征是相关的这一信 息作为出发点，设计实验进一步弄清特征与模式之间的关系。另一方面，如果我们 只是对利用该模型进行预测，例如：作为语言处理系统的一部分，那么可以使用该 模型预测新的数据，而不用担心潜在的因果关系。

**6.8** **小结**

· 在语料库中的语言数据建模，可以帮助我们理解语言模型，也可以用于预测新 语言数据。

· 监督式分类器使用加标签的训练语料库来建立模型，该模型可基于输入的特 征，预测该输入的标签。

● 监督式分类器可以执行多种NLP 任务，包括文档分类、词性标注、语句分割、 对话行为类型识别、确定蕴涵关系及很多其他任务。

● " 训练监督式分类器时，应该把语料分为3个数据集：用于构造分类器模型的训 练集，用于帮助选择和调整模型特性的开发测试集，以及用于评估最终模型性 能的测试集。

● 评估监督式分类器时，应使用没有包含在训练集或开发测试集中的新数据。否 则，你的评估结果可能会不切实际地乐观。

● 决策树是自动构建树结构的流程图，用于基于输入值的特征添加标签，虽然易

于解释，但不适合在决定合适标签过程中处理相互影响的特性值。

· 在朴素贝叶斯分类器中，每个特征独立地决定应该使用哪个标签。它允许特征 值间有关联，但当两个或更多的特征高度相关时将会出现问题。

· 最大熵分类器使用的基本模型与朴素贝叶斯相似；不过，它们使用迭代优化寻 找使训练集概率最大化的特征权值集合。

● 大多数从语料库自动构建的模型都是描述性的，也就是说，它们可以让我们知 道哪些特征与给定的模式或结构相关，但它们无法给出关于这些特征和模式之 间的因果关系的任何信息。

**6.9** **深入阅读**

关于本章和如何安装外部机器学习包，如 Weka、Mallet、TADM 和 MegaM 的进一 步的材料请查询<http://www.nltk.org/>。更多使用NLTK 进行分类和机器学习的例子， 请参阅在 <http://www.nltk.org/howto> 的分类HOWTO。

有关机器学习的一般介绍我们推荐 (Alpaydin,2004) 。 更多机器学习理论的数学 专著,参见 (Hastie,Tibshirani&Friedman,2009) 。 使用机器学习技术进行自然 语言处理的优秀图书包括： (Abney,2008) 、(Daelemans &Bosch,2005) 、(Feldman

&Sanger,2007) 、(Segaran,2007) 和 (Weiss et al.,2004)。语言问题上有关平滑

技术的更多信息，请参阅 (Manning &Schütze ,1999)。 序列建模尤其是隐马尔可 夫模型的更多信息，请参阅 (Manning& Schütze,1999) 或 (Jurafsky &Martin, 2008) 。(Manning,Raghavan &Schütze,2008) 中的第13章讨论了如何利用朴素 贝叶斯分类文本。

在本章中讨论的机器学习算法大都是数值计算密集的，使用纯Python编码时它们运行 很慢。关于如何提高Python中的数值密集型算法效率，请参阅 (Kiusalaas,2005)。

本章中所描述的分类技术应用范围可以非常广泛。例如：(Agirre&E dmonds,2007) 使用分类器处理词义消歧； (Melamed,2001) 使用分类器创建平行文本。最近包括文 本分类的教科书有(Manning,Raghavan &Schütze,2008)和(Croft,Metzler &Strohman, 2009)。

目前机器学习技术应用到 NLP 问题的研究大部分都是由政府主办的“挑战赛”来 驱动的，他们向研究机构提供相同的开发语料库，要求建立一个系统，并使用保留 的测试集比较系统的结果。这些挑战比赛包括 CoNLL 共享任务 (CoNLL Shared Tasks)、文字蕴含识别比赛 (Recognizing Textual Entailment competitions),ACE 比 赛 (ACE competitions) 和 AQUAINT比赛 (AQUAINT competitions)。这些挑战的 网页链接列表请参阅 <http://www.nltk.org/>。

**6.10** **练习**

1.O 阅读在本章中提到的任意一种语言技术，如词意消歧、语义角色标注、问答 系统、机器翻译，命名实体识别。找出开发这种系统需要什么类型和多少数量的已 标注的数据。为什么你认为需要大量的数据?

2.O 使用本章所述的3种分类器之一，以及你能想到的任何特征，尽可能好地建 立一个名字性别分类器。从将名字语料库分成3个子集开始：500个词为测试集， 500个词为开发测试集，剩余6900 个词为训练集。然后从示例的名字性别分类器 开始，逐步改善。使用开发测试集检查你的进展。 一旦对分类器感到满意，在测试 集上检查它的最终性能。相比在开发测试集上的性能，它在测试集上的性能如何? 这是你期待的吗?

3.OSenseval2 语料库包含了旨在训练词-词义消歧分类器的数据。它包含4个词 的数据： hard 、interest 、line 和 serve。选择这4个词中的其中一个，加载相应的 数据。

>>>from nltk.corpus import senseval

>>>instances =senseval.instances('hard.pos')

>>>size =int(len(instances) \*0.1)

>>>train\_set,test\_set =instances[size:],instances[:size]

使用这个数据集建立一个分类器，使其能正确地预测出一个给定实例的词意标签。关 于如何使用Senseval 2语料库返回实例对象的信息请参阅<http://www.nltk.org/howto> 上 的语料库HOWTO。

4.O 使用本章讨论过的电影评论文档分类器，生成30个对分类器最有用的特征列 表。你能解释为什么这些特定特征具有信息量吗?你能在它们中发现什么?

5.O 选择本章所描述的任意一个分类任务，如名字性别检测、文档分类、词性标 注或对话行为分类。使用相同的训练和测试数据，相同的特征提取器，为该任务建 立3个分类器：决策树、朴素贝叶斯分类器和最大熵分类器。比较这3个分类器在 所选任务上的性能。如果使用了不同的特征提取器，结果可能会不同，对应于这一 点你如何看待?

6.O 同义词strong 和powerful的模式不同(尝试将它们与 chip和 sales 结合)。哪 些特征与这种区别有关?建立一个分类器，预测每个词何时该被使用。

7.①对话行为分类器为每个帖子分配标签，不必考虑帖子的上下文背景。然 而，对话行为是高度依赖上下文的， 一些序列的对话行为可能比别的更相似。 例如： ynQuestion 对话行为更容易被 **yanswer** 回答而不是被 **greeting** 回答。 利用这一事实，建立一个连续的分类器为对话行为加标签。 一定要考虑哪些 特征可能是有用的。参见例6-5的词性标记的连续分类器的代码，以获得一些 想法。

8.①词特征在处理文本分类中是非常重要的，因为在文档中出现的词对于其语义 内容具有强烈的指示作用。然而，很多词很少出现， 一些在文档中的最有信息量的 词可能永远不会出现在训练数据中。 一种解决方法是使用**词典，**描述词之间的不同。 使用 WordNet 词典，加强本章介绍的电影评论文档分类器，使用能概括某个文档 中出现词的特征，使之与训练数据中发现的词匹配得更容易。

9.●P P附件语料库是描述介词短语附属决策的语料库。语料库中的每个实例被编 码**为** **PPAttachment 对象**。

>>>from nltk.corpus import ppattach

>>>ppattach.attachments('training')

[PPAttachment(sent='0',verb='join',noun1='board',

prep='as',noun2='director',attachment='V'),

PPAttachment(sent='1',verb='is',nounl='chairman',

prep='of',noun2='N.V.',attachment='N'),

>

>>>(inst.noun1,inst.prep,inst.noun2)

('chairman', 'of', 'N.V.')

**只选择** **inst.atachment 为N 的实例。**

>>>nattach =[inst for inst in ppattach.attachments('training')

if inst.attachment =='N']

使用此子语料库，建立一个分类器，尝试预测哪些介词是用来连接一对给定名词的。 例如：给定的名词对 team和researchers,分类器应该预测出介词of。更多使用PP 附件语料库的信息，参阅 <http://www.nltk.org/howto> 上的语料库 HOWTO。

10.●假设要自动生成一个场景的散文描述，且每个实体都已有唯一描述此实体的 词，例如：the book,只需要决定在有关的各项目中是否使用in 或 on, 例如：the book is in the cupboard 对the book is on the shelf。通过查找语料数据，编写需要的程序， 探讨这个问题，并思考下面的例子。

(13)a.in the car versus on the train

b.in town versus on campus

c.in the picture versus on the screen

d.in Macbeth versus on Letterman



**第7章**

**从文本提取信息**

对于任何给定的问题，很可能已经有人把答案写在某个地方了。以电子化提供的自 然语言文本的数量真的惊人，并且与日俱增。然而，自然语言的复杂性使访问这些 文本中的信息非常困难。NLP 还需要一定的发展，才能适用于各种复杂的文本。 如果我们不将我们的精力集中在一个问题或“实体关系”的有限集合上，例如：“不 同的设施位于何处”或“谁被什么公司雇用”上，我们就能取得重大进展。本章的 目的是要回答下列问题。

(1)如何能构建一个系统，以至从非结构化文本中提取结构化数据?

(2)有哪些稳健的方法识别一个文本中描述的实体和关系?

(3)哪些语料库适合这项工作，如何使用它们来训练和评估模型?

在此过程中，我们将应用最后两章中的技术来解决分块和命名实体识别的问题。

**7.1** **信息提取**

信息有很多种“形状”和“大小”。 一个重要的形式是**结构化数据：** 实体和关系的 规范和可预测的组织。例如：我们可能对公司和地点之间的关系感兴趣。给定一个 公司，我们希望能够确定它做业务的位置；反过来，给定位置，我们会想发现哪些 公司在该位置做业务。如果我们的数据是表格形式，如表7-1中的例子，那么就可 以很明确地回答这些问题了。

**表7-1**

**位置数据**

|  |  |
| --- | --- |
| **机** **构** **名** | **位** **置** **名** |
| Omnicom | New York |
| DDB Needham | New York |
| Kaplan Thaler Group | New York |
| BBDO South | Atlanta |
| Georgia-Pacific | Atlanta |

如果这个位置数据被作为一个元组(entity,relation,entity) 的链表存储在Python 中 ， 那么这个问题：“哪些组织在亚特兰大经营?”可翻译如下。

>>>print [org for(el,rel,e2)if rel=='IN'and e2=='Atlanta']

['BBDO South','Georgia-Pacific']

如果我们尝试从文本中获得相似的信息，事情就比较麻烦了。例如：思考下面的片 段(来自 nltk.corpus.ieer,fileid 为NYT19980315.0085)。

(1)The fourth Wells account moving to another agency is the packaged paper-products division of Georgia-Pacific Corp.,which arrived at Wells only last fall.

Like Hertz and the History Channel,it is also leaving for an Omnicom-owned agency,the BBDO South unit of BBDO Worldwide.BBDO South in Atlanta, which handles corporate advertising for Georgia-Pacific,will assume additional duties for brands like Angel Soft toilet tissue and Sparkle paper towels,said Ken Haldin,a spokesman for Georgia-Pacific in Atlanta.

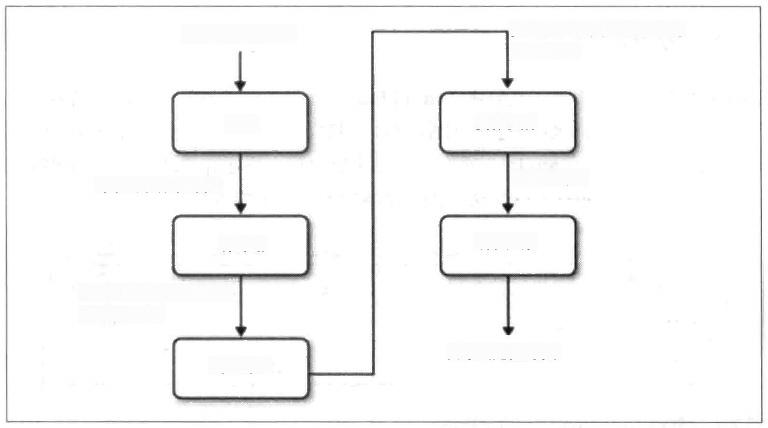
如果你通读了(1),你将收集到回答例子问题所需的信息。但我们如何能让一台机 器理解(1)然后返回链表[BBD O South,'Georgia-Pacific] 将其作为答案呢?这显然 是一个困难得多的任务。与表7-1不同，(1)不包含连接组织名和位置名的结构。

这个问题的解决方法之一是建立一个非常一般的含义(见第10章)。在这一章中， 我们采取不同的方法，提前定为只查找文本中非常具体的各种信息，如：组织和地 点之间的关系。不是试图用文字像(1)那样直接回答这个问题，我们首先将自然 语言句子这样的 **非结构化数据转**换成表7-1 的结构化数据。然后，利用强大的查询 工具，如 SQL。这种从文本获取意义的方法被称为信息提取。

信息提取有许多应用，包括商业智能、简历收获、媒体分析、情感检测、专利检索 及电子邮件扫描。当前研究的 一 个特别重要的领域是提取出电子科学文献的结构化 数据，特别是在生物学和医学领域。

**信息提取结构**

图7-1 显示了一个简单的信息提取系统的结构。它开始于使用第3章和第5章讨论 过的几个程序处理文档：首先，使用句子分割器将该文档的原始文本分割成句，使 用分词器将每个句子进一步细分为词。接下来，对每个句子进行词性标注，在下一 步**命名实体识别**中将证明这是非常有益的。在这 一 步，寻找每个句子中提到的潜在 的有趣的实体。最后，使用**关系识别**搜索文本中不同实体间的可能关系。



pos 标记的句子(列表中的 元祖列表)

实体识别

分块的句子

确认关系

关系(元祖列表)

原始文本(一串)

断句

句子(字符串列表)

标计划

标记划的句子(列表中

的元祖列表)

词性标注

**图7-** **1** **信息提取系统的简单流程。该系统将文档的原始文本作为输入，将生成的(entity,relation,** enti**y)** **元组链表作为输出。例如：给定的文档假设** **Georgia-Pacific** **公司位于** **Atlanta,可能产生** 元组([ORG:Georgia-Pacific]'in'[LOC:'Atlanta')

要执行前面3项任务，需定义 一 个函数，简单地连接NLTK 中默认的句子分割器①,

分词器②和词性标注器③。

>>detsk(cnut\_mtize(document)①

... sentences =[nltk.word\_tokenize(sent)for sent in sentences]② sentences =[nltk.pos\_tag(sent)for sent in sentences]③



**提示**

请记住上述例子程序的前提是假设以import nltk,re,pprint开始交互式 会话或程序。

接下来，在命名实体识别中，我们分割和标注可能组成具有某种关系的实体。通常 情况下，其被定义为名词短语，例如 the knights who say"ni"或者适当的名称如 Monty Python。在一些任务中，还需要考虑不明确的名词或名词块，如 every student 或 cats, 它们不一定非要用与定义NP 和适当名称一样的方式指示实体。

最后，在提取关系时，我们搜索文本中出现在彼此附近的实体对之间的特殊模式， 并使用这些模式建立元组记录实体之间的关系。

**7.2** **分块**

用于实体识别的基本技术是**分块** **(chunking )**, 分割和标注如图7-2所示的多标识 符序列。小框显示词级标识符和词性标注，同时，大框显示较高级别的程序分块。 较大的框叫**做组块** **(chunk)** 。就像分词忽略空白符，程序分块通常选择标识符的 一个子集。同分词一样，分块构成的源文本中的片段不能重叠。

|  |
| --- |
|  |

**图7-2** **词标识符和块级别的分割与标注**

在本节中，我们将在较深的层面上探讨程序分块，以组块的定义和表示开始。 我们将看到正则表达式和 n-gram 方法分块，使用 CoNLL-2000 分块语料库开 发和评估分块器。我们将在第7.5节和第7.6节继续介绍命名实体识别和关系 抽取。

**名词短语分块**

首先思考**名词短语分块，**或 NP-分块(**NP-chunking**),寻找单独名词短语对应的块。

例如：下面是一些《华尔街日报》文本，其中的NP-分块用方括号标记。

(2)[The/DT market/NN ]for/IN [system-management/NN software/NN ]for/IN [Digital/NNP ]['s/POS hardware/NN ]is/VBZ fragmented/JJ enough/RB that/IN [a/DT giant/NN ]such/JJ as/IN [Computer/NNP Associates/NNPS ]

should/MD do/VB well/RB there/RB ./.

正如我们可以看到， NP-分块往往是比完整的名词短语更小的片段。例如： the market for system-management software for Digital's hardware 是一个单独的名词短语(含两 个嵌套的名词短语),它中间有一个简单的NP-分块 the market。这种差异的动机 之一是 NP-分块被定义为不包含其他的 NP-分块。因此，修饰一个名词的任何介 词短语或从句将不包括在相应的NP-分块内，因为它们几乎可以肯定包含更多的 名词短语。

NP-分块信息最有用的来源之一是词性标记。这是在信息提取系统中进行词性标注的 动机之一。在例7-1中使用已经标注词性的例句来演示这种方法。为了创建NP-分块， 首先定义分**块语法，**规定句子应如何分块。在本例中，使用一个正则表达式规则定义 一个简单的语法②。这条规则是 NP-分块由可选的且后面跟着任意数目形容词(JJ) 的限定词 (DT) 和名词 (NN) 组成。使用此语法，我们创建了组块分析器③,测试 我们的例句④。结果得到树状图，可以输出⑤或显示图形⑥。

例7-1 一个简单的基于正则表达式的 NP 分块器的例子

>>>sentence=[("the","DT"),("little","JJ"),("yellow","JJ"),

..("dog","NN"),("barked","VBD"),("at","IN"), ("the","DT"),("cat","MN")]

>>>grammar ="NP:{<DT>?<JJ>\*<NN>}"

>>>cp =nltk.RegexpParser(grammar)

>>>result =cp.parse(sentence)

>>>print result

(S

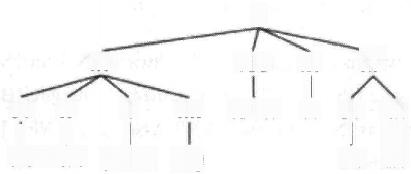
(NP the/DT little/JJ yellow/JJ dog/NN)

/rIkNed/VBD

(NP the/DT cat/NN))

>>>result.draw() ⑥





S

**VBD IN NP**

1

**NN** barked at DT

1

dog the

DT JJ l | the little

NN | cat

yellow

**NP**

JJ

**标记模式**

组成块语法的规则利用**标记模式**描述已标注的词的序列。标记模式是用尖括号分隔 的词性标记序列，如<DT>?<JJ>\*<NN> 。标记模式类似于正则表达式模式(见3.4 节)。现在，思考下面来自《华尔街日报》的名词短语。

another/DT sharp/JJ dive/NN

trade/NN figures/NNS

e/JnJeRwJtJagpeoyN/N measures/NNS

Panamanian/JJ dictator/NN Manuel/NNP Noriega/NNP

略微改进上述第一个标记模式，以匹配这些名词短语。如<DT>?<JJ.\*>\*<NN.\*>+。 将任何以一个可选的限定词开头，后面跟零个或多个任意类型形容词(包括相对形 容词，如earlier/JR), 以及一个或多个任意类型名词的标识符序列分块。然而，还 存在许多该规则不包括的、更复杂的例子。

his/PRPS Mansion/NNP House/NNP speech/NN

the/DT price/NN cutting/VBG

3/CD 8/NN to/TO 4/CD %/NN

more/JJR than/IN 10/CD 8/NN

the/DT fastest/JJS developing/VBG trends/NNS

's/POS skill/NN

**轮到你来：**

尝试用标记模式覆盖这些案例。使用图形界面 nltk.app.chunkparserO 测试它们。并且使用此工具提供的帮助资料继续完善你的标记模式。

**用正则表达式分块**

为了找到给定句子的分块结构， RegexpParser分块器以一个平面结构开始，其中的 标识符都未被分块。轮流应用分块规则，依次更新块结构。所有的规则都被调用后，



返回块结构。

例7-2显示了一个由2个规则组成的简单的分块语法。第一条规则匹配一个可选的 限定词或所有格代名词，其后带有零个或多个形容词及一个名词。第二条规则匹配 一个或多个专有名词。我们还定义了一个进行分块的例句①,并在此输入上运行这 个分块器②。

**例7-2** **简单的名词短语分块器**

grammar m rnnn

NP:{<DT|PP\S>?<JJ>\*<NN>} #chunk determiner/possessive,adjectives and nouns

{<NNP>+} #chunk sequences of proper nouns

n n H

sentence =【("Rapunzel","NNP"),("let","VBD"),("down","RP"),① ("her","PPS"),("long","JJ"),("golden","JJ"),("hair","NN")]

cp =nltk.RegexpParser(grammar)

>>>print cp.parse(sentence)②

(S

(NPRapunzel/NNP)

let/VBD

n/r/PP$long/JJ golden/JJ hair/NN))

$符号是正则表达式中的一个特殊字符，必须使用转义符\匹配 PP$标

记。

如果标记模式匹配位置重叠，最左边的优先匹配。例如：如果将匹配两个连续名词 的文本的规则应用到包含3个连续名词的文本中，则只有前两个名词被分块。

>>>nouns =[("money","NN"),("market","NN"), ("fund","NN")] >>>grammar ="NP:{<NN><NN>) #Chunk two consecutive nouns"

>>>cp =nltk.RegexpParser(grammar)

>>>print cp.parse(nouns)

(S(NP money/NN market/NN)fund/NN)

一旦创建了块money market, 就说明已经消除了允许 fund 被包含在块中的上下文。 使用一种更加宽容的块规则就可以避免这个问题，如： NP:{<NN>+} 。



我们已经为每个分块规则添加了一个注释。这些都是可选的。当它们 存在时，分块器将它作为其跟踪输出的一部分输出这些注释。

**探索文本语料库**

在5.2节中，我们看到了如何在已标注的语料库中提取匹配特定词性标记序列的短 语。使用分块器可以更轻松地完成这项工作，具体如下。

>>>cp =nltk.RegexpParser('CHUNK:{<V.\*><TO><V.\*>}')

>>>brown =nltk.corpus.brown

>>>for sent in brown.tagged\_sents():

· tree =cp.parse(sent)

for subtree in tree.subtrees():

. · · if subtree.node =='CHUNK':print subtree



(CHUNK combined/VBN to/TO achieve/VB)

(CHUNK continue/VB to/TO place/VB)

(CHUNK serve/VB to/TO protect/VB)

(CHUNK wanted/VBD to/TO wait/VB)

(CHUNK allowed/VBN to/TO place/VB)

(CHUNK expected/VBN to/TO become/VB)

(CHUNK seems/VBZ to/TO overtake/VB)

(CHUNK



**缝隙**

want/VB to/TO buy/VB)

**轮到你来：**

将上面的例子放在函数 find\_chunks()内，以一个如"CHUNK:{<V.\*> <TO><V.\*>}"的块字符串作为参数。用它来搜索语料库以寻找其他几 个模式，如4个或更多的连续名词即"NOUNS:{<N,\*>{4,}}"。

有时可以很容易地定义我们想从块中排除什么。可以为不包括在大块中的标识符序 列定义一个缝隙。在下面的例子中， barked/VBD at/IN 是一个缝隙。

[the/DT little/JJ yellow/JJ dog/NN ]barked/VBD at/IN [the/DT cat/NN ]

加缝隙是从大块中去除标识符序列的过程。如果匹配的标识符序列贯穿整块，那么 这个整块将被去除；如果标识符序列出现在块中间，这些标识符会被去除，在以前 只有一个块的地方留下两个块。如果序列在块的周边，这些标记会被去除，留下一

个较小的块。表7-2演示了这3种可能性。

**表7-2** **3个加缝隙规则应用于同一个块**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **整** **个** **块** | **块** **中** **间** | **块** **结** **尾** |
| 输入 | [a/DT little/JJ dog/NN] | [a/DT little/JJ dog/NN] | [a/DT little/JJ dog/NN] |
| 操作 | Chink“DT JJNN” | Chink“JJ” | Chink“NN” |
| 模式 | }DT JJNN{ | }JJ{ | }NN{ |
| 输出 | a/DT little/JJ dog/NN | a/DT]little/JJ[dog/NN] | [a/DT little/JJ]dog/NN |

在例7-3中，我们将整个句子作为一个块，然后练习加缝隙。

例7-3 简单的加缝器

grammar =rnmn

NP:

{<.\*>+}

}<VBD|IN>+{

#

#Chunk everything

#Chink sequences of VBD and IN

sentence=[("the","DT"),("little","JJ"),("yellow","JJ"),

("dog","NN"),("barked","VBD"),("at","IN"),("the","DT"), ("cat","NN")]

cp =nltk.RegexpParser(grammar)

>>>print cp.parse(sentence)

(S

ked/T little/JJ yellow/JJ dog/NN)

at/IN

(NP the/DT cat/NN))

**分块的表示：标记与树状图**

作为标注和分析之间的中间状态(见第8章),块结构可以使用标记或树状图来表 示。使用最广泛的表示是 **IOB 标记。**在这个方案中，每个标识符被用3个特殊的 块标签之一标注，I(inside, 内部),O(outside, 外部)或B(begin, 开始)。标 识符被标注为B, 则标志着它是分块的开始。块内的标识符子序列被标注为I 。 所 有其他的标识符被标注为O。B 和 I 标记是块类型的后缀，如 B-NP,I-NP。 当然， 没有必要指定出现在块外的标识符类型，所以这些都只标注为O。这个方案的例子 如图7-3所示。



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | W e | S | a | W | t | h | e | y | e | 1 | 1 | 0 | W | d 0 8 | | PRP  B-NP | VBD  0 | | | DT  B-NP | | | JJ  I-NP | | | | | | NN  I-NP | |

**图7-3** **分块结构的标记标识符**

IOB 标记已成为文件中表示块结构的标准方式，我们也将使用这种格式。图7-3展 示了信息是如何出现在一个文件中的。

We PRP B-NP

saw VBD O

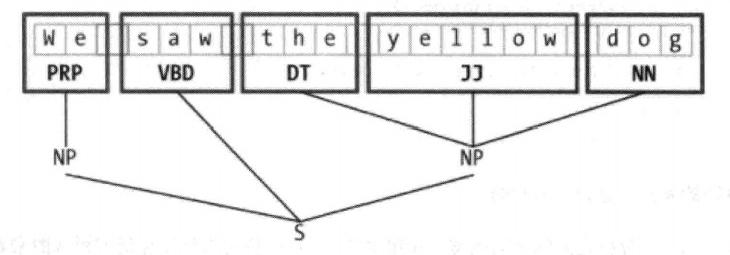
the DT B-NP

little JJ I-NP

yellow JJ I-NP

dog NN I-NP

在此表示方式中，每个标识符一行，和它的词性标记与块标记一起。这种格式允许 表示多个块类型，只要块不重叠。正如我们前面所看到的，块的结构也可以使用树 表示。这有利于使每块作为一个组成部分可以直接操作。 一个例子如图7-4所示。



**图7-4** **块结构的树状图表示**

NLTK 用树状图作为分块的内部表示，却提供这些树状图与IOB 之间 格式转换的方法。

**7.3** **开发和评估分块器**

现在对分块的作用已经有了一定的了解，但我们并没有解释如何评估分块器。和往 常一样，这需要一个合适的已标注语料库。首先寻找将IOB 格式转换成 NLTK 树 状图的机制，然后是如何在一个更大的规模上使用已分块的语料库完成上述过程。 我们将学习如何为一个分块器相对于一个语料库的准确性打分，以及利用一些数据 驱动方式搜索 NP 分块。本节的重点在于扩展分块器的覆盖范围。

**读取** **IOB** **格式与** **CoNLL2000** **分块语料库**

使用 corpora 模块，我们可以加载已标注的《华尔街日报》文本，然后使用IOB 符 号分块。这个语料库提供的分块类型有 NP 、VP 和 PP 。正如我们已经看到的，每 个句子使用多行表示，如下所示。

he PRP B-NP

accepted VBDB-VP

the DT B-NP

position NN I-NP

转换函数 chunk.conllstr2tree()用这些多行字符串建立一个树状图表示。此外，我们 可以选择使用3个分块类型的任何子集，下例中只使用了NP 分块。

>>> text = !

...he PRP B-NP

...accepted VBD B-VP

...the DT B-NP

...position NN I-NP

.of IN B-PP

...vice NN B-NP

...chairman NN I-NP

..of IN B-PP

...Carlyle NNP B-NP

..Group NNP I-NP

/ C

..a DT B-NP

...merchant NN I-NP

...banking NN I-NP

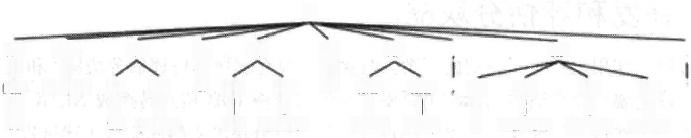
.. concern NN I-NP

0

>>>ltk.chunk.conllstr2tree(text,chunk\_types=['NP']).draw()

君一

—





NP

|  |  |  |  |
| --- | --- | --- | --- |
| DT  l  0 | NN  1  meichant | NN  1  banking | NN 1  conxe |

|  |  |
| --- | --- |
| NNP  1  (anlyle | NNP  1  Group |

BD asxepted

NN

1

positkn

NN

1

chgmn

NN

1

wice

NP

1

PRP

DT 1 the

W

NP

NP

NP

能

—

我们可以使用 NLTK 的 corpus 模块访问较大量的已分块文本。CoNLL2000 分块语 料库包含27万词的《华尔街日报文本》,分为“训练”和“测试”两部分，标注有 词性标记和IOB 格式分块标记。我们可以使用 nltk.corpus.conll12000 访问这些数据。 下面是一个读取语料库的“训练”部分的100个句子的例子。

>>>from nltk.corpus import con112000

>>>print conll2000.chunked\_sents('train.txt')[99]

(S

(PP Over/IN)

(NPa/DT cup/NN)

(PP of/IN)

(NP coffee/NN)

,/,

(NP Mr./NNP Stone/NNP)

(VP told/VBD)

(NP his/PRP$story/NN)

./.)

正如你看到的， CoNLL2000 分块语料库包含3种分块类型： NP 分块，我们已经学 过了； VP 分块，如 has already delivered;PP 分块，如 because of。因为现在我们唯 一感兴趣的是 NP 分块，所以可以使用 chunk\_types 参数选择它们。

>>>print conl12000.chunked\_sents('train.txt',chunk\_types=['NP'])[99]

(S

Over/IN

(NP a/DT cup/NN)

of/IN

(NP coffee/NN)

,/,

(NP Mr./NNP Stone/NNP)

l/iBsPRP$story/NN)

./.)

**简单评估和基准**

现在，我们可以访问已分块语料，还可以评估分块器。下例是为琐碎的不创建任何

**块的块分析器** **cp 建立一个基准** **(baseline)。**

>>>from nltk.corpus import conll2000

>>>cp =nltk.RegexpParser("")

>>>test\_sents =con112000.chunked\_sents('test.txt',chunk\_types=['NP'])

>>>print cp.evaluate(test\_sents)

ChunkParse score:

IOB Accuracy: 43.4%

Precision: 0.0%

Recall: 0.0%

F-Measure: 0.0%

IOB 标记准确性表明超过三分之一的词被标注为 O, 即没有在NP 分块中。然而， 由于我们的标注器没有找到任何分块，其精度、召回率和 F-度量均为零。下例中 利用一个初级的正则表达式分块器，查找以名词短语标记的特征字母(如 CD 、DT 和 JJ) 开头的标记。

>>>grammar =r"NP:{<[CDJNP].\*>+}"

>>>cp=nltk.RegexpParser(grammar)

>>>print cp.evaluate(test\_sents)

ChunkParse score:

IOB Accuracy:87.7%

Precision: 70.6%

Recall: 67.8%

F-Measure: 69.2%

正如你看到的，通过这种方法可得到相当好的结果。但是，我们仍可以采用更多数据驱 动的方法改善它，在这里我们使用训练语料找到对每个词性标记最有可能的块标记 (I、 O或 B) 。换句话说，我们可以使用unigram标注器(见5.4节)建立一个分块器。但不 是要确定每个词的正确词性标记，而是给定每个词的词性标记，尝试确定正确的块标记。

在例7-4中，我们定义了 UnigramChunker 类，使用 unigram 标注器给句子加块标 记。这个类的大部分代码只是用来在NLTK 的 ChunkParserl 接口使用的分块树表示 方式和嵌入式标注器使用的 IOB 表示之间镜像转换。这一分类定义了两个方法： 一个是构造函数°,当建立新的 UnigramChunker 时调用；另一个是 parse 方法°, 用来给新句子分块。

例7-4 使用 unigram 标注器对名词短语分块

class UnigramChunker(nltk.ChunkParserI):

deftrain\_is[):①in nltk.chunk.tree2conlltags(sent)]

for sent in train\_sents]

self.tagger =nltk.UnigramTagger(train\_data)②

def parse(self,sentence):③

pos\_tags =[pos for(word,pos) in sentence]

gugneks\_ta[chunktaself.tfaogr.a,cgh(i tagged\_pos\_tags]

conlltags=[(word,pos,chunktag)for((word,pos),chunktag)

in zip(sentence,chunktags)]

return nltk.chunk.conlltags2tree(conlltags)

构造函数①需要一个训练句子链表，以分块树状图的形式表示。它首先将训练数据 转换成适合训练标注器的形式，使用 tree2conlltags 映射每个分块树状图到一个词、 标记、块三元组的链表。然后使用转换好的训练数据训练 unigram 标注器，并存储 在 self.tagger 供以后使用。

parse 方法°取一个已标注的句子作为其输入，以从该句提取词性标记开始。然后使 用在构造函数中训练过的标注器 selftagger, 为词性标记标注IOB 块标记。接下来， 提取块标记，与原句组合，产生 conlltags。最后，使用 conlltags2tree 将结果转换成 分块树状图。

现在我们得到了UnigramChunker, 可以使用CoNLL2000 分块语料库训练它，并测 试其性能。

>>>test\_sents=con112000.chunked\_sents('test.txt',chunk\_types=['NP'])

>>>train\_sents =conl12000.chunked\_sents('train.txt',chunk\_types=['NP'])

>>>unigram\_chunker =UnigramChunker(train\_sents)

>>>print unigram\_chunker.evaluate(test\_sents)

ChunkParse score:

IOB Accuracy:92.9%

Precision: 79.9%

Recall: 86.8%

F-Measure: 83.2%

**这个分块器相当不错，** **F-Measure: 的结果为83%。让我们看看使用** **unigram 标注**

器标记每个在语料库中出现的词性标记，将会发生什么。

>>>postags =sorted(set(pos for sent in train\_sents

for(word,pos)in sent.leaves()))

>>>print unigram\_chunker.tagger.tag(postags)

[('#','B-NP'),('S','B-NP'),("''",'O'),('(','O'),(')','O'),

(',','o'),('.','O'),(':','o'),('CC','O'),('CD','I-NP'),

('DT','B-NP'),('EX','B-NP'),('FW','I-NP'),('IN','O'),

('JJ','I-NP'),('JJR','B-NP'),('JJS','I-NP'),('MD','O'),

('NN',‘I-NP'),('NNP',‘I-NP'), ('NNPS',‘I-NP'),('NNS',‘I-NP¹),

('PDT', 'B-NP'), ('POS','B-NP'), ('PRP','B-NP'),('PRPS','B-NP'), ('RB','O'),('RBR','O'),('RBS','B-NP'),('RP','O'),('SYM','O'),

('TO','O'),('UH','O'),('VB','O'),('VBD','O'),('VBG','O'),

('VBN','O'),('VBP','O'),('VBZ','O'),('WDT','B-NP'),

('WP','B-NP'),('WPS','B-NP'),('WRB','O'),('`','O')]

结果表明除了两种货币符号#和$,大多数标点符号出现在 NP 分块以外。限定词 (DT) 和所有格(PRP$ 和 WP$) 出现在NP 分块的开头，而名词类型(NN,NNP,NNPS,

NNS)大多出现在NP 的分块之内。

unigram 分块器建立完成后，可以很容易地建立 bigram 分块器：只需要改变分类的 名称为 BigramChunker, 修改例7 - 4 中行②来构造 BigramTagger 而不是 UnigramTagger。由此产生的分块器的性能略高于unigram 分块器。

>>>bigram\_chunker >>>print

=BigramChunker(train\_sents)

bigram\_chunker.evaluate(test\_sents)

ChunkParse score:

IOB Accuracy:93.3%

Precision: 82.3%

Recall: 86.8%

F-Measure: 84.5%

**训练基于分类器的分块器**

无论是基于正则表达式的分块器还是 n-gram 分块器，创建什么样的分块完全取决 于词性标记。然而，有时词性标记不足以确定一个句子应如何分块。例如：考虑下 面的两个语句。

(3)a.Joey/NN sold/VBD the/DT farmer/NN rice/NN J.

b.Nick/NN broke/VBD my/DT computer/NN monitor/NN /.

这两句话的词性标记相同，但分块方式不同。在第一句中，the farmer 和rice 都是 单独的分块，而在第二个句子中相应部分，the computer monitor, 是单独的分块。 显然，如果想要最大限度地提升分块的性能，需要使用词的内容信息作为词性标记 的补充。

我们包含词的内容信息的方法之一是使用基于分类器的标注器对句子分块。如在上 一节使用的 n-gram分块器，这个基于分类器的分块器分配IOB 标记给句子中的词， 然后将这些标记转换为块。对于基于分类器的标注器本身，我们将使用与在6.1 节 建立词性标注器的相同方法。

基于分类器的NP 分块器的基础代码如例7-5所示。它包括两个类：第一个类几乎 与例6-5中 ConsecutivePosTagger 类相同。仅有的两个区别是它调用一个不同的特 征提取器”,使用MaxentClassifier 而不是NaiveBayesClassifier; 第二个类③基本上 是标注器类的一个包装器，将它变成一个分块器。训练期间，第二个类映射训练语 料中的分块树状图到标记序列；在parse()方法中，它将标注器提供的标记序列转换 回块树。

**例7-5** **使用连续分类器对名词短语分块**

class ConsecutiveNPChunkTagger(nltk.TaggerI):①

def \_init\_(self,train\_sents):

train\_set = []

for tagged\_sent in train\_sents:

nt =nltk.tag.untag(tagged\_sent)

for feahunken\_fuema\_t)ent,i,history)②

train\_set.append((featureset,tag))

history.append(tag)

self.classifier =nltk.MaxentClassifier.train(③

train\_set,algorithm='megam',trace=0)

def hientence):

for ercefntfy(f)sa:(e,i,history)

history.append(tag)

return zip(sentence,history)

class ConsecutiveNPChunker(nltk.ChunkParserI):④

def \_init\_(self,train\_sents):

tagged sents =[[((w,t),c)for(w,t,c)in

\_

nltk.chunk.tree2conlltags(sent)]

for sent in train\_sents]

=ConsecutiveNPChunkTagger(tagged\_sents)

self.tagger

parse(self,sentence):

def

tagged\_sents =self.tagger.tag(sentence)

=[(w,t,c)for((w,t),c)in tagged\_sents] nltk.chunk.conlltags2tree(conlltags)

conlltags return



留下来唯一需要填写的就是特征提取器。首先，定义一个简单的特征提取器，它只 提供当前标识符的词性标记。使用此特征提取器，基于分类器的分块器的性能与 unigram 分块器非常类似。

>>>def worunk\_ ntence,i,history):

.. · return {"pos":pos}

>>>chunker =ConsecutiveNPChunker(train\_sents)

>>>print chunker.evaluate(test\_sents)

ChunkParse score:

IOB Accuracy:92.9%

Precision: 79.9%

Recall: 86.7%

F-Measure: 83.28

还可以添加特征：前面词的词性标记。添加此特征允许分类器模拟相邻标记之间的

相互作用，由此产生的分块器与 bigram 分块器非常接近。

>>>def npchunk features(sentence,i,history):

... r,nce<START>","<START>"

··

·

..

else:

prevword,prevpos =sentence[i-1]

return {"pos":pos,"prevpos":prevpos)

>>>chunker =ConsecutiveNPChunker(train\_sents)

>>>print chunker.evaluate(test\_sents)

ChunkParse score:

IOB Accuracy:93.6%

Precision: 81.9%

Recall: 87.1%

F-Measure: 84.4%

下一步，我们将为当前词增加特征，假设这个词的内容对分块有用。我们发现 这个特征确实提高了分块器的性能，大约1.5个百分点(相应的错误率减少大约 10%)。

>>>def npchunk features(sentence,i,history):

·

.

\*

·

·



\_

word,pos

=sentence[i]

if i==0:

prevword,prevpos

="<START>","<START>"

=sentence[i-1]

else:

prevword,prevpos

return {"pos":pos,"word":word,"prevpos":prevpos}

>>\*>chunker =ConsecutiveNPChunker(train\_sents)

>>>print chunker.evaluate(test\_sents)

ChunkParse score:

IOB Accuracy:94.2%

Precision: 83.4%

Recall: 88.6%

F-Measure: 85.9%

最后，我们使用多种附加特征扩展特征提取器，例如：预取特征°、配对功能②和复 杂的语境特征③。最后一个特征，被称为 tags-since-dt, 用其创建一个字符串，描述 自最近限定词以来遇到的所有词性标记。

>>> def ncohrudn,e, i, history):

if pd,prevpos ="<START>","<START>"

else:

if srce)- 1: =sentence[i-l]

nextword,nextpos ="<END>","<END>"

else:

nextword,nextpos =sentence[i+1]

vpos,

return{"pos":pos,

"nextpos":nextpos,'

①

s+-tcpeo-%\_ (epo\_,i))

"prevpos+pos":"8s+8s"8(prevpos,pos),°

def tags\_since\_dt(sentence,i):

tags =set()

for word,pos in sentence[:i]:

if pos =='DT':

tags =set()

else:

tags.add(pos)

return '+'.join(sorted(tags))

>>>chunker =ConsecutiveNPChunker(train\_sents)

>>>print chunker.evaluate(test\_sents)

ChunkParse score:

IOB Accuracy:95.9%

Precision: 88.38

Recall: 90.7%

F-Measure: 89.5%



**轮到你来：**

尝试为特征提取器函数 npchunk\_features 增加不同的特征，看看是否 可以进一步改善NP 分块器的性能。

**7.4** **语言结构中的递归**

**用级联分块器构建嵌套结构**

到目前为止，我们的分块结构一直是相对单调的。已标注标识符组成的树状图在如 NP 这样的块节点下任意组合。然而，只需创建一个包含递归规则的多级的分块语 法，就可以建立任意深度的分块结构。例7-6展示了名词短语、介词短语、动词短 语和句子的模式。这是一个四级分块语法器，可以用来创建深度最多为4的结构。

例7-6 分块器，用于处理NP、PP、V P 和 S

grammar = rnnn

NP: {<DT|JJ|NN.\*>+}

PP:{<IN><NP>}

VP: {<VB.\*><NP|PPICLAUSE>+$}

CLAUSE:{<NP><VP>}

n n H

#Chunk sequences of DT,JJ,NN

#Chunk prepositions followed by NP #Chunk verbs and their arguments

#Chunk NP,VP

sent"nsciet"VMr),"(aI,V,(tDh"e),("""T,"",(**N**")N,")]

cp =nltk.RegexpParser(grammar)

>>>print cp.parse(sentence)

(S

(NP Mary/NN)

saw/VBD

(CLAUSE

(NP the/DT cat/NN)

(VP sit/VB(PP on/IN(NP the/DT mat/NN)))))

遗憾的是，这一结果丢掉了以 saw 为首的 VP。此外它还有其他缺陷。如果将此分 块器应用到是有更深嵌套的句子中，那么会发生什么。请注意，它无法识别°开始 的 VP 块。

>>>sentence =[("John","NNP"),("thinks","VBZ"),("Mary","NN"), ("saw", "VBD"), ("the", "DT"), ("cat", "NN"),("sit","VB"),

("on","IN"), ("the","DT"), ("mat","NN")]

>>>print cp.parse(sentence)

(S

(NP John/NNP)

thinks/VBZ

(NP Mary/NN)

saw/VBD ①

(CLAUSE

(NP the/DT cat/NN)

(VP sit/VB(PP on/IN(NP the/DT mat/NN)))))

问题的解决方案是：让分块器在它的模式中循环：尝试完所有模式之后，重复此过 程。添加第二个可选的参数 loop, 指定这套模式应该循环的次数。

>>>cp =nltk.RegexpParser(grammar,loop=2)

>>>print cp.parse(sentence)

(S

(NP John/NNP)

thinks/VBZ

(CLAUSE

(NP Mary/NN)

(VP

saw/VBD

(CLAUSE

(NP the/DT cat/NN)

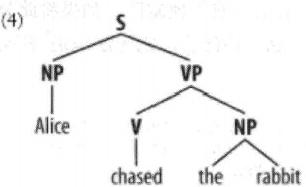
(VP sit/VB(PP on/IN(NP the/DT mat/NN)))))))



**树状图**

通过这个级联过程，我们可以创建深层结构。然而，创建和调试级联 过程都是困难的，关键点是进行全面的分析能更有效(见第8章)。 另外，级联过程只能产生固定深度的树状图(不超过级联级数),这 是不完整的句法分析。

**树状图是**一组相互连接的加标签节点，从一个特殊的根节点沿一条唯一的路径到达 每个节点。下面是树状图的例子(注意：它们标准的画法是颠倒的)。



我们用“家庭”来比喻树中节点的关系：例如： S 是VP的父母；反之 VP是 S 的 **孩子。**此外，由于NP和VP同为S 的两个孩子，它们也是兄弟。为方便起见，也 有特定树的文本格式。

(S

(NP Alice)

(VP

(V chased)

(NP

(Det the)

(N rabbit))))

虽然我们将只集中关注语法树状图，但它可以用来编码任何同构的、超越语言形式 序列的层次结构(如形态结构、篇章结构)。 一般情况下，叶子和节点值不一定要 是字符串。

在 NLTK 中，创建树状图，方法是给节点添加标签和一个子链表。

>>>treel =nltk.Tree('NP', ['Alice'])

>>>print treel

) =nltk.Tree('NP',['the','rabbit'])

>>>print tree2

(NP the rabbit)

**我们可以将这些不断合并成更大的树状图，如下所示。**

>>>tree3 =nltk.Tree('VP', ['chased',tree2])

>>>tree4 =nltk.Tree('S',[treel, tree3])

>>>print tree4

(S(NP Alice) (VP chased(NP the rabbit)))

**下面是构成树状图对象的** **一** **些方法。**

>>>print tree4[1]

t](de rabbit))

'VP'

>>>tree4.leaves()

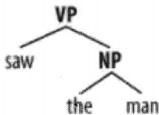
e4 ,]'the', 'rabbit']

'rabbit!



复杂的树状图用括号表示方法不利于阅读。在这种情况下， draw 方法是非常有用的。 它会打开一个新窗口，其中包含树状图的图形表示。树状图的显示窗口可以放大和缩 小，子树可以折叠和展开，并将图形表示输出为 postscript 文件(包含在一个文档中)。

>>>tree3.draw()



**树遍历**

使用递归函数来遍历树是标准的做法。例7-7中的列表演示了该做法。

例7-7 递归函数遍历树状图

def traverse(t):

try:

t.node

except AttributeError:

print t,

else:

#Now we know that t.node is defined

print'(',t.node,

for child in t:

traverse(child)

print ')',

>>>t=nltk.Tree('(S(NP Alice) (VP chased(NP the rabbit)))')

>>>traverse(t)

(S(NP Alice)(VP chased(NP the rabbit )))

使用一种叫做动态类型的技术检测t 是树状图(如：定义了t.node)。

**7.5** **命名实体识别**

在本章开头，我们简要介绍了命名实体 (NEs) 。 命名实体是确切的名词短语，指

特定类型的个体，如组织、人、日期等。表7-3罗列出了一些较常用的NEs 类型。 这些应该是不言自明的，除了“FACILITY”: 建筑和土木工程领域的人造产品。以 及“GPE”: 地缘政治实体，如城市、州/省、国家。

**表7-3** **常用命名实体类型**

|  |  |
| --- | --- |
| **N** **E** **类** **型** | **例子** |
| 组织 | Georgia-Pacific Corp.,WHO |
| 人 | Eddy Bonte,President Obama |
| 地点 | Murray River,Mount Everest |
| 日期 | June,2008-06-29 |
| 时间 | two fifty a m,1:30 p.m |
| 货币 | 175 million Canadian Dollars,GBP 10.40 |
| 百分数 | twenty pct,18.75 % |
| 设施 | Washington Momument,Stonehenge |
| 地缘政治实体 | South East Asia,Midlothian |

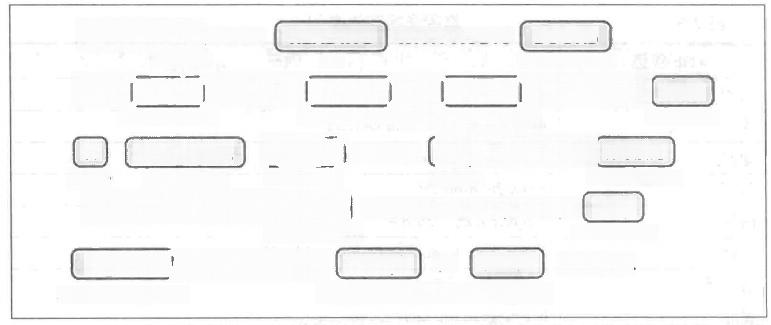
**命名实体识别**(NE R) 系统的目标是识别所有文字提及的命名实体。这可以分解成 两个子任务：确定NE 的边界和确定其类型。命名实体识别经常是信息提取中关系 识别的前奏，也有助于其他任务。例如：在问答系统 (QA) 中，我们试图提高信 息检索的精确度，不用返回整个页面而只是包含用户问题的答案的那部分。大多数 QA系统利用标准信息检索返回的文件，然后尝试分离文档中包含答案的最小的文 本片段。现在假设问题是 Who was the first President of the US? 被检索的文档中包 含下面这段话。

(5)The Washington Monument is the most prominent structure in Washington, D.C.and one of the city's early attractions.It was built in honor of George Washington,who led the country to independence and then became its first

President.

分析该问题时我们想到答案应该是X was the first President of the US 的形式，其中 X 不仅是一个名词短语也是一个 PER 类型的命名实体。这样我们就可以忽略段落 中的第一句话。虽然它提示 Washington 出现过两次，但命名实体识别告诉我们： 它们都不是正确的类型。

如何识别命名实体呢?一种办法是查找适当的名称列表。例如：识别地点时，我们 可以使用地名辞典，如亚历山大地名辞典或盖蒂地名辞典。然而，盲目这样做会出 问题，如图7-5所示。



**YOUR READING WITH AUDIO BOOKS**

Vietnam UK *Louisiana,USA*

Audio books arehighly Dopular with llbrary]patronsin the( town

Louisiana,U5A *5.Carolina.USA* Pennsylvania,USA Mass.USA

County, MO.Peopleare〔 mobile Norway *Alabama,USA*

and busier,and audio books fit into that lifestyle"says( Gary Louisiana,U5A indiana.USA

Sanchez.who oversees the library's $2 million budget …

*Dominican Republic* Pennsylvania,USA Kentucky,USA

图7 - 5 地点检测，在新闻故事中进行简单的查找：查找地名辞典中的每个词很容易出错；可进

Springfield,

*Virginia,U5A*

Greene Maine,USA

*of*

Turkey

**KEEP UP**ON

行案例区分，但它们不是总有的

地名辞典涵盖了很多国家的地点，却错误地认为 Sanchez 在多米尼加共和国而 On 在越南。当然，我们可以在地名辞典中忽略这些地名，但这样一来当它们出现在文 档中时，我们将无法识别它们。

人或组织名称的情况更加困难。任何这些的列表都肯定覆盖不全。每天都有新的组 织出现，如果我们要处理当代文本或博客条目，仅凭查找名称辞典来识别众多实体 是不可能的。

困难的另一个原因是许多命名实体措辞有歧义。例如，May 和 North 可能是日期和 地点类型的命名实体，但也可以都是人名；相反的， Chris-tian Dior 看上去像是一 个人名，但更可能是组织类型。词 Yankee 在某些上下文中是普通的修饰语，但在 短语 Yankee infielders 中会被标注为组织类型的一个实体。

更大的挑战来自如 Stanford University 这样的多词名称和包含其他名称的名称，如 Cecil H.Green Library 和 Escondido Village Conference Service Center。因此，在命名 实体识别中，我们需要能够识别多标识符序列的开头和结尾。

命名实体识别是一个非常适合用于分类器类型的方法，这些方法我们在名词短语分

块时用到过。特别是，我们可以建立一个标注器，为使用 IOB 格式的句子中的每 个词加标签，而每个块都添加了适当类型标签。下面是 CONLL 2002(conll2002)

荷兰语训练数据的一部分。

Eddy N B-PER

Bonte N I-PER

is V O

woordvoerder N O

van Prep O

diezelfde Pron O

Hogeschool N B-ORG

.Punc O

在上面的表示中，每个标识符一行，与它的词性标记及命名实体标记一起。基于这个训 练语料，我们可以构造一个可以用来标注新句子的标注器，使用nltk.chunk.conlltags2tree) 函数将标记序列转换成一个块树。

NLTK 提供了一个已经训练好的可以识别命名实体的分类器，使用函数 nltk.ne\_chunk() 访问。如果设置参数 binary=True°, 那么命名实体只被标注为NE;

否则，分类器会添加类型标签，如 PERSON 、ORGANIZATION and GPE。

>>>sent =nltk.corpus.treebank.tagged\_sents()[22]

>>>print nltk.ne\_chunk(sent,binary=True)①

(S

The/DT

(NE U.S./NNP)

is/VBZ

one/CD

accor·ding/VBG

to/TO

(NE Brooke/NNP T./NNP Mossman/NNP)

..)

>>>print nltk.ne\_chunk(sent)

(S

The/DT

(GPE U.S./NNP)

is/VBZ

one/CD

·

.

according/VBG

to/TO

(PERSON Brooke/NNP T./NNP Mossman/NNP)

..)

**7.6** **关系抽取**

只要文本中的命名实体被识别，我们就可以提取它们之间存在的关系。如前所述， 通常会寻找指定类型的命名实体之间的关系。方法之一是首先寻找所有(X,a,Y) 形式的三元组，其中X 和 Y 是指定类型的命名实体，α表示X 和 Y 之间关系的 字符串。然后使用正则表达式从α的实体中抽出正在查找的关系。下面的例子为搜 索包含词 in 的字符串。特殊的正则表达式(?!Nb.+inglb)是一个否定预测先行断言， 允许忽略如 success in supervising the transition of中的字符串，其中 in 后面跟着动 名词。

>>>IN =re.compile(r'.\*\bin\b(?!\b.+ing)')

>>>for doc in nltk.corpus.ieer.parsed\_docs('NYT\_ 19980315'):

for rel in nltk.sem.extract\_rels('ORG','LOC',doc,

corpus='ieer',pattern

=IN):

. · · print nltk.sem.show\_raw\_rtuple(rel)

[ORG:'WHYY']'in' [LOC:'Philadelphia']

[ORG:'McGlashan &AMP;Sarrail']'firm in'[LOC:'San Mateo']

[ORG:'Freedom Forum']'in'[LOC:'Arlington']

[ORG:'Brookings Institution']',the research group in'[LOC:

[ORG:'Idealab']',a self-described business incubator based in'[LOC:

'Washington']

'Los Angeles']

[ORG:'Open Text']',based in'[LOC:'Waterloo']

**[ORG** 'il]lirLaO' :''iBno'st'O]C:'Paris']

[ORG:'Omnicom']'in'[LOC:'New York']

[ORG: 'DDB Needham'] 'in' [LOC:'New York']

[ORG:'Kaplan Thaler Group']'in'[LOC:'New York']

[ORG:'BBDO South'] 'in' [LOC:'Atlanta']

[ORG: 'Georgia-Pacific'] 'in' [LOC:'Atlanta']

搜索关键字 in 的结果相当不错，即使它的检索结果也可能误报，例如：[ORG:House Transportation Committee],secured the most money in the [LOC:New York]; 利用简 单基于字符串的方法排除这样的填充字符串似乎不太可能。

如前文所示， CoNLL 2002 命名实体语料库的荷兰语部分不只包含命名实体标注， 也包含词性标注。我们可以设计对这些标记敏感的模式，如下面的例子所示。 show\_clause() 方法以分条形式输出关系，其中二元关系符号作为参数 relsym 的值被 指定①。

>>>from nltk.corpus import conll2002

>>> vny =nm



|  |  |  |
| --- | --- | --- |
| ·.. ( |  |  |
| is/VI | #3rd sing present and |  |
| ...was/V| | #past forms of the verb | zijn('be') |
| ...werd/V | #and also present |  |
| ...wordt/v | #past of worden('become') |  |
| . ) |  |  |
| .. ★ | #followed by anything |  |
| ...van/Prep | #followed by van('of') |  |



>>>VAN =re.compile(vnv,re.VERBOSE)

>>>for doc in conll2002.chunked\_sents('ned.train'):

for r in nltk.sem.extract\_rels('PER', 'ORG',doc,

corpus='conl12002',pattern=VAN):

print nltk.sem.show\_clause(r,relsym="VAN")① VAN("cornet\_d'elzius",'buitenlandse\_handel')

VAN('johan\_rottiers','kardinaal\_van\_roey\_instituut')

VAN('annie\_lennox','eurythmics')

**轮到你来：**

替换最后一行“为print show\_raw\_rtuple(rel,Icon=True,rcon=True)。结 果将显示表示两个 NE 之间关系的词以及左右默认十词窗口的上下 文。利用荷兰语词典，你也许能够找出为什么 VAN(annie\_lennox', 'eurythmics)结果是个误报。

**7.7** **小结**

● 信息提取系统搜索大量非结构化文本，寻找特定类型的实体和关系，并用它们 来填充有组织的数据库。这些数据库可以用来寻找特定问题的答案。

● 信息提取系统的典型结构以断句开始，然后是分词和词性标注。接下来在产生 的数据中搜索特定类型的实体。最后，信息提取系统着眼于文本中提到的相互 临近的实体，并试图确定这些实体之间是否有指定的关系。

· 实体识别通常采用分块器，它分割多标识符序列，并用适当的实体类型给它们 加标签。常见的实体类型包括组织、人员、地点、日期、时间、货币、GPE(地 缘政治实体)。

· 利用基于规则的系统可以构建分块器，例如：NLTK中提供的RegexpParser类； 或使用机器学习技术，如本章介绍的 ConsecutiveNPChunker。在这两种情况中，

词性标记往往是搜索分块时的一个重要特征。

· 虽然分块器专门用来建立相对平坦的数据结构，其中任意两个块不允许重叠， 但它们可以被串联在一起，建立嵌套结构。

· 关系抽取可以使用基于规则的系统，它通常查找文本中的连结实体和相关词的 特定模式；或使用机器学习系统，通常尝试从训练语料自动学习这种模式。

**7.8** **深入阅读**

本章额外的材料，包括网上免费提供的资源链接，发布在 <http://www.nltk.org/>。更 多关于使用NLTK分块的例子，请参阅 <http://www.nltk.org/howto>上的分块HOWTO。

分块的普及很大一部分取决于 Abney的开创性工作，如(Abney,1996a)。http://www. vinartus.net/spa/97a.pdf 中描述了 Abney 的Cass 分块器。

根据 Ross 和Tukey 在1975年的论文，词 **chink** 最初的意思是停用词序列(Abney, 1996a)。

IOB 格式(有时也称为**BIO Format)** 由 (Ramshaw&Marcus,1995) 开发，用来 NP 分块，并被由Conference on Natural Language Learning(CoNLL) 在1999年用 于NP 加括号共享任务。CoNLL2000 采用相同的格式标注了华尔街日报的文本，

将其作为NP 分块共享任务的一部分。

(Jurafsky &Martin,2008) 中的第13.5节包含有关分块的讨论。第22章讲述了信 息提取，包括命名实体识别。有关生物学和医学中的文本挖掘的信息，请参阅 (Ananiadou&McNaught,2006)。

亚历山大地名辞典和盖蒂地名辞典的更多信息请参阅 http:/en.wikipedia.org/wiki/ Getty\_Thesaurus\_of\_Geographic\_Names 和<http://www.alexandria.ucsb.edu/gazetteer/>。

**7.9** **练习**

1.OIOB 格式分类标注标识符为I 、O和 B。为什么3个标签必须同时使用?如果 我们只使用I 和 O 标记会造成什么问题?

2.O 编写一个标记模式匹配包含复数中心名词在内的名词短语，如 many/J researchers/NNS 、two/CD weeks/NNS 、both/DT new/JJ positions/NNS。尝试使用泛 化处理单数名词短语的标记模式。

3.O 选择 CoNLL-2000 分块语料库中3种块类型之一。查看这些数据，并观察组 成这种类型的块的 POS 标记序列的任 一 模式。使用正则表达式分块器 nltk.RegexpParser 开发一个简单的分块器。讨论任何难以可靠地分块的标记序列。

4.O 块的早期定义是出现在缝隙之间的材料。开发一个分块器以将完整的句子作 为单独的块开始，然后其余的工作完全由加缝隙完成。在你自己的应用程序中，确 定哪些标记(或标记序列)最有可能组成缝隙。相对于完全基于块规则的分块器， 比较这种方法的性能和易用性。

5.①编写一个标记模式，涵盖包含动名词在内的名词短语，如 the/DT receiving/VBG end/NN 、assistant/NN managing/VBG editor/NN。将这些模式加入到 grammar 中， 每行一个。使用自己设计的已标注的句子，测试工作。

6.①编写一个或多个标记模式处理有连接词的名词短语，如： July/NNP and/CC August/NNP 、all/DT your/PRP$managers/NNS and/CC supervisors/NNS 、company/NN courts/NNS and/CC adjudicators/NNS。

7.①利用之前已经开发的分块器执行下列评估任务。(请注意，大多数分块语料库 包含一些内部的不一致，以至于任何合理的基于规则的方法都将产生错误。)

a. 利用分块语料库中的100个句子评估分块器，报告精度、召回率和F 量度。

b. 使用 chunkscore.missed() 和 chunkscore.incorrect() 方法识别分块器的错误， 并讨论。

c. 将本章中评估部分讨论的基准分块器与你的分块器的性能相比较。

8.①使用基于正则表达式的块语法 RegexpChunk, 为 CoNLL 分块语料库中块类型 中的一种开发分块器。使用分块、加缝隙、合并或拆分规则的任意组合。

9.①有时词的标注不正确，例如12/CD or/CC so/RB cases/VBZ 的中心名词。替代 手工校正标注器的输出，好的分块器使用标注器的错误输出也能运作。查找其他使

用不正确的标记也能正确为名词短语分块的例子。

10.Obigram 分块器的准确性得分约为90%。研究它的错误，并试图找出它为什 么不能获得100%的准确率。实验trigram 分块。你能够进一步提高性能吗?

11.●在IOB 块标注上应用n-gram 和 Brill 标注方法。不是给词分配 POS 标记， 而是给 POS 标记分配IOB 标记。例如：如果标记 DT (限定符)经常出现在块的 开头，它将被标注为B(begin) 。 相对于本章中讲到的正则表达式分块方法，评估 这些分块方法的性能。

12.●在第5章中我们看到通过查找有歧义的 n-grams, 即在训练数据中有多种可 能的方式标注的n-grams, 可以得到标注性能的上限。应用同样的方法来确定n-gram 分块器的上限。

13.●挑选CoNLL 分块语料库中3种块类型之一。编写一个函数为你选择的类型 完成以下任务：

a. 列出与此块类型的每个实例一起出现的所有标记序列。

b. 计数每个标记序列的频率，并产生一个按频率减少的顺序排列的列表；每 行要包含一个整数(频率)和一个标记序列。

c. 检查高频标记序列。使用这些作为开发更好地分块器的基础。

14.●在评估一节中提到的基准分块器往往会产生比它预计产生的块更大的块。例 如：短语[every/DT time/NN][she/PRP]sees/VBZ [a/DT newspaper/NN]包含两个连续 的块，我们的基准分块器不正确地将前两个结合： [every/DTtime/NN she/PRP]。编 写一段程序，找出这些通常出现在块开头的块内部标记有哪些，然后设计一个或多 个规则分裂这些块。将这些与现有的基准分块器组合，重新评估它，看看你是否已 经发现了改进的基准。

15.●开发NP 分块器，转换POS 标注文本为元组的链表，其中每个元组由后面跟 着名词短语和介词的动词组成，如： the little cat sat on the mat becomes('sat','on', NP')。

16.●宾州树库样例包含一部分已标注的《华尔街日报》文本，已经按名词短语分

块。格式使用方括号，我们已经在本章遇到过几次。语料可以使用 for sent in nltk.corpus.treebank\_chunk.chunked\_sents(fileid) 来访问。这些都是单调的树状图，正 如我们使用 nltk.corpus.conll2000.chunked\_sents) 得到的一样。

a. 函数 nltk.tree.pprint()和 nltk.chunk.tree2conllstr() 可以利用树状图创建树库和 IOB字符串。编写函数 chunk2brackets()和 chunk2iob(), 以单独的分块树状 图为唯一的参数，返回所需的多行字符串表示方式。

a. 编写命令行转换工具 bracket2iob.py 和 iob2bracket.py, (分别)读取树库或 CoNLL 格式的文件，将它转换为其他格式。(从NLTK 语料库获得一些原 始的树库或 CoNLL数据，保存到文件，然后使用 for line in open(filename) 在 Python 访问它。)

17. ●n-gram 分块器可以使用除当前词性标记和n-1 个前面的块的标记以外其他信 息。调查其他的上下文模型，如 n-1 个前面的词性标记，或带有前后词性标记的前 面块标记组合。

18.●思考 n-gram 标注器使用临近标记的方式。观察分块器如何重新使用这个序 列信息。例如：这两个任务都使用名词往往跟在形容词后面(英文中)的信息。这 会出现相同的信息被保存在两个地方的情况。随着规则集规模的增长，这会成为一 个问题吗?如果会，推测解决这个问题的任何可能方式。



**第8章**

**分析句子结构**

前面的章节重点关注词：如何识别它们，分析它们的结构，给它们分配词汇类别， 以及获得它们的含义。我们还学习了如何识别词序列或n-grams 的模式。然而，这 些方法只是触碰到了管理句子用到的复杂约束条件的表面。我们需要一种能处理自 然语言中显著歧义的方法。我们还需要应对这样一个事实：句子有无限的可能，而 我们只能写有限的程序来分析其结构和发现它们的含义。

本章的目的是要回答下列问题。

(1)如何使用形式化语法来描述无限的句子集合的结构?

(2)如何使用句法树来表示句子结构?

(3)解析器如何分析句子并自动构建语法树?

在本章中，我们将介绍英语语法的基础，并学习句子含义系统化的一面，只要确定 了句子结构，将很容易掌握。

**8.1** **一些语法困境**

**语言数据和无限可能性**

前面的章节中已经讲述了如何处理和分析文本语料库，我们一直强调处理大量且每 天都在增加的电子语言数据是NLP 的挑战。细致地思考这些数据，大胆想象：我 们有一个巨大的语料库，包括在过去50年中，英文表达或写作的一切。我们称这

个语料库为“现代英语”合理吗?有许多原因可以解释为什么我们的回答可能是否 定的。回想一下，在第3章中，我们让你搜索网络查找 the of模式的实例。虽然很 容易在网上找到包含这个词序列的例子，例如： New man at the of IMG (见 <http://www.telegraph.co.uk/sport/2387900/New-man-at-the-of-IMGhtml>), 但说英语的人会 觉得这些例子中的大多数都是错误的，因为它们根本不是英语。

因此，我们可以说，“现代英语”与我们想象中的在语料库有非常大量的词序列集

合并不等同。说英语的人可以判断这些序列，并将拒绝其中不合文法的。

同样，可以很容易地构造一个新的句子，并让说话者认为它是非常好的英语。例如： 句子的一个属性：可以嵌入更大的句子中。考虑下面的句子。

(1)a.Usain Bolt broke the 100m record.

b.The Jamaica Observer reported that Usain Bolt broke the 100m record.

c.Andre said The Jamaica Observer reported that Usain Bolt broke the 100m record. d.I think Andre said the Jamaica Observer reported that Usain Bolt broke the

100m record.

如果用符号S 表示整个句子，我们将会看到像这样的模式：Andre said S 和 Ithink S。 这些模板用于获得一个句子，并构建一个更大的句子。我们还可以使用其他的模板， 如 ：S but S和 S when S。稍微花些心思，我们就可以使用这些模板构建一些长句 子。这里有一个令人印象深刻的例子，来自A.A.Milne 的《小熊维尼的故事》:《In Which Piglet Is Entirely Surrounded by Water》。

[You can imagine Piglet's joy when at last the ship came in sight of him.]In after-years he liked to think that he had been in Very Great Danger during the Terrible Flood,but the only danger he had really been in was the last half-hour of his imprisonment,when Owl,who had just flown up,sat on a branch of his tree to comfort him,and told him a very long story about an aunt who had once laid a seagull's egg by mistake,and the story went on and on, rather like this sentence,until Piglet who was listening out of his window without much hope,went to sleep quietly and naturally,slipping slowly out of the win-dow towards the water until he was only hanging on by his toes,at which moment,luckily,a sudden loud squawk from Owl,which was really part of the story,being what his aunt said,woke the Piglet up and just gave him time to jerk himself back into safety and say,"How interesting,

and did she?"when—well,you can imagine his joy when at last he saw the good ship, Brain of Pooh (Captain,C.Robin;1st Mate,P.Bear)coming over the sea to rescue him …

这个长句子其实结构很简单，以S but S whenS开始。从这个例子我们可以看到： 语言提供给我们的结构看上去是可以无限扩展的。同样，我们能理解从来没有听说 过的任意长度的句子：假造一个全新的句子并不难，它是一个在语言的历史上可能 从来没有使用过的句子，然而所有说这种语言的人都能理解它。

文法的目的是给出一个明确的语言描述。而我们思考文法的方式与我们认为什么是一种 语言紧密联系在一起。观察到的言语和书面文本是否是一个大却有限的集合呢?关于文 法句子是否存在一些更抽象的东西，如有能力的说话者能理解的隐性知识?或者是两者 的某种组合?我们不会解决这个问题，而是将介绍主要的方法。

在本章中，我们将采取“生成文法”的形式化框架，其中一种“语言”被认为仅仅 是所有合乎文法的句子的大集合，文法是一个形式化符号，可用于“生成”这个集 合的成员。文法使用S→S andS 形式的递归**产生式，**将在8.3节中探讨。第10章 中，我们将学习如何从句子组成部分的意思自动建立一个句子的意思。

**普遍存在的歧义**

一个众所周知的关于歧义的例子如(2)所示，来自 Groucho Marx的电影， Animal Crackers(1930)。

(2)While hunting in Africa,I shot an elephant in my pajamas.How an elephant got into my pajamas I'll never know.

让我们仔细看看短语：I shot an elephant in my pajamas 中的歧义。首先，我们需要 定义一个简单的文法。

>>>groucho\_grammar =nltk.parse\_cfg("""

S ->NP VP

.PP->P NP

...NP ->Det N l Det N PP !'I'

...VP ->V NP I VP PP

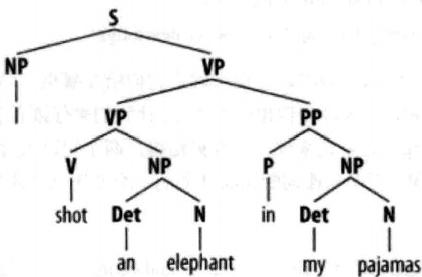
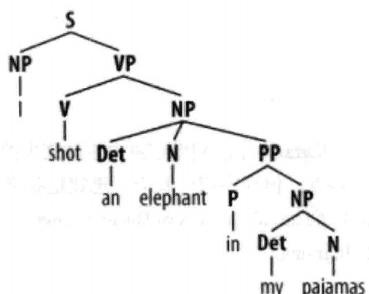
...N ->'elephant'l 'pajamas'

...Det ->'an'l 'my'

.V->'shot

.P-> 'in

""")



这个文法允许以两种方式分析句子，这取决于介词短语 in my pajamas 描述的是大

**象还是枪击事件。**

>>>sent=['I','shot','an','elephant','in','my','pajamas']

>>>parser =nltk.ChartParser(groucho\_grammar)

>>>trees =parser.nbest\_parse(sent)

>>>for tree in trees:

print tree

(S

(NP I)

(VP

(V shot)

(NP(Det an)(N elephant)(PP(P in) (NP(Det my)(N pajamas))))))

(S

(NP I)

(VP

(VP(V shot)(NP(Det an)(N elephant)))

(PP(P in)(NP(Det my)(N pajamas)))))

程序产生两个由括号括起的结构，我们可以用树来表示它们，如(3)所示。

(3) a.

b.



**轮到你来：**

思考下面的句子，看看你是否能想出两种完全不同的解释：Fighting animals could be dangerous.Visiting relatives can be tiresome. 个别词是 否有歧义?如果有，造成歧义的原因是什么?

请注意，相关的所有词的含义是没有歧义的。例如：词 shot 在第一句话中不表示 “使用枪的动作”,而在第二句中表示“使用相机”。

本章介绍文法和分析，例如以形式化的和可计算的方法对我们一直在讨论的语言现 象进行调查和建模。正如我们所看到的，词序列中符合语法规则的和不符合语法规 则的模式相对于短语结构和依赖性是可以被理解的。我们可以使用文法和解析器来 开发这些结构的形式化模型。与以前一样，重要的目的是自然语言understanding。 当识别一个文本所包含的语言结构时，可以从中获得多少文本的含义?一段程序在 通读了一个文本后，它能否足够“理解”文本，并回答一些简单的问题，如“发生 了什么事”或“谁对谁做了什么”?还像以前一样，我们将开发简单的程序来处理 已注释的语料库，并执行有用的任务。

**8.2** **文法的用途**

**超越n-grams**

在第2章中，我们给出了一个如何使用bigrams 中的频率信息来生成文本的例子， 生成短的词序列看上去似乎完全可以接受，但词序列一长会就迅速退化成无稽之 谈。下面是另一对例子，计算儿童故事《The Adventures of Buster Brown》( 包含在 古登堡工程选集语料库中)文本中的bigrams。

(4)a.He roared with me the pail slip down his back

b.The worst part and clumsy looking for whoever heard light

凭感觉能知道，错误的序列是“词、沙拉”,但很难确定它们错在哪里。学习文法 的一个好处是，凭借概念框架和词汇表能培养出这些直觉。让我们来仔细看看序列： the worst part and clumsy looking。这看起来像一个**并列结构，**两个短语通过并列连 词如 and 、but或 or 连接在一起。下面是连词的语法功能的一个非正式(并且简单) 的描述。

并列结构：如果v1和v2 都是文法类型X 的短语，那么v1andv2 也是X 类型的短语。

下面是几个相关例子。首先，两个NP (名词短语)连接在一起组成一个NP, 其次， 两个AP (形容词短语)连接在一起组成一个AP。

(5)a.The book's ending was (NP the worst part and the best part)for me. b.On land they are (AP slow and clumsy looking).

我们不能做的是连接NP 和 AP, 这就是the worst part and clumsy looking 不合语法的原 因。在我们形式化这些想法之前，需要了解成分结构的概念。

成分结构是词与词结合在一起组成的单元。通过词汇的可替代性可以证明词序列是 怎样形成这样一个单元的——也就是说，在符合语法规则句子中的词序列可以被一 个更小的且不会导致句子不符合语法规则的序列替代。为了弄清这个想法，思考下 面的句子。

(6)The little bear saw the fine fat trout in the brook.

事实上，我们可以将He 代替为 The little bear, 这表明后者是一个单元。相比之下， 我们不能以同样的方式代替little bear saw。(在句子开头加注星号表示它不符合语 法规则。)

(7)a.He saw the fine fat trout in the brook.

b.\*The he the fine fat trout in the brook.

在图8-1 中，我们系统地用较短的序列替代较长的序列，并使其依然符合语法规则。 事实上，形成单元的每个序列都可以被单独的词替换，最终得到两个元素。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | the | little | **bear** | **saw** | **the** | **fine** | fat | trout | in | the | **brook** | | the | **bear** | | **saw** | the | trout | | | in | it | | | He | | | **saw** | it | | | | there | | | | He | | | ran | | | | | there | | | | He | | | ran | | | | | | | | |

图8-1 词序列的替代：从最上面一排开始，我们用单个词(如：it)替换特定的词序列(如：the brook); 重复这一过程，最终得到一个符合语法规则且只有两个词的句子



(8)

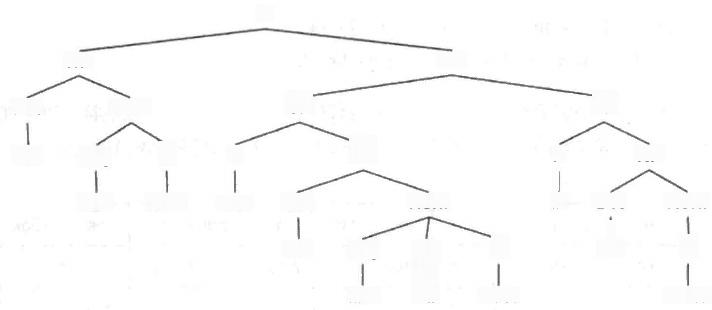
S

在图8-2中，为前面的图中的词增加了文法类别标签。标签NP 、VP 和 PP 分别表 示名词短语、动词短语和介词短语。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Det  the | Adj little | N  bear | V  saw | Det  the | Adj  fine | Adj  fat | N  trout | P  in | Det  the | N  brook | | Det  the | Nom  bear | | V  saw | Det  the | Nom  trout | | | P  in | NP  it | | | NP  He | | | V  saw | NP  it | | | | PP  there | | | | NP  He | | | VP  ran | | | | | PP  there | | | | NP  He | | | VP  ran | | | | | | | | |

图8-2 词序列的替换，附加文法分类：此图再现了图8-1并给名词短语(NP)、 动词短语(VP)、 介词短语(PP) 及名词性词(Nom) 也添加了相应的文法分类

如果现在将最上面的词汇分离，增加一个S 节点，再把图翻转，最终我们得到一个 标准的短语结构树，如(8)所示。此树的每个节点(包括词)被称为**组成部分(成** **分，** **constituent)** 。S 的**直接组成部分**是 NP 和 VP。



NP

Det Nom

the Adj N

little bear

|

N

trout

Nom

N

brook

Nom

Adj

一

Det

the

Adj

fine

P

n

the

Np

Det

Pp

VP

saw

NP

vF

V

域

正如我们在8.1 节中所看到的，句子长度是任意的。因此，短语结构 树可以有任意深度。我们在7.4节中看到的级联块解析器只能产生有 限深度的结构，所以分块方法在这里并不适用。

在下一节中，我们将看到一个指定的文法如何将句子细分成它的直接成分，以及如 何将这些进一步细分，直到成为单独词汇。

**8.3** **上下文无关文法**

**一种简单的文法**

首先，学习一个简单的上下文无关文法 (context-free

grammar,CFG) 。 按照惯例，

第一条生产式的左端是文法的**开始符号，**通常是 S, 所有符合语法规则的树都必 须有这个符号作为它们的根标签。NLTK 中，上下文无关文法定义在 nltk.grammar 模块。在例8-1 中，我们定义了文法，并示范了如何分析一个简单的符合文法的 句子。

例8-1 一个简单的上下文无关文法的例子

grammar1 =nltk.parse\_cfg("""

S ->NP VP

VP ->V NP |V NP PP

PP ->P NP

V->"saw"|"ate"|"walked"

NP ->"John"|"Mary"|"Bob"l Det N l Det N PP

Det ->"a"l"an"l"the"l"my"

N->"man"|"dog"l"cat"l"telescope"l"park"

P ->"in"l"on"l"by"|"with"

>

rfpr n\_.i\_scpeanrtsPea(t()g:rammar1)

print tree

。

(S(NP Mary) (VP(V saw) (NP Bob)))

例8-1中的文法包含涉及各种句法类型的产生式，如表8-1中所列出的。在这里使 用的递归下降解析器也可以通过图形界面查看，如图8-3所示；我们在8.4节中将 更详细地讨论这个解析器。

|  |  |  |
| --- | --- | --- |
| X Recursive Descent Parser Demo | | |
| **File Apply View**  **Available Expansions**  S>NP VP  NP->Det N PP  NP ->Det N  VP-xV NP PP  vP->VNP  VP->V  PP-SP NP  NP->中  Det->he  Det->'a'  N->'man'  N->'park'  N->'dog  N->'telescope'  V->'ate'  V ->'saw'  P->'in'  P-\*'under'  P->'with'  Last Operation: Match:the | **Animate Help** | |
| §  四  Det N P | *Vp* |
| the  the dog saW a man in the.park | |
| **Step Autostep Expand Match Backtrack** | | |

**图8-3** **递归下降解析器演示：在这个递归下降解析器生长出解析树并与其输入匹配时，此工具**

**可以让你看到它的操作过程**

**表8-1**

**句法类型**

|  |  |  |
| --- | --- | --- |
| 符 号 | **意** **思** | **例** 子 |
| S | 句子 | the man walked |
| NP | 名词短语 | a dog |
| VP | 动词短语 | saw a park |
| PP | 介词短语 | with a telescope |
| Det | 限定词 | the |
| N | 名词 | dog |
| V | 动词 | walked |
| P | 介词 | in |

产生式如VP->VNP|VNPP P 右侧脱节了，中间显示|,这是两个产生式VP->VNP

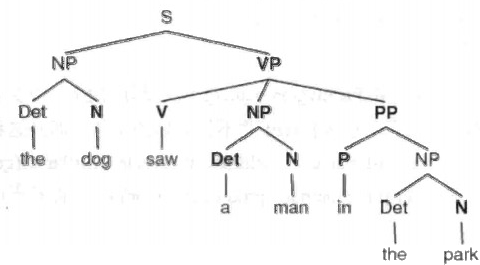
和VP->VNP PP 的缩写。

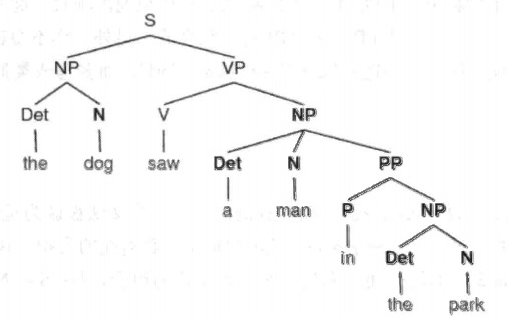
如果我们使用例8-1所显示的文法分析句子 The dog saw a man in the park, 结果将 得到两棵树，类似于我们在(3)中所看到的。

(9)

a.

b.





由于这句话的两棵树都符合我们的文法规则，这句话被称为**结构上有歧义**。正如我 们在本章前面看到的，这个问题中的歧义被称为**介词短语附着歧义**。如果你还记得， 这是一个附着歧义，因为此 PPin the park 需要附着在树中两个位置中的任意一个： 要么是 VP 的“孩子”要么是NP 的“孩子”。当 PP 附着在 VP 上时，合理的解释 是：看到公园里发生的事情。然而，如果 PP 附着在 NP 上，意思就是：在公园里 的人，作为主语，看到(狗)可能已经坐在公寓的阳台上俯瞰公园了。

**编写你自己的文法**

如果你有兴趣尝试编写上下文无关文法 CFG, 你会发现在一个文本文件，比如 mygrammar.cfg 中创建和编辑你的语法会很有帮助。然后，你可以将它加载到 NLTK

中，并按如下方式进行分析。

>>gsrary sab.d"ai)('file:mygrammar.cfg')

>>>rd\_parser =nltk.RecursiveDescentParser(grammar1)

>>>for tree in rd\_parser.nbest\_parse(sent):

print tree

确保文件名后缀为.cfg, 并且字符串'file:mygrammarcfg'中间没有空格符。如果命令 print tree 没有产生任何输出，可能是因为句子sent 并不符合你的文法。遇到这种情况可 以将解析器的跟踪设置打开 rd\_parser =nltk.RecursiveDescentParser(grammarl, trace=2) 。你还可以使用命令for p in grammarl.productions():print p,查看当前使用的 文法中的产生式。

当你在 NLTK 中为了解析编写 CFG 时，不能将文法类型与词汇项目一起写在同一 个产生式的右侧。因此，产生式PP->'of NP是不允许的。另外，你不得在产生式

右侧放置多字词汇项。因此，不能写成 NP>New York', 而要写成类似 NP >

New\_York '的样子。

**句法结构中的递归**

如果出现在产生式左侧的文法类型也出现在右侧，那么这个文法被认为是递归的，

如例8-2所示。产生式Nom >Adj Nom (其中 Nom 是名词性的类别)包含Nom 类型的直接递归，而S 上的间接递归来自于两个产生式的组合，即： S>NPVP 与

VP->VS。

**例8-2 递归的上下文无关文法**

grammar2 =nltk.parse\_cfg("""

S ->NP VP

NP ->Det Nom l PropN

Nom ->Adj Nom l N

VP ->V Adj lV NP I VS lV NP PP

PP ->P NP

PropN ->'Buster'l'Chatterer'l'Joe'

Det ->'the' l 'a'

N->'bear'l'squirrel'l'tree'l'fish'l'log'

Adj ->'angry'l 'frightened'I 'little' l 'tall'

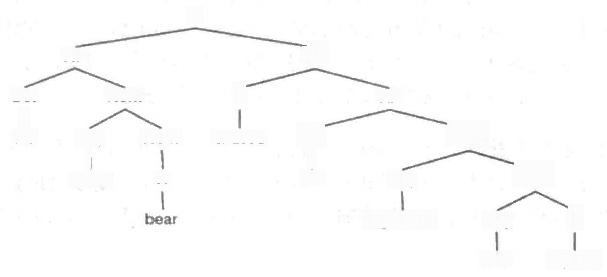
V-> 'chased' |'saw'l.'said'|'thought'l'was'l 'put'

P ->'on'

""")

要看递归如何从这个语法产生的，先思考下面的树。(10)a 包括嵌套的名词短语， 而(10)b 包含嵌套的句子。

(10)

a. S

NP

Det Nom

te Adj Nom

1

angry N

√

chased

vp

Dlet

the

NP

Adj

frightened

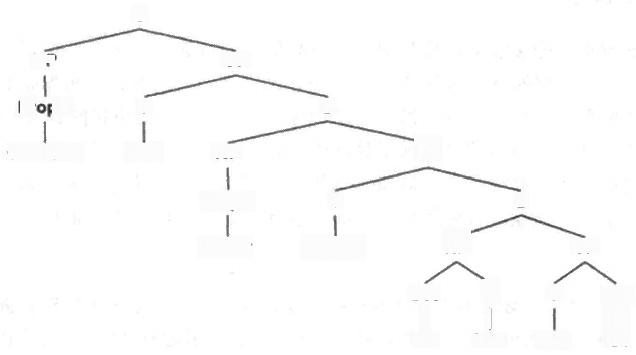
Nom

Nom

Adj N

little Squirrel

b.

NFP

**Pr pN**

Chatterer

S

V

said

**VP**

**NP**

**PropN**

Bustei

S

V

thought

vp

S

|  |  |  |
| --- | --- | --- |
| NP | | VP |
| Det  l  the | N  I  tree |
| V Adj  was 晶 |

我们在这里只演示了两个层次的递归，但递归深度是没有限制的。你可以尝试分析 更深层次嵌套结构的句子。请注意， RecursiveDescentParser 是无法处理形如X>X

Y 的左递归产生式的；我们将在第8.4节谈及这部分内容。

**8.4** **上下文无关文法分析**

**解析器根**据文法产生式处理输入的句子，并建立一个或多个符合文法的组成结构。 文法是一个格式良好的声明规范——实际上它只是一个字符串，而不是程序。解析

器是一种文法解释程序。用于搜索符合文法的所有树的空间，并找出一棵边缘有所 需句子的树。

解析器使用一组测试句子对文法进行评估，可以帮助语言学家发现在他们的文法分 析中存在的错误。解析器可以作为心理语言处理模型，帮助解释人类在处理某些句 法结构时遇到的困难。许多自然语言应用程序都某种程度上涉及文法分析，例如我 们会期望自然语言问答系统对提交的问题首先进行文法分析。

在本节中，我们将看到两个简单的分析算法， 一种自上而下的方法称为下降递归分 析， 一种自下而上的方法称为移进-归约分析。我们也将看到更复杂的算法， 一种 称为左角落分析的带自下而上过滤的自上而下的方法； 一种称为图表分析的动态规 划技术。

**递归下降解析器**

最简单的解析器把文法解释成如何将一个高层次的目标分解成几个低层次的子 目标的规范。顶层的目标是找到一个S 。S→NP VP生产式允许解析器把这个目 标替换为两个子目标：先找到NP, 然后再找到VP 。通过使用左侧有NP 和 VP 的产生式，这些子目标将再次被子目标的子目标替代。最终，这种扩张过程完 成了子目标，如：找到词 telescope。这样的子目标可以直接与输入序列比较， 如果下一个单词匹配就说明找到了。如果没有匹配，解析器必须备份，并尝试 其他选择。

递归下降解析器在上述过程中建立分析树。以最初的目标(找到一个 S), 创建 S 根节点。随着在上述过程中使用文法的产生式来递归扩展达到其目标，分析树不断 向下延伸(故名为下降递归)。我们可以在图形化示范 nltk.app.rdparser()中看到这 个过程。执行此解析器的6个阶段，如图8-4所示。

在这个过程中，解析器往往被迫在多种可能的产生式中选择。例如：第3步到第4 步，它试图找到左侧有N 的产生式。第一个是 N→man。不起作用时就回溯，按顺 序尝试其他左侧有N 的产生式，直到它找到N→dog, 这与输入句子中的下一个词 相匹配。 一段时间后，如第5步所示，找到了完整的分析树。这是一个涵盖了整个 句子的树，没有任何悬着的边。 一旦分析树被发现，我们可以让解析器继续寻找其 他额外的分析树。它将再次回溯以探索选择其他产生式，以免漏掉任何一个产生分



析树的情况。

|  |  |  |
| --- | --- | --- |
| **1.初始期**  **4.无法与man匹配** | 2.第二成果    **5,完整的解析** | 3.与the匹配  图  P  H  he  he  **6.回溯** |

**图8-4** **递归下降解析器的6个阶段：解析器以一棵包含节点S** **的树开始；每个阶段它都会查询** **文法来找到一个可以用于扩大树的产生式；当遇到一个词汇产生式时，将它的词与输入比较；发** **现一个完整的分析树后，解析器将回溯以寻找更多的分析树**

NLTK提供了一个递归下降解析器。

s'Mary san .rcentParser(grammar1)

>>>for t in rd\_parser.nbest\_parse(sent):

· print t

(S(NP Mary)(VP(V saw)(NP(Det a)(N dog))))

**RecursiveDescentParser()接受一个可选的参数** **trace。如果** **trace 大于零，**

**则解析器将报告它解析文本的步骤。**

递归下降分析有3个主要的缺点。首先，左递归产生式，如： NP>NP PP, 会进

入死循环。第二，解析器在处理不符合输入句子的词和结构时会浪费很多时间。第 三，回溯过程中可能会丢弃分析过的成分，需要再次重建。例如：从 VP>V NP 上回溯将丢弃为NP 创建的子树。如果解析器处理 VP->V NP PP, 那么 NP 子树必 须重新创建。

递归下降分析是一种**自上而下分析。**自上而下解析器会使用文法来预测输入是什

么。然而，由于输入对解析器一直是可用的，所以需要从一开始就考虑输入的句子。

这种方法被称为**自下而上分析，**在下一节中我们将看到相关例子。

**移进-归约分析**

**移进-归约解析器是**一种简单的自下而上解析器。与所有自下而上的解析器一样， 移进-归约解析器试图找到对应文法产生式右侧的词和短语的序列，用左侧的序列 来替换它们，直到整个句子只剩下S。

移位-规约解析器反复将下一个输入词推到堆栈(见4.1节),这叫移位操作。如果 堆栈上的前n 项，匹配某个产生式右侧的n 个项目，那么就把它们弹出栈，并把产 生式左边的项目压入栈。这种替换前n 项为一项的操作就是规约操作。此操作只适 用于堆栈的顶部；规约栈中的项目必须在后面的项目被压入栈之前完成。当所有的 输入都使用过，堆栈中只剩余一个项目，也就是当分析树把S 节点作为它的根时， 解析器便完成此步骤。移位-规约解析器通过上述过程建立了一颗分析树。每次弹 出堆栈n 个项目，它就将它们组合成为一颗部分分析树，然后将其压回推栈。我们 可以使用图形化示范nltk.app.srparser()看到移位-规约分析算法步骤。执行此解析器 的6个阶段如图8-5所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** **.** **初** **始** **期**  Stack | Remaining Tezt tte dog saw a man in the pak | **2.** **一次偏移后**   |  |  | | --- | --- | | Stack | Remaining Text |   the dog saw a mun in the pak |
| **3.** **减少偏移后** | | **4.** **识别出第2个NP** **后**  Remaning Texi the park  Det  màn  MP n  如  SLack  ?  MP |
| suack  0et N  T…  the | Remaining Text saw a man in the pak |
| 一星  一星  - 星  5. 构建一个复杂的NP 后  V 甲  p    四 p  Remaining Tert  Suack HP | |
| **6.** **构建一个完整的解析树** |

**图8-5** **移进-归约解析器的6个阶段：解析器一开始把输入的第一个词转移到堆栈；** **一旦堆栈** **顶端的项目与一个文法产生式的右侧匹配，就可以将它们用那个产生式的左侧部分替换；当所有** **输入都被使用过且堆栈中只有剩余一个项目S时，解析器就成功结束以上步骤**



NLTK 中提供了 ShiftReduceParser(), 一个简单的移进-归约解析器实现。这个 解析器不执行任何回溯，所以它不能保证一定能找到文本的解析，即使确实存 在一个这样的解析。此外，它最多只能找到一个解析，即使有多个解析存在。 我们可以提供一个可选的 trace 参数，用来控制解析器报告它分析文本步骤的 繁琐程度。

s'Mary .hdie' upcleiser(grammarl)

>>>print sr\_parse.parse(sent)

(S(NP Mary)(VP(V saw)(NP(Det a)(N dog))))

**轮到你来：**

**以跟踪模式运行上述解析器，看看序列的移进和规约操作，使用**

**sr\_parse =nltk.ShiftReduceParser(grammarl,trace=2)。**

移进-规约解析器可能会进入一个死胡同，而找不到任何解析，即使输入的句子是 符合语法的。这种情况发生时，没有剩余的输入，而堆栈所包含的项目不能被规约 成S 。问题出现的原因是：较早前做出的选择不能被解析器撤销(虽然图形演示中 用户可以撤消它们的选择)。解析器可以做两种选择：(1)当有多种规约可能时选 择哪种规约；(2)当移进和规约都可以时选择哪个动作。

可以通过移进-规约解析器改进执行策略来解决这些冲突。例如：可以通过规定只 有在不能规约时才移进，来解决移进-规约冲突；可以通过优先执行规约操作从堆 栈移除大部分的项目，来解决规约-规约冲突。(一个通用的移进-规约解析器，是 一个“超前LR 解析器”,在编程语言编译器中普遍使用。)

移进-规约解析器相比递归下降解析器的好处是，它们只建立与输入中的词对应 的结构。此外，对每个子结构它们只建立一次。例如： NP(Det(the),N(man))

只建立和压入栈一次，不管以后VP->VNP PP规约或者NP->NP PP规约会 不会用到。

**左角落解析器**

递归下降解析器的问题之一是当它遇到一个左递归产生式时，会进入无限循环。这 是因为它盲目应用文法产生式而不考虑实际输入的句子。左角落解析器是自下而上 与自上而下方法的混合体。

**左角落解析器**是一个带自下而上过滤的自上而下的解析器。不像普通的递归下降解 析器，它不会陷入左递归产生式的陷阱。在开始工作之前，左角落解析器会预先处 理上下文无关文法以建立一个表，其中每行包含两个单元，第一个存放非终结符， 第二个存放此非终结符可能有的左角落集合。表8-2 用 grammar2 的文法演示了这 一点。

**表8-2**

**grammar2** **的左角落**

|  |  |
| --- | --- |
| **类** **型** | **左角落(非终结符)** |
| S | NP |
| NP | Det,PropN |
| VP | V |
| PP | P |

解析器每次考虑产生式时，它会检查下一个输入词是否与左角落表格中至少一种预 终结符的类别相容。

**符合语句规则的子串表**

上面讨论的简单的解析器在完整性和效率上都有限制。为了弥补这些，我们将运用 **动态规划**算法设计技术来分析这些问题。正如我们在4.7节中所看到的，动态规划 存储中间结果，并在适当的时候重用它们，能显著提高效率。这种技术可以应用到 句法分析，使我们能够存储分析任务的部分解决方案，然后在必要的时候查找它们， 直到达到最终解决方案。这种分析方法被称为**图表分析。**我们在本节中将介绍它的 主要思想，更多的实施细节请看网上提供的关于本章的材料。

通过动态规划我们只能够建立一次 PP in my pajamas。第一次建立时把它存入一个 表格中，然后在需要将它作为对象NP 或高一级的 VP 的次成分时查找表格。这个 表格被称为**符合语法规则的子串表**或简称为 WFST。 (术语“子串”指一个句子中 的连续的词序列。)我们将展示如何自下而上地构建 WFST, 以便系统地记录已经 找到的句法成分。

设置我们的输入为句子(2)。WFST 指定数值的跨度让人联想起 Python 的分片符 号(见3.2节)。这种数据结构的另一种方式如图8-6所示，是一个称为**图表的**数据 结构。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ◎ | ① | shot | ② - | an | eleph ③ | ant | ④ | in | ⑤ - | my ⑥ pajamas ⑦ |

**图8-6** **图表数据结构：词作为线性图结构边上的标签**

在 WFST 中，通过在三角矩阵中填充单元来记录词的位置：纵轴表示一个子串的 起始位置，而横轴表示结束位置(从而shot 将出现在坐标(1,2)的单元中)。为 了简化这个演示，假定每个词有一个独特的词汇类别，我们将会在矩阵中存储词

汇类别(不是词)。所以单元(1,2)将包含条目V 。更一般的，如果输入的字符 串是 a₁a₂…an, 且文法包含一个形为 A→a;的产生式，那么我们把 A 添加到单元

(i-1,i)。

所以，对于 text 中的每个词，都可以在文法中查找它所属的类别。

>>>text

[V ->'shot']

=['I','shot','an','elephant','in','my','pajamas']

对于 WFST, 创建一个 (n-1)×(n-1) 的矩阵作为 Python 链表的链表，在例8-3

的函数 init\_wfst) 中，用每个标识符的词汇类型对它进行初始化。定义一个实用的 函数 displayO以输出 WFST 。正如预期的那样，在(1,2)单元中存在V。

例8-3 使用符合语句规则的子串表接收器

def init\_wfst(tokens, grammar):

numtokens =len(tokens)

wfst=[[None foriin range(numtokens+1)]forjin range(numtokens+1)]

productions =grammar.productions(rhs=tokens[i]) wfst[i][i+1]=productions[0].lhs()

for i in range(numtokens):

return wfst

def complete\_wfst(wfst,tokens,grammar,trace=False):

index =dict((p.rhs(),p.lhs())for p in grammar.productions())

numtokens =len(tokens)

for span in range(2,numtokens+1):

for =artrmtokens+1-span):

for nt,ein**d**])wfst[mid][end]

if fstarn,[)t1]

if trace:

print "[8 s] 83s [8s] 83s [8s ]=>[8 s] 83s [8s ]" 名\ (start , nt 1, mid, nt 2, end , start , index [( nt1, nt 2)], end)

return wfst

def display (wfst , tokens ):

print '\ nWFST '+''. join ([("8-4 d "% i ) foriin range (1, len (wfst ))])

for i in range ( len ( wfst )-1):

print "8 d "8 i,

for j in range (1, len (wfst )):

print " g-4 s " 名 (wfst [ i ][ j ] or '.'),

print

>>> tokens = "I shot an elephant in my pajamas ". split ()

>>> wfst 0 = init \_wfst ( tokens , groucho \_grammar)

>>> display (wfst 0, tokens)

WFST 12 3 4 5 6 7

0 NP ·

1 V

2 Det .

3 N

4 P

5 Det .

6 . N

>>> wfst 1= complete \_wfst (wfst 0, tokens , groucho \_grammar)

>>> display (wfst 1, tokens)

WFST 1:2 3 4 5 67

0 NP S S

1 V VP VP

2 Det NP

3 N

4 PP

5 Det NP

6 N

回到表格表示法，假设对于词 an 在 单 元 ( 2 , 3 ) 有 Det ,对于词 elephant 在单元(3,4) 有 N , 对 于 an elephant 应该在(2,4)输入什么?我们需要找到 一 个形如 A → Det N 的

产生式。查询文法，知道我们可以在单元(0,2)输入NP。

更 一 般的，如果有 一 个产生式A → BC ,我们可以在(i , j) 中 输 入A ,并且在(i, k)中找到非 终 结 符B , 在 (k, j )中找到非终结符C。例8 - 3 中的程序使用此规则来完成 WFST 。 通 过 在调用函数 complete \_wfst )时设置 trace 为 True ,跟踪结果显示 WFST 正在被创建。

>>> wfst 1 = complete \_wfst ( wfst 0, tokens , groucho \_grammar , trace =True)

[2] Det [3] N [4]==>[2] NP [[4]](#bookmark120)

[5] Det [6] N [7]==> [5] NP [[7]](#bookmark121)

[1]V[2] NP[4]==>[1] VP [[4]](#bookmark122)



[4] P[5] NP[7]==>[4] PP [[7]](#bookmark117)

[0] NP[1] VP [4]==>[0] S [[4]](#bookmark123)

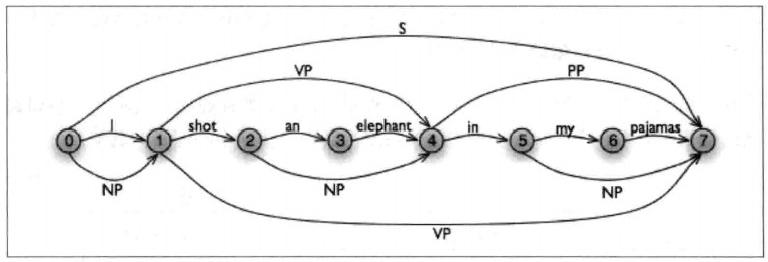
[1] VP [4] PP [7]==>[1] VP [[7]](#bookmark124)

[0] NP[1] VP[7]==>[0] S[[7]](#bookmark125)

为了更简便地通过产生式的右侧检索产生式，可为文法创建一个索 引。空间-时间权衡的一个例子：对语法做反向查找，每次通过右侧查 找产生式时，不必遍历整个产生式链表。

例如：由于在 wfst[0][1]找到了 Det, 在 wfst[1][2] 找到了N, 可以添加 NP 到 wfst[0] [2]中。

得出结论：只要在单元(0,7)中构建 S 节点，就表明我们已经找到了能涵盖整个输 入的句子，即为整个输入字符串找到了一个解析。最后的WFST 状态如图8-7所示。



**图8-7** **图数据结构：图中额外的边表示非终结符**

请注意，在这里我们没有使用任何内置的分析函数。我们已经可以从头开始建立一 个完整的初级图表解析器了!

WFST 有几个缺点。首先，WFST 本身不是一个分析树，所以严格地说该技术认识 到句子是被文法承认，而不是分析它。其次，它要求每个非词汇文法生产式都必须 是二元的。虽然可以将任意的 CFG 转换为这种形式，但是我们宁愿使用没有这样 规定的方法。最后，作为一个自下而上的方法，它潜在地存在着浪费，它会在不符 合文法的地方提出成分。

最后， WFST 并不能表示句子中的结构歧义(如两个动词短语的读取)。单元(2,



8)中的 VP 实际上被输入了两次， 一 次是读取VNP, 一 次是读取 VP PP。这 是 不同的假设，第二个会覆盖第 一 个(虽然如此，但这并不重要，因为左侧是相 同的。)图表解析器将使用更丰富的数据结构和 一 些有趣的算法来解决这些问题 (见8 . 8节)。

轮到你来：

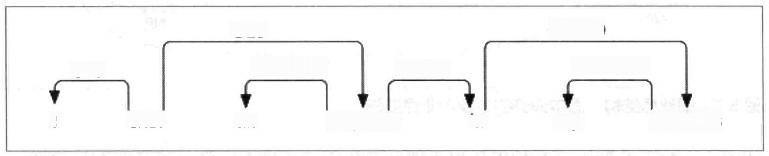
尝试交互式图表解析器应用程序 nltk.app.chartparser() 。

**8.5** **依存关系和依存文法**

短语结构文法是关于词和词序列如何结合形成句子成分的。 一种独特且互补的方 式， **依** **存** **文** **法** **，**集 中关注的是词与其他词之间的关系。依存关系是一个中心词与其 **从属之**间的二元非对称关系。 一个句子的中心词通常是动词，所有其他词要么依赖 **于中心词，** 要 么 通**过依赖**路径与它相关联。

依存关系表示是加标签的有向图，其中节点是词汇项，加标签的弧表示从中心词到

从属的依赖关系。图8-8所示为依存关系图，其中箭头从中心词指向其从属。



OBJ PMOD

SBJ DETMOD NMOD DETMOD

1 shot an elephant in my pajamas

图8-8 依存结构：箭头从中心词指向其从属；标签表示从属的语法功能，如主语、宾语或

修饰语

图8-8 中的弧添加了从属词与中心词之间的语法功能标签。例如： I 是 shot ( 整 个

句子的中心词)的 SBJ ( 主 语 ) ,in 是 NMOD(elephant 的名词修饰语)。与短语结

构文法相比，依存文法可以作为一种依存关系用来直接表示语法功能。

下面是 NLTK 中为依存文法编码的一种方式，注意它只能捕捉依存关系信息，不能

指定依存关系类型。

>>>groucho\_dep\_grammar =nltk.parse\_dependency\_grammar("""

.'shot'->'I'l'elephant'|'in

...'elephant'->'an'l'in'

··'in'->'pajamas!

..'pajamas'->'my'

>>>p”rngroucho\_dep\_grammar

Dependency grammar with 7 productions

'shot'->'elephant'

'shot'->'I'

'elephant'->'an'

'shot'->'in'

'elephant'->'in'

'in'->'pajamas'

'pajamas'->'my'

依存关系图是一个**投影，**若所有的词都按线性顺序书写，则用边连接这些词且保证 边不交叉。这等于是说一个词及其所有子节点(从属及其从属的从属，等等)在句 子中形成一个连续的词序列。图8-8所示为一个投影，我们可以使用投影依存关系 解析器分析更多的英语句子。下面的例子演示了groucho\_dep\_grammar 如何使用另 一种替代方法来捕捉附着歧义，这在我们之前研究短语结构语法中所遇到过的。

>>>pdp=nltk.ProjectiveDependencyParser(groucho\_dep\_grammar)

espa(sant in my pajamas'.split()

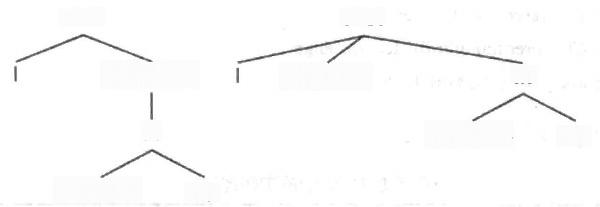
>>>for tree in trees:

· print tree

(shot I(elephant an(in(pajamas my))))

(shot I(elephant an)(in(pajamas my)))

(11)

shot

**elephant**

in

pajamas my

shot

elephant

in

pajamas my

这些括起来的依存关系结构也可以显示为树，从属作为它们的中心词的“孩子”。

在比英语具有更灵活词序的语言中，非投影的依存关系更加常见。

在一个成分C 中决定哪个是中心词H, 哪个是依赖D, 对于这个问题已经提出了很 多不同的标准。最重要的有以下几种。

(1)H 决定类型C 的分布；或者， C 的外部句法属性取决于H。

(2)H 定义C 的语义类型。

(3)H 必须有而D 是可选的。

(4)H 选择 D 并且决定它是必须有的还是可选的。

(5)D 的形态由H 决定(如 agreement 或 case government)。

若在一个短语结构文法中：PP 的直接成分是P 和 NP 时，需要注意中心词/从属之 间的区别。介词短语是一个短语，其中心词是一个介词。此外，NP 是 P 的从属。 之前讨论过的其他类型的短语结构文法中也存在着同样的区别。这里要注意的关键 点是：虽然短语结构文法看上去似乎与依存关系文法非常不同，但它们隐含着依存 关系。虽然 CFG 不会直接捕获依存关系，但最近的语言框架更多的是采用这两种 方法的结合形式。

**配价与词汇**

让我们来仔细看看动词和它们的从属。例8-2中的文法生成了类似(12)的例子。

(12) a.The squirrel was frightened.

b.Chatterer saw the bear.

c.Chatterer thought Buster was angry.

d.Joe put the fish on the log.

以上句子对应表8-3中的产生式。

**表8-3** **VP** **产生式和它们的中心词汇**

|  |  |
| --- | --- |
| 产 生 式 | 中心词汇 |
| VP->VAd | was |
| VP ->VNP | saw |
| VP ->VS | thought |
| VP->VNP PP | put |



也就是说，was 可以与跟在其后的Adj一起出现，saw 可以与跟在其后的NP一起出现， thought可以与跟在其后的S 一起出现，put可以与跟在其后的NP 和 PP一起出现。从属 的 Adj 、NP 、PP 和 S 通常被称为各自动词的补语，什么动词可以和什么补语一起出现 是具有很强约束性的。与(12)对比，(13)中的词序列是不符合语法规则的。

(13) a.\*The squirrel was Buster was angry.

b.\*Chatterer saw frightened.

c.\*Chatterer thought the bear.

d.\*Joe put on the log.

稍微用点儿想象力就可以设计一个上下文，其中动词和补语的不常见 的组合也可以解释的通。然而，我们假设(13)中的例子在中性语境 是可以解释得通的。

在依存文法的传统中，在表8-3中的动词被认为具有不同的配价。配价限制不仅适 用于动词，也适用于其他类的中心词。

在基于短语结构文法的框架内，提出了利用各种技术来排除(13)中不合语法的例子。 在CFG 中，我们需要一些限制文法产生式的方式，使得扩展了VP 后的动词只与正确 的补语一同出现。我们也可以将动词划分成更多“子类别”,每个子类别都与一组不同 的补语相关联。例如：**及物动词，**如： chased 和 saw, 需要后面跟 NP 对象补语；它 们是 NP 大类别的**子类别**。如果我们为及物动词引入一个新的类别标签，叫做 TV (transitive verb),那么我们可以在下面的产生式中使用它。

VP ->TV NP

TV ->'chased'|'saw'

现在\*Joe thought the bear 被排除了，因为我们没有将 thought 作为 TV 列出，但 Chatterer saw the bear 仍然是允许的。表8-4列举了更多动词子类别标签的例子。

**表8-4** **动词子类别**

|  |  |  |
| --- | --- | --- |
| **符** **号** | **含** **义** | **例** **子** |
| IV | Intransitive verb | barked |
| TV | Transitive verb | saw a man |
| DatV | Dative verb | gave a dog to a man |
| SV | Sentential verb | said that a dog barked |

配价是一个词项的属性，我们将在第9章进一步讨论它。补语往往与修饰语对照， 虽然两者都是一种从属。介词短语、形容词和副词通常充当修饰语。与补充不同， 修饰语是可选的，经常可以进行迭代，不像补语那样受中心词选择。例如：副词 really 在(14)所有的句子中都可以被添加为修饰语。

(14) a.The squirrel really was frightened.

b.Chatterer really saw the bear.

c.Chatterer really thought Buster was angry.

d.Joe really put the fish on the log.

我们已经在短语结构和依存文法中说过的 PP 附着的结构歧义，在语义上对应着修 饰语范围上的模糊含义。

**扩大规模**

到目前为止，我们只考虑了“玩具文法”,用于演示分析过程中关键环节的小型文 法。但有一个明显的问题就是这种做法是否可以扩大到能覆盖自然语言中的大型语 料库。手工构建这样的一套产生式有多么困难?一般情况下，答案是：非常困难。 即使我们允许自己使用各种正规工具，它们可以为文法产生式提供更简洁的表示。 但覆盖某种语言的主要成分所需要的众多产生式之间有着复杂的相互作用，所以对 这一过程保持控制仍然是极其困难的。换句话说，很难将文法模块化，以致每部分 文法都需独立开发。反过来意味着，在一个语言学家团队中分配编写文法的任务是 最困难的。另一个困难是当文法扩展到包括更加广泛的成分时，适用于任何一个句 子的分析也会相应增加。换句话说，歧义会随着覆盖范围的扩大而增加。

尽管存在这些问题，为几种语言开发基于规则的文法的一些大的合作项目已取得了 积极的和令人印象深刻的结果。例如：词汇功能语法(Lexical Functional Grammar, LFG)Pargram 项目、中心词驱动短语结构文法 (Head-Driven Phrase Structure Grammar,HPSG)LinGO 矩阵框架、邻接着文法XTAG 的词汇化树项目。

**8.6** **文法开发**

解析器根据短语结构文法在句子上建立树。之前给出的所有例子只涉及包含少数 的产生式的玩具文法。如果通过扩大这种方法的规模来处理现实的语言语料库会

发生什么?在本节中，我们将学习如何访问树库，及开发覆盖广泛文法所具有的 挑战。

**树库和文法**

corpus 模块定义了树库语料的阅读器，其中包含了宾州树库语料10%的样本。

oeidm\_a\_01.mrg')[0]

>>>print t

(S

(NP-SBJ

(NP(NNP Pierre) (NNPVinken))

(,,)

(ADJP(NP(CD 61) (NNS years)) (JJ old))

(,,))

(VP

(MD will)

(VP

(VB join)

(NP(DT the)(NN board))

(PP-CLR

(IN as)

(NP(DT a)(JJ nonexecutive)(NN director)))

(NP-TMP(NNP Nov.) (CD 29))))

( · ·))

我们可以利用这些数据开发文法。例如：例8-4中的程序使用简单的过滤器找出带 句子补语的动词。假设形如VP>SV S的产生式，利用该信息能够识别那些包括 在SV 的扩张中的所指定的动词。

例8-4 搜索树库找出句子的补语

def filter(tree):

child\_nodes =[child.node for child in tree

if isinstance(child,nltk.Tree)]

return (tree.node =='VP')and('S'in child\_nodes) >>>from nltk.corpus import treebank

>>>[subtree for tree in treebank.parsed\_sents()

for subtree in tree.subtrees(filter)]

[Tree('VP',[Tree('VBN',['named']),Tree('S',[Tree('NP-SBJ',…)], …)),.]

**PP** 附着语料库， **nltk.corpus.ppattach,** 是另一个有关特别动词配价的信息源。在这 里，我们演示一种能挖掘此语料库的技术。它找出具有固定介词和名词的介词短语

**对，其中介词短语附着到** **VP 还是** **NP 由选择的动词来决定。**

>>>entries =nltk.corpus.ppattach.attachments('training')

>>>table =nltk.defaultdict(lambda:nltk.defaultdict(set))

>>>for ry in=un1 +'-'+entry.prep+'-'+entry.noun2

table[key][entry.attachment].add(entry.verb)

>>>for ey l(eled[(1:

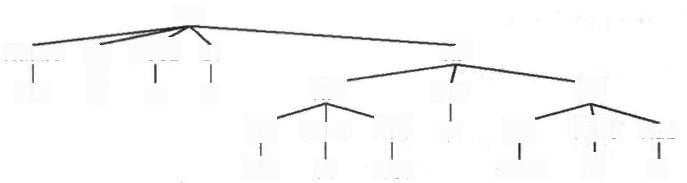
print key,'N:',sorted(table[key]['N']),'V:',sorted(table[key]['V'])

在这个程序的输出行中我们发现 offer-from-group N:[rejected]]V:[received]], 表示 received 期望单独的 PP 补充到 VP 而 rejected 却不是。和以前一样，我们可以使用 此信息来构建文法。

NLTK 语料库包含了来自 PEO8 跨框架和跨领域解析器评估共享任务的数据。为了 比较不同的解析器，准备一个更大的文法集合，可以通过下载 large\_grammars 包获 得 ( 如 ：python -m nltk.downloader large\_grammars)。

NLTK 语料库也包含了中央研究院树库语料，其中包括现代汉语中央研究院平衡 语料库中的10000 句已分析的句子。这样我们可以加载并显示这个语料库中的一 棵树。

>>>nltk.corpus.sinica\_treebank.parsed\_sents()[3450].draw()



VP

DaVC32 Di

带 了

DM

四张

NP

Caa

Nab 和 DM

牛皮 十二头

NP

VH16

肥

NF

VH11

熟

Ndabe

同时

Nab

牛

一 就

**有害的歧义**

随着文法覆盖范围的增加和输入句子长度的增长，分析树的数量也迅速增长。事实 上，它以无法想象的速度增长。

利用一个简单的例子来探讨这个问题。词 fish 既是名词又是动词。我们可以造这样

的句子： fish fish fish,意思是 fish like to fish for other fish。(也可以尝试police。) 下面是 “fish” 句子的玩具文法。

>>>grammar =nltk.parse\_cfg("""

S ->NP V NP

..NP ->NP Sbar

.Sbar ->NP y

..NP ->'fish

V->'fish!

.. ”“”)

下面，尝试分析这个较长的句子： fish fish fish fish fish, 其中一个意思是： “fish that

**other fish fish are in the habit of fishing fish themselves.” 使用在本章前面介绍过的**

NLTK 图表解析器。这句话有两种读法。

>>>tokens =["fish"]\*5

>>>cp =nltk.ChartParser(grammar)

>>>for tree in cp.nbest\_parse(tokens):

(S·(NP( iSbar(NP fish)(V fish)))(V fish)(NP fish))

(S(NP fish) (V fish) (NP(NP fish) (Sbar(NP fish)(V fish))))

当句子长度增加到(3,5,7,…),我们得到的分析树数量是：1;2;5;14;42;132; 429;1,430;4,862;16,796;58,786;208,012;…(这是 **Catalan 数**，我们在第4章的练 习中见过。)最后一个是句子长度为23的分析树的数目，这是宾州树库 WSJ 部分 句子的平均长度。对于长度为50的句子有超过10¹²的解析，这只是 Piglet 句子长 度的一半(见8.1 节),可以毫不费力地处理这些子句。实际的自然语言处理系统 不可能为一个句子构建数以百万计的树，而是根据上下文选择一个合适的。

请注意，这个问题不只是存在于我们选择的例子中。Church&Patil(1982) 指出像 (15)这样的句子，中PP 附着的句法歧义也是按 Catalan数的比例增长的。

(15) Put the block in the box on the table.

这么多的结构歧义；那么词汇歧义又怎样呢?只要我们试图建立一个广泛覆盖的文 法，我们就不得不让词汇条目对其词性高度含糊。在玩具文法中，a 是限定词， dog 是名词，而runs 是动词。然而，在覆盖广泛的文法中， a 是名词(如： part a),dog 是动词(意思是密切跟踪),而 runs 却是名词(如： ski runs)。事实上，所有的词 都可以作为名字被提及，例如： the verb'ate'is spelled with three letters; 在讲话中，

我们不需要使用引号。此外，大多数名词都可以动词化。因此，覆盖广泛的文法解 析器将对歧义不堪重负。即使是完全的胡言乱语往往也能读出来的，例如：the a are of I。正如 Abney(1996) 指出的，这不是词“沙拉”而是一个符号语法的名词短 语，其中 are 是名词，意思是一公顷(或100平方米)的百分之一，而 a 和 I 是名 词指定坐标，如图8-9所示。

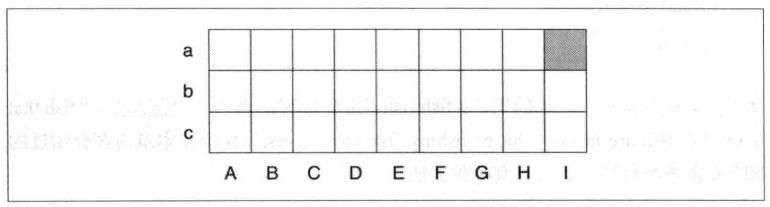


图8-9 The a are ofl:27个围场的示意图，每一块是一个 are,并用坐标确定；左上角的单元

格是a are of column A(after Abney)

即使这句话是不可能的，但它仍是符合语法的，广泛覆盖的解析器能够为它建 造分析树。同样，看上去没有歧义的句子，例如： John saw Mary, 结果却出现 我们没有想到的其他读法(正如Abney 解释的)。这种歧义是不可避免的，在分 析看似平淡无奇的句子时能导致可怕的低效率。概率分析提供了解决这些问题 的方法，它使我们能够以来自语料库的证据为基础，对歧义句的解析进行排名。

**加权文法**

正如我们刚才看到的，处理歧义是开发覆盖广泛的解析器的主要任务。图表解析器 提高了计算同一个句子的多个分析的效率，但它们仍然会因可能的分析数量过多而 不堪重负。加权文法和概率分析算法为这些问题提供了有效的解决方案。

在介绍这些之前，我们需要理解为什么符合语法的概念可能是具有梯度的。思考动 词 give。这个动词既需要一个直接宾语(被给予的东西)也需要一个间接宾语(接 收者)。这些补语可以按任何顺序出现，如(16)所示。在(16)a 中的“介词格” 中，直接宾语先出现，然后包含间接宾语的介词短语再出现。

(16)a.Kim gave a bone to the dog.

b.Kim gave the dog a bone.

在(16)b 中的“双宾语”中，间接宾语先出现，然后是直接宾语。在这种情况 下，两种顺序都是可以接受的。然而，如果间接宾语是代词，则更倾向于双宾 语结构。

(17)a.Kim gives the heebie-jeebies to me (prepositional dative).

b.Kim gives me the heebie-jeebies (double object).

使用宾州树库样本检查所有包含 give 介词格和双宾语结构的实例，如例8-5所示。

例8-5 宾州树库样本中 give 和 gave 的用法

def give(t):

return t.node =='VP'and len(t)>2 and t[1].node =='NP'\ and(t[2].node =='PP-DTV'or t[2].node =='NP')\

and('give'in t[0].leaves()or 'gave'in t[0].leaves())

def sent(t):

return''.join(token for token in t.leaves()if token[0]not in'\*-0') def print\_node(t,width):

output ="%s %s:%s /8s:8s"8\

(sent(t[0]),t[1].node,sent(t[1]),t[2].node,sent(t[2]))

if tpuwidth]+"..."

print

>>>for tree

for t

·

..

output

in nltk.corpus.treebank.parsed\_sents():

in tree.subtrees(give):

· print\_node(t,72)

gave NP:the chefs /NP:a standing ovation

giveNP:advertisers /NP:discounts for maintaining or increasing ad sp.. give NP:it /PP-DTV:to the politicians

gave NP:them /NP:similar help

give NP:them /NP:

give NP:only French history questions /PP-DTV:to students in a Europe. give NP:federal judges /NP:a raise

give NP:consumers /NP:the straight scoop on the U.s.waste crisis

gave NP:Mitsui /NP:access to a high-tech medical product

give NP:Mitsubishi /NP:a window on the U.S.glass industry

give NP:much thought /PP-DTV:to the rates she was receiving ,nor to.

giveNP:your Foster Savings Institution /NP:the gift of hope and free.·.. give NP:market operators /NP:the authority to suspend trading in futu... gave NP:quick approval /PP-DTV:to $3.18 billion in supplemental appr... give NP:the Transportation Department /NP:up to 50 days to review any.

give NP:the president /NP:such power

give NP:me /NP:the heebie-jeebies

give NP:holders /NP:the right,but not the obligation,to buy a cal..

gave NP:Mr.Thomas /NP:only a`qualified ''rating ,rather than“. give NP:the president /NP:line-item veto power

观察发现更倾向于最短的补语先出现。然而，这并没有解释，例如： give NP:federal judges/NP:a raise 的形式，其中生命度起了重要作用。事实上，根据 Bresnan &Hay (2008)的调查，其中存在大量的影响因素。这些倾向可以用加权文法来表示。

**概率上下文无关文法** (probabilistic context-free grammar,PCFG) 是一种上下文无 关文法，它把每一个概率与其产生式关联起来。它会产生与相应的上下文无关文法 相同的文本解析，并给每个解析分配一个概率。由PCFG 产生的解析概率仅仅是其 所用到的产生式的概率的乘积。

定义 PCFG 最简单的方法是从由加权产生式序列组成的特殊格式的字符串加载它， 其中权值出现在括号里，如例8-6所示。

例8-6 定义一个概率上下文无关文法 (PCFG )

|  |  |  |
| --- | --- | --- |
| grammar | =nltk.parse\_pcfg(""" |  |
| S | -> NP VP | [1.0] |
| VP | ->TV NP | [0.4] |
| VP | ->IV | [0.3] |
| VP | ->DatV NP NP | [0.3] |
| TV | -> 'saw' | [1.0] |
| IV | ->'ate' | [1.0] |
| DatV | ->'gave' | [1.0] |
| NP | ->'telescopes' | [0.8] |
| NP | ->'Jack' | [0.2] |

""")

ar gwrairproductions (start state =S)

S ->NP VP [1.0]

VP ->TV NP[0.4]

VP ->IV [0.3]

VP ->DatV NP NP [0.3]

'ate' **[**0]0]

DatV ->'gave'[1.0]

NP ->'telescopes' [0.8]

NP ->'Jack'[0.2]

有时可以将多个产生式组合成一行，这样会方便，如： VP->TV NP [0.4]|IV [0.3]| DatV NP NP [0.3]。为了确保由文法生成的树能形成概率分布，PCFG 文法强加了约

束：所有产生式给定的左侧的概率之和必须为1。例8-6中的文法符合这个约束： 对 S 只有一个产生式，它的概率是1.0;对于 VP,0.4+0.3+0.3=1.0; 对于 NP, 0.8+0.2=1.0。parse()返回的分析树中包含了概率。

>>>viterbi\_parser =nltk.ViterbiParser(grammar)

>>>print viterbi\_parser.parse(['Jack','saw','telescopes']) (S(NP Jack) (VP(TV saw) (NP telescopes))) (p=0.064)

现在，分析树被分配了概率，给定的句子拥有数量庞大的可能解析也不再是个问题 了。解析器将负责寻找最有可能的解析。

**8.7** **小结**

· 句子的内部组织结构用树来表示。组成结构的显著特点是：递归、中心词、补 语和修饰语。

● 文法是可能句子的集合的紧凑型特性；我们可以说， 一棵树是符合语法规则的， 或文法可以授权一棵树。

● 文法是一种用于描述给定短语是否可以被分配特定成分或依存结构的形式化 模型。

· 给定一组句法类别，上下文无关文法使用生产式表示某类型 A的短语是如何被 分析成较小序列α₁…an的。

· 依存文法使用产生式指定给定中心词的依赖是什么。

● 当句子有一个以上的文法分析时，就会产生句法歧义(如介词短语附着歧义)。

● 解析器是寻找一个或多个与符合语法规则句子相对应树的程序。

· 下降递归解析器是一个简单的自上而下解析器，利用文法产生式，递归可扩 展开始符号(通常是 S), 并尝试匹配输入的句子。该解析器并不能处理左递 归产生式(如： NP->NP PP)。它盲目扩充类别而不检查它们是否与输入字 符串兼容，导致此方式效率低下，而且会重复扩充同样的非终结符，然后丢 弃结果。

● 移位-规约解析器是一个简单的自下而上的解析器，它把输入移到堆栈中，并 尝试匹配堆栈顶部的项目和文法产生式右边的部分。该解析器不能保证能为输 入找到有效的解析，即使确实存在。建立子结构但不检查它是否与全部文法 一致。

**8.8** **深入阅读**

本章的额外材料发布在 <http://www.nltk.org/>, 包括网上提供的免费资源的链接。更 多使用NLTK 分析的例子，请看<http://www.nltk.org/howto> 上的 Parsing HOWTO。

有许多关于句法的入门书籍：(OGrady et al.,2004)是一本语言学概论，而(Radford, 1988)以读者更容易接受的方式介绍了转换文法，我们推荐阅读其中无限制依赖结 构的转换文法。在形式语言学中最广泛使用的术语是**生成文法，**但它与 generation 并没有关系 (Chomsky,19 65)。

(Burton-Roberts,1997)是一本面向实践的关于如何分析英语成分的教科书，包含广泛的 例子和练习。(Huddleston&Pullum,2002) 介绍了有关英语句法现象的最新的综合分析。

(Jurafsky &Martin,2008)的第12章讲述了英语的形式文法；13.1～13.3节讲述了简 单的分析算法和歧义处理技术；第14章讲述了统计分析；第16章讲述了乔姆斯基层 次和自然语言形式的复杂性。(Levin,1993) 根据英语动词的句法属性，将其划分成 更细的类。

下面是几个正在研究的大规模基于规则的文法项目，如： LFG Pargram 项目

(<http://www2.parc.com/ist/groups/nlt/pargram/>),HPSG LinGO 矩 阵 框 架 (htp://www.

delph-in.net/matrix/) 及 XTAG 项 目(<htp://www.cis.upenn.edu/xtag/>)。

**8.9** **练习**

1.O 你能想出符合语法的但之前提过的句子吗?(与伙伴轮流进行。)从中你学习 到哪些有关人类语言的知识?

2.O 回想一下 Strunk 和 White 不许在句子开头使用however 表示 “although” 意思 的禁令。在网上搜索开头使用 however 的句子。使用范围有多广泛?

3.○思考句子： Kim arrived or Dana left and everyone cheered。用括号表示and 和 or 的相对范围。对应这两种解释构建相应的树结构。

4.OTree 类实现了各种其他有用的方法。请查阅 Tree 帮助文档的更多细节(如： 导入Tree 类，然后输入help(Tree))。

5.O 在本练习中，手动构造一些分析树。

a. 编写代码构建两棵树，分别对应短语old men and women的不同读法。

b. 本章中用括起标签的形式表示任一树编码，使用 nltk.Tree()检查它是否符 合语法。使用 draw()显示树。

c. 如 (a) 中，为 The woman saw a man last Thursday 画一棵树。

6.O 编写递归函数，遍历树，再返回树的深度，只有一个节点的树的深度应为0。 (提示：子树的深度是其子女的最大深度加1。)

7.O 分析A.A.Milne 关于Piglet 的句子，为其中的所有句子标注下划线，然后用 S替换它们(如：第一句话变为 SwhenS) 。 为这种“压缩”的句子画出树形结构。 用于建立这样一个长句的主要句法结构是什么?

8.O 在下降递归解析器中，选择“编辑”菜单上的“编辑文本”改变实验句子。

9.Ogrammarl 中的文法(见例8-1)可以用来描述词长超过20 的句子吗?

10.O 使用图形化图表解析器接口，尝试不同的规则调用策略。想出可以使用图形 界面手动执行的策略。描述该步骤，并报告其在效率方面提高的部分(例如：用结 果图表示大小)。这些改进取决于文法结构吗?你觉得利用更优的规则调用策略能 显著提升性能吗?

11.O 对于一个你已经见过的或一个你自己设计的 CFG, 利用笔和纸，手动跟踪 下降递归解析器和移位-规约解析器的执行情况。

12.O 我们知道图表解析器只增加边但从来不从图表中删除边，为什么?

13.O 思考词序列： Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo。

正如在 <http://en.wikipedia.org/wiki/Buffalo>\_buffalo\_Buffalo\_buffalo\_buffalo\_buffalo\_ Buffalo\_buffalo 上所解释的，这是一个语法正确的句子。思考此维基百科页面上表 示的树形图，并编写一个合适的文法。正常情况下是小写，模拟听到这句话时听者 会遇到的问题。你找到这句话的其他解析吗?当句子变长时分析树的数量如何增 长 ? (类似这些句子的更多的例子可以在 <http://en.wikipedia.org/wiki/List>\_of\_ homophonous phrases 上找到。)

14.①通过选择“编辑”菜单上的“编辑文法”可修改下降递归解析器的演示程序。 改变第一次扩充产生式，即NP->DetN PP修改为NP->NPPP 。 使用 Step 按钮， 尝试建立一个分析树。观察会发生什么?

15.①使用产生式扩展 grammar2 中的文法，将介词扩展为不及物的、及物的和需 要PP 补语的。基于这些产生式，使用前面练习中的方法为句子Lee ran away home 画一棵树。

16.①挑选一些常用动词，完成以下任务。

a. 编写一段程序在 PP 附着语料库 nltk.corpus.ppattach 找到这些动词。即同一动 词展示两种不同的附属，其中第一个是名词，或者第二个是名词，或者介词

保持不变(正如我们在第8.2节句法歧义中讨论过的)。

b. 设计 CFG文法产生式涵盖其中一些情况。

17.①编写一段程序，比较自上而下的图表解析器与下降递归解析器的效率(见 8.4节)。使用相同的文法和输入句子。使用timeit 模块比较它们的性能(见4.7节 的例子)。

18.①使用相同的文法和3个语法测试句子，比较自上而下、自下而上和左角落解 析器的性能。使用timeit 记录每个解析器在同一句子上花费的时间。编写函数，在 这3句话上分别运行这3个解析器，输出3×3格的时间，以及行和列的总计。讨论 你的结果。

19.①阅读 “garden path” 句子。解析器的计算工作与人类处理这些句子的困难性 有什么关系? (见 <http://en.wikipedia.org/wiki/Garden>\_path\_sentence)

20.①为了在窗口中比较多棵树，可以使用 draw\_trees() 方法。定义一些树，尝试 一下内容。

>>>from nltk.draw.tree import draw\_trees

>>>draw\_trees(treel,tree2, tree3)

21.①使用树中的位置，列出宾州树库前100个句子的主语。为了使结果更容易查 看，规定从高度最高为2的子树中提取主语。

22.O 查看 PP 附着语料库，尝试提出一些影响 PP 附着的因素。

23.①在8-2节中，我们说过简单地用术语n-grams 不能描述所有语言学规律。思考下 面的句子，尤其是短语in his tum的位置。这反应了基于n-grams 方法的问题吗?

What was more,the in his turn somewhat youngish Nikolay Parfenovich also tumed out to be the only person in the entire world to acquire a sincere liking to our“dis-criminated-against”public procurator.(Dostoevsky:The Brothers Karamazov)

24.①编写一个递归函数生成一棵嵌套了括号形式的树，去掉叶节点并在子树之后 显示非终结符。于是8.6 节中的关于 Pierre Vinken 的例子会产生：[[[NNP NNP]NP ,[ADJP [CD NNS]NP JJ]ADJP ,JNP-SBJ MD [VB [DT NN]NP [IN [DT JJ NNJNP]PP-CLR[NNP CD]NP-TMP]VP.]S。连续的类别应用空格分隔。

25.①从古登堡工程中下载一些电子书。编写一段程序扫描出这些文本中任何极长 的句子。你能找到的最长的句子是什么?这个长句子的语法结构是什么?

26.①修改函数 init\_wfst) 和 complete\_wfst(), 使 WFST 中每个单元的内容是一组 非终端符而不是一个单独的非终结符。

27.①思考例8-3 中的算法。你能解释为什么上下文无关文法是与n³ 成正比的，其中n

是输入句子的长度。

28.○处理宾州树库语料库样本 nltk.corpustreebank 中的每棵树，在 Tree.productionsO 下提取产生式。丢弃只出现一次的产生式。具有相同左侧和类似右侧的产生式可以被折 叠，产生一个等价的却更紧凑的规则集。编写代码输出此紧凑的文法。

29.●英语中一种定义句子S 主语的常见方法是以S 的“孩子”和VP 的“兄弟”

作为的名词短语。编写一个函数，以一句话作为树的参数，返回与句子主语对应的 子树。如果传递给这个函数的树的根节点不是S 或它缺少一个主语，应该怎么做?

30.●编写一个函数，以一个文法(如例8-1定义的文法)为参数，返回由这个文法随 机产生的句子。(使用grammarstart() 找出文法的开始符号； grammar productions(lhs)得到 具有指定左侧内容的文法产生式的链表； production.ths()得到一个产生式的右侧。)

31.●使用回溯实现移位-规约解析器，找出一个句子所有可能的解析，它可以被 称为“递归上升解析器”。咨询维基百科关于回溯的条目 <http://en.wikipedia.org> /wiki/Backtracking.

32.●正如我们在第7章中所看到的，可以将块表示成它们的块标签。当对包含 gave 的句子进行相同的操作时，我们发现如以下模式。

gave NP

gave up NP in NP

gave NP up

gave NP NP

gave NP to NP

a. 使用这种方法学习感兴趣的动词互补模式，编写合适的文法产生式。(此 任务有时也被称为**词汇获取**。)

b. 识别一些英语动词的近-同义词，如 dumped/filled/loaded, 这是来自第9章 中(64)的例子。使用分块方法研究这些动词的互补模式。创建一个文法 来包含这些情况。这些动词能够自由地取代对方，或者有什么限制吗?讨 论你的结果。

33.●开发一种基于下降递归解析器的左角落解析器，从ParseI 继承。

34. ● 扩展 NLTK 中的移位-规约解析器使其包含回溯，这样就能保证找到所有存 在的解析(也就是说，它是**完整的**)。

35 . ● 修改函数 init\_wfst() 和 complete\_wfst), 当非终结符添加到 WFST 中的单元 时，它包括了它所派生的单元的记录。实现一个函数，将一棵分析树的 WFST 转 换成这种形式。



**第9章**

**建立基于特征的文法**

自然语言具有广泛的文法结构，用第8章中所描述的简单方法很难处理如此广泛 的文法结构。为了获得更大的灵活性，可改变对待文法类别如 S 、NP 和 V 的方 式。我们将这些原子标签分解为类似字典的结构，以便可以提取一系列的值作为 特征。

本章的目的是要回答下列问题。

(1)怎样用特征扩展无关上下文文法的框架，以获得对文法类别和产生式的更细粒

度的控制?

(2)特征结构的主要形式化属性是什么,如何使用它们来计算?

(3)我们现在用基于特征的文法能获得哪些语言模式和文法结构?

在此过程中，我们将介绍更多的英语句法主题，包括：约定、子类别和无限制依赖 成分等现象。

**9.1** **文法特征**

在第6章，我们描述了如何通过检测文本的特征建立分类器。其中的特征可能非常 简单，例如提取一个单词的最后一个字母；或者更复杂的，如分类器自己预测的词 性标签。在本章中，我们将探讨建立在基于规则上的文法中特征的作用。对比特征 提取，记录已经自动检测到的特征，我们现在要介绍词和短语的特征。我们以下面

这个简单的例子为例，介绍使用字典存储特征和它们的值。

>>>kim={'CAT': 'NP', 'ORTH': 'Kim','REF': 'k'}

>>>chase =('CAT':'V','ORTH':'chased','REL':'chase'}

对象 kim 和 chase 有一些共同的特征， CAT (文法类别)和 ORTH(正字法，即拼 写)。此外，每一个都具有面向语义的特征： kim[REF] 表示 kim 的指示物，而 chase[REL] 表示chase 表示的关系。在基于规则的文法上下文中，这样的特征和特 征值对被称为特征结构，我们将在下面的内容中看到它们的替代符号。

特征结构包含各种有关文法实体的信息。这些信息不需要详尽无遗的介绍，而是要 进一步添加属性。例如：需要知道一个动词所扮演的“语义角色”。对于 chase, 主 语扮演 “agent (施事)”的角色，而宾语扮演 “patient (受事)”角色。添加这些信 息时，可使用'sbj' (主语)和'obj” (宾语)作为占位符，然后一旦动词和文法参数 结合时，它会被填充。

>>>chase['AGT']='sbj'

>>>chase['PAT']='obj'

下面处理句子 Kim chased Lee, 要“绑定”动词的施事角色和主语，受事角色和宾 语，可以通过链接到相关的NP的 REF特征。在下面的例子中，简单的假设：在动 词左侧和右侧的NP 分别是主语和宾语。我们还在例子结尾为Lee 添加了一个特征 结构。

>>>sent ="Kim chased Lee"

>>>tokens =sent.split()

>>>lee ={'CAT':'NP','ORTH':'Lee','REF':'l'}

>>>def lex2fs(word):

for fs in [kim,lee,chase]:

…· if fs['ORTH']==word:

>

lex2fs(tokens [2])

>>>verb['AGT']=subj['REF']#agent

of 'chase'is Kim

>>>verb['PAT']=obj['REF'] #patient of 'chase'is Lee

>>>for k in ['ORTH','REL','AGT','PAT']:#check featstruct of 'chase'

· print "%-5s =>8s"%(k,verb[k])

ORTH =>chased

REL =>chase

AGT =>k

PAT =>1

同样的方法可以应用于不同的动词 ay 和surprist  -尽管在这种情况下，主语将扮 演 “source(源事)”(SRC) 的角色，宾语扮演“experiencer(体验者)”(EXP) 的角色。

>>>surprise

={'CAT':'V','ORTH':'surprised','REL':'surprise', 'SRC':'sbj','EXP':'obj'}

特征结构是非常强大的，但我们操纵它们的方式是极其特别的。本章接下来的任务 是，显示无关上下文文法和分析如何能扩展到合适的特征结构，使我们可以以一种 更通用的和有原则的方式建立像这样的分析。以查看句法协议的现象作为开始；展 示如何使用特征表示协议约束，并在简单的文法中说明它们的用法。

由于特征结构是表示任何形式信息的通用的数据结构，我们将从更形式化的视点简 要地介绍它们，并演示NLTK 提供的特征结构的支持。在本章的最后一部分，我们 将介绍，特征的额外表现力开辟了用于描述语言结构复杂性的可能性。

**句法协议**

下面的例子所示为词序列对，其中第一个是符合语法的而第二个不是。(我们在词 序列的开头用星号表示它是不符合语法的。)

(1)a.this dog

b.\*these dog

(2)a.these dogs

b.\*this dogs

在英语中，名词通常被标记为单数或复数。范例中的形式也各不相同： this(单数), these (复数)。例(1)和例(2)表示在名词短语中使用指示词和名词是有限制的。 要么两个都是单数要么都是复数。主语和谓词间也存在类似的约束。

(3)a.the dog runs

b.\*the dog run

(4)a.the dogs run

b.\*the dogs runs

这里我们可以看到，动词的形态属性同主语名词短语的句法属性一起变化。该过程 被称为协议(**agreement)**。如果我们进一步研究英语动词协议，将发现动词的现在

'dog'

时态通常有两种屈折形式： 一为第三人称单数，另一个为人称和数量的所有其他组 合，如表9-1 所示。

**表9-1** **英语规则动词的协议范式**

|  |  |  |
| --- | --- | --- |
|  | **单** **数** | **复** **数** |
| 第一人称 | Irun | we run |
| 第二人称 | you run | you run |
| 第三人称 | he/she/it runs | they run |

进一步明确形态学特性的作用，如例(5)和例(6)所示。这些例子表明动词与它 的主语在人称和数量上保持一致。(我们用3作为第三人称的缩写， SG 表示单数， PL表示复数。)

(5) the dog

dog.3.SG

(6) the dog-s

dog.3.PL

run-s

run-3.SG

run

run-3.PL

当我们在一个无关上下文文法中编码时，这些协议约束会发生什么。我们以例(7) 中简单的 CFG 开始。

(7) S ->

NP ->

VP ->

Det -> N -> V ->

NP VP

Det N

V

'this'

''runs'

通过文法(7)能够产生句子this dog runs。然而，我们真正想要做的也是产生 these dogs run, 且同时阻止不必要的序列，如\*this dogs run 和\*these dog runs。最简单的 方法是为文法添加新的非终结符和产生式。

(8) S ->NP\_SG VP\_SG

S ->NP\_PL VP\_PL

NP\_SG ->Det\_SGN\_SG

NP\_PL->Det\_PL N\_PL

VP SG ->V\_SG

VP\_PL ->V\_PL

Det\_SG ->'this'

Det\_PL ->'these'

N\_SG ->'dog'

N\_PL->'dogs'

V\_SG ->'runs'

V\_PL ->'run'

除了唯一的扩展S 产生式，我们现在有两个产生式， 一个是包括单数主语 NP 和 VP 的句子，另一个是包含复数主语NP 和 VP 的句子。事实上，例(7)中的每个 产生式在例(8)中都有两个对应项。在小规模的文法中这算不上是什么真正的问 题。然而，在一个更大的涵盖了一定量英语成分子集的文法中，很难将文法规模增 加一倍。假设现在用同样的方法处理第一、第二和第三人称，还有单数和复数。这 将导致原有的文法乘以因子6,一定要避免这种情况。我们可以做得比这更好吗? 在下一节，我们将展示数量和人称协议的一致性，并且不需要以“爆炸式”地增加 文法和人称协议的数量为代价。

**使用属性和约束**

我们说过非正式的语言类别具有属性，例如：名词具有复数的属性。让我们把这个

问题弄得更明确，如(9)所示。

(9) N[NUM=pl]

在(9)中，我们介绍了一些新的符号，它的意思是类别N 有一个(文法)特征叫做 NUM(“number 数字”的简写),此特征的值是 pl(“plural 复数”的简写)。我们 可以添加类似的注解给其他类别，并在词汇条目中使用它们。

(10)Det[NUM=sg]->'this'

Det[NUM=pl] ->'these'

]->'dogs'

V[NUM=sg]->'runs'

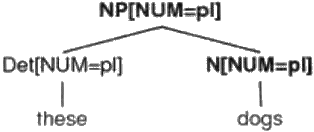
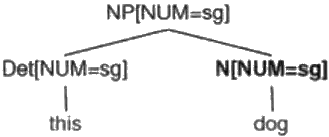
V[NUM=pl]->'run'

这确实有帮助吗?迄今为止，它看起来就像(8)中指定内容的一个稍微详细的替 代方式。当允许使用特征值变量表示限制时，事情变得更复杂。

(11)S ->NP[NUM=?n]VP[NUM=?n]

NP[NUM=?n]->Det[NUM=?n]N[NUM=?n]

VP[NUM=?n]->V[NUM=?n]



我们使用?n 作 为NUM 值上的变量，它可以在给定的产生式中被实例化为sg 或 pl。 我们可以读取第一条产生式——无论 NP 为特征 NUM 取什么值， VP 必须取同样 的值。

(12)

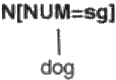
a. **Det[NUM=sg**

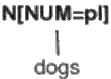
this

b. et[NUM=pl

these

(13)

a. 

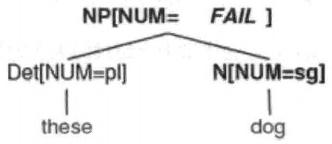
b. 

要想了解这些特征限制如何工作，需要思考如何建立一棵树。词汇产生式将承认下 面的树(树的深度为1)。

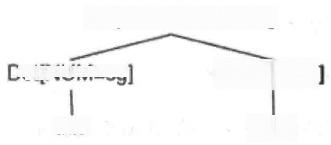
(14)

a.

b.



(15)

**a. NP[NUM= FAIL]**

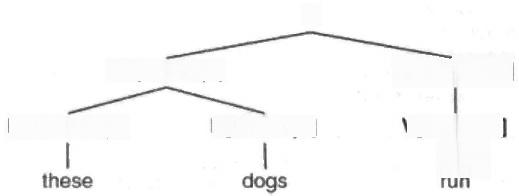
Det[NUM=sg] **N[NUM=pl**

this dogs

b.

现在NP[NUM=?n]>Det[NUM=?n]N[NUM= ?n]表示无论N 和 Det的 NUM的值是 什么,且它们都是相同的。因此，这个产生式允许(12)a和(13)a 组合成NP, 如(14)a 所示，它也允许(12)b 和(13)b 组合在一起，如(14)b 中所示。相 比之下，(15)a 和(15)b 被禁止组合在一起，因为它们的子树根的NUM 值不同。 这种值的不相容可以用顶端节点的值 FAIL 非正式地表示。

产生式 VP[NUM=?n]> V[NUM=?n]表示核心动词的NUM 必须与它的VP 根源的 NUM 值相同。结合扩展 S 的产生式，我们得出结论：如果主语核心名词的 NUM 值是pl, 那么 VP核心动词的 NUM值也是pl。

(16) S

NP[NUM=pl]

Det[NUM=pl] N[NUM=pl]

VP[NUM=pl]

V[NUM=pl]

!

文法(10)演示了限定词的词汇产生式，如 this 和 these, 它们分别需要单数 或复数的核心名词。然而，英语中其他限定词不限定与它们结合的名词的文法 数量。描述方法是在文法中添加两个词汇条目，单数和复数版本的限定词如 the 一样。

Det [NUM=sg]->'thel'some'l'several'

Det[NUM=pl]->'the'l'some'l'several'

然而， 一种更好的解决方法是不指定NUM的值，让它保存相关的词汇的数量值。 为NUM 设定一个变量是解决方法之一：

Det[NUM=?n]->'the' l 'some'l'several'

但事实上可以更简单些，在这样的产生式中不给 NUM任何指定。我们只需要在这 个限制制约同一个产生式的其他地方的值时，明确地输入变量的值。

例9-1中的文法演示到目前为止我们已经在本章中介绍过的大多数想法，再加上几 个新的想法。

**例9-1** **基于特征的文法的例子**

>>>nltk.data.show\_cfg('grammars/book\_grammars/feat0.fcfg')

start S

####################

#Grammar Productions

####################

#S expansion productions

S ->NP[NUM=?n]VP[NUM=?n]

#NP expansion productions

NP[NUM=?n]->PropN[NUM=?n]

NP[NUM=?n]->Det[NUM=?n]N[NUM=?n]

PV[NM]productions

VP[TENSE=?t,NUM=?n]->IV[TENSE=?t,NUM=?n]

VP[TENSE=?t,NUM=?n]->TV[TENSE=?t,NUM=?n]NP

# ###################

#Lexical Productions

t排]-'iery'

Det[NUM=pl]->'these'l'all'

Det ->'the'|'some'|'several'

N-g>]y' l'car'l'child'

N[NUM=pl]->'dogs'l'girls'l'cars'l'children'

IV[TENSE=pres, NUM=sg]->'disappears'l'walks'

**V**[=rEes=,NpUr]-pl]->'disappealk'

TV[TENSE=pres,NUM=pl]->'see'l'like'

IV[TENSE=past]->'disappeared'l TV[TENSE=past]->'saw'l'liked'

'walked'

注意一个句法类别可以有多个特征，例如：V[TENSE=pres,NUM=pl]。在一般情况

下，我们更倾向于任意添加特征。

关于例9-1 的最后的细节是语句%start S。这个“指令”表示解析器以 S 作为文法

的开始符号。

一般情况下，当我们尝试开发一个小型文法时，把产生式放在文件中可以更方便地 进行编辑、测试和修改。我们将例9-1以NLTK 的数据格式保存为文件 feat0.fcfg。 你可以使用nltk.data.load()制作你自己的副本以进行进一步实验。

NLTK 中使用 Earley 图表分析器分析基于特征的文法(更多信息见第9.5 节),例 9-2演示了该实施过程。输入分词之后，导入 load\_parser 函数，以文法文件名为 输入，返回图表分析器cp²。 调用分析器的nbest\_parse()方法返回分析树的 trees 链 表。如果文法无法分析输入， trees 将为空，否则将包含一个或多个分析树，这取决 于输入是否有句法歧义。

例9-2 跟踪基于特征的图表分析器

>>>tokens ='Kim likes children'.split()

>>>from nltk import load\_parser①

>>>cp =load\_parser('grammars/book\_grammars/feat0.fcfg',trace=2)②

>>>trees =cp.nbest\_parse(tokens)

l.Kim .like.chil.I

l[----] .I PropN[NUM='sg']->'Kim'\*

l [----] .|NP[NUM='sg']->PropN[NUM='sg']\*

l[----> .IS[]->NP[NUM=?n]\*VP[NUM=?n]{?n:'sg'}

1. [----] .I TV[NUM='sg',TENSE='pres']->'likes'\*

1. [----> .l VP[NUM=?n,TENSE=?t]->TV[NUM=?n,TENSE=?t]\*NP[]

[ {N:'sg['N,?t:eMs'}='pl']->'children'\*

[----]|NP[NUM='pl']->N[NUM='pl']\*

l. [---->|S[]->NP[NUM=?n]\*VP[NUM=?n]{?n:'pl'}

1. [--------]|VP[NUM='sg',TENSE='pres']

->TV[NUM='sg',TENSE='pres']NP[]\*

1[=============]|S[]->NP[NUM='sg']VP[NUM='sg']\*

分析过程中的细节对于当前的目标并不重要。然而，有一个实施上的问题与我们前 面讨论的文法的大小有关。分析包含特征限制的产生式的一种可行的方法是编译出 问题中特征的所有可接受的值，使我们最终得到一个大的完全指定的如(8)中那 样的 CFG。 相比之下，前面例子中显示的分析器过程直接与给定文法的未指定的 产生式一起运作。特征值从词汇条目“向上流动”,变量值于是通过如{?n:'sg',?t:

'pres'}这样的绑定(即字典)与那些值关联起来。当分析器装配有关它正在建立的树 的节点的信息时，绑定这些变量用来实例化这些节点中的值。从而通过查找绑定中?n

和?t 的值，将未指定的VP[NUM=?n,TENSE=?t]>TV[NUM=?n,TENSE=?t]NP[] 实例化为 VP[NUM='sg',TENSE='pres]->TV[NUM='sg',TENSE='pres']NP[]。

最后，检查生成的分析树状图(在这种情况下只有一个)。

>>>for tree in trees:print tree

(S[]

(NP[NUM='sg'](PropN[NUM='sg']Kim))

(VP[NUM='sg',TENSE='pres']

'pl']children))))

**术语**

到目前为止，我们学习到像sg 和pl 这样的特征值。这些简单的值通常被称为**原子—**— 也就是，它们不能被分解成更小的部分。原子值的一种特殊情况是**布尔值，**也就是说， 值仅仅指定一个属性是真还是假。例如：我们可能要用布尔特征AUX 区分**助动词，**如： can、may、will和 do。那么产生式V[TENSE=pres,aux=+]>'can '意味着 can 接受TENSE 的值为 pres, 并且 AUX 的值为+或 true。有一个广泛采用的约定：用缩写表示布尔特 征f, 我们分别用+aux 和-aux 表示，而不用 aux=+或 aux= 。这些都是缩写，然而，分 析器就像+和-等其他原子值一样解释它们。(17)列举了一些有代表性的产生式。

(17)V[TENSE=pres,+aux]->'can'

V[TENSE=pres,+aux]->'may'

V[TENSE=pres,-aux] ->'walks'

V[TENSE=pres,-aux]->'likes'

之前介绍过分配“特征注释”给句法类别。表示整个类别的更激进的做法——也就 是，非终结符加注释——作为特征的绑定。例如： N[NUM=sg] 包含词性信息，可 以表示为POS=N 。因此，这个类别的替代符号是[POS=N,NUM=sg]。

除了原子值特征，特征可能需要本身就是特征结构的值。例如：我们可以将协议特 征组合在一起(例如：人称、数量和性别)作为一个类别的不同部分，表示为AGR 的值。这种情况中，AGR 是一个复杂值。(18)描述的结构，在格式上称为**属性值** **矩阵** (attribute value matrix,AVM)。

(18) [POS =N

[AGR =[PER =3 ]]

[NUM=pl ]]

[GND =fem ]]

在传递中，我们应该指出显示AVM的替代方 法；图9-1 展示了相关的例子。虽然特征结构 呈现的(18)中的风格不太“悦目”,但我们 将坚持采用这种格式，因为它对应着我们将 会从NLTK得到的输出。

关于表示，我们要注意特征结构，例如字典，

|  |  |  |
| --- | --- | --- |
| POS  AGR | N  PER NUM  GND | 3  *pl fem* |

**图9-** **1** **以属性值矩阵呈现一个特征结构**

对特征的顺序没有指定特别的意义。因此，(18)等价于如下形式

(19) [AGR=[NUM =pl ]]

3 ]]

(GND=fem

AA S N

当使用如 AGR 这样的特征时，我们可以重构如例9-1 这样的文法，使协议特征捆 绑在一起。 一个小型的文法演示了这个想法，如(20)所示。

(20)S->NP[AGR=?n] VP[AGR=?n]

NP[AGR=?n]->PropN[AGR=?n]

VP[TENSE=?t,AGR=?n]->Cop[TENSE=?t,AGR=?n]Adj

Cop[TENSE=pres, AGR=[NUM=sg,PER=3]]->'is'

PropN[AGR=[NUM=sg,PER=3]]->'Kim'

Adj->'happy'

**9.2** **处理特征结构**

在本节中，我们将学习如何构建特征结构，并在NLTK 中操作。我们还将讨论统一

的基本操作，能够结合两个不同的特征结构中的信息。

NLTK 中的特征结构使用构造函数 FeatStruct()声明。原子特征值可以是字符串或整数。

>>>fs1

>>>print

=nltk.FeatStruct(TENSE='past',NUM='sg')

fsl

[NUM ='sg' ]

[TENSE ='past’]

特征结构实际上只是一种字典，所以我们可以平常的方式通过索引访问它的值。可 以用我们熟悉的方式指定值给特征。

>>>fs1 =nltk.FeatStruct(PER=3,NUM='pl',GND='fem')

>>>print fs1['GND']

fem

>>>fs1['CASE']='acc'

我们还可以为特征结构定义更复杂的值，如前所述。

>>>fs2 =nltk.FeatStruct(POS='N',AGR=fs1)

>>>print fs2

[ [CASE ='acc']]

[AGR =[GND ='fem′ ]]

[NUM ='pl' ] ]

【 PER =3 ] ]

1

[POS ='N'

>>>print fs2['AGR']

[CASE ='acc']

[GND ='fem' ]

[NUM ='pl'

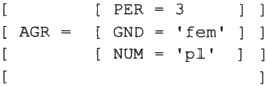
[PER =3

>>>print fs2['AGR']['PER']

3

另一种指定特征结构的方法是使用包含 feature=value 格式的特征-值对的用方括号 括起的字符串，其中值本身可能是特征结构。

>>>print nltk.Featstruct("[POS='N',AGR=[PER=3,NUM='pl',GND='fem']]")



[POS ='N'

特征结构本身并不依赖于语言对象；它们是表示知识的通用目的的结构。例如：我 们可以将一个人的信息用特征结构编码。

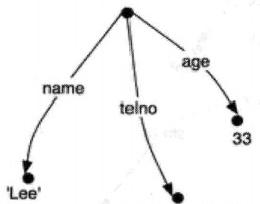
>>>print nltk.Featstruct(name='Lee',telno='0127864296',age=33)

[ age =33



在接下来的内容中，我们会使用这样的例子来探讨特征结构的标准操作。

可以将特征结构看作是图，更具体的，作为**有向无环图** (directed acyclic graph, DAG) 。(21) 相当于前面的AVM。

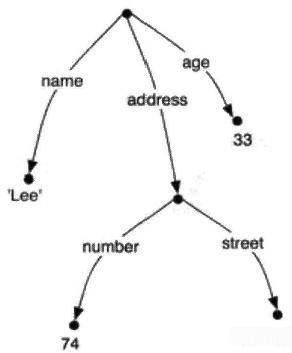


(21)

0127864296'

特征名称作为弧上的标签，特征值作为弧指向的节点的标签。

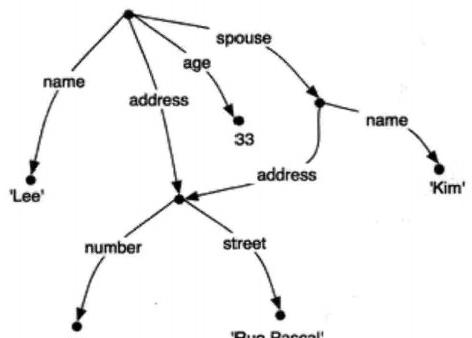
像以前一样，特征值也可以是复杂的。



(22)

rue Pascal'

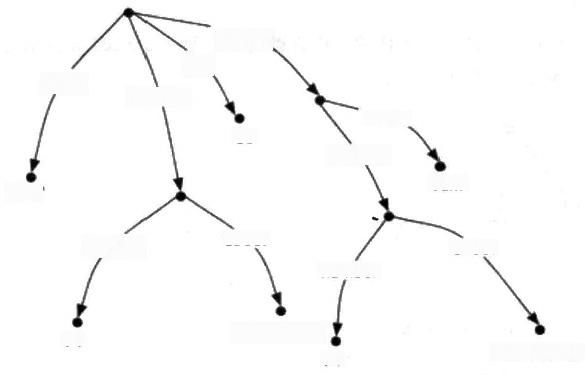
标签的元组表示路径。因此，(ADDRESS’,'STREET) 就是一个特征路径，它在(22) 中的值是标签为'rue Pascal的节点。



(23)

现在让我们思考 一 种情况： Lee 有 个 配 偶 叫Kim,Kim 的 地 址 与Lee 相同。可以如

(23)那样表示这种情况。



spouse

age

name address

name

33

address

Kim'

Lee'

number street

number

'Rue Pascal

Rue Pascal'

74

street

74

然而 ， 除了重复特征结构中的地址信息 ， 我们可以 “ 共享 ” 不同的弧之间的同

一子图 。

(24)

74

换 句 话 说，(24)中路径(ADDRESS) 的 值 与 路 径(SPOUSE’,'ADDRESS) 的值相同。 如 ( 2 4 ) 的 DAG 被称为包含结构共享或重入。当两条路径具有相同值时，它们被

362

第 9 章

称为是等价的。

为了在我们的矩阵式中表示重入，我们将在共享的特征结构第一次出现的地方加一 个用括号括起的数字前缀，例如(1)。以后任何对这个结构的引用将使用符号>(1), 如下所示。

>>>print nltk.FeatStruct("""[NAME='Iee',ADDRESS=(1)[NUMBER=74,STREET='rue Pascal'], SPOUSE=[NAME='Kim',ADDRESS->(1)]]""")

[ADDRESS=(1)[NUMBER=74 ]]

[ [STREET ='rue Pascal']]

[NAME ='Lee'

[

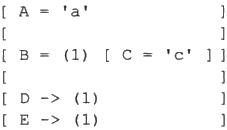
1

[SPOUSE =[ADDRESS ->(1) ]

[NAME ='Kim']

括号内的整数有时也被称为标记或同指标志 (coindex)。 整数的选择并不重要，可 以有任意数目的标记在单独的特征结构中。

>>>print nltk.FeatStruct("[A='a',B=(1)[C='c'],D->(1),E->(1)]")



包含和统一

认为特征结构提供一些对象的部分信息是很正常的，在这个意义上，我们可以根据 它们通用的程度给特征结构排序。例如：(25)a 比 ( 2 5 )b 更一般(更少特征), (25)b 比 ( 2 5 )c 更一般。

(25)a.[NUMBER =74]

b.[NUMBER =74 ]

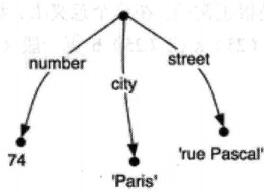
[STREET ='rue Pascal']

c.[NUMBER =74

[STREET ='rue Pascal']

[CITY='Paris' ]

这个顺序被称为包 含 (subsumpt ion); 一个更一般的特征结构包含(subsumes) 一 个较一般的。如果FSo 包 含FS₁ (正式的，我们写成 FS₀EFS₁), 那么 FS₁ 必须包含



FS₀ 所有路径和等价路径，也可能有额外的路径和等价路径。因此，(23)包括(24) 因为后者有额外的等价路径。包含只提供了特征结构上的偏序，这应该是显而易见 的，因为一些特征结构是不可比较的。例如：(26)既不包含(25)a 也不被(25) a包含。

(26) [TELNO =0127864296]

所以，我们已经看到了， 一些特征结构比其他的更具体。我们如何去具体化一个给 定的特征结构?例如：决定地址应不仅包括门牌号码和街道名称，还应该包括城市。 也就是说，我们可能要用(27)b 合并图(27) a 以产生(27)c。

(27)

a.

number street

rue Pascal'

74

b.

city

"Paris'

C.

合并两个特征结构的信息被称为**统** **一** **，**由方法 unifyO 支持。

n(CITY='Paris')

>>>fs1 =nltk.FeatStruct(NUMBER=74,STREET='rue Pascal')



统一被正式定义为二元操作：FSoUFS₁。统一是对称的，所以 FSoUFS₁=FS₁UFSo。 Python 中也是如此如下。

>>>print fs2.unify(fs1)

[CITY ='Paris'

[NUMBER =74

[STREET ='rue Pascal']

如果我们统一两个具有包含关系的特征结构，那么统一的结果是两个中更具体的 那个。

(28)IfFS₀≤FS₁,then FSoU FS₁=FS₁

例子中统一(25)b 和(25)c 的结果是(25)c。

如果这两个特征结构共享路径π,但在 FSo中的π值与在 FS₁中的π值是不同的 原子值，那么,FS₀ 和FS₁ 之间的统一将失败。通过设置统一的结果为None 解决 这个问题。

>>>fs0 =nltk.FeatStruct(A='a')

>>>fs1 =nltk.FeatStruct(A='b')

>>>fs2 =fs0.unify(fs1)

>>>print fs2

None

如果我们现在考虑统一是如何与结构共享相互作用的，那么事情就变得很“有趣” 了。首先，在Python中定义(23)。

>>>fs0 =nltk.FeatStruct("""[NAME=Lee,

· ADDRESS=[NUMBER=74,

· STREET='rue Pascal'],

! SPOUSE=[NAME=Kim,

. · ADDRESS=[NUMBER=74,

· STREET='rue Pascal']]]""")

>>>print fs0

[ADDRESS

[NAME =

=[NUMBER

[STREET

'Lee'

=74

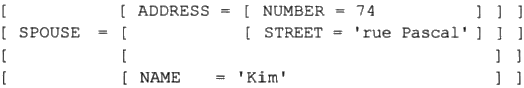
='rue

Pascal']

1

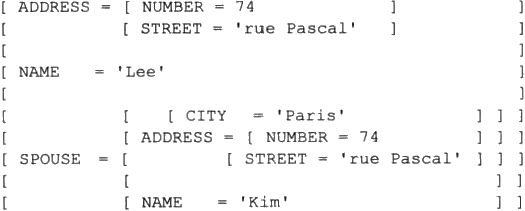
]

1



如果指定 CITY作为 Kim的地址参数会发生什么?请注意， fs1 需要包括从特征结 构的根到CITY 的整个路径。

>>>fs1=nltk.FeatStruct("[SPOUSE =[ADDRESS =[CITY =Paris]]]") >>>print fs1.unify(fs0)



通过对比，如果fs1 与 fs2 的结构共享版本统一，结果是非常不同的，如(24) 所示。

>>>fs2=nltk.FeatStruct("""[NAME=Lee,ADDRESS=(1)[NUMBER=74,STREET='rue Pascal'],

>>>print [

[ADDRESS

[

[NAME

[

[SPOUSE

SPOUSE=[NAME=Kim,ADDRESS->(1)]]""")

fs1.unify(fs2)

[CITY ='Paris' ] ]

=(1) [NUMBER =74

1

」

]

1

」

]

[ STREET ='rue Pascal'

]

='Lee'

->(1) ]

=[ADDRESS [NAME

='Kim']

除了更新 Lee 地址中的 Kim 地址的“副本”,同时更新他们两个的地址。更一般 的，如果统一包含指定一些路径π的值，那么统一同时指定与π等价的所有路径 的值。

正如我们已经看到的，结构共享也可以使用变量表示，如?x。

>>>fs1=nltk.FeatStruct("[ADDRESS1=[NUMBER=74,STREET='rue Pascal']]") >>>fs2 =nltk.Featstruct("[ADDRESS1=?x,ADDRESS2=?x]")

>>>print fs2

[ADDRESS1 [ADDRESS2 >>>print [ADDRESS1

[ADDRESS2

=?x ]

=?x ]

fs2.unify(fsl) =(1) [NUMBER

[STREET

->(1)

=74

='rue

] ]

Pascal']]

**9.3** **扩展基于特征的文法**

在本节中，我们继续介绍基于特征的文法，探索各种语言问题，以及将特征纳入文 法的好处。

**子类别**

第8章中，通过增强类别标签来表示不同类别的动词，分别用标签IV 和 TV 表示 不及物动词和及物动词。这样可编写如下的产生式。

(29)VP ->IV

VP->TV NP

虽然IV 和TV 是V 的两种类型，但它们只是CFG 中的原子非终结符，同任何其他符 号对之间的区别是一样的。这个符号不会告诉我们任何有关一般动词的信息，例如： 我们不能说“V 类的所有词汇都按时态标记”。因为 walk 是 IV 类的项目，而不是 V 类的。因此，通过V 的特征，我们是否可以替换如TV 和 IV 类别的标签? (V 的特 征表明动词是否与后面的NP对象结合或者能否不带补语)

最初为文法框架开发的简单方法称为广义短语结构文法 (Generalized Phrase Structure Grammar,GPSG), 通过允许词汇类别支持 SUBCAT特征尝试解决这个问 题，设 SUBCAT 特征表明该项目所属的子类别。相比 GPSG 使用整数值表示 SUBCAT, 下面的例子采用了更容易记忆的值，即 intrans 、trans 和 clause。

(30)VP[TENSE=?t,NUM=?n]->V[SUBCAT=intrans,TENSE=?t,NUM=?n] VP[TENSE=?t,NUM=?n]->V[SUBCAT=trans,TENSE=?t,NUM=?n]NP

VP[TENSE=?t,NUM=?n]->V[SUBCAT=clause,TENSE=?t,NUM=?n]SBar

V[SUBCAT=intrans,TENSE=pres,NUM=sg]->'disappears' l'walks'

V[SUBCAT=trans,TENSE=pres,NUM=sg]->'sees'l'likes'

V[SUBCAT=clause,TENSE=pres,NUM=sg]->'says'l 'claims'

V[SUBCAT=intrans,TENSE=pres,NUM=pl]->'disappear'l'walk'

V[SUBCAT=trans,TENSE=pres,NUM=pl]->'see'l 'like'

V[SUBCAT=clause,TENSE=pres,NUM=pl] ->'say' l 'claim'

V[SUBCAT=intrans,TENSE=past]->'disappeared' l 'walked'

V[SUBCAT=trans,TENSE=past]->'saw'l 'liked'

V[SUBCAT=clause,TENSE=past]->'said'l'claimed'

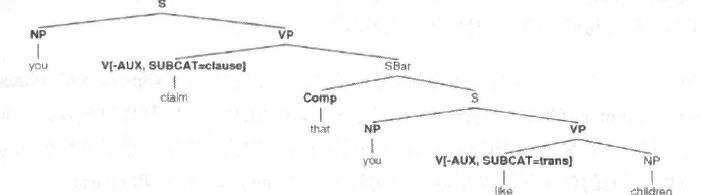
若一个词汇类别，如 V[SUBCAT=trans], 则可以将 SUBCAT 解释为指向一个产生 式，其中 V[SUBCAT=trans] 作为 VP 产生式中的核心词引入。按照约定， SUBCAT 值和引入核心词的产生式之间存在对应关系。按照这种做法，SUBCAT 只能出现在 词汇类别中，且它是没有意义的，例如：要在VP 上指定SUBCAT。 与要求的一样， walk 和 like 都属于类别V 。然 而 walk 将只在特征 SUBCAT=intrans 右侧产生式扩 展 VP 中出现，而like 则相反，它需要一个 SUBCAT=trans。

在(30)中的动词的第三个类别中，我们指定了一个类SBar。 这是一个从句标签， 就像例子 You claim that you like children中 claim 的补语。我们需要两个的产生式来 进一步分析这样的句子。

(31) SBar ->Comp S

Comp ->'that'

产生的结构如下。

(32)

由于文法类别，子类别的替代方式展示在基于特征的框架中，如 PATR 和核心 驱动短语结构文法 (Head-driven Phrase Structure Grammar)。不采用 SUBCAT 值作为索引产生式的方式，SUBCAT 值直接为中心词的配价(它能结合的参数 列表)编码。例如：动词 put 带 有NP 和 PP 补 语 (put the book on the table) 可

以表示为(33)。

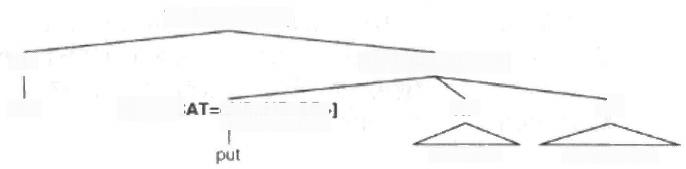
(33)V[SUBCAT=<NP,NP,PP>]

也就是说动词可以结合3个参数。列表最左边元素是主语 NP, 而其他所有的 这个例子中后面跟着 PP 的 NP 包括了补语子类别。当动词如 put 与适应的补语 结合时，SUBCAT 中指定的要求被免除，只需要主语 NP。这个类别相当于传统上 被认为的 VP, 可以表示如下。

(34)V[SUBCAT=<NP>]

最后，句子成为一种不需要更多参数的动词类别，同时具有一个值为空列表的 SUBCAT。树状图(35)展示了在短语 Kim put the book on the table 中这些类别是 如何组合的。

(35)

NP

Kim

V[SUBCAT=<>]

NP,NP,PP>I

V[SUBC

V[SUBCAT=<NP>]

NP

the book

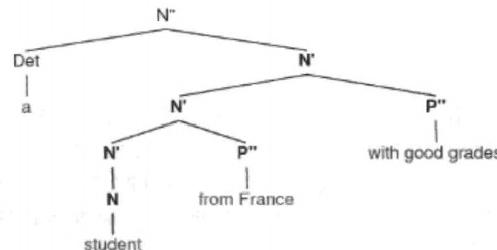
Pp

on the table

**核心词回顾**

在上一节中，通过从主类别标签分解出子类别信息，我们可以概括更多有关动词属 性的内容。类似的属性如下： V 类的表达式是 VP 类短语的核心。同样， N 是 NP 的核心词， A (即形容词)是AP 的核心词， P (即介词)是 PP的核心词。并非所 有的短语都有核心词，例如： 一般认为连词短语(如：the book and the bel) 缺乏核 心词。然而，我们希望我们的文法形式能表达它所具有的根源/核心关系。现在， V 和 VP只是原子符号，我们需要找到一种用特征将它们关联起来的方法(就像我们 以前关联IV 和 TV那样)。

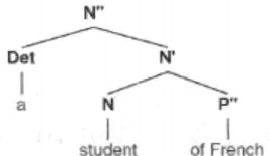
X-bar 句法通过抽象出**短语级别**的概念，解决了这个问题。它通常认为有3个这样 的级别。如果N 表示词汇级别，那么N表示更高一层级别，对应较传统的级别Nom; N"表示短语级别，对应类别NP 。(36)a 展示了这种表示结构，而(36)b 是更传 统的对应形式。

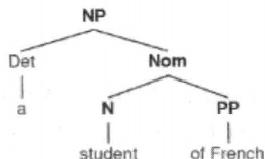


(36)

a.

b.





结构(36)a 的核心词是N, 而 N'和N" 被称为N 的 **(短语的)投影**。N"是**最大的** **投影，** N 有时也被称为零投影。X-bar 文法的中心思想是所有成分都有结构的类似 性。使用X 作为 N 、V 、A 和 P 的变量，我们说词汇核心X 的直接补语子类别总 是位于核心词同级的位置，而修饰成分位于中间类别X'同级的位置。因此，(37) 中两个 P"修饰成分的分布与(36)a 中补语P"形成对比。

(37)

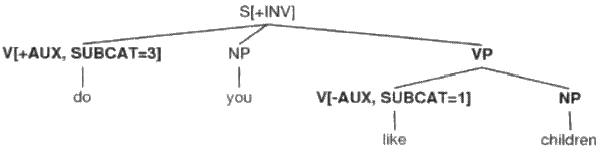
(38)中的产生式展示了X-bar 的级别如何用特征结构进行编码。(37)中的嵌入结 构是通过扩展N[BAR=1] 递归规则的两个应用得到的。

(38)S->N[BAR=2]V[BAR=2]

N[BAR=2]->Det N[BAR=1]

N[BAR=1]->N[BAR=1]P[BAR=2]

N[BAR=1]->N[BAR=0] P[BAR=2]



**助动词与倒装**

倒装从句 主语和动词顺序互换 -出现在英语疑问句中，也出现在“否定”副 词之后。

(39)a.Do you like children?

b.Can Jody walk?

(40)a.Rarely do you see Kim.

b.Never have I seen this dog,

然而，我们不能把任意的动词放在主语前面。

(41)a.\*Like you children?

b.\*Walks Jody?

(42)a.\*Rarely see you Kim.

b.\*Never saw I this dog

放置在倒装从句开头的动词术语叫做**助动词，** 如 ：do 、can 和 have, 也包括 be、 will 和 shall 。获得这种结构的方法之一是使用以下产生式。

(43)S[+INV]->V[+AUX]NP VP

也就是说，标记有[+inv]的从句包含助动词，其后跟着 VP。(在更详细的文法中， 我们需要在VP 的形式上加一些限制，但取决于选择的助动词。)(44)所示为一个 倒装从句的结构。

(44)

**无限制依赖成分**

考虑下面的对比。

(45)a.You like Jody.

b.\*You like.

(46)a.You put the card into the slot.

b.\*You put into the slot.

c.\*You put the card.

d.\*You put.

动词 like 需要NP 补语，而 put 需要跟随其后的NP 和 PP 。(45) 和(46)表明， 这些补语是必须的：省略它们会导致语句不符合语法。然而，在有些上下文中强制 性的补语可以省略，如(47)和(48)所示。

(47)a.Kim knows who you like.

b.This music,you really like.

(48)a.Which card do you put into the slot?

b.Which slot do you put the card into?

也就是说，强制性补语可以被省略，如果句子中有适当的**填充的话，** 例如(47)a 中 的疑问词 who,(47)b 中前置的主题 this music,或(48)中 wh 短语 which card/slot。 通常在类似(47)和(48)中的句子会包含一个**缺口，**即强制性补语被省略了，这

些缺口一般用下划线标出。

(49)a.Which card do you put\_\_into the slot?

b.Which slot do you put the card into\_\_?

所以，只有填充词许可，才能出现缺口。相反，填充词只会出现在句子可以有缺口 的地方，如下面的例子所示。

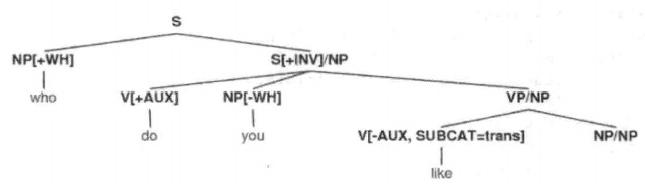
(50)a.\*Kim knows who you like Jody.

b.\*This music,you really like hip-hop.

(51)a.\*Which card do you put this into the slot?

b.\*Which slot do you put the card into this one?

填充词和缺口之间的相互关系有时被称为“依赖”。在理论语言学中一直在研究填 充词和它许可的缺口之间可以相互作用的原因，以及能否简单地列出一个分开这两 个序列的有限集合吗?答案是否定的，因为填充词和缺口之间的距离没有上界。通 过包含句子补语的成分可以很容易地证实这一点，如(52)所示。



(52)a.Who do you like\_\_?

b.Who do you claim that you like\_\_?

c.Who do you claim that Jody says that you like\_\_?

因为我们可以无限地加深句子补语的递归，所以在整个句子中缺口可以无限远地被 填充。这一属性导致无限依赖成分的概念，也就是填充词-缺口依赖，即填充词和 缺口之间的距离没有上界。

目前存在多种处理形式化文法中无限依赖的机制；在这里我们说明广义短语结构文法中 使用的方法，其中包含斜线类别。斜线类别的形式是Y/XP: 类别Y 的短语缺少类别 XP 的子成分。例如： S/NP 是缺少NP的 S。斜线类别的使用说明如(53)所示。

(53)

树状图的顶端部分引入了填充词 who (作为 NP[+wh]类表达式对待)和相应包含成 分S/NP 的缺口。缺口信息通过VP/NP 类别“预填充”树状图的下方，直到到达类 别VP/NP。这时，由于意识到缺口信息为空字符串直接受控于NP/NP, 依赖被排除。

我们需要认为斜线类别是一种全新的对象吗?通过将斜线作为特征，其右边的类别 作为值，可以将它们纳入现有的基于特征的框架中，也就是说， S/NP 可变为 S[SLASH=NP]。在实践中，这也是分析器解释斜线类别的方法。

例9-3中的文法演示了斜线类别的主要原则，而且还包括倒装从句的产生式。为了 简化演示，我们省略了动词上的任何时态规范。

例9-3 具有倒装从句和长距离依赖的产生式的文法，使用斜线类别

>>>nltk.data.show\_cfg('grammars/book\_grammars/feat1.fcfg') 8 start S

####################

#Grammar Productions

##################

S[-INV]->NP VP

S[-INV]/?x ->NP VP/?x

S[-INV] ->NP S/NP

S[-INV]->Adv[+NEG] S[+INV]

S[+INV]->V[+AUX]NP VP

S[+INV]/?x ->V[+AUX]NP VP/?x

SBar ->Comp S[-INV]

a-rmApT=n[]-X]

VP ->V[SUBCAT=trans,-AUX] NP

VP/?x ->V[SUBCAT=trans,-AUX]NP/?x

VP ->V[SUBCAT=clause,-AUX]SBar

VP/?x ->V[SUBCAT=clause,-AUX]SBar/?x

VP ->V[+AUX]VP

VP/?x ->V[+AUX]VP/?x

# ###################

#Lexical Productions

,AUX]->'walk'l 'sing'

V[SUBCAT=trans,-AUX]->'see'| 'like'

V[SUBCAT=clause,-AUX]->'say'l'claim'

V[+AUX]->'do'l 'can'

NP[-WH]->'you'l'cats'

NP[+WH]->'who'

Adv[+NEG]->'rarely'l 'never'

NP/NP ->

Comp ->'that'

例9-3中的文法包含有“缺口引进”产生式，即 S[-INV]>NPS/NP。 为了正确地预 填充斜线特征，我们需要为扩展S 、VP 和 NP 的产生式中箭头两侧的斜线添加变量 值。例如： VP/?x>VSBar/?x 是 VP>VSBar 的斜线版本，也就是说，可以为一个 成分的根源VP 指定斜线值，前提是为核心SBar 指定同样的值。最后， NP/NP-> 允 许 NP 上的斜线信息为空字符串。使用例9-3中的文法，我们可以分析序列： who do you claim that you like。

>>>tokens ='who do you claim that you like'.split()

>>>from nltk import load\_parser

>>>cp =load\_parser('grammars/book\_grammars/feat1.fcfg')

>>>for tree in cp.nbest\_parse(tokens):

 print tree

(S[-INV]

(NP[+WH]who)

(S[+INV]/NP[]

(V[+AUX] do)

(NP[-WH] you)

(VP[]/NP[]

(54)

(V[-AUX,SUBCAT='clause']claim)

(SBar[]/NP[]

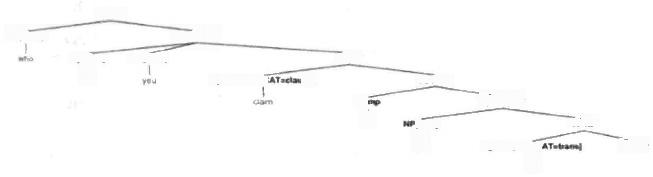
(Comp[]that)

(S[-INV]/NP[]

(NP[-WH] you)

(VP[]/NP[] (V[-AUX,SUBCAT='trans']like) (NP[]/NP[])))))))

**更易读的树状图版本如(54)所示。**



M]

moe

Em

SRar NP

Can

m-

!

1.

VH-AUX,SUBCA

VF-AUX,SUBC

NF-WJ

ymP

a|

V|+AUR]

NPP



NPr-WJ

vFNP

。

**例9-** **3** **中的文法也可以分析没有缺口的句子。**

>>>tokens ='you claim that you like cats'.split()

>>>for tree in cp.nbest\_parse(tokens):

.\* print tree

(S[-INV]

(NP[-WH]you)

(VP[]

(V[-AUX,SUBCAT='clause']

claim)

(SBar[]

(Comp[]that)

(S[-INV]

(NP[-WH]you)

(VP[] (V[-AUX,SUBCAT='trans']like)

(NP[-WH] cats))))))

**此外，它还允许没有** **wh 结构的倒装句。**

>>>tokens ='rarely do you sing'.split()

>>>for tree in cp.nbest\_parse(tokens):

 print tree

(S[-INV]

(Adv[+NEG] rarely)

(S[+INV]

(V[+AUX]do)

(NP[-WH]you)

(VP[] (V[-AUX,SUBCAT='intrans']sing))))

**德语中的格和性别**

与英语相比，德语的协议具有相对丰富的形态。例如：在德语中定冠词根据格、性 别和数量变化，如表9-2所示。

**表9-2** **德语定冠词的形态范式**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **格** | **男** **性** | **女** **性** | **中** **性** | **复** **数** |
| 主格 | der | die | das | die |
| 所有格 | des | der | des | der |
| 与格 | dem | der | dem | den |
| 宾格 | den | die | das | die |

德语中的主语采用主格，大多数动词采用宾格支配它们的宾语。不过，也有例外， 例如： helfen 支配与格。

(55)a.Die

the.NOM.FEM.SG

'the cat sees b.\*Die

the.NOM.FEM.SG c.Die

the.NOM.FEM.SG

'the cat helps d.\*Die

the.NOM.FEM.SG

Katze

cat.3.FEM.SG the dog'

Katze

cat.3.FEM.SG

Katze

cat.3.FEM.SG the dog'

Katze cat.3.FEM.SG

sieht den

see.3.SG the.ACC.MASC.SG

sieht dem

see.3.SG the.DAT.MASC.SG

hilft dem

help.3.SG the.DAT.MASC.SG

hilft den

help.3.SG the.ACC.MASC.SG

Hund

dog.3.MASC.SG

Hund

dog.3.MASC.SG

Hund

dog.3.MASC.SG

Hund

dog.3.MASC.SG

**例9-4中的文法演示带格协议(包括人称、数量和性别)的相互作用。**

**例9-4** **基于特征的文法的例子**

>>>nltk.data.show\_cfg('grammars/book\_grammars/german.fcfg')

8 start S

#Grammar Productions

S ->NP[CASE=nom,AGR=?a]VP[AGR=?a]

NP[CASE=?c,AGR=?a]

NP[CASE=?c,AGR=?a]

->PRO[CASE=?c,AGR=?a]

->Det[CASE=?c,AGR=?a]N[CASE=?c,AGR=?a]

VP[AGR=?a]->IV[AGR=?a]

VP[AGR=?a] ->TV[OBJCASE=?c,AGR=?a] NP[CASE=?c]

#Lexical #Singular #masc

Det

Productions

determiners

[CASE=nom,AGR=[GND=masc,PER=3,NUM=sg]]->'der'

Det[CASE=dat,AGR=[GND=masc,PER=3,NUM=sg]]->'dem'

Det[CASE=acc,AGR=[GND=masc,PER=3,NUM=sg]]->'den'

#fem

Det[CASE=nom,AGR=[GND=fem,PER=3,NUM=sg]]->'die'

Det[CASE=dat,AGR=[GND=fem,PER=3,NUM=sg]]->'der'

Det[CASE=acc,AGR=[GND=fem,PER=3,NUM=sg]]->'die'

#Plural determiners

Det [CASE=nom,AGR=[PER=3,NUM=p1]]->'die'

Det[CASE=dat,AGR=[PER=3,NUM=pl]]->'den'

Det[CASE=acc,AGR=[PER=3,NUM=pl]]->'die'

#Nouns

N[AGR=[GND=masc,PER=3,NUM=sg]]->'Hund'

N[CASE=nom,AGR=[GND=masc,PER=3,NUM=pl]]->'Hunde'

N[CASE=dat,AGR=[GND=masc,PER=3,NUM=pl]]->'Hunden'

N[CASE=acc,AGR=[GND=masc,PER=3,NUM=pl]]->'Hunde'

N[AGR=[GND=fem,PER=3,NUM=sg]] ->'Katze'

N[AGR=[GND=fem,PER=3,NUM=pl]]->'Katzen'

# Pronouns

PRO[CASE=nom,AGR=[PER=1,NUM=sg]]->'ich'

PRO[CASE=acc,AGR=[PER=1,NUM=sg]]->'mich'

PRO[CASE=dat,AGR=[PER=1,NUM=sg]] ->'mir'

PRO[CASE=nom, AGR=[PER=2,NUM=sg]]->'du'

PRO[CASE=nom, AGR=[PER=3,NUM=sg]]->'er'l 'sie'l'es'

PRO[CASE=nom,AGR=[PER=1,NUM=pl]]->'wir'

PRO[CASE=acc,AGR=[PER=1,NUM=pl]]->'uns'

PRO[CASE=dat,AGR=[PER=1,NUM=pl]]->'uns'

PRO[CASE=nom,AGR=[PER=2,NUM=pl]] ->'ihr'

PRO[CASE=nom,AGR=[PER=3,NUM=pl]]->'sie'

#Verbs

IV[AGR=[NUM=sg,PER=1]]->'komme'

IV[AGR=[NUM=sg,PER=2]]->'kommst'

IV[AGR=[NUM=sg,PER=3]]->'kommt'

IV[AGR=[NUM=pl,PER=1]]->'kommen'

IV[AGR=[NUM=pl,PER=2]]->'kommt'

IV[AGR=[NUM=pl,PER=3]]->'kommen'

TV[OBJCASE=acc, AGR=[NUM=sg,PER=1]] ->'sehe'l'mag'

TV [OBJCASE=acc, AGR=[NUM=sg,PER=2]] ->'siehst'l 'magst' TV[OBJCASE=acc, AGR=[NUM=sg,PER=3]] ->'sieht'l'mag'

[Odat, =R,PE-g>e'olgstt'

TV[OBJCASE=dat, AGR=[NUM=sg,PER=3]]->'folgt'l 'hilft'

|  |  |
| --- | --- |
| TV[OBJCASE=acc, | AGR=[NUM=pl,PER=1]]->'sehen' l'moegen' |
| TV[OBJCASE=acc, | AGR=[NUM=pl,PER=2]]->'sieht'l 'moegt' |
| TV[OBJCASE=acc, | AGR=[NUM=pl,PER=3]]->'sehen'l'moegen' |
| TV [OBJCASE=dat, | AGR=[NUM=pl,PER=1]]->'folgen'l'helfen' |
| TV[OBJCASE=dat, | AGR=[NUM=pl,PER=2]]->'folgt'l 'helft' |

TV[OBJCASE=dat,AGR=[NUM=pl,PER=3]]->'folgen'I 'helfen'

正如你可以看到的，特征 objcase 用来指定动词支配对象的格。下面的例子演示了

**包含支配与格动词的句子的分析树状图。**

>>>tokens ='ich folge den Katzen'.split()

>>>cp =load\_parser('grammars/book\_grammars/german.fcfg') >>>for tree in cp.nbest\_parse(tokens):

 print tree

(S[]

(NP[AGR=[NUM='sg',PER=1],CASE='nom']

(PRO[AGR=[NUM='sg',PER=1],CASE='nom']ich))

(VP[AGR=[NUM='sg',PER=1]]

(TV[AGR=[NUM='sg',PER=1],OBJCASE='dat']folge)

(NP[AGR=[GND='fem',NUM='pl',PER=3],CASE='dat']

R=3]]Katzen))))

在开发文法时，排除不符合语法的词序列往往与分析符合语法的词序列同样具有挑 战性。为了能知道序列分析在哪里和为什么失败，设置 load\_parser)方法的 trace 参数可能是至关重要的。考虑下面的故障分析。

>>>tokens ='ich folge den Katze'.split()

>>>cp =load\_parser('grammars/book\_grammars/german.fcfg',trace=2) >>>for tree in cp.nbest\_parse(tokens):

print tree

l.ich.fol.den.Kat.I

l[---] . .I PRO[AGR=[NUM=¹sg',PER=1],CASE='nom']->'ich'\*

l[---] . . .|NP[AGR=[NUM='sg',PER=1],CASE='nom']

->PRO[AGR=[NUM='sg',PER=1],CASE='nom']\*

l[---> . .|S[]->NP[AGR=?a,CASE='nom']\*VP[AGR=?a]

{?a:[NUM='sg',PER=1]}

l. [---] .I TV[AGR=[NUM='sg',PER=1],OBJCASE='dat']->'folge'\*

1. [---> .|VP[AGR=?a]->TV[AGR=?a,OBJCASE=?c]

\*NP[CASE=?c]{?a:[NUM='sg',PER=1],?c:'dat'}

1..[---] .I Det[AGR=[GND='masc',NUM='sg',PER=3],CASE='acc']->'den'\*

l. · [---] ·Det[AGR=[NUM='pl',PER=3],CASE='dat']->'den'\*

1. [---> .|NP[AGR=?a,CASE=?c]->Det[AGR=?a,CASE=?c]

\*N[AGR=?a,CASE=?c]{?a:[NUM='pl',PER=3],?c;'dat'}

1… [---> {.?I,]?\*cN:a,CASE=?c]

1 … [---]IN[AGR=[GND='fem',NUM='sg',PER=3]]->'Katze'\*

跟踪中的最后两个 Scanner 行显示 den 被识别为两个可能的类别： Det[AGR=[GND='masc',NUM='sg',PER=3],CASE='acc']和 Det[AGR=[NUM=pl, PER=3],CASE='dat]。我们从例9-4中的文法知道Katze的类别是N[AGR=[GND= fem,NUM=sg,PER=3]]。然而，产生式中的变量? a 没有被绑定。

NP[CASE=?c,AGR=?a] ->Det[CASE=?c,AGR=?a] N[CASE=?c,AGR=?a]

这样就可以满足这些限制，因为Katze的 AGR 值将不与 den 的任何一个 AGR 值统 一，也就是[GND='masc',NUM='sg,PER=3] 或[NUM=pl,PER=3]。

**9.4** **小结**

● 上下文无关文法的传统分类是原子符号。特征结构的重要作用之一是捕捉精细 的区分，否则将需要数量翻倍的原子类别。

● 通过使用特征值的变量，我们可以表达出文法产生式中的限制，使得不同的特 征规格之间相互依赖。

● 通常情况下，在词汇层面指定固定的特征值，并限制短语中的特征值使其与它 们“孩子”的对应值相统一。

● 特征值可以是原子的，也可以是复杂的。原子值的特定类别是布尔值，按照惯 例用[+/- feat]表示。

● 两个特征可以共享一个值(原子的或复杂的)。具有共享值的结构被称为重入。 共享值被表示为AVM中的数字索引(或标记)。

● 特征结构中的路径是特征元组，对应着从图底部开始的弧序列上的标签。

● 如果两条路径共享一个值，那么它们是等价的。

● 包含的特征结构是偏序的。FSo包含 FS₁, 当 FS₀ 比 FS₁ 更一般(较少信息)的 时候。

● 两种结构 FS₀ 和 FS₁ 的统一形式，如果存在，就是包含 FS₀ 和 FS₁ 合并信息的 特征结构 FS₂。

● 如果统一在FS 中指定一条路径π,那么它也指定与π等效的每个路径π'。

· 我们可以使用特征结构对大量广泛语言学现象进行简洁的分析，包括动词子类 别、倒装结构、无限制依赖结构和格支配。

**9.5** **深入阅读**

有关本章的进一步材料请参考 <http://www.nltk.org/>, 包括特征结构的 HOWTO 、 特

征文法、Earley 分析和文法测试套件。

如果想更好地了解协议现象，请参阅 Corbett(2006)。

理论语言学中使用特征的最初目的是捕捉语音的音素特性。例如：音/b/可能会被分 解成结构[+labial,+voice]([+ 唇，+语音])。 一个重要的目的是获取分割类别之间的 一般性，例如：/n/在任一+labial (唇)辅音前面都读作/m/。在乔姆斯基文法中， 原子特征可用来处理一些如协议的现象，使用原子特征是很标准的，同时通过类比 与音韵也用来获取跨句法类别的概括。在句法理论中特征的扩展使用是广义短语结 构语法 (GPSG;[Gazdar et al.,1985]), 特别是使用带有复杂值的特征。

从计算语言学的角度来看， (Kay,1985) 提出了语言功能可以被属性-值结构统一 捕获，类似的方法由 (Grosz&Stickel,1983) 在 PATR-Ⅱ 体系中提出。早期的词

汇功能文法 (Lexical-Functional grammar,LFG) 介 绍 了f-structure 概 念 (Kaplan &

Bresnan,1982), 主要目的是表示文法关系和与成分结构短语关联的谓词参数结构。 Shieber(1986) 更好地介绍了基于特征文法方面的研究。

当研究人员试图建立反例模型时，在特征结构代数方法概念上出现了问题。另一种

观点，由 (Kasper&Rounds,1986)and(Johnson,198 8) 开创，认为文法涉及结

构功能的描述而不是结构本身。这些描述使用逻辑操作，而否定仅仅是特征描述上 的普通逻辑运算。这种面向描述的观点对LFG 从一开始就是不可或缺的 (Kaplan, 1989),也被中心词驱动短语结构文法 (HPSG) 的较高版本采用(Sag&Wasow , 1999)。在 <http://www.cl.uni-bremen.de/HPSG-Bi> b/上有关于 HPSG 文献的全面的参

考书目。

本章介绍的特征结构无法获取语言信息中的重要限制。例如：有没有办法只将NUM 的值表达为sg 和 pl, 而指定[NUM=masc] 是反常的。同样，我们不能确定AGR 的 复合值必须包含特征 PER、NUM 和 GND 的指定，而不能包含如[SUBCAT=tr ans] 这样的指定。**指定类型的特征结构**用来弥补这方面的不足。 一本早期的全面总结了 关于指定类型的特征结构的书籍是 (Emele &Zajac,1990)。有关形式化基础的全

面的检查可以在 (Carpenter,1992) 中 找 到 ，(Copestake,20 02) 重点关注利用面

向 HPSG 的方法指定类型的特征结构。

还有很多著作是关于德语基于特征文法框架分析的。(Nerbonne ,Netter &Pollard, 1994)是 H P S G 著作的开创巨著,而(Müller,2002) 给 出HPSG 中广泛而详细的 德语句法分析。

Jurafsky &Martin(2008) 的第15章讨论了特征结构、统一算法和如何将统一整合 到分析算法中。

**9.6** **练习**

1.O 正确分析词序列需要哪些限制，如Iam happy 和 she is happy 而不是\*you is happy 或\*they am happy? 实现英语中动词be 的现在时态范例有两个解决方案，首 先以文法(8)作为起点，然后以文法(20)为起点。

2.O 对例9-1 中文法进行改变，使用特征 COUNT 来区分下面的句子。

(56)a.The boy sings.

b.\*Boy sings.

(57)a.The boys sing.

b.Boys sing.

(58)a.The water is precious.

b.Water is precious.

3.O 编写函数 subsumes()判断两个特征结构fs1 和 fs2,fs1 是否包含 fs2。

4.O 修改(30)中所示的文法，引入特征BAR 来处理短语投影。

5.O 修改例9-4中的德语文法，引入9.3节中介绍的子类别处理办法。

6.①开发一个基于特征的文法，并正确描述下面的西班牙语名词短语。

(59)un cuadro hermos-o

INDEF.SG.MASC picture beautiful-SG.MASC

'a beautiful picture'

(60) un-os

INDEF-PL.MASC

cuadro-s

picture-PL

hermos-os

beautiful-PL.MASC

'beautiful pictures'

(61)un-a cortina hermos-a

INDEF-SG.FEM curtain beautiful-SG.FEM

'a beautiful curtain'

(62) un-as cortina-s hermos-as

INDEF-PL.FEM curtain beautiful-PL.FEM

'beautiful curtains'

7.①开发 earley\_parser 的包装程序，只在输入序列分析出错时才输出跟踪。

8.①思考例9-5的特征结构。

例9-5 探索特征结构。

fs1=nltk.FeatStruct("[A=?x,B=[C =?x]]")

("[[(1]=]"b)],C->(1)]")

fs2 =nltk.FeatStruct("[B=[D =d]]")

fs5=nltk.FeatStruct("[A=(1)[D=?x],C=[E ->(1),F=?x]]")

fs6 =nltk.FeatStruct("[A= [D=d]]")

**([**?x,E->(]]]"])")

fs9 =nltk.FeatStruct("[A=[B=b],C=[E = [G=e]]]")

fs10 =nltk.FeatStruct("[A=(1)[B =b],C ->(1)]")

在纸上计算下面统一结果是什么。(提示：可使用绘制图结构)

a.fsl and fs2

b.fs1 and fs3

c.fs4 and fs5

d.fs5 and fs6

e.fs5 and fs7

f.fs8 and fs9

g.fs8 and fsl0

使用NLTK 检查你的答案。

9.①列出两个包含[A=?x,B=?x]的特征结构。

10.①忽略结构共享，给出一个统一两个特征结构的非正式算法。

11.①扩展例9-4中的德语语法，使它能处理所谓的动词第二顺位结构，如下所示。

(63)Heute sieht der Hund die Katze.

12.①同义动词的句法属性看上去略有不同(Levin,1993) 。 思考下面的动词 loaded、 filled 和 dumped 的文法模式。你能写出文法产生式处理这些数据吗?

(64)a.The farmer loaded the cart with sand

b.The farmer loaded sand into the cart

c.The farmer filled the cart with sand

d.\*The farmer filled sand into the cart

e.\*The farmer dumped the cart with sand

f.The farmer dumped sand into the cart

13.●形态范例很少是完全正规的，矩阵中的每个单元的意义有不同的实现方式。 例如：词 walk 的现在时态词性变化只有两种不同形式：第三人称单数的 walks 和 所有其他人称和数量组合的 walk 。一个好的分析不应该额外要求6个可能的形态 组合中有5个具有相同的实现方式。设计和实施一个方法处理这个问题。

14. ●所谓的核心特征在父节点和核心子节点之间共享。例如： TENSE 是核心特 征，在VP 和它的子核心V 之间共享。更多细节见 (Gazdar et al.,1985)。我们看 到的结构中大部分是核心结构 除了 SUBCAT 和 SLASH 。由于核心特征的共享 是可以预见的，它不需要在文法产生式中明确表示。开发一种方法自动计算核心结 构的这种规则行为的比重。

15. ● 扩展NLTK中特征结构的处理，允许统一值为链表的特征，使用它对HPSG 风格的子类别进行分析，核心类别的 SUBCAT 是其补语的类别和直接“父母” SUBCAT 值之间的连接。

16. ● 扩展NLTK 的特征结构处理，允许带未指定类别的产生式，例如： S[-INV]>? x

S/?x。

17. ● 扩展NLTK 的特征结构处理，允许指定类型的特征结构。

18.●挑选 (Huddleston&Pullum,2002) 中描述的文法结构，建立一个基于特征

的文法进行计数。



**第10章**

**分析语句的含义**

我们已经了解到利用计算机的能力来处理大规模文本是多么有用。现在既然已经有 了分析机制和基于特征的文法，那么能否做一些类似分析语句的含义的事情?本章 的目的是要回答下列问题。

(1)如何表示自然语言的含义，并能通过计算机进行处理?

(2)怎样才能将意思表示与无限制的语句集相关联?

(3)怎样才能通过连接意思表示与句子的程序来存储信息?

本章中，我们将学习一些逻辑语义方面的规范化技术，看看如何使用它们来查询存 储有客观真理的数据库。

**10.1** **自然语言理解**

**查询数据库**

假设有一个程序，输入一个自然语言问题，返回得到正确的答案。

(1) a.Which country is Athens in?

b.Greece.

编写这样的程序有多困难?我们可以使用到目前为止在这本书中学到的技术吗? 或者还需要新的技术?在本节中，我们将学习解决特定领域的任务是相当简单的。



但如果要以一种更通用的方式解决这个问题，就必须开辟一个全新的涉及意思表示 的理念和技术框架。

因此，首先假设有关于城市和国家的结构化数据。具体的，我们将使用一个前几行 如表10-1所示的数据库表。

表10-1所示的数据来自80聊天系统(Warren& Pereira,1982)。 人口 数以千计算，注意在这些例子中使用的数据可以追溯到至少20世纪 80年代，在 (Warren&Pereira,1982) 出版时，某些已经有些过时。

**表10-1** **city\_table:** **关于城市、国家和人口的表格**

|  |  |  |
| --- | --- | --- |
| City | Country | Population |
| athens | greece | 1368 |
| bangkok | thailand | 1178 |
| barcelona | spain | 1280 |
| berlin | east\_germany | 3481 |
| birmingham | united\_kingdom | 1112 |

从表格数据中检索答案最简单的方式是在数据库查询语言(如 SQL) 中编写查询 语句。

SQL(Structured Query Language, 结构化查询语言)是为在关系数据 库中检索和管理数据而设计的语言。如果你想了解更多有关SQL 的信 息，请参考 <http://www.w3schools.com/sqV>.

例如：执行查询(2)将得到值： 'greece'。

(2)SELECT Country FROM city\_table WHERE City ='athens'

这条语句得到由所有City 列的值为'athens'的数据行中Country 列的所有值组成的结 果的集合。

怎样才能使用英语作为输入得到与我们在查询系统中得到的相同效果呢?利用第9 章中描述的基于特征的文法规范，我们可以很容易地将英语翻译为 SQL 。文法 sql0.fefg 说明了如何使用语句分析组装串联语句的意思表示。每个短语结构规则为



特征 SEM 构建值作补充。你可以看到这些方法非常简单；在每一种情况下，将子 成分用字符串连接操作“+”来组成父成分的值。

>s>tnal.dta.show\_cfg('grammars/book\_grammars/sq10.fcfg')

S[SEM=(?np +WHERE +?vP)]->NP[SEM=?np]VP[SEM=?vp]

VP[SEM=(?v+?pp)]->IV[SEM=?v] PP[SEM=?pp]

VP[SEM=(?v+?ap)]->IV[SEM=?v]AP[SEM=?ap]

NP[SEM=(?det +?n)]->Det[SEM=?det]N[SEM=?n]

PP[SEM=(?p+?np)]->P[SEM=?p]NP[SEM=?np]

AP[SEM=?pp]->A[SEM=?a]PP[SEM=?pp]

NP[SEM='Country="greece"']->'Greece'

NP[SEM='Country="china"'] ->'China'

S[]O-tcyh\_'I]->'cities'

IAV[->'located'

P[SEM='']->'in'

**这样我们就能够分析SQL 查询了。**

>>>from nltk import load\_parser

>>>cp =load\_parser('grammars/book\_grammars/sq10.fcfg')

>>>query ='What cities are located in China'

>>>trees =cp.nbest\_parse(query.split())

>>>answer =trees[0].node['sem']

>>>q=''.join(answer)

>>>print q

SELECT City FROM city\_table WHERE Country="china"

**轮到你来：**

**设置跟踪为最大的运行分析器，即** **cp=load\_parser('grammars/book\_**

grammars/sql0.fcfg',trace=3), 研究当完整的边被加入到图表中时，如 何建立 SEM 的值。

最后，在数据库 city.db 上执行查询，检索得出的结果如下。

s 0 ismq\_orqt uorpora/city\_database/city.db',q)

>>>for r in rows:print r[0],①

canton chungking dairen harbin kowloon mukden peking shanghai sian tientsin

由于每行r 是一元数组，所以输出的是数组的成员，而不是数组本身°。

定义一个任务：计算机对自然语言查询做出反应，并返回有用的数据。通过将一个 小的英语子集翻译成 SQL 来实现这个任务。可以说，NLTK 代码已经“理解”了 SQL, 只要Python 能够对数据库执行 SQL 查询，通过扩展，它也能“理解”如“What cities are located in China” 这样的查询。这相当于自然语言理解的例子能够从荷兰 语翻译成英语。假设你是一个以英语为母语的人，现在开始学习荷兰语。你的老师 问你是否理解(3)的意思。

(3)Margrietje houdt van Brunoke.

如果你知道(3)中单个词的意思，并且知道如何将它们结合在一起组成整个句子 的意思，你就应该知道(3)的意思与 Margrietje loves Brunoke 相同。

观察者 Olga——可能会将此作为你已理解(3)的意思的证据。但是，前提是Olga 懂英语。如果她不懂英语，那么你从荷兰语到英语的翻译就不能向她证明你理解了 荷兰语。

文法 sql0.fcfg, 连同 NLTK 的 Earley 分析器是实现从英语翻译到 SQL 的工具。这 个文法够用吗?整个句子的 SQL 翻译是由句子成分的翻译组成起来的。然而，这 些成分的意思表示似乎没有很多的合理性。例如：名词短语 Which cities 的分析， 限定词和名词分别对应SQL 片 段SELECT 和 City FROM city\_table。但这两个都没 有单独的符合语法的意思。

语法还有另一种缺陷：“硬生生”地把一些数据库的细节加入其中。我们需要知道 有关表(如： city\_table)和字段的名称。但我们的数据库中可能确实存在相同的数 据行但用的是不同的表名和字段名，在这种情况下， SQL 就查询不到。同样，我 们可以不同的格式存储数据，如 XML, 在这种情况下，为了检索相同的结果需要 将英语查询语句翻译成 XML 查询语言而不是 SQL。这些因素表明我们应该将英语 翻译成比 SQL 更加抽象和通用的语言。

为了突出这一点，下面进行另一个英语查询及其翻译。

(4)a.What cities are in China and have populations above 1,000,000?

b.SELECT City FROM city\_table WHERE Country ='china'AND Population >1000



**轮到你来：**

扩展文法 sq10.fcfg使它能将(4)a 转换为(4)b, 检查查询所返回的 值。记得在聊天80数据库中数字以千为单位，因此(4b) 中的1000 代表一百万人口。

可能会发现最简单的方法是在处理之前，扩展文法以处理查询 What cities have populations above 1,000,000。完成这个任务之后，可以将你 的方案与 NLTK 数据分布的 grammars/book\_grammars/sql1.fcfg 进行 比较。

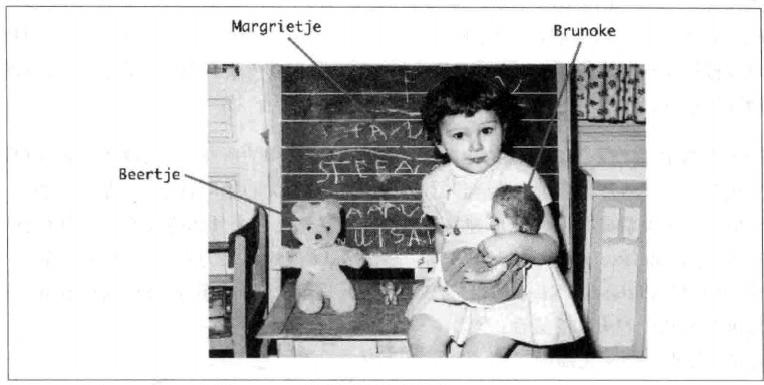
观察(4)a 中连接词and 被翻译成(4)b 中 SQL 对应的 AND。后者告诉我们 从两个条件都为真的行中选择结果： Country 列的值是'china',Population 列的 值是大于1000。and 一个新的解释：它表示是在某些特定情况下什么是真的， 在条件s 中 Cond1AND Cond2为真，当且仅当条件 Cond1 在 s 中为真且 Cond2 也在s 中为真。虽然这并不是and 在英语中的全部意思，但它具有一个很好的 属性：独立于任何查询语言。事实上，我们已经给了它一个经典逻辑的标准解 释。在下面的章节中，我们将探讨将自然语言的句子翻译成逻辑而不是如 SQL 这样的可执行查询语言的方法。逻辑形式的一个优势是它们更抽象，因此更通 用。 一旦翻译成了逻辑，只要我们想要，就可以再翻译成其他各种具有特殊用 途的语言。事实上，大多数通过自然语言查询数据库的重要尝试都是使用这种 方法。

**自然语言、语义和逻辑**

我们一开始尝试捕捉(1)a 的意思，通过将它翻译成另一种计算机可以解释和执 行的语言——SQL 来查询。但是，这仍然在回避问题的实质：翻译是否正确。结 束数据库查询，我们注意到and 的意思似乎取决于能否指定在特定情况下语句的真 假。通过将S 与现实的情况关联，而不是将句子S 从一种语言翻译到另一种，来尝 试解释S 是关于什么的。进一步地说，假设有一种情况s, 其中有两个实体 Margrietje 和她最喜欢的娃娃Brunoke。此外，两个实体之间有一种称之为爱的关系。如果你 理解(3)的意思，那么你知道在情况s 中它为真。因为你知道 Margrietje 指的是 Margrietje,Brunoke 指的是Brunoke,houdt van 指的是爱的关系。

引进语义中的两个基本概念。第一个是在确定的情况下，陈述句非真即假。第二个 是名词短语和专有名词的定义指的是世界上的东西。所以(3)在 Margrietje 喜欢

娃娃Brunoke 这一情形中为真，如图10-1所示。



**图10-1** **对** **Margrietje 喜欢** **Brunoke 的情形进行的描绘**

一旦采取了在特定情况下事件真假的概念，我们就有了进行推理的强大工具。特别 是，我们可以推理语句集在某些情况下是否能同时为真。例如：(5)中的句子可以 都为真，而(6)和(7)中的则不能。换句话说，(5)中的句子是**一致的，** 而 ( 6) 和(7)中的是**不一致的。**

(5)a.Sylvania is to the north of Freedonia.

b.Freedonia is a republic.

(6)a.The capital of Freedonia has a population of 9,000.

b.No city in Freedonia has a population of 9,000.

(7)a.Sylvania is to the north of Freedonia.

b.Freedonia is to the north of Sylvania.

选择有关 fictional countries(虚构的国家)(选自Marx Brothers 1933 年的电影《Duck Soup》) 的句子强调你对这些例子的推理并不取决于真实世界中的真与假。如果你 知道词 no 的意思，也知道一个国家的首都是一个位于该国的城市，那么你应该能 够得出这样的结论：(6)中的两个句子是不一致的，不管 Freedonia 在哪里和它的 首都有多少人口。也就是说，不存在使这两个句子同时都为真的情况。同样的，如 果你知道 to the north of所表达的关系是不对称的，那么你应该能够得出这样的结 论：(7)中的两句话是不一致的。

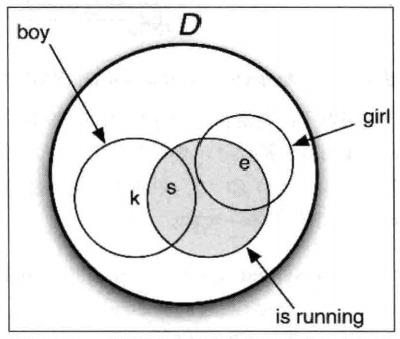
从广义上讲，基于逻辑方法的自然语言语义关注于那些指导我们判断自然语言的一 致性和不一致性的方面。设计一种逻辑语言的句法是为了使这些特征更标准更明 确。结果是如一致性这样的确定性属性往往可以简化成符号操作，也就是说， 一种 可以被计算机实施的任务。为了实现这种方法，我们首先要开发一种表示某种可能 情况的技术。我们做的这些逻辑学家称之为“模型”。

语句集W 的模型是某种情况的规范化表示，其中W中的所有句子都为真。表示模 型通常的方式涉及集合论。讨论到的域D (我们当前关心的所有实体)是个体的一 个集合，而关系被当作是从D 建立的集合。现在来看一个具体的例子。域 D 包括 3个孩子， Stefan 、Klaus和 Evi, 分别用s 、k 和 e 表示，记为D={s,k,e} 。 表达 式boy是包含Stefan和Klaus的集合，表达式girl 是包含Evi的集合，表达式is running

是包含 Stefan 和 Evi 的集合。图10-2

是这个模型的图形化描绘。

在本章后面，我们将使用模型来帮助 评估英语句子的真假，并用这种方式 来说明表示意思的一些方法。然而， 在更详细描述之前，让我们从更广阔 的角度进行讨论，回到我们在1.5节简 要提到过的主题。 一台计算机可以理 解句子的意思吗?我们该如何判断它 是否能理解?这和问题“计算机能思 考吗”类似。阿兰·图灵提出了通过检 查计算机与人类进行理智的对话的能

图10-2 一个模型图，包含一个域D 及D 的子集

分别对应描述 boy、girl和 is running

力 (Turing,1950) 来回答这一问题的著名方法。假设有一个与人聊天的会话和一 个与计算机聊天的会话，但一开始你并不知道哪个是哪个。在与它们两个聊天后， 如果你不能识别对方哪一个是计算机，那么说明这台计算机成功地模仿了人类。如 果一台计算机被当做人类成功地通过了这个“模仿游戏”(或“图灵测试”,这是它 是俗称),那么根据图灵的说法，就可以说这台计算机能思考，也即是说它具有了 智能。所以图灵从侧面回答了这个问题，不是检查计算机的内部状态，而是检查它 的行为来作为具有智能的证据。同样的道理，要说一台计算机懂英语，只需要它 的行为表现看上去它懂。这里最重要的不是图灵模仿游戏的细节，而是以可观察 的行为为依据来判断自然语言理解能力的想法。

**10.2** **命题逻辑**

设计一种逻辑语言的目的是使推理更明确规范。因此，它可以在自然语言中捕捉决 定一组句子是否是一致的方面。作为这种方法的一部分，我们需要开发一个句子φ 的逻辑表示，它能规范地捕捉φ为真的条件。我们先从一个简单的例子开始。

(8)[Klaus chased Evi]and [Evi ran away].

分别用φ和ψ替代(8)中的两个子句，并用&替代对应的英语词and 的逻辑操作： φ&ψ。这种结构是(80)的**逻辑形式。**

**命题逻辑**只表示对应特定语句连接词的语言结构部分，如刚才看到的and。其他类 似的连接词还有 not 、or和 if…,then…。命题逻辑形式中，这些连接词的对应形式 有时叫做**布尔运算符。**命题逻辑的基本表达式是**命题符号，** 通常写作P、Q、R 等。 表示布尔运算符的约定很多。由于我们将重点探索NLTK中的逻辑表示方式，所以 将使用下列 ASCⅡ版本的运算符。

>>>nltk.boolean\_ops()

negation

conjunction

disjunction

implication

equivalence

6

->

<->

有了命题符号和布尔运算符，我们可以建立命题逻辑的**规范公式**的无限集合(简称公 式)。首先，每个命题字母是一个公式。然后，如果φ是一个公式，那么-φ也是一个 公式。如果φ和ψ都是公式，那么(φ&ψ)、(φ|ψ)、(φ>ψ)及(φ<>ψ)也是公式。

表10-2指定了包含这些运算符的公式为真的条件。和以前一样，我们使用φ和ψ 作为表示句子的变量，iff 作为ifand only if(当且仅当)的缩写。

**表10-2** **命题逻辑的布尔运算符的真值条件**

|  |  |  |  |
| --- | --- | --- | --- |
| **布尔运算符** | **真** **值** **条** **件** | | |
| 非(否定的情况) | 在s中-φ为真 | iff | 在s中φ为真 |
| 与(and) | 在s中(φ&ψ)为真 | iff | 在s中φ为真并且在s中w为真 |
| 或(or) | 在s中(φ|ψ)为真 | iff | 在s中φ为真或者在s中w为真 |

续表

|  |  |  |  |
| --- | --- | --- | --- |
| **布尔运算符** | **真** **值** **条** **件** | | |
| 蕴含(if...,then …) | 在s中(φ>ψ)为真 | iff | 在s中φ为假或者在s中y为真 |
| 等价(ifand only i) | 在s中(φ<>ψ)为真 | iff | 在s中φ和ψ同时为真或者同时为假 |

这些规则通常是简单的，即使蕴含的真值条件跟大多英语条件句例子的通常形式不 相一致。公式 (P->Q) 为假只有当P 为真且Q 为假时。如果P 为假(比如说， P 对应 The moon is made of green cheese) 而 Q 为真(比如说，Q 对应Two plus two equals four),那么 P->Q 的结果为真。

NLTK 的 LogicParser()将逻辑表达式分析成表达式的各种子类。

>>>lp =nltk.LogicParser()

>>>lp.parse('-(P&Q)')

<NegatedExpression -(P &Q)>

>>>lp.parse('P&Q')

<AndExpression(P&Q)>

>>>lp.parse('Pl (R ->Q)')

<0rExpression(P l (R ->Q))>

>>>lp.parse('P<->--P')

<IffExpression(P <->--P)>

从计算的角度来看，逻辑是进行推理的重要工具。假设表达： Freedonia is not to the north of Sylvania(Freedonia 不在 Sylvania北边),而理由为：Sylvania is to the north of Freedonia(Sylvania 在 Freedonia 北边)。在这种情况下，你就已经给出了论证。 句子：Sylvania is to the north of Freedonia 是论证的假设，而 Freedonia is not to the north of Sylvania 是结论。从假设一步一步推到结论，被称为推理。通俗地说，就 是我们以在结论前面写 therefore 这样的格式编写一个论证。

(9)Sylvania is to the north of Freedonia.

Therefore,Freedonia is not to the north of Sylvania.

一个论证，如果不存在所有的前提都为真而结论为假的情况，那么它是**有效**的。

现在，(9)的有效性关键取决于短语 to the north of的含义，特别的，它实际上是 一个非对称的关系。

(10)ifx is to the north of y then y is not to the north of x.

然而，在命题逻辑中，我们不能表达这样的规则：我们能用的最小的元素是原子命 题，我们不能“向内看”来谈论个体x 和 y 之间的关系。在这种情况下，最好的办 法是捕捉不对称的特定案例。使用命题符号 SnF 表示 Sylvania is to the north of Freedonia, 用 FnS 表示 Freedonia is to the north of Sylvania 。Freedonia is not to the north of Sylvania,我们写成-FnS。也就是说，我们将 not 当做短语 it is not the case that 的等价，并用一元布尔运算符-来翻译。分别替代(10)中的x 和 y 为 Sylvania 和 Freedonia, 于是我们得到一个蕴含，可以写作如下形式。

(11) SnF ->-FnS

给出一个完整的论证版本会怎样?我们将(9)的第一句话替代为两个命题逻辑公 式：Snf和(11)中的蕴含，它表示(相对贫乏的)背景知识： to the north of 的意 思。[A1,…,An]/C 表示一个从假设[A1,…,An]得出结论C 的论证。这使得论证(9) 可以表示为如下形式。

(12)[SnF,SnF ->-FnS]/-FnS

这是一个有效的论证：如果SnF 和 SnF->-FnS 在情况s 中都为真，那么-FnS 也一 定在s 中为真。相反，如果FnS 为真，这将与我们的理解冲突：在任何可能的情况 下两个事物不能同时在对方的北边。同样的， 链表[SnF,SnF->-FnS,FnS] 是不一致 的——这些句子不能全为真。

参数可以通过使用证明系统来测试“句法有效性”。在之后的10.3节中我们会进行 更多的介绍。NLTK 中的推理模块通过一个引用第三方定理 Prover9 的接口，可以 进行逻辑证明。推理机制的输入首先必须用LogicParser) 分析成逻辑表达式。

>>>lp =nltk.LogicParser()

>>>SnF =lp.parse('SnF')

>>>NotFnS =lp.parse('-FnS')

>>>R =lp.parse('SnF ->-FnS')

>>>prover =nltk.Prover9()

>>>prover.prove(NotFnS,[SnF,R])

True

还有另一种方式可以观察到结论是如何得出的。SnF>-FnS 在语义上等价与-SnF|-FnS, 其中是对应or 的二元运算符。在一般情况下，φ|ψ在条件s 中为真，要么φ在s 中为 真，要么ψ在s 中为真。现在，假设 SnF和-SnF|-FnS 都在s 中为真。如果 SnF 为

真，那么-SnF 不可能也为真。经典逻辑的一个基本假设是： 一个句子在一种情况 下不能同时为真和为假。因此， -FnS 必须为真。

回想一下，接着我们来解释相对于一个模型的一种逻辑语言的语句，它们是这个世 界的一个非常简化的版本。 一个命题逻辑的模型需要为每个可能的公式分配值 True 或 False 。一步步地进行：首先，为每个命题符号分配一个值，然后通过查询确定 布尔运算符的含义(即表10-2)并用它们替代这些公式的组成成分的值，来计算合 成公式的值。估值是从逻辑的基本符号映射到它们的值。下面是一个例子。

>>>val =nltk.Valuation([('P',True),('Q',True),('R',False)])

使用配对的链表初始化估值，每个配对由语义符号和语义值组成。所产生的对象基 本上只是作为一个字典，映射逻辑符号(作为字符串处理)为适当的值。

>>>val['P']

True

正如我们之后将看到的，我们的模型需要稍微更加复杂些，以便处理将在下一节中 讨论的更复杂的逻辑形式；在下面的声明中先忽略参数 dom 和 g。

|  |  |
| --- | --- |
| >>>dom | =set([]) |
| >>>g | =nltk.Assignment(dom) |

现在，让我们用 val 初始化模型m。

>>>m =nltk.Model(dom,val)

每一个模型都有一个 evaluate() 方法，可以用于确定逻辑表达式(如命题逻辑 的公式)的语义值；当然，这些值取决于最初我们分配给命题符号的真值(如 P 、Q 和 R)。

|  |  |  |
| --- | --- | --- |
| >>>print | m.evaluate('(P | &Q)',g) |
| True |  |  |
| >>>print | m.evaluate('-(P | &Q)',g) |
| False |  |  |
| >>>print | m.evaluate('(P | &R)',g) |

False

>>>print m.evaluate('(P|R)',g)

True



轮到你来：

尝试为不同的命题逻辑公式估值。模型是否给出你所期望的值?

到目前为止，我们一直在将英文语句翻译成命题逻辑。由于限于用字母如 P 和 Q 表示原子语句，不能深入其内部结构。从效果上来看，将原子语句分成主语、宾语 和谓词并没有语义上的好处。然而，这似乎是错误的：如果我们想标准化如(9) 这样的论证，就必须要能“看到内部”的基本语句。因此，我们将超越命题逻辑到 一个更有表现力的东西，也就是一阶逻辑。这正是我们下一节要讲的。

**10.3** **一阶逻辑**

本章的剩余部分，我们将通过翻译自然语言表达式为一阶逻辑来表示它们的意思。 并不是所有的自然语言语义都可以用一阶逻辑表示。但它是计算语义的一个不错的 选择，因为它具有足够的表现力来表达语义的很多方面，并且另一方面，有出色现 成的系统可用于开展一阶逻辑自动推理。

下一步我们将描述如何构造一阶逻辑公式，然后在模型中如何评估这样的公式。

**语法**

一阶逻辑保留了所有命题逻辑的布尔运算符，但它增加了一些重要的新机制。首先， 命题被分析成谓词和参数，这接近于自然语言的结构的距离。 一阶逻辑的标准构造 规则识别以下术语：独立变量和独立常量、带不同数量的参数的谓词。例如： Angu s walks 可以被标准化为 walk(angus),Angus sees Bertie 可以被形式化为 see(angus, bertie)。我们称 walk 为一元谓词， see 为**二元谓词。**作为谓词使用的符号不具有内 在的含义。(13)a 和(13)b 之间没有逻辑区别。

(13)a.love(margrietje,brunoke)

b.houden\_van(margrietje,brunoke)

一阶逻辑本身没有什么实质性的关于词汇语义的表示 单个词的意思 虽然 一些词汇语义理论可以用一阶逻辑编码。原子谓词如 see(angus,bertie) 在情况s 中 是真还是假不是一个逻辑的问题，而是依赖于特定的估值，即为常量 see 、angus

和 bertie 选择的值。出于这个原因，这些表达式被称为**非逻辑常量**。相比之下，**逻** **辑常量** (如布尔运算符)在一阶逻辑的每个模型中的解释总是相同的。

我们应该在这里提到：有一个二元谓词具有特殊的地位，它就是等号，如在angus =aj 中的等号。等号被视为一个逻辑常量，因为对于单独的术语t1 和 t2, 公式t1 =t2 为真当且仅当t1 和t2 指同一个实体。

通常需要检查一阶逻辑表达式的语法结构，方法是为表达式指定**类型。**下面是Montague 文法的约定，我们将使用**基本类型**：e 是实体类型，而t是公式类型，即有真值的表达式 类型。给定这两种基本类型，我们可以形成函数表达式的**复杂类型**。也就是说，给定任 何类型σ和t,<o,> 是一个对应于从othings”到 'things’函数的复杂类型。例如：<,T> 是从实体到真值(即一元谓词)的表达式的类型。可以调用LogicParser 进行类型检查。

>>>tlp =nltk.LogicParser(type\_check=True)

>>>parsed =tlp.parse('walk(angus)')

>>>parsed.argument

<ConstantExpression angus>

>>>parsed.argument.type

e

>>>parsed.function

<ConstantExpression walk>

>>>parsed.function.type

<e,?>

为什么会在上例的结尾看到<e,?>呢?虽然类型检查器会尝试推断出尽可能多的类 型，但是在这种情况下，它并没有能够推断出 walk 的类型，所以其结果的类型是 未知的。虽然我们期望 walk 的类型是<e,b, 依据类型检查器可知，但在这个上下 文环境中可能是一些其他类型，如<e,e> 或<e,<e,t>> 。要帮助类型检查器，需要指 定信号，将其作为字典实现与非逻辑常量类型之间的关联。

sed ={':'alk(angus)',sig)

>>>parsed.function.type

<e,t>

二元谓词的类型为<e,<e,t>>。虽然这是先将类型e 的参数组合成一元谓词的类型， 我们可以组合二元谓词的两个参数来表示二元谓词。例如：在 Angus sees Cyril 的 翻译中谓词 see 会与它的参数结合得到结果 see(angus,cyril)。

在一阶逻辑中，谓词的参数也可以是独立变量，如 x 、y 和 z。在 NLTK 中，我们 采用的惯例： e 类型的变量都是小写。独立变量类似于人称代词，如 he 、she 和 it, 为了弄清楚它们的含义需要知道它们使用的上下文。解释(14)中的代词的方法之 一是指向上下文中相关的个体。

**(14)He disappeared.**

另一种方法是为代词he 提供文本中的先行词，例如：指出(15) a 在(14)之前。 在这里，我们说he 与名词短语 Cyril **同指称。**在这个语境中，(14)与(15)b 语 义上是等价的。

(15)a.Cyril is Angus's dog.

b.Cyril disappeared.

相比之下，思考(16)a 中出现的he 。在这种情况下，它受不确定的NP:a dog的 **约束，**这是一个与同指称关系不同的关系。如果替换代词he 为 adog, 结果(16)b 就在语义上就不等效与(16)a。

(16)a.Angus had a dog but he disappeared.

b.Angus had a dog but a dog disappeared.

对应于(17)a, 我们可以构建一个开放公式 (17) b, 变量x 共出现了两次(我们 忽略了时态，以简化论述。)

(17)a.He is a dog and he disappeared.

b.dog(x)&disappear(x)

通过在(17)b 前面指定一个**存在量词**日x( “存在某些x”), 可以绑定这些变量， 如在(18)a 中，它的意思是(18)b, 或者更习惯的写法(18)c。

(18)a. 日x.(dog(x)&disappear(x))

b.At least one entity is a dog and disappeared.

c.A dog disappeared.

下面是(18)a 在 NLTK 中的表示。

(19) exists x.(dog(x)&disappear(x))

**除了存在量词，** **一阶逻辑为我们提供了全称量词** **√X(“对所有x”), 如(20)所示。**

(20)a.3x.(dog(x)→disappear(x))

b.Everything has the property that if it is a dog,it disappears.

c.Every dog disappeared.

下面是(20) a 在NLTK 中的表示。

(21)all x.(dog(x)->disappear(x))

虽然(20)a 是(20)c 的标准一阶逻辑翻译，其真值条件不一定是你所期望的。 公式表示如果某些x 是狗，那么x 就会消失 但它并没有说有狗存在。在没有狗 存在的情况下，(20)a 仍然会为真。(请记住，当P 为假时，(P>Q) 为真。)现 在你可能会说，所有的狗都消失是以狗的存在为前提条件的，逻辑规范化表示的是 完全错误的。但也可能找到其他没有这样前提的例子。例如：我们也许可以解释 Python 表达式 astring.replace(ate','8')是替代 astring中出现的所有'ate'为'8的结果， 即使事实上'ate'可能没有出现(见表3-2)。

我们已经看过了一些量词约束变量的例子。下面的公式中会发生什么?

((exists x.dog(x))->bark(x))

量词existsx 的范围是 dog(x), 所以 bark(x)中x 的出现是不受限制的。因此，它可

以被其他量词约束，例如：下面公式中的all x。

all x.((exists x.dog(x))->bark(x))

在一般情况下，如果变量x 在公式φ中但没有出现在all x或 exists x范围内，那么它 在φ中的出现是**自由的**。相反，如果x 在公式φ中是自由的，那么它在 all x.φ和 exists x.φ限制范围内。如果公式中所有变量都是受限的，那么我们说这个公式是**封闭的。**

在此之前我们提到过，NLTK 中的LogicParser 的 parse) 方法返回 Expression类的对 象。这个类的每个实例expr 都有 free()方法，返回一个在 expr 中自由变量的集合。

>>>lp =nltk.LogicParser()

>>>lp.parse('dog(cyril)').free()

set([])

>>>lp.parse('dog(x)').free()

set([Variable('x')])

>>>lp.parse('own(angus,cyril)').free()

set([])

>>>lp.parse('exists x.dog(x)').free()

set([])

'ists x.own(y,x)').free()

>>>lp.parse('((some x.walk(x))->sing(x))').free()

set([Variable('y')])

**一阶定理证明**

回顾一下我们较早前在(10)中提出的to the north of上的限制。

(22)if x is to the north of y then y is not to the north of x.

我们观察到命题逻辑不足以表示与二元谓词相关的概括，因此，我们不能正确地捕 捉论证： Sylvania is to the north of Freedonia.Therefore,Freedonia is not to the north ofSylvania。

毫无疑问，可以使用一阶逻辑规范化这些规则。

all x. all y.(north\_of(x,y)->-north\_of(y,x))

更妙的是，我们可以进行自动推理来证明论证的有效性。

定理证明在一般情况下是为了确定要证明的公式(**证明目标)**是否可以由一个有限 序列的推理步骤从一个假设的公式列表派生出来。写作 A-g, 其中 A 是一个假设 的列表(可能为空),g 是证明目标。我们将用NLTK 中的定理证明接口Prover9 进 行演示。首先，我们分析所需的证明目标和两个假设。然后我们创建一个Prover9 实例°,并在给定的假设列表条件下调用它的 proveO⑤方法证明目标。

>>>NotFnS =lp.parse('-north\_of(f,s)')、①

>>>SnF=lp.parse('north\_of(s,f)') ②

>>>R=lp.parse('all x.all y.(north\_of(x,y)->-north\_of(y,x))') ③

>>>prover =nltk.Prover9() ④

>>>prover.prove(NotFnS,[SnF,R]) ⑤

True

令人高兴的是，定理证明器证明我们的论证是有效的。相比之下，它同样得出结论： 不能从我们的假设推到出 north\_of(f,s)。

>>>FnS =lp.parse('north\_of(f,s)')

>>>prover.prove(FnS, [SnF,R])

False

**一阶逻辑语言总结**

我们将借此机会重新表述前面的命题逻辑的语法规则，并添加量词的标准规则；所 有这些一起组成了一阶逻辑语法。此外，我们会明确相关表达式的类型。我们将采 取约定： <en,> 是 由n 个类型为e 的参数组成产生类型为t 的表达式的谓词类型。 在这种情况下，我们说n 是谓词的**元数。**

(1)如果P 是类型<en,t> 的谓词，al,…an 是 e 类型的术语，那么P(al,…an) 的

类型是 t。

(2)如果α和β都是e 类型，那么(a=β) 和 (a!=β) 是 t 类型。

(3)如果φ是t 类型，那么是-φ也是t 类型。

(4)如果φ和ψ是t 类型，那么(φ&y) 、(φly) 、(φ>ψ) 和(φ<>ψ)也是t 类型。

(5)如果φ是t 类型， x 是类型为e 的變量，那么 exists x.φ 和 all x.φ是 t 类型。

表10-3总结了logic 模块的新的逻辑常量，以及 Expressions 模块的两个方法。

**表10-3** **一阶逻辑所需的新的邏辑关系和运算符总结**

|  |  |
| --- | --- |
| **示** **例** | **描** **述** |
| = | 等于 |
| != | 不等于 |
| exists | 存在量词 |
| all | 全称量词 |

**真值模型**

我们已经学习过了一阶逻辑句法，在第10.4 节中我们会检查将英语翻译成一阶邏 辑的任务。然而，正如在第10.1 节中提到的，只有赋予一阶逻辑的句子以含义， 才能进一步做下去。换句话说，需要给出一阶逻辑的真值条件的语义。从计算语义 学的角度来看，采用这种方法具有一定的限制。虽然我们要谈的是语句在某种情况

下的真假，但我们只能在电脑中以符号的方式来表示这些情况。尽管有这些限制， 但是通过NLTK 编码模块仍然可以获得较清晰的真值条件语义。

给定一阶逻辑语言L,L 的模型M 是一个<D,Val> 对，其中 D 是一个非空集合，称 为模型的域， Val 是一个函数，称为**估值函数，**它按如下方式从D 中分配值给L 的 表达式。

(1)对于L 中每一个独立常量 c,Val(c) 是D 中的元素。

(2)对于每一个元数n≥0 的谓词符号P,Val(P) 是从Dn 到{True,False} 的函数。(如 果P 的元数为0,则Val(P)是简单的真值，P 被认为是命题符号。)

对于(2),如果P 的元数是2,那么 Val(P)将是从D 元素配对到{True,False} 的函 数。我们将在NLTK 中建立的模型中采取更简便的替代方法，其中 Val(P)是一个配 对的集合S, 定义如下。

**(23)S={s |f(s)=True}**

这样的f 被称为S 的**特征函数** (和扩展阅读中一致)。

NLTK 的语义关系可以用标准的集合论方法表示：作为元组的集合。例如：假设我 们讨论的域包括 Bertie 、Olive 和Cyril, 其中 Bertie 是男孩， Olive 是女孩，而Cyril 是小狗。为了方便记述，我们用 b 、o 和 c 作为模型中相应的标签。我们可以声明 域如下。

>>>dom =set(['b', 'o','c'])

我们使用工具函数 parse\_valuation()将“符号=>值”形式的字符串序列转换成 Valuation 对象。



..oil 

.girlboy ...dog ...walk

...see

()

=>{c)

=>{o,c}

=>{(b,o), (c,b),(o,c)}



>>>l =nltk.parse\_valuation(v)

>>>print val

{'bertie':'b',

'boy':set([('b',)]),

'cyril':'c',

le't[s(')t])(, [('o',)]),

'olive': 'o',

'see':set([('o','c'),('c','b'),('b','o')]),

'walk':set([('c',), ('o',)])}

根据这一估值，则 see 的值是一个元组的集合，包含： Bertie 看到 Olive,Cyril 看 到 Bertie 以及 Olive 看到Cyril。

轮到你来：

仿照图10-2绘制另一个图，描述域 dom 和相应的每个一元谓词的集合。

你可能已经注意到，我们的一元谓词(即 boy 、girl 、dog) 是以单个元组的集合而 不是个体的集合出现的。这样能够方便地统一处理任何元数的关系。

一个形式为P(T1,…Tn)的谓词，其中P 是 n 元的，为真的条件是对应于(T1,…Tn) 值的元组属于P 值的元组集合。

>>>('o','c')in val['see']

True

>>>('b',)in val['boy']

True

**独立变量和赋值**

在模型中，上下文对应的使用是为变量**赋值。**这是一个从独立变量到域中实体的映 射。赋值使用构造函数 Assignment 实现，它也以论述的模型的域为参数。我们无 需实际输入任何绑定，但如果我们要这样做，它们是以类似于之前面看到的估值形 式 (variable,value) 进行绑定。

>>>g=nltk.Assignment(dom, [('x', 'o'), ('y','c')])

>>>g

{'y':'c','x':'o'}

此外，还可以使用print()查看赋值，它使用的是与逻辑学课本中经常出现的符号类 似的符号。

>>>print g

g[c/y][o/x]

现在让我们看看如何为一阶逻辑的原子公式估值。首先，创建一个模型，然后调用 evaluate) 方法计算真值。

>>>m=nltk.Model(dom,val)

>>>m.evaluate('see(olive,y)',g)

True

这里发生了什么?我们正在为一个公式估值，类似与我们前面的例子： see(olive, cyril)。然而，当解释函数遇到变量y 时，没有检查val中的值，而是在变量赋值g 中查询这个变量的值。

>>>g['y']

'c'

由于我们已经知道o 和c 在 see 关系中表示的含义，所以 true值是我们所期望的。 在这种情况下，我们可以说赋值g 满足公式 see(olive,y)。相比之下，下面的公式 相对g 的评估结果为False(检查为什么会是你看到的这样)。

>>>m.evaluate('see(y,x)',g)

False

在我们的方法中(虽然不是标准的一阶逻辑),变量赋值是部分的。例如：g 中除 了x 和y 没有其他变量。方法purge()从一个赋值中清除了所有的绑定。

>>>g·purge()

>>>g

{)

如果我们现在尝试为公式，如 see(olive,y), 相对于g 估值，就像在我们不知道him 是指什么的情况下试图解释一个包含him的句子。因此，在这种情况下，估值函数 未能如愿提供真值。

>>>m.evaluate('see(olive,y)',g)

'Undefined'

由于我们的模型已经包含了解释布尔运算的规则，因此可以组合和评估任意复杂的 公式。

>>>m.evaluate('see(bertie,olive)&boy(bertie)&-walk(bertie)',g) True

确定模型中公式的真假的一般过程称为**模型检查。**

**量化**

现代逻辑的关键特征之一就是变量满足的概念可以用来解释量化的公式。让我们以 (24)作为一个例子。

(24)exists x.(girl(x)&walk(x))

上述公式什么时候为真?想想我们的域，即在 dom 中的所有个体。我们要检查这 些个体中是否有属性是女孩并且正在走路的。换句话说，我们想知道 dom 中是否 有某个u 使 g[u/x]满足开放公式(25)。

(25)girl(x)&walk(x)

思考下面的例子。

>>>m.evaluate('exists x.(girl(x)&walk(x))',g)

True

在这里 evaluate()返回True,因为 dom 中有某些u 通过绑定x 到 u 的赋值满足(25)。 事实上， o 满足下述公式的u。

>>>m.evaluate('girl(x)&walk(x)',g.add('x','o'))

True

NLTK中提供了一个有用的工具： satisfiers()方法。它返回满足开放公式的所有个 体的集合。该方法的参数是一个分析过的公式、 一个变量和一个赋值。下面是一 些相关例子。

>>>fmlal =lp.parse('girl(x) l boy(x)')

>>>m.satisfiers(fmlal,'x',g)

 'o']) =lp.parse('girl(x)->walk(x)')



>>>m.satisfiers(fmla2,'x',g)

set(['c','b','o'])

>>>fmla3 =lp.parse('walk(x)->girl(x)')

>>>m.satisfiers(fmla3,'x',g)

set(['b','o'])

需要思考为什么 fmla2 和 fmla3 是那样的值。>的真值条件是 fimla2 等价于-girl(x) |walk(x), 要么不是女孩要么正在步行的个体满足该条件。因为 b(Bertie)和 c(Cyril) 都不是女孩，根据模型m, 它们都满足整个公式。 o 当然也满足公式，因为o 两项 都满足。现在，因为论述域的每一个成员都满足fmla2, 相应的一般量化公式也为 真。

>>>m.evaluate('all

True

换句话说，只有在对每一个u,φ 关于g 为真。

x.(girl(x)->walk(x))',g)

关于g[u/x]为真的条件下， 一般量化公式Vx.φ 才

**轮到你来：**

先用笔和纸，然后用 m.evaluate(), 尝试弄清楚 all x.(girl(x)&walk(x)) 和 exists x.(boy(x)->walk(x))的真值。确保你能理解它们得到这些值的 原因。

**量词范围歧义**

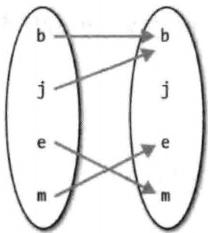
当我们用两个量词规范化地表示一个语句时，会发生什么?

**(26)Everybody admires someone.**

使用一阶逻辑有(至少)两种方法表示(26)。

(27)a.l .**(**)(-

这两种方法我们都能用吗?答案是肯定的，但它们的含义不同。(27)b 在逻辑上 强于(27)a: 它声称只有一个人，也就是 Bruce, 被所有人钦佩。而相对的，(27) a 只要其对于每一个u 我们可以找到u 钦佩的一些人u'; 但每次找到的人u'可能不 同。我们使用术语量词范围来区分(27)a 和(27)b 。首先， √ 的范围比日广，



而在(27)b 中范围顺序颠倒了，所以现在我们有两种方式表示(26)的意思， 它们都相当合理。换句话说，我们称(26)在量词范围方面有歧义，(27)中的 公式提供了一种能够使这两个读法明确的方法。然而，我们不只是关系与(26) 相关联的两个不同的表示；我们也想要详细地展示模型中的两种表述是如何导 致不同的真值条件的。

为了更仔细地检查这种歧义，让我们对估值做如下修正。

>>>v2 =

.bril

...elspeth

...julia

...matthew

...person

...admire



>>>val2



>e

=>m

=>{b,e,j,m}

=>{(j,b),(b,b),(m,e),(e,m),(c,a)}

=nltk.parse\_valuation(v2)

admire 关系可以使用如(28)所示的映射图来进行观察。

(28)

在(28)中，个体x 和y 之间的箭头表示x 钦佩 y。因此，j 和 b 都钦佩 b(布鲁斯 很虚荣),而e 钦佩 m 且 m 钦佩 e。在这个模型中，公式(27) a 为真而(27)b 为假。探索这些结果的方法之一是使用Model 对象的satisfiers) 的方法。

>>>dom2 =val2.domain



>>>m2 =nltk.Model(dom2,val2)

>>>m2.satisfiers(fmla4,'x',g2)

set(['a','c','b','e','j','m'])



**这表明** **fmla4包含域中每一个个体。相反，思考下面的公式** **fmla5; 没有满足** **y**

的值。

5pt.ipsafris's((5y,)'2)

set([])

x.(person(x)->admire(x,y)))')

也就是说，没有大家都钦佩的人。再看看另一个开放的公式 fmla6, 我们可以验证 有一个人，即 Bruce, 它被 Julia 和 Bruce同时钦佩。

>>>fmla6=lp.parse('(person(y)kallx.((x=bruce|x=julia)->admire(x, y)))'>)>>m2.satisfiers(fmla6,'y',g2)

set(['b'])

**轮到你来：**

基于m2 设计一个新的模型，使(27)a 在你的模型中为假；类似的， 再设计一个新的模型使(27)b 为真。

**模型的建立**

我们一直假设已经有了一个模型，并要检查模型中的每一个句子的真值。相比之下， 模型的建立是在给定一些句子集合的条件下，尝试创造一个新的模型。如果成功， 那么集合是一致的，因为有模型的存在作为证据。

通过创建Mace()的一个实例并调用它的 build\_model) 方法来调用 Mace4 模型生成 器，与调用Prover9 定理证明器的方法类似。 一种选择是将候选的句子集合作为假 设，保留目标为未指定。下面的结果显示了[a,cl]和[a,c2]都是一致的链表，因为 Mace 成功地为它们都建立了一个模型，而[cl,c2]不一致。

 p.kp.ppMr-oionsxwalks(x))')

>>>print mb.build\_model(None,[a3,cl])

True

>>>print mb.build\_model(None,[a3,c2])

True

>>>print mb.build\_model(None,[cl,c2])

False

我们也可以使用模型建立器作为定理证明器的辅助。假设正试图证明 A+g, 即 g 是 假设A=[al,a2,…,an] 的逻辑派生。我们可以提供同样的输入给 Mace4, 模型建立 器将尝试找出一个反例，就是要表明 g 不是从 A 中得出的。因此，给定此输入， Mace4 将尝试为假设A 连同g 的否定找到一个模型，即列表A'=[al, a2,…,an,-g] 。 如果g 不能从S 证明出来，那么 Mace4 会返回一个反例，比 Prover9 更快地得出结 论：无法找到所需的证明。相反，如果g 可以从S 证明出来， Mace4 可能要花费很 长时间也不无法成功地找到一个反例模型，最终会放弃。

让我们思考一个具体的方案。假设链表[There is a woman that every man loves, Adam is a man,Eve is a woman]。我们的结论是 Adam loves Eve 。Mace4 能找到使 假设为真而结论为假的模型吗?在下面的代码中，使用MaceCommand) 检查已建 立的模型。

>>>a4=lp.parse('exists y.(woman(y)&all x.(man(x)->love(x,y)))')

>>>a5 =lp.parse('man(adam)')

>>>a6 =lp.parse('woman(eve)')

=lp.parsned),assumptions=[a4,a5,a6])

>>>mc.build\_model()

True

因此答案是肯定的： Mace4 发现了一个反例模型，其中 Adam 爱某个女人而不是 Eve 。但是，让我们细看 Mace4 的模型，将其转换成用来估值的格式。

>>>print mc,valuation

{'C1':'b',

'adam':'a',

'eve':'a',

'love':set([('a','b')]),

'man':set([('a',)]),

'woman':set([('a',),('b',)])}

这个估值的一般形式应是你熟悉的：它包含了一些单独的常量和谓词，每一个都有 适当类型的值。可能令人费解的是 C1 。它是一个 “Skolem 常量”,模型生成器作 为存在量词的表示被引入的。也就是说，当模型生成器遇到a4 里面的 exists y时， 它知道，域中有某个个体b 满足a4 中的开放公式。然而，它不知道 b 是否也是它 的输入中的某个地方的一个独立常量的标志，所以它为b 凭空创造了一个新名字， 即 C1。现在，由于我们的假设中没有关于独立常量 adam 和 eve 的信息，模型生成

器认为没有任何理由将它们当做表示不同的实体，于是它们都映射到 a。此外，我 们并没有指定 man 和 woman 表示不相交的集合，因此，模型生成器让它们相互重 叠。用来解释情境的知识被隐藏了，而模型生成器对此一无所知。因此，添加新的 假设，使man 和 woman 不相交。模型生成器仍然产生了一个反例模型，但这次的 情况更符合我们的要求。

>>>a7 =lp.parse('all x.(man(x)->-woman(x))')

>>>g=lp.parse('love(adam,eve)')

>>>mc

>>>mc.build\_model() True

=nltk.MaceCommand(g,assumptions=[a4,a5,a6,a7])

>>>print mc.valuation

{'Cl':'c',

'adam':'a',

'eve':'b',

'love':set([('a','c')]),

'man':set([('a',)]),

'woman':set([('b',),('c',)])}

结果表明假设中并没有说 Eve 是论述域中唯一的女性，所以反例模型其实是可以接 受的。如果想排除这种可能性，我们必须添加进一步的假设，如 exists y.all x. (woman(x)->(x=y)), 以确保模型中只有一个女性。

**10.4** **英语语句的语义**

**基于特征文法的成分语义学**

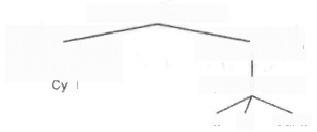
在本章开头，我们简要说明了一种在句法分析的基础上建立语义表示的方法，其中 使用了在第9章开发的语法框架。在这里，我们不构建 SQL 查询，而是建立一种 逻辑形式。设计这样的语法的指导思想之一是**组合原则。**(也称为 Frege 原则，具 体内容见[Partee,1995] 。)

**组合原则：** 整体的含义是部分的含义与它们的句法结合方式的函数。

假设一个复杂的表达式的语义相关部分由句法分析理论给出。在本章中，我们将认

为表达式已经用上下文无关语法分析过。然而，这不是组合原则的内容。

我们现在的目标是以一种可以与分析过程平滑对接的方式整合语义表达的构建。



**S[SEM=<bark(cyril)>]**

VPISEM-cbarksl

*rseu,oN*

IVISEM=<x.bark x >] barks

ril

**(29)类似于我们想建立的分析形式。**

(29)

(29)中，根节点的SEM 值显示了整个句子的语义表示，而较低节点处的 SEM 值 显示句子成分的语义表示。由于SEM 值必须以特殊的方式来对待，它们被括在尖 括号里面以区别于其他特征值。

如何编写能给出这样结果的语法规则呢?方法与本章开始采用的语法 sq10.fcfg 类 似，其中为词汇节点指定语义表示，然后将子节点每个部分的语义表示组合起来。 然而，在目前情况下，我们将使用函数应用而不是字符串连接作为组成的模式。更 具体的说，假设 SEM 节点带有适当值的NP 和 VP 成分。那么,S 的 SEM 值由(30) 这样的规则处理。(SEM 的值是一个变量时，省略尖括号)。

(30)S[SEM=<?vp(?np)>]->NP[SEM=?subj]VP[SEM=?vp]

(30)告诉我们，在给定某个 SEM 值?subj 表示主语 NP并且某个 SEM 值?vp 表示 VP的情况下，父体S 的 SEM 值通过将?vp当做?np 的函数表达式来构建。由此， 我们可以得出结论?vp 必须表示一个在它的域中有?np的表示的函数。(30)是一个 使用组合原则建立语义的好例子。

要完成语法是非常简单的，我们需要的规则全部显示如下。

PP[S?Mv]-i

IV[SEM=<\x.bark(x)>]->'barks'

VP规则说的是父体语义与子核心的语义相同。两个词法规则提供非逻辑常数分别 作为Cyril 和 barks的语义值。另外，barks 入口处有一个额外的符号，我们将进行 简短的解释。

在详细讲述组合语义规则之前，需要为我们的工具箱添加一个新的工具，称为λ 运算。这是一个宝贵的工具，用于在组装英文句子的意思表示时组合一阶逻辑

表达式。

**λ运算**

在1.3节中，指出数学集合符号对于指定文档中所选择词的属性P 有帮助。我们用 (31)来说明，注解为“所有w 的集合，其中w 是 V (词汇表)的元素且 w 有属 性P”。

(31){w l we&P(w)}

事实证明添加一些能达到同样的效果的量到一阶逻辑中是非常有用的。我们添加λ **运算符** (发音为 “lambda”) 。(31) 的λ对应是(32)。(由于不是要讲述集合论，

所以只将V 当作一元谓词。)

(32)λw.(V(w)&P(w))

λ是一个约束运算符，就像一阶逻辑量词。如果有一个开放公式，如(33)a, 那 么可以将变量x 与λ运算符绑定，如(33)b 所示。(33)c 给出了相应的NLTK 中 的表示。

(33)a.(walk(x)&chew\_gum(x))

b.λx.(walk(x)&chew\_gum(x))

c.\x.(walk(x)&chew\_gum(x))

请记住\是 Python 中的特殊字符。我们要么使用转义字符(另一个\),要么使用如 下所示的“原始字符串”(见3.4节)。

=lp..**(**)&chew\_gum(x))')

*>>>e*

<LambdaExpression \x.(walk(x)&chew\_gum(x))>

>>>e.free()

set([])

>>>print lp.parse(r'\x.(walk(x)&chew\_gum(y))')

\x.(walk(x)6 chew\_gum(y))

我们给绑定表达式中的变量结果定义了一个特殊的名字： **λ-抽象。**第一次遇到λ- 抽象时，很难对它们的意思有一个直观的感觉。(33)b 的英语表示是“一个x, 其 中x 步行且x 嚼口香糖”或“具有步行和嚼口香糖的属性”。通常认为λ-抽象可以

很好地表示动词短语(或无主语从句),尤其是当它作为参数出现在它自己的右侧 时。如(34)a 和它的翻译(34)b 中所展示的。

(34)a.To walk and chew gum is hard

b.hard(\x.(walk(x)&chew\_gum(x))

所以一般的描绘是这样的：开放公式φ有自由变量x,x 抽象为属性表达式λx.q 满足φ的x 的属性。下面是如何建立抽象的官方版本。

(35)如果α是 t 类型，x 是 e 类型的变量，那么\x.α 是<e,t> 类型。

(34)b 所示为一个阐述某物属性的例子，也就是说该表达是困难的。但我们通常 提到属性时，是为它们指定个体。而事实上，如果φ是一个开放公式，那么抽象λx.φ 可以被用来作为一元谓词。在(36)中，术语gerald 满足(33)b。

(36)\x.(walk(x)&chew\_gum(x))(gerald)

(36)表示Gerald 具有步行和嚼口香糖的属性，与(37)的意思相同。

(37)(walk(gerald)&chew\_gum(gerald))

我们在这里所做的是将x 从lx.(walk(x)&chew\_ gum(x)) 的开头去掉，并将所有在 (walk(x)&chew\_gum(x) 中出现的x 替换为 gerald。我们利用a[B/x] 将在a 中出现 的所有x 替换为表达式β,如下所示。

(walk(x)&chew\_gum(x))[gerald/x]

与(37)表示的相同。从(36)到(37)的约简是在简化语义表示时非常有用的操 作，本章的其余部分也将大量用到。该操作通常被称为**β-约简**。为了使它在语义上 合理，我们希望λx.a(β) 能保持与α[β/x]有相同的语义值。这的确是真实的，虽然稍 微有些复杂，但我们马上会遇到。为了在 NLTK 中实施表达式的β-约简，我们可 以调用simplifyO方法°。

>>>e=lp.parse(r'\x.(walk(x)&chew\_gum(x))(gerald)')

>>>print e

\x.(walk(x)&chew\_gum(x))(gerald)

>>>print e.simplify() ①

(walk(gerald)&chew\_gum(gerald))

虽然我们迄今只考虑了λ-抽象的主体是某种类型t 的开放公式，但这并不是必要的 限制；主体可以是任何符合文法的表达式。下面是一个有两个λ的例子。

(38)\x.\y.(dog(x)&own(y,x))

正如(33)b 中起到一元谓词的作用，(38)就像一个二元谓词：它可以直接应用 到两个参数°。LogicParser 允许嵌套λ,如lx.ly., 写成缩写形式lx y.。

>>>print lp.parse(r'\x.\y.(dog(x)&own(y,x))(cyril)').simplify() \y.(dog(cyril)6 own(y,cyril))

>>>print lp.parse(r'\x y.(dog(x)&own(y,x))(cyril,angus)').simplify()① (dog(cyril)&own(angus,cyril))

所有的λ-抽象到目前为止只涉及熟悉的一阶变量： x、y 等——类型e 的变量。但 假设要将一个抽象，例如：\x.walk(x), 作为另一个λ-抽象的参数?我们不妨试试 下面的例子。

\y.y(angus)(\x.walk(x))

由于自由变量y 规定是e 类型，因此\y.y(angus)只适用于e 类型的参数，而且lx.walk(x) 是<e,t> 类型的!相反，我们需要允许在更高级的类型的变量抽象。让我们用P 和 Q 作为<e,> 类型的变量，那么我们可以得到抽象，如：\P.P(angus)。由于P 是<e,> 类型，整个抽象是<<e,b,t> 类型。那么\P.P(angus)(lx.walk(x))是合法的，可以通过 β-约简化简为\x.walk(x)(angus), 然后再次化简为 walk(angus)。

在进行β-约简时，对变量有些注意事项。例如，思考(39)a 和(39)b 的λ-术语， 只是用不同自由变量的同一表达式。

(39)a.\y.see(y,x)

b.\y.see(y,z)

现在假设将λ-术语\P.exists x.P(x)应用到以下两项。

(40)a.\P.exists x.P(x)(\y.see(y,x))

b.\P.exists x.P(x)(\y.see(y,z))

较早前提到过这个应用程序的结果应该是语义等价的。但是，如果我们让(39)a 中的自由变量x 落入(40)a 中存在量词的范围内，那么约简之后的结果会发生不同。

(41)a.exists x.see(x,x)

b.exists x.see(x,z)

(41)a 是指有某个x 能看到他/她自己，而(41)b 的意思是有某个x 能看到一个 未指定的个体 z。出了什么错?显然，我们要禁止(41)a 所示的变量“捕获”。

我们使用的变量的特定名字是否受到(40)a 函数表达式的存在量词的约束呢?答 案是否定的。事实上，给定任何绑定变量的表达式(包含 √ 、日或λ),为绑定的 变量选择名字完全是任意的。例如： exists x.P(x)和 exists y.P(y)是等价的；它们 被称为**a- 等价，或字母变体。**重新标记绑定的变量的过程被称为**α-转换。**当在 logic 模块中测试 VariableBinderExpressions 是否相等(即使用==)时，其实是 测试α-等价。

>>>e1 =lp.parse('exists x.P(x)')

>>>print el

2x=.Pe(llpha\_convert(nltk.Variable('z'))

>>>print e2

exists z.P(z)

>>>el ==e2

True

当β-约简在一个应用 f(a)中实施时，检查是否有自由变量在a 中同时也作为 f 的子 术语中绑定的变量出现。如同在上面讨论的例子，假设x 是 a 中的自由变量， f包 括子术语 exists x.P(x)。在这种情况下，将产生一个 exists x.P(x)的字母变体，也就 是说，exists z1.P(z1), 然后再进行约简。这种重新标记由logic 中的β-约简代码自 动进行，可以在下面的例子中看到的结果。

>>>e3 =lp.parse('\P.exists x.P(x)(\y.see(y,x))')

>>>print e3

(\P.exists x.P(x))(\y.see(y,x))

>>>print e3.simplify()

exists z1.see(z1,x)

当你在下面的章节运行这些例子时，你可能会发现返回的逻辑表达式 的变量名不同，例如：你可能在前面公式的 zl 的位置看到z14。这种 标签的变化是无害的 事实上，它仅仅是一个字母变体的例子。

在以上补充说明之后，让我们继续英语语句逻辑形式建立的任务。

**量化的** **NP**

从本节开始，我们简要介绍如何为Cyril barks构建语义表示。你一定会以为这都太 容易了 但实际上还有更多关于建立组合语义的内容。例如：量词?没错，这是 一个至关重要的问题。例如，我们要给出(42)a 的逻辑形式(42)b 。如何才能 实现呢?

(42)a.A dog barks.

b.exists x.(dog(x)&bark(x))

假设建立复杂语义表示的唯一操作是函数应用。那么我们的问题是：如何给量化的 NP:ado g 一个语义表示，使它可以在(42)b 的结果中与bark 结合?作为第一步， 让我们将主语的 SEM 值作为函数表达式，而不是参数(这有时被称为**类型提升**)。 现在，我们寻找使[SEM=<?np(x.bark(x)>]等价于[SEM=<exists x.(dog(x)& bark(x)>] 的?np 的实例。这是否看上去有点让人联想到λ-运算中的β-约简?换句话 说，我们想要一个λ-术语M 来取代?np, 从而将M 应用到x.bark(x) 产生(42)b。

要做到这一点，我们将(42)b 中出现的x.bark(x) 替代为一个谓词变量 P, 并用λ

绑定变量，如(43)所示。

(43)\P.exists x.(dog(x)&P(x))

在(43)中使用不同风格的变量——是'P'而不是x '或'y'——来表明在不同类型的对 象上的抽象 不是单个个体，而是类型为<e,t>的函数表达式。因此，作为一个整 体(43)的类型是<<e,>,> 。 我们将把这个作为NP 的类型。为了进一步说明，普 通量化的NP 看起来像(44)。

(44)\P.all x.(dog(x)->P(x))

现在我们几乎完成了，除了还需要进行进一步的抽象，以及将限定词 a, 即(45), 和dog 的语义组合的过程的应用。

(45)\Q P.exists x.(Q(x)&P(x))

将(45)作为一个函数表达式应用到x.dog(x)中产生(43),如果应用到x.bark(x)则

得到\Pexists 即(42)b。

x.(dog(x)&P(x))(x.bark(x))。最后，进行β-约简产生我们想要的结果，

**及物动词**

我们的下一个挑战是处理含及物动词的句子，如(46)。

(46)Angus chases a dog.

要建立的输出语义是 exists x.(dog(x)&chase(angus,x))。让我们来看看如何能够 利用λ-抽象得到这样的结果。 一个重要制约因素是需要adog 的语义表示与 NP 是否充当句子的主语或宾语独立。换句话说，我们希望得到上述公式作为输出， 同时坚持(43)作为 NP 的语义。第二个制约因素是： VP 应该有统一的解释类 型，不管它们是否只是不及物动词或及物动词加对象组成。更具体地说，我们 始终规定VP 的类型是<e,t> 。鉴于这些制约因素，下面是 chases a dog 的正确的 语义表示。

(47)ly.exists x.(dog(x)&chase(y,x))

将(47)作为y 的属性，使得对于某只狗x, 有 y 追逐x; 或者更通俗地讲，作为 一个追逐狗x 的 y。我们现在的任务是为chases 设计语义表示，它可以与(43)结 合从而派生出(47)。

在(47)上进行β-约简的逆操作，得到(48)。

(48)\P.exists x.(dog(x)&P(x))(z.chase(y,z))

(48)一开始可能会稍微难读，你需要了解的是，它涉及从(43)到zchase(y,z) 应 用量化的NP 表示。(48)通过β-约简与exists x.(dog(x)&chase(y,x)) 等效。

现在，让我们用与NP 类型相同的变量X, 也就是<<e,t>,t>类型，替换(48)中的 函数表达式。

(49)X(z.chase(y,z))

及物动词的表示将必须使用X 类型的参数来产生VP 类型(<e,t>类型)的函数表达 式。我们可以通过在(49)的X 变量和主语变量 y 上进行抽象来确保这一点。因 此，完整的解决方案是给出(50)中所示的 chases的语义表示。

(50)\X y.X(x.chase(y,x))

如果将(50)应用到(43),β-约简后的结果与(47)等效，这是我们一直都想要

**得到的结果。**

nsr(\y.chase(x,y))')

>>>np=lp.parse(r'(\P.exists x.(dog(x)&P(x)))')

>>>vP=nltk.ApplicationExpression(tvp,np)

>>>print vp

(\X x.X(\y.chase(x,y)))(\P.exists x.(dog(x)6P(x)))

>>>print vp.simplify()

\x.exists z2.(dog(z2)6 chase(x,z2))

为了建立一个句子的语义表示，我们也需要组合主语 NP 的语义。如果后者是一 个量化的表达式，例如every girl, 一切都与我们前面讲过的 a dog barks 采用一 样的处理方式；主语转换为被用于VP 的语义表示的函数表达式。然而，我们似 乎因为使用固有的名称而引发了另一个问题。到目前为止，这些被作为单个常 量进行了语义的处理，而且不能作为像(47)那样的表达式的函数进行应用。 因此，我们需要为它们提出一种不同的语义表示。我们在这种情况下所做的是 重新解释适当的名称，使它们也成为如量化的 NP 那样的函数表达式。下面是 Angus 的λ表达式。

(51)\P.P(angus)

(51)表示相应与Angus 为真的所有属性的集合的特征函数。前面简单的提到过， 从独立常量 angus 转换为\P.P(angus)是类型提升的另一个例子，并且这种转换允许 我们用等效的函数应用\PP(angus)(x.walk(x)) 替 换 布 尔 值 的 应 用 ， 如 ： \x.walk(x)(angus)。通过β-约简，两个表达式都约简为 walk(angus)。

语法 sem.fcfg 包含一个用于分析和翻译简单例子的一个小型规则集合。下面是一个 稍微复杂的例子。

r nltlp\_rss/book\_grammars/simple-sem.fcfg',trace=0)

>>>sentence ='Angus gives a bone to every dog'

 =par\_.se(tokens)

>>>for tree in trees:

print tree.node['SEM']

al… z2.(dog(z2)->exists z1.(bone(zl)6 give(angus,z1,z2)))

**(**'''ls x.(P(x)&Q(x))>]an)

NLTK提供了一些实用工具使获得和检查的语义解释变得更容易。函数batch\_interpret) 用于批量解释输入语句的列表。它建立一个字典d, 其中对每个输入的句子 sent,d[sent] 是包含sent 的分析树和语义表示(synrep,semrep)对的链表。该值是一个链表，因为 sent 可能有句法歧义；在下面的例子中，链表中的每个语句只有一个分析树。

**(**[M(ir)'>,EM=<\P.P(irene)>]

(PropN[-LOC,NUM='sg',SEM=<\P.P(irene)>]Irene))

(VP[NUM='sg',SEM=<\x.walk(x)>]

(IV[NUM='sg',SEM=<\x.walk(x)>,TNS='pres']walks)))

(S[SEM=<exists z1.(ankle(zl)&bite(cyril,z1))>]

(NP[-LOC,NUM='sg',SEM=<\P.P(cyril)>]

(PropN[-LOC,NUM='sg',SEM=<\P.P(cyril)>]Cyril))

(VP[NUM='sg',SEM=<\x.exists z1.(ankle(zl)&bite(x,z1))>]

(TV[NUM='sg',SEM=<\X x.X(\y.bite(x,y))>,TNS='pres']bites)

(NP[NUM='sg',SEM=<\Q.exists x.(ankle(x)&Q(x))>]

(N[NUM='sg',SEM=<\x.ankle(x)>]ankle)))))

现在我们已经懂得了英文句子如何转换成逻辑形式，之前学习了在模型中如何检查 逻辑形式的真假。把这两个映射放在一起，可以检查给定的模型中的英语语句的真 值。让我们看看前面定义的模型m 。工具batch\_evaluate()类似于batch\_interpret(), 除此之外我们还需要传递一个模型和一个变量赋值作为参数。输出是三元组(synrep, semrep,value), 其中 synrep,semrep 和以前一样，value 是真值。为简单起见，下面 的例子中只处理了一个简单的句子。

>>>y=

. bertie =>b

...olive =>o

.cyril =>c

.birl (}

alk =co,c}

...see =>{(b,o),(c,b),(o,c)

>>>vl =nltk.parse\_valuation(v)

>>>g =nltk.Assignment(val.domain)

>>>m=nltk.Model(val.domain,val)

>>>sent ='Cyril sees every boy'

>>>grammar\_file='grammars/book\_grammars/simple-sem.fcfg'

>>>results =nltk.batch\_evaluate([sent],grammar\_file,m,g)[0] >>>for(syntree,semrel,value)in results:

... print value

... print semrep

exists z3.(ankle(z3)&bite(cyril,z3))

True

**重述量词歧义**

上述方法的一个重要的限制是它们没有处理范围歧义。我们的翻译方法是句法驱动 的，在认为语义表示与句法分析紧密耦合的前提下，因此语义中量词的范围也反映 句法分析树中相应的NP 的相对范围。因此，像前面(26)这样的句子总是会被翻 译为(53) a 而不是(53)b。

(52)Every girl chases a dog.

(53)a.all x.(girl(x)->exists y.(dog(y)&chase(x,y)))

b.exists y.(dog(y)&all x.(girl(x)->chase(x,y)))

目前有许多处理范围歧义的方法，我们只学习最简单的一种。首先，简要地考虑具 有范围的公式的结构。图10-3描绘了(52)的这两种不同的读法。

|  |  |
| --- | --- |
| Vx.(girl(x)→φ)  3y.(dog(y)A ψ)  chase(x,y) | 3y.(dog(y)A ψ) |

**图10-3** **量词范围**

先考虑左侧的结构。在顶端，有对应于every girl 的量词。φ可以被看作是量词范围内 的一个占位符。向下，是可以插入相对于a dog的量词作为φ的实例。这样就产生了 一种新的占位符ψ用来表示adog 的范围，通过这一点，我们可以插入语义的“核心”, 即对应于xchases y的开放语句。右侧的结构是相同的，除了两个量词交换了顺序。

在被称为 **Cooper 存储**的方法中，语义表示不再是一阶逻辑表达式，而是由“核心” 语义表示与绑定操作符链表组成的配对。就目前而言，可以认为**绑定操作符**是如

(44)或(45)那样的量化NP 的语义表示。沿图10-3所示的线向下，假设已经构 建了Cooper存储风格的句子(52)的语义表示，并将开放公式 chase(x,y)作为核心。 给定有关(52)中两个NP 的绑定操作符的列表，将绑定操作符从列表挑出来，并 与核心结合。

\P.exists y.(dog(y)6 P(y))(\z2.chase(z1,z2))

然后，将列表中的另一个绑定操作符应用到结果中。

\P.all x.(girl(x)->P(x))(\z1.exists x.(dog(x)&chase(z1,x)))

当列表为空时，可得到句子传统的逻辑形式。以这种方式结合绑定操作符与核 心的操作被称为 **S-检索**。如果允许绑定操作符的每个可能的顺序，例如：通过 将列表进行全排列(见4.5 节),那么将能够产生量词的每一个可能范围的 排序。

接下来要解决的问题是如何建立一个“核心+存储表示”的组合。如前所述，语法 中每个短语和词法规则将有一个 SEM 特征，但现在将还有嵌入特征 CORE 和 STORE。通过一个简单的例子： Cyril smiles, 来说明这里的机制。下面是动词 smiles 的词法规则(取自语法 storage.fcfg)。

IV[SEM=[CORE=<\x.smile(x)>,STORE=(/)]]->'smiles'

固有名称 Cyril的规则更为复杂。

NP[SEM=[CORE=<@x>,STORE=(<bo(\P.P(cyril),@x)>)]]->'Cyril'

谓词bo 有两个子部分： 一个适当名称的标准(类型提升)表示，以及表达式@x, 被称为绑定操作符的地址。(之后我们会简要解释为什么需要地址变量)@x 是原 变量，也就是范围在逻辑独立变量之上的变量，同时，它也给出了核心的值。VP 的规则只是向上渗透IV 的语义，并关注由S 规则完成的工作。

VP[SEM=?s]->IV[SEM=?s]

S[SEM=[CORE=<?vp(?subj)>,STORE=(?b1+?b2)]]->

NP[SEM=[CORE=?subj,STORE=?b1]]VP[SEM=[core=?vp,store=?b2]]

S 节点的核心值是应用VP 的核心值(即\x.smile(x)) 到主语 NP的值的结果。后者

不会是@x,而是一个@x的实例，也就是z3。β-约简后，<?vp(?subj>将与<smile(z3> 统一。当@x 实例化为分析过程的一部分时，它将会同样地被实例化。特别是主语 NP 的 STORE 中出现的@x 也将被映射到 z3, 并产生元素 bo(P.P(cyril),z3)。这些 步骤可以在下面的分析树中看到。

(S[SEM=[CORE=<smile(z3)>,STORE=(bo(\P.P(cyril),z3))]]

(NP[SEM=[CORE=<z3>,STORE=(bo(\P.P(cyril),z3))]]Cyril)

(VP[SEM=[CORE=<\x.smile(x)>,STORE=()]]

(IV[SEM=[CORE=<\x.smile(x)>,STORE=()]] smiles)))

让我们回到更复杂的例子(52),看看在用文法storage.fcfg 分析后，存储风格的 SEM

值是什么。

CORE =<chase(z1,z2)>

STORE=(bo(\P.all x.(girl(x)->P(x)),z1),bo(\P.exists x.(dog(x)6

P(x)),z2))

现在，应该很清楚为什么地址变量是绑定操作符的一个重要组成部分。回想在 S- 检索过程中，我们将绑定操作符从STORE 列表移出，先后将它们应用到CORE 。 假设从与chase(z1,z2)结合的 bo(Pall x.(girl(x)>P(x)),z1)开始。绑定操作符的量词 部分是\Pall x.(girl(x)->P(x)),为了与 chase(z1,z2)结合，后者需要先被转换成λ-抽 象。我们怎么知道在哪些变量上进行抽象? z1 的地址会告诉我们，即 every girl 都 有被追赶而不是去追赶的角色。

模块 nltk.sem.cooper\_storage 将存储形式的语义表示转换成标准逻辑形式。首先， 构造一个 CooperStore 实例，并检查它的 STORE 和CORE。

>>>from nltk.sem import cooper\_storage as cs

>>>sentence ='every girl chases a dog'

>>>trees =cs.parse\_with\_bindops(sentence,grammar='grammars/book\_grammars/ storage.fcfg')

>>>semrep =trees[0].node['SEM']

>>>cs\_semrep =cs.Cooperstore(semrep)

>>>print cs\_semrep.core

chase(z1,z2)

>>>for

bo

in bo

print

cs\_semrep.store:

b\* o· (\P.all x.(girl(x)->P(x)),z1)

bo(\P.exists x.(dog(x)&P(x)),z2)

**然后，调用s\_retrieve)检查读法。**

>>>cs\_semrep.s\_retrieve(trace=True)

Permutation 1

(\P.all x.(girl(x)->P(x)))(\z1.chase(z1,z2))

(\P.exists x.(dog(x)&P(x)))(\z2.all x.(girl(x)->chase(x,z2))) Permutation 2

(\P.exists x.(dog(x)&P(x)))(\z2.chase(z1,z2))

(\P.all x.(girl(x)->P(x)))(\z1.exists x.(dog(x)&chase(z1,x)))

>>>for reading in cs\_semrep.readings:

exists iogll z3.(girl(z3)->chase(z3,x)))

all x.(girl(x)->exists z4.(dog(z4)&chase(x,z4)))

**10.5** **段落语义层**

段落是语句的序列。很多时候，段落中句子的解释依赖它前面的句子。明显的例子 来自照应代词，如 he、she 和 it。给定段落如：Angus used to have a dog.But he recently disappeared.你可能会把 he 理解为指的是Angus的狗。然而，在 Angus used to have a dog.He took him for walks in New Town.中，你更可能把 he 理解为指的是 Angus 自己。

**段落表示理论**

一阶逻辑中的量化标准方法仅限于单个句子。然而，似乎存在量词的范围可以扩大 到两个或两个以上的句子的情况。我们之前看到过一个，下面是第二个例子和它的 翻译。

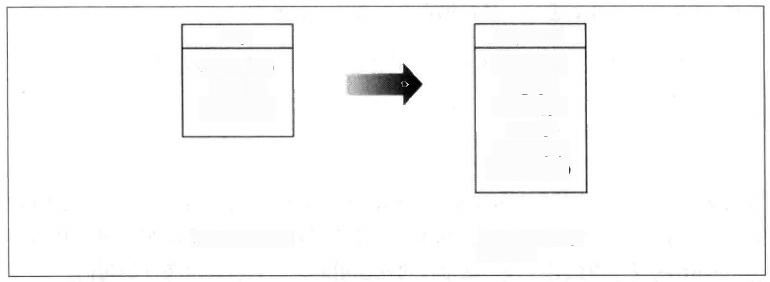
(54)a.Angus owns a dog.It bit Irene.

b.3x.(dog(x)&own(Angus,x)&bite(x,Irene))

也就是说， NP:a dog 的作用是绑定第二句话中的 it 的量词。段落表示理论 (Discourse Representation Theory,DRT) 的目标是提供处理这个和其他段落特征 的语义现象的方法。**段落表示结构** (discourse representation structure,DRS) 根据 一个**段落指称**的列表和一个条件列表表示段落的意思。段落指称是段落中正在讨 论的事情，它对应一阶逻辑的单个变量。**DRS** **条件**应用于段落指称，对应于一阶 逻辑的原子开放公式。图10-4演示了(54)a 中第一句话的DRS 如何增强为两个 句子的 DRS。

在处理(54)a 第二句时，以图10-4左侧呈现的上下文来进行解释。代词 it 触发另 外一个新的段落指称，也就是u, 我们需要为它找一个先行词 也就是，我们要 找出 it 指的是什么。在 DRT 中，为一个参照代词寻找先行词的任务包括将它连接 到已经在当前 DRS 中的话题指称， y 是显而易见的选择。(我们会在下面的内容中 继续讲述更多关于指代分解的内容。)处理步骤产生了一个新的条件 u=y。第二句 增加的其余内容也与第一个的内容合并，如图10-4右侧所示。

图10-4说明了 DRS 如何表示多个句子。图10-4中的只是一个两句话的段落，但原 则上一个 DRS 可以对应整个文本的解释。我们可以查询图10-4中 DRS 右侧的真 值。通俗地讲，如果在情况s 中有实体 a 、c 和 i, 对应 DRS 中的段落指称使s 中 所有条件都为真，那么它为真；也就是说， a是Angus,c 是 adog,a 拥有c,i 是 Irene,c 咬 了i。



x y u z

Angus(x)

dog(y)

own(x,y)

u =y

Irene(z)

bite(u,z)

*Angus ocons a dog Angus owns a dog.*

It bit Irene

**图10-4** **建立一个** **DRS: 左侧的** **DRS 表示段落中第一句话的处理结果，而右侧的** **DRS 显示处**

x y

Angus(x) dog(y) own(x,y)

**理并整合第二句上下文的效果**

为了处理 DRS 计算，我们需要将其转换成线性格式。下面是一个相关例子，其中

DRS 是由一个段落指称列表和一个 DRS 条件列表组成的配对。

([x,y],[angus(x),dog(y),own(x,y)])

在NLTK 建立DRS 对象最简单的方法是通过解析字符串表示°。

>>>dp =nltk.DrtParser()

>>>drs1 =dp.parse('([x,y],[angus(x),dog(y),own(x,y)1)')①

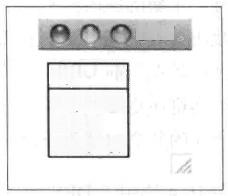
>>>print drs1

([x,y],[angus(x),dog(y),own(x,y)])

我们可以使用drawO方法“将结果可视化，如图10-5所示。

>>>drs1.draw()①

我们讨论图10-4中 DRS 的真值条件时，假设最上面的 段落指称被解释为存在量词，而条件解释为联合。事实 上，每一个DRS 都可以转化为一阶逻辑公式， fol) 方法 就可以实现这种转换。

DRT

|x y

angus(x)

dog(y)

own(x,y)

>>>print drs1.fol()

exists x y.((angus(x)&dog(y))&own(x,y))

**图10-5** **DRS** **截图**

作为一阶逻辑表达式功能补充， DRT 表达式有 DRS-连接运算符，用“+”符号表 示。两个 DRS 的连接是指一个单独的DRS 包含合并的段落指称和来自两个论证的 条件。DRS-连接自动进行α-转换绑定变量避免名称冲突。

>>>drs2

>>>print drs2

=dp.parse('([x],[walk(x)])+([y],[run(y)])')

(([x],[walk(x)])+([y],[run(y)]))

>>>print drs2.simplify()

([x,y],[walk(x),run(y)])

虽然迄今为止见到的所有条件都是原子的，但一个 DRS 可以内嵌入另一个DRS, 这是一般量词被处理的方式。在 drs3 中，没有顶层的段落指称，唯一的条件由两 个子 DRS 组成，由蕴含连接。这里，再次使用 folO来获得真值条件的句柄。

>>>drs3 =dp.parse('([],[(([x],[dog(x)])->([y],[ankle(y),bite(x, y)]))])')

>>>print drs3.fol()

all x.(dog(x)->exists y.(ankle(y)&bite(x,y)))

我们较早前指出 DRT 旨在通过链接照应代词和现有的段落指称来解释照应代词。 DRT 设置约束条件使段落指称可以像先行词那样“可访问”,但并不打算解释如何 从候选集合中选出特定的先行词。模块nltk.sem.drt\_resolve\_anaphora采用了类似的 保守策略：如果DRS 包含PRO(x) 形式的条件，方法resolve\_anaphora() 将其替换为 x=[…]形式的条件，其中[…]是一个可能先行词的列表。

>>>drs4 =dp.parse('([x,y],[angus(x),dog(y),own(x,y)])')

>>>drs5 =dp.parse('([u,z],[PR0(u),irene(z),bite(u,z)])')

t 4sidprlsiy()

([x,y,u,z],[angus(x),dog(y),own(x,y),PR0(u),irene(z),bite(u,z)])

>>>print drs6.simplify().resolve\_anaphora()

([x,y,u,z],[angus(x),dog(y),own(x,y),(u=[x,y,z]),irene(z),bite(u,z)])

由于指代分解算法已分离到它自己的模块，这有利于在替代程序中进行交换，使对 先行词的猜测更加准确。

我们对 DRS 的处理与处理λ-抽象的现有机制是完全兼容的，因此可以直接基于 DRT 而不是一阶逻辑建立组合语义表示。这种技术将在下面的不确定性规则(是 文法 drt.fcfg的一部分)中说明。为便于比较，我们已经从 simplesem.fcfg 增加了不 确定性的平行规则。

Det[NUM=sg,SEM=<\P Q.([x],[])+P(x)+Q(x)>]->'a'

Det[NUM=sg,SEM=<\PQ.exists x.(P(x)&Q(x))>]->'a'

为了更好地了解 DRT规则是如何运作的，请看下面的NP:adog 的子树。

(NP[NUM='sg',SEM=<\Q.(([x],[dog(x)])+Q(x))>] (Det[NUM'sg',SEM=<\P

(Nom[NUM='sg',SEM=<\x.([],[dog(x)])>]

(N[NUM='sg',SEM=<\x.([],[dog(x)])>]dog)))))

Q.((([x],[])+P(x))+Q(x))>]a)

不确定性的λ-抽象作为一个函数表达式被应用到x.(D,[dog(x)]), 得到Q.(([x],D)+ ([],[dog(x)])+Q(x))。简化后，我们得到Q.(([x],[dog(x)])+Q(x))整体作为NP 的表示。

为了解析语法 drt.fcfg, 我们在load\_parser) 调用中指定特征结构中的 SEM 值，用 DrtParser 解析替代默认的LogicParser。

>>>from nltk import load\_parser

>>>parser =load\_parser('grammars/book\_grammars/drt.fcfg',

logn\_parse('Angus owns a dog'.split())

>>>print trees[0].node['sem'].simplify()

([x,z2],[Angus(x),dog(z2),own(x,z2)])

**段落处理**

解释一句话时会利用丰富的上下文背景知识， 一部分取决于前面的内容， 一部分 取决于背景假设。DRT 提供了如何将句子的含义集成到前面段落表示中的理论，

s1-r0:all x.(student(x)->person(x))

但是在前面的讨论中明显缺少以下两个部分。首先， 一直没有尝试纳入任何一种 推理；第二，我们只处理了单个的句子。这些遗漏由模块 nltk.inference.discourse 纠正。

段落是一个句子的序列sl,…,sn, 段落线是读法的序列sl-ri,…,sn-rj,每个序列 对应段落中的一个句子。该模块按增量处理句子，当有歧义时保持追踪所有可能的 线。为简单起见，下面的例子中忽略了范围歧义。

>>>dt =nltk.DiscourseTester(['A student dances','Every student is a

person'])

>>>dt.readings()

s0 readings:s0-r0:exists x.(student(x)&dance(x))

s1 readings:s1-r0:all x.(student(x)->person(x))

一个新句子添加到当前的段落时，设置参数 consistchk=True 会通过每条线，即每 个可接受的读法序列的检查模块，来检查一致性。在这种情况下，用户可以选择收 回有问题的句子。

>>>dt.add\_sentence('No person dances',consistchk=True)

Inconsistent discourse d0 ['s0-r0', 'sl-r0','s2-r0']:

s0-r0:exists x.(student(x) &dance(x))

s2-r0:-exists x.(person(x)&dance(x))

>>>dt.retract\_sentence('No person dances',verbose=True)

Current sentences are

s0:A student dances

s1:Every student is a person

以类似的方式，我们使用 informchk=True 检查新句子φ是否对当前的段落来说具 有信息量。定理证明器将段落线中现有的句子当做假设，尝试证明φ;如果没有发 现可行的证明，那么说明它是有信息量的。

>>>dt.add\_sentence('A person dances', informchk=True)

Sentence 'Aperson dances'under reading'existsx.(person(x)6dance(x))': Not informative relative to thread 'd0'

也可以将另一种假设作为背景知识，并使用这些筛选出不一致的读法；详情请参阅 <http://www.nltk.org/howto> 上的段落 HOWTO。

discourse 模块可适应语义歧义，筛选出不可接受的读法。下面的例子调用 Glue 语

义和DRT。由于Glue 语义模块被配置为使用覆盖面广的 Malt 依存关系分析器，输

**入** **(Every dog chases a boy.He runs.) 需要分词和标注。**

>>>from nltk.tag import RegexpTagger

>>>tagger =RegexpTagger(

.. · [[('^(chaseslruns)$', 'VB'),

· ('^(a)$','ex\_quant'),

· ('^(every)s','univ\_quant'),

· ('^(doglboy)$', 'NN'),

\* ('^(He)S','PRP')

>rc =nltk.DrtGlueReadingCommand(depparser=nltk.MaltParser(tagger=

tagdt=nltk.DiscourseTester(['Every dog chases a boy','He runs'],rc) >>>dt.readings()

s0 readings:

s0-r0:([],[(([x],[dog(x)])->([z3],[boy(z3),chases(x,z3)]))])

s0-r1:([z4],[boy(z4),(([x],[dog(x)])->([],[chases(x,z4)]))])

s1 readings:

s1-r0: ([x],[PRO(x),runs(x)])

段落的第一句有两种可能的读法，取决于量词的作用域。第二句的唯一读法通过条 件 PRO(x)表示代词 He。现在让我们看看段落线的结果。

>>>dt.readings(show\_thread\_readings=True)

d0:['s0-r0','s1-rO']:INVALID:AnaphoraResolutionException

d1:['s0-r1','s1-r0']:([z6,z10],[boy(z6),(([x],[dog(x)])->

([],[chases(x,z6)])), (z10 =z6),runs(z10)])

当我们检查段落线 d0和 d1 时，我们看到读法sO-r0, 其中 every dog 超出了a boy 的范围，被认为是不可接受的，因为第二句的代词得不到解释。相比之下，段落线 d1中的代词(重写为z10) 通过等式(z10=z6) 绑定。

不可接受的读法可以通过传递参数 filter=True 过滤掉。

>>>dt.readings(show\_thread\_readings=True,filter=True)

d1:['s0-r1','s1-r0']:([z12,z15],[boy(z12),(([x],[dog(x)])->

([],[chases(x,z12)])), (z17 =z15),runs(z15)])

虽然本节关于段落语义的内容是极其有限的，但应该能让你对多于单个句子后产生

的语义处理问题，以及用来解决它们的技术有所了解。

**10.6** **小结**

● 一阶逻辑是一种适合在计算环境中表示自然语言含义的语言，因为它很灵活， 足以表示自然语言含义的很多有用的方面，并且有一些使用一阶逻辑推理的高 效的定理证明器。(同时，自然语言语义中也有其他各种各样的现象，需要更 强大的逻辑机制。)

● 在将自然语言句子翻译成一阶逻辑的同时，我们可以通过检查一阶公式模型表 述这些句子的真值条件。

· 为了表示成分组合的含义，我们为一阶逻辑补充了λ-运算。

● λ-运算中的β-约简在语义上与函数传递参数对应。句法上，它包括将被函数表 达式中的λ绑定的变量替换为函数应用中表达式提供的参数。

● 构建模型的关键部分在于建立估值，为非逻辑常量分配解释。这些非逻辑常量 被解释为n 元谓词或独立常量。

● 开放表达式是包含一个或多个自变量的表达式。开放表达式只在它的自变量被 赋值时才能获得解释。

● 量词的解释是对于具有变量x 的公式φ[x],构建个体的集合，赋值g 分配它们 作为x 的值使φ[x]为真。然后量词对这个集合加以约束。

· 封闭的表达式是没有自由变量的表达式。也就是，变量都是被绑定的。封闭的 表达式的真假取决于所有变量的赋值。

● 如果两个公式只是由绑定操作符(即λ或量词)绑定的变量的标签不同，那么 它们是α-等价。重新标记公式中的绑定变量的过程被称为α-转换。

● 给定有两个嵌套量词Q1 和 Q2 的公式，最外层的量词Q1 有较广的范围(或范 围超出 Q2) 。英语句子往往由于它们包含的量词的范围而产生歧义。

● 在基于特征的语法中英语句子可以通过将 SEM 作为特征与语义表达关联。 一 个复杂表达式的 SEM 值通常包括成分表达式的 SEM 值的函数应用。

**10.7** **深入阅读**

关于本章的进一步材料及如何安装Prover9定理证明器和Mace4模型生成器的内容 请查阅 <http://www.nltk.org/>。这两个推论工具一般情况参阅 (McCune,2008)。

更多关于用 NLTK 进行语义分析的例子，请参阅 <htp://www.nltk.org/howto> 上的语 义和逻辑 HOWTO。请注意，范围歧义还有其他两种解决方法，即(Blackburn &Bos, 2005)描述的Hole 语义和 (Dalrymple et al,1999) 描述的 Glue 语义。

自然语言语义中还有很多现象没有在本章中涉及，主要有：

(1)事件、时态和动词的体；

(2)语义角色；

(3)广义量词，如 most;

(4)内涵结构，例如像may 和 believe 这样的动词。

(1)和(2)可以使用一阶逻辑处理，(3)和(4)需要不同的逻辑。下面的读物中 很多都介绍了这些问题。

建立自然语言前端数据库方面的结果和技术的综合概述可以在 (Androutsopoulos, Ritchie &Thanisch,1995) 中找到。

任何一本现代逻辑的入门书都将提出命题和一阶逻辑的有关内容。强烈推荐 (Hodges,1977), 书中有很多有关自然语言的有趣且有洞察力的文字和插图。

要说范围广泛，请参阅两本关于逻辑的教科书 (Gamut,1991a,1991b), 以及有关 自然语言形式语义的当代材料，例如：Montague 文法和内涵逻辑。(Kamp&Reyle, 1993)提出了段落表示理论的权威报告，其中涵盖大量且有趣的自然语言片段，包 括时态、动词的体和形态。另一个对自然语言结构语义的全面研究是 (Carpenter, 1997)。

还有许多作品介绍语言学理论框架内的逻辑语义。(Chierchia &McConnell-Ginet, 1990)与句法相对无关，而 (Heim&Kratzer,1998) 和 (Larson&Segal,1995) 都

很明确地倾向于将语义真值条件整合到乔姆斯基框架中。

(Blackburn &Bos,2005) 是致力于计算语义的第一本教科书，为该领域提供了极好 的介绍。它扩展了许多本章涵盖的主题，包括量词范围歧义的未指定、 一阶逻辑推 理及段落处理。

要获得更先进的当代语义方法的概述，包括处理时态和广义量词，尝试查阅(Lappin, 1996)或 (van Benthem &ter Meulen,1997)。

**10.8** **练习**

1.O 将下列句子翻译成命题逻辑，并用LogicParser 验证结果。说明翻译后的命题 变量是如何对应英语表达的。

a.If Angus sings,it is not the case that Bertie sulks.

b.Cyril runs and barks.

c.It will snow if it doesn't rain.

d.It's not the case that Irene will be happy if Olive or Tofu comes.

e.Pat didn't cough or sneeze.

f.If you don't come ifI call,I won't come if you call.

2.O 将下面的句子翻译为一阶逻辑的谓词参数公式。

a.Angus likes Cyril and Irene hates Cyril.

b.Tofu is taller than Bertie.

c.Bruce loves himself and Pat does too.

d.Cyril saw Bertie,but Angus didn't.

e.Cyril is a four-legged friend.

**f.Tofu and Olive are near each other.**

3.O 将下列句子翻译为一阶逻辑的量化公式。

a.Angus likes someone and someone likes Julia.

b.Angus loves a dog who loves him.

c.Nobody smiles at Pat.

d.Somebody coughs and sneezes.

e.Nobody coughed or sneezed.

f.Bruce loves somebody other than Bruce.

g.Nobody other than Matthew loves Pat.

h.Cyril likes everyone except for Irene.

i.Exactly one person is asleep.

4.O 使用λ-抽象和一阶逻辑的量化公式，翻译下列动词短语。

a.feed Cyril and give a capuccino to Angus

b.be given'War and Peace'by Pat

c.be loved by everyone

**d.be loved or detested by everyone**

**e.be loved by everyone and detested by no-one**

**5.O** **思考下面的语句。**

>>>lp =nltk.LogicParser()

>>>e2 =lp.parse('pat')

>>>e3 =nltk.ApplicationExpression(el,e2)

>>>print e3.simplify()

exists y.love(pat,y)

显然缺少了el 值的声明，但为了ApplicationExpression(el,e2)能被β-转换为 exists y.love(pat,y),el 必须是以 pat为参数的λ-抽象。你的任务是构建这样的一个抽象， 将其绑定到el, 且上面的语句都是满足的(直到字母方差)。此外，提供e3.simplifyO 的非正式的英文翻译。

现在根据更多 e3.simplifyO的例子(如下所示)完成同样的任务。

>>>print e3.simplify()

exists y.(love(pat,y) l love(y,pat))

>>>print e3.simplify()

exists y.(love(pat,y) l love(y,pat))

>>>print walk(fido)

e3.simplify()

**6.O** **如前面的练习中那样，找到λ-抽象el, 产生如下面显示的等效结果。**

 =trnExpression(el,e2)

>>>print e3.simplify()

\x.all y.(dog(y)->chase(x,pat))

>>>e2 =lp.parse('chase')

>>>e3 =nltk.ApplicationExpression(el,e2)

\x.exists y.(dog(y)&chase(pat,x))

>>>print e3.simplify()

>>>e2 =lp.parse('give')

>>>e3=nltk.ApplicationExpression(el,e2)

>>>print e3.simplify()

\x0 x1.exists y.(present(y)6 give(x1,y,x0))

7.O 如前面的练习中那样，找到λ-抽象e1, 产生如下面显示的等效结果。

 =tt)ionExpression(el,e2)

>>>print e3.simplify()

exists y.(dog(x)&bark(x))

>>>e2 =lp.parse('bark')

>>>e3 =nltk.ApplicationExpression(el,e2)

>>>print e3.simplify()

bark(fido)

>>>e2 =lp.parse('\\P.all x.(dog(x)->P(x))')

>>>e3 =nltk.ApplicationExpression(el,e2)

>>>print e3.simplify()

all x.(dog(x)->bark(x))

8. ①开发一种方法，将英语句子翻译为带有二元广义量词的公式。在此方法中， 给定广义量词Q, 量化公式的形式为Q(A,B), 其 中A 和 B 是<e,> 类型的表达式。 那么,例如： all(A,B) 为真当且仅当A 是 B 一个子集。

9. ①扩展前面练习中的方法，使量词如 most 和 exactly three 的真值条件可以在模 型中计算。

10 . ①修改 sem.evaluate 代码，当表达式不在模型的估值函数域中时，能提供有用 的错误消息。

11 . ● 从儿童读物中选择3个或4 个连续的句子。 一个可行的例子来源是 nltk. corpus.gutenberg:bryantstories.txt,burgess-busterbrown.txt 和 edgeworth-parents.txt 中的 故事集。开发一种语法，能将你的句子翻译成一阶逻辑，建立一个模型，使它能检查 这些翻译的真假。

12. ● 完成前面的练习，但使用DRT 作为意思表示。

13 . ● 以 (Warren&Per eira,1982) 为出发点，开发一种技术，将自然语言查询转 换为一种可以更加有效地在模型中评估的形式。例如：给定一个(P(x)&Q(x)) 形式 的查询，将它转换为(Q(x)&P(x)), 如 果Q 的范围比P 小的话。



**第11章**

**语言数据管理**

已标注的语言数据的结构化集合在 NLP 的大部分领域都是至关重要的；然而，在 使用它们的过程中我们仍面临着许多问题。本章的目的是要回答下列问题。

(1)如何设计一种新的语言资源，并确保它的覆盖面、平衡及文档支持广泛的

用途?

(2)现有数据对某些分析工具格式不兼容，如何才能将其转换成合适的格式?

(3)有什么好的方法来记录已经创建的资源，从而使其他人可以很容易地找 到它?

在此过程中，我们将研究当前设计、创建一个语料库的典型工作流程，及语料库的 生命周期。与在其他章节中一样，会有很多语言数据管理的实际例子，包括在语言 学现场教学课程、实验室的工作和网络中收集的数据。

**11.1** **语料库结构：案例研究**

TIMIT 语料库是第一个广泛发布的已标注语音数据库，它有一个特别清晰的组织结构。 TIMIT 由克萨斯仪器公司和麻省理工学院共同开发，也由此得名。主要用来为获取声 学-语音知识提供数据，并支持自动语音识别系统的开发和评估。

**TIMIT** **的结构**

与布朗语料库显示文章风格和来源的平衡选集一样，TIMIT 包括方言、说话者和材

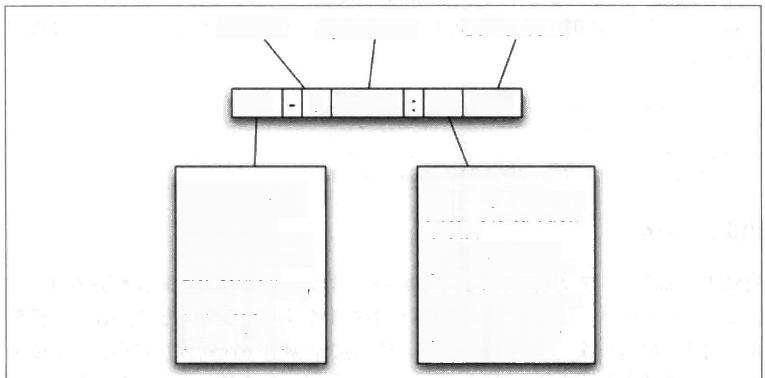
料的平衡选集。用8个方言，具有一定年龄和教育背景的50个男性和女性，每人 读10个句子。所有说话者都读的两句话将体现出方言变化：

(1)a.she had your dark suit in greasy wash water all year

b.don't ask me to carry an oily rag like that

其余的句子语音丰富，包含所有单音(音)和全部的双音(二元音)。此外，设计 在以下两方面取得平衡：多个说话者说相同的句子以便说话者之间进行比较，还有 语料库涵盖较大范围的句子，以便获得最大的双音覆盖。每个说话者读的句子中有 5个也会被其他6个说话者读到(为了可比性)。每个说话者读的其余3个句子是 不同的(为了覆盖面)。

NLTK 包 括 TIMIT 语料库的样本。你可以使用常用的方式，也就是使用 help(nltk.corpus.timit)访问它的文档。nltk.corpus.timit.fileids() 可以看到语料库样本中 160个录制的话语列表。每个文件名的内部结构如图11-1所示。

Gender Speaker ld Sentence ld

dr1 f vmh0 SX 206

|  |  |
| --- | --- |
| **Dialect Region:** | **Sentence Type:** |
| dr1:New England | sa:read by all speakers, |
| **dr2:** Northem  **dr3:** North Midland  **dr4:** South Midland  **dr5** :Southern  **dr6** :New York City  dr7;Western | chosen to reveal dialect  variation  sl:read by individual  speakers,chosen from  diverse text sources  including Brown Corpus |
| dr8:Army Brat  (moved around) | sx:read by subset of speakers,chosen for phonetic complexity |

**图11-1** **TIMIT标识符的结构：每个记录将说话者的方言区、性别、说话者标识符、句子类型、**

**句子标识符组成的字符串作为标签**

每个项目都有音标，可以使用phones()方法访问。我们可以按习惯的方式访问相应 的词标识符。两种访问方法都允许一个可选的参数 offset=True, 其中包括音频文件 相应跨度的开始和结尾偏移。

>>>phonetic =nltk.corpus.timit.phones('drl-fvmh0/sal')

>>>phonetic

'',s'ixy',v''t,',l''e,'y'n,'ix'',cgl','cd',a','',,'r','iy','s','iy','w','aa',

'sh','epi','w','aa','dx','ax','q','ao','l','y','ih','ax','h#']

>>>nltk.corpus.timit.word\_times('drl-fvmh0/sal')

[('she',7812,10610),('had',10610,14496),('your',14496,15791),

('dark',15791,20720), ('suit',20720,25647),('in',25647,26906),

('greasy',26906,32668),('wash',32668,37890),('water',38531,42417), ('all',43091,46052),('year',46052,50522)]

除了这种文本数据， TIMIT 还包括词汇，提供每一个可与特定的话语相比较的规范 发音。

>>>timitdict =nltk.corpus.timit.transcription\_dict()

>>>timitdict['greasy']+timitdict['wash']+timitdict['water']

['g','r','iy1','s','iy','w','aol','sh','w','aol','t','axr']

>>>phonetic[17:30]

['g','r','iy','s','iy','w','aa','sh','epi','w','aa','dx','ax']

这给了我们语音处理系统在处理或识别这种特殊方言(新英格兰)的语音中必须进 行操作。最后， TIMIT 包括说话人的人口学统计，允许更细致地研究声音、社会和 性别特征。

>>>nltk.corpus.timit.spkrinfo('drl-fvmh0')

SpeakerInfo(id='VMHO',sex='F',dr='l',use='TRN',recdate='03/11/86',

birthdate='01/08/60',ht='5\'05"',race='WHT',edu='BS',

comments='BEST NEW ENGLAND ACCENT SOFAR')

**主要设计特点**

TIMIT 演示了语料库设计中的几个主要特点。首先，语料库包含语音和字形两个标 注层。 一般情况下，文字或语音语料库可能在多个不同的语言学层次标注，包括形 态、句法和段落层次。此外，即使在给定的层次仍然有不同的标注策略，甚至标注 者之间也会有分歧，因此我们要表示多个版本。TIMIT 的第二个特点是：它在多个 维度的变化与方言地区和二元音覆盖范围之间取得平衡。人口学统计的加入带来了 许多独立的变量，这有助于解释数据中的变化，便于以后用于在建立语料库时没有 想到的目的，例如社会语言学。第三个特点是：将原始语言学事件作为录音来捕捉 和作为标注来捕捉之间有明显的区分。两者一致表示文本语料库正确，原始文本通 常有不可改变的作品的外部来源。包含人类判断的作品的任何转换——即使如分词

一样简单——也是后来的修订版；因此以尽可能接近原始的形式保留源材料是十分 重要的。

TIMIT 的第四个特点是语料库的层次结构。每个句子4个文件，500个说话者每人 10个句子，共有20000个文件。这些被组织成一个树状结构，示意图如图11-2所 示。在顶层分成训练集和测试集，用于开发和评估统计模型。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **TIMIT**  **doc:** | | | | | | | |
| phoncode  prompts  spkrinfo  spkrsent  testset  timitdic | dr1  M | | **train**  d2 | **test**  … **dr8 dr1 dr2 dr8** | | | |
| **fcjfo fetb0 fsaho** | | **faks0:**  sal-phn  sa2-p2o  611573.pha =12201.pha 81941.phn Bx43.pha  8x133,par sx223.pan sx110.pan sx403.phn | | | sa1.txt  sa2,txt  s11573.txt  si2203.txt  B1943.txt  Bx43,txt  Bx133,txt  sx223,txt  *sx113.txt*  sx40),txt | sa1.WV  sa2.wiv  si1573.wav si2203.wav  *81941.wav*  BX43.wav Sx133,wav sx22],wav sx]13,wav 58400.wy | ma1,wrd  Ba2.wru  wi1573.wrd  si2203.wrd  *ai941.wrd*  8x43,wra  Px133.WT0  Bx223,wrd  mx313.wrd  nx403.wrd |

**图11-2** **发布的TIMIT语料库的结构；** **CD-ROM** **包含文档、顶层的训练(train)和测试(test)** **目录；训练和测试目录都有8子目录，每个方言区一个；这些目录又包含更多子目录，每个说话** **者一个；列出的目录是女性说话者** **aks0的目录内容，显示10个wav 文件配以一个录音文本文** **件、一个录音文本词对齐文件和一个音标文件**

最后，请注意尽管 TIMIT 是语音语料库，但它的录音文本和相关数据只是文本， 而且可以像任何其他的文本语料库那样用程序处理。因此，许多在这本书中所描述 的计算方法都适用。此外，注意 TIMIT 语料库包含的所有数据类型分为词汇和文 字，我们将在下面讨论。甚至，说话者人口学统计数据只不过是词汇数据类型的另 一个实例。

我们认识到文字和记录结构是关注数据管理的两个计算机科学子领域的首要内容， 两个子领域即为全文检索领域和数据库领域。语言数据管理的一个显着特点往往是 将这两种数据类型放在一起，可以利用这两个领域的成果和技术。

**基本数据类型**

不考虑它的复杂性，TIMIT 语料库只包含两种基本数据类型：词典和文本。正如我 们在第2章中所看到的那样，大多数词典资源都可以使用记录结构表示，即一个关 键字加一个或多个字段，如图11-3 所示。词典资源可能是一个传统字典或比较词 表，如下所示。它也可以是一个短语词典，其中的关键字是一个短语而不是一个词。 词典还包括记录结构化的数据，我们可以通过对应主题的非关键字字段来查找条 目。我们也可以构造特殊的表格(称为范例)来进行对比和说明系统性的变化，图 11-3显示了3个动词。TIMIT 的说话者表也是一种词典资源。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Lexicon**  Abstraction:fielded records   |  |  |  |  |  | | --- | --- | --- | --- | --- | | key | field | field | field | field | | key | field | field | field | field |   Eg:dictionary  **wake**:werk,[V],cease to sleep … .  *walk:wo.k,[v],progress by lifing and setting down each foot.*  Eg:comparative wordlist  wake;aufwecken;acordar  walk;gehen;andar  write;schreiben;enscrever  Eg:verb paradigm  wake woke woken  write wrote written  wring wrung wrung | Text  Abstraction:time series   |  |  |  |  | | --- | --- | --- | --- | | token attrs | token attrs | token attrs |  |   time  Eg:written text  A long time ago,Sun and Moon  ived fogether.They were good  brothers. …  Eg:POS-tagged text  A/DT long/JJ time/NN ago/RB J,  Sun/NNP and/CC Moon/NNP  ived/VBD together/RB ./.  Eg:interlinear text  Ragaipa irai vateri  ragai -pa ira - vate -ri  PR1.SG-BEN RP3.SGM-ABS give-2SG |

图11-3 基本语言数据类型——词汇和文本：它们的多样性中，词汇具有记录结构，而已标注 文本具有时间组织

在最抽象的层面上，文本表示真实的或虚构的讲话事件，该事件的时间过程也在文 本本身存在。文本可以是一个小单位，如一个词或句子，也可以是一个完整的叙述 或对话。它可能会有标注如词性标记、形态分析、话语结构等。正如我们在 IOB 标注技术(见第7章)中所看到的，可以使用单个词的标记表示更高层次的成分。 因此，图11-3所示为丰富的文本抽象形式。

不考虑单独的语料库的复杂性和特质，最基本的，它们是带有记录结构化数据的

文本集合。语料库的内容往往偏重于这些类型中的一种或多种。例如：布朗语料 库包含500个文本文件，但我们仍然可以使用表将这些文件与15中不同风格关 联。在事情的另一面，WordNet 包含117659个同义词集记录，也包含许多例子句 子(小文本)来说明词的用法。TIMIT 处在中间，含有大量独立的文本和词汇类 型的材料。

**11.2** **语料库生命周期**

语料库并不是从天而降的，需要精心的准备和长时期的输入。原始数据需要进行收 集、清理、记录并以系统化的结构存储。标注可分为各种层次， 一些需要语言形态 或句法的专门知识。要完成这个阶段取决于建立一个高效的工作流程，包括适当的 工具和格式转换器。质量控制程序可以将寻找标注中的不一致落实到位，确保尽最 大可能在标注者之间达成一致。由于任务的规模和复杂性，大型语料库可能需要几 年的准备，包括几十或上百人多年的努力。在本节中，我们简要地回顾语料库生命 周期的各个阶段。

**创建语料库的3种方案**

语料库的一种类型是设计在创作者的探索过程中逐步展现。这是典型的传统“领域 语言学”模式，即来自会话的材料在它被收集的同时就被分析了，明天的想法往往 基于今天分析而产生的。在随后几年的研究中产生的语料不断被使用，并可能用作 不确定的档案资源。计算机化明显有利于这种类型的工作，以广受欢迎的程序 Shoebox 为例，它作为 Toolbox 重新发布，现在已有超过二十年的历史(见2.4节)。 其他的软件工具，甚至是简单的文字处理器和电子表格，通常也可用于采集数据。 在下一节，我们将着眼于如何从这些来源提取数据。

另一种语料库创建方案是典型的实验研究，其中一些精心设计的材料被从一定范围 的人类受试者中收集，然后进行分析来评估一个假设或开发一种技术。此类数据库 经常在实验室或公司内被共享和重用，经常被更广泛地发布。这种类型的语料库是 “共同任务”的科研管理方法的基础，这在过去的二十年已成为政府资助的语言技 术研究项目。在前面的章节中，我们已经遇到很多这样的语料库；我们将学习如何 编写 Python 程序实践这些语料库发布前必要做的一些任务。

最后，还要为特定的语言收集“参考语料”,如美国国家语料库(A merican National



Corpus,ANC) 和英国国家语料库(British National Corpus,BNC)。这里的目标已 经成为产生使用各种形式、风格和语言的一个全面的记录。除了规模庞大的挑战， 还特别依赖自动标注工具和后期编辑共同修复错误。然而，我们可以编写程序来查 找和修复错误，还可以分析语料库是否平衡。

**质量控制**

为自动和手动的数据准备好的工具是必不可少的。然而， 一个高质量的语料库的建 立很大程度取决于文档、培训和工作流程等普通的东西。标注指南确定任务并记录 标记约定。它们可能会定期更新以覆盖不同的情况，同时制定实现更一致的标注的 新规则。在此过程中标注者需要接受训练，包括指南中没有的情况的解决方法。需 要建立工作流程，尽可能与支持软件一起，跟踪哪些文件已被初始化、标注、验证、 手动检查等。可能有多层标注，由不同的专家提供。不确定或不一致的情况可以进 行裁决。

大的标注任务需要多个标注者，由此产生一致性的问题。 一组标注者如何能一致地 处理呢?我们可以通过将一部分独立的原始材料由两个人分别标注，很容易测量标 注的一致性。这揭示了指南中或标注任务的不同功能的不足。在对质量要求较高的 情况下，整个语料库可以标注两次，由专家裁决不一致的地方。

报告标注者之间对语料库(如通过两次标注10%的语料库)达成的一致性被认 为是最佳实践。这个分数是作在此语料库上训练的所有自动化系统的期望性能 的上限。

**注意!**

应谨慎解释标注者之间一致性得分，因为标注任务的难度差异巨大。 例如：90%的一致性得分对于词性标注是可怕的得分，但对语义角 色标注是可以预期的得分。

**Kappa** 系数 k 测量两个人判断类别，修正预期期望的一致性。例如：假设要标注 一个项目，4种编码选项可能性相同。这种情况下，两个人随机编码预计有25%可 能达成一致。因此，25%一致性将表示为k=0, 相应的较好水平的一致性将依比 例决定。对于50%的一致性，我们将得到k=0.333, 因为50是从25到100之间 距离的三分之一。还有许多其他一致性测量方法，详情请参阅 help(nltk.metrics. agreement)。

我们还可以测量语言输入的两个独立分割的一致性，例如：分词、句子分割、命名 实体识别。在图11-4中，我们看到3 种可能的由标注者(或程序)产生的项目序 列的分割。虽然没有一个完全一致，但S1 和 S2 是接近一致的，这是我们想要一个 合适的测量。windowdiff 是评估两个分割一致性的一种简单算法，通过在数据上移 动一个滑动窗口计算近似差错的部分得分。如果我们将标识符预处理成0和1的序 列，当标识符后面跟着边界符号时记录下来，我们就可以用字符串表示分割，应用 windowdiff打分器。

>>>s1="00000010000000001000000"

>>>s2 ="00000001000000010000000"

>>>s3="00010000000000000001000"

>>>nltk.windowdiff(sl,s1,3)

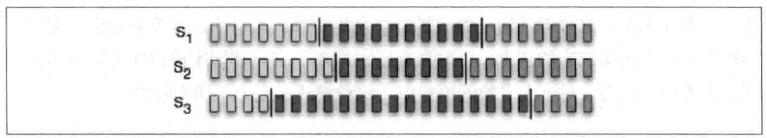
0

>>>nltk.windowdiff(s1,s2,3)

4

>>>nltk.windowdiff(s2,s3,3)

16



**图11-4** **一个序列的3种分割：小矩形代表字、词、句，总之，任何可能被分为语言单位的序**

**列；S1 和** **S2 是接近一致的，两者都与** **S3 显著不同**

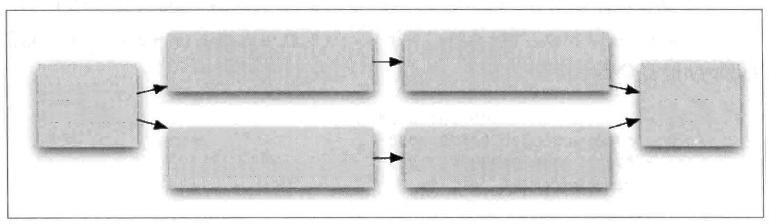
上面的例子中，窗口大小为3。windowdiff 计算在一对字符串上滑动这个窗口。在 每个位置它计算两个字符串在这个窗口内的边界的总数，然后计算差异，最后累加 这些差异。我们可以通过放大或缩小窗口的大小来控制测量的敏感度。

**维护与演变**

随着大型语料库的发布，研究人员将研究立足于均衡的、集中的子集，该子集是从 目的完全不同的语料库中派生出的。例如： Switchboard 数据库，最初是为识别说 话人的研究而收集的，已被用作语音识别、单词发音、口吃、句法、语调和段落结 构研究的基础。重用语言语料库的动机包括希望节省时间和精力，希望在别人可以 复制的材料上工作，有时希望研究语言行为更加自然的形式。为这样的研究选择子 集的过程本身可视为一个伟大的贡献。



除了选择适当的语料库子集，还包括重新格式化文本文件(如转换为XML), 重命 名文件，重新为文本分词，选择数据的子集来充实等。多个研究小组可以独立地完 成这项工作，如图11-5 所示。在以后的日子，应该有人想要组合不同的版本的源 数据，这项任务可能会非常繁重。

研究小组3:

研究小组1:

丢弃标签，注释命名实体

选择句子的子集和注释指代

出版文集：

话语研究：

结合源???

标计划和标签

研究小组2;

研究小组4:

句子和标签语义角色的选择子集

句子划分，舍弃标点符号，解析

**图11-5** **语料库随着时间的推移而演变：语料库发布后，研究小组将独立地使用它，选择和丰** **富不同的部分；然后研究努力整合单独的标注，面临校准注释的艰巨挑战**

由于缺乏有关派生的版本是如何创建的，哪个才是最新版本等记录，使用派生的语料 库变得更加困难。这种混乱情况的改进方法是集中维护语料库，专家委员会定期修订 和扩充它，考虑第三方的意见，不时发布新的版本。字典和国家语料库可能以这种方 式集中维护。然而，对于大多数的语料库，这种模式是完全不切实际的。

出版原始语料库的中间过程是要有一个能识别其中任何一部分的规范。每个句子、 树、或词条都有全局的唯一标识符，每个标识符、节点或字段(分别)都有一个相 对偏移。标注包括分割，可以使用规范的标识符(一个被称为**对峙注释**的方法)引 用原材料。这样，新的标注可以与源材料独立分布，同一来源的多个独立标注可以 对比和更新而不影响源材料。

如果语料库有多个版本，那么可以使用版本号或日期进行识别。整个语料的版本标 识符之间的对应表，将使任何不同的注释更容易被更新。

**注意!**

有时一个更新的语料包含对一直在外部标注的基本材料的修正。标 识符可能会被分拆或合并，成分可能已被重新排列。新老标识符之 间可能不会一一对应。使不同标注打破新版本的这些组件比默默允 许其标识符指向不正确的位置要好。

**11.3** **数据采集**

**从网上获取数据**

网络是语言分析的一个丰富的数据源。我们已经讨论了访问单个文件的方法，如 RSS 订阅、搜索引擎的结果(见3.1 节)。然而，在某些情况下，我们要获得大量 的 Web 文本。

最简单的方法是获得出版的网页文本的文集。Web 语料库 ACL 特别兴趣组 (The ACL Special Interest Group on Web as Corpus,SIGWAC)在 <http://www.sigwac.org.uk/> 上维护了一个资源列表。使用定义好的 Web 语料库的优点是它们有文档、稳定并 允许重复性实验。

如果所需的内容在一个特定的网站上，有许多实用程序能获取网站的所有可访问 内容，如 GNU Wget(<http://www.gnu.org/software/wget/>)。为了最大的灵活性和可 控制，可以使用网络搜索器如 Heritrix(<http://crawler.archive.org/>) 。 搜索器允许细 致地控制去看哪里，获取哪个链接及如何组织结果。例如：如果要编译双语文本 集合，对应两种语言的文档对，搜索器需要检测站点的结构及提取文件之间的对 应关系，它需要按照捕获的对应方式组织下载的页面。编写你自己的网页搜索器 是很有帮助的，但也有很多陷阱需要克服，如检测 MIME 类型、转换相对地址为 绝对 URL、避免被困在循环链接结构、处理网络延迟、避免使站点超载或被禁止 访问该网站等。

**从文字处理器文件获取数据**

文字处理软件通常用来在具有有限的可计算基础设施的项目中手工编制文本和词 汇。这些项目往往提供数据录入模板，通过文字处理软件并不能保证数据结构正确。 例如：每个文本可能需要有一个标题和日期。同样，每个词条可能有一些必须的字 段。随着数据规模和复杂性的增长，用于维持其一致性的时间的比重也增大。

我们怎样才能提取这些文件的内容，使我们能够在外部程序中操作?此外，我们如 何才能验证这些文件的内容，以帮助作者创造结构良好的数据，在原始的创作过程 中最大限度地提高数据的质量?

考虑一个字典，其中的每个条目都有一个词性字段，从具有20个可能值的集合中 选取，在发音字段显示，以11号黑体字呈现。传统的文字处理器没有能够验证所 有的词性字段已正确输入和显示的搜索函数或宏。这个任务需要彻底的手动检查。 如果文字处理器允许保存文档为一种非专有的格式，如 text、HTML 或 XML, 有 时我们可以写程序自动做这个检查。

思考下面的词条片段：“sleep [sli:p]vi.condition of body and mind…”。我们可以在

MSWord 中输入这些词，然后“另存为网页”,然后检查生成的HTML 文件。

<p class=MsoNormal>sleep

<span style='mso-spacerun:yes'></span>

[<span class=SpellE>sli:p</span>]

span stl=e='f'-- :pan></b>

<span style='mso-spacerun:yes'></span>

 condition of body and mind ..<o:p></o:p></i>

观察发现该条目的 HTML 段落表示，使用了<p>元素，词性出现在<span style='font- size:11.0pt>元素内。下面的程序定义了合法的词性集合：legal\_pos 。 然后从 dicthtm 文 件提取所有11号字的内容，并存储在集合used\_pos 中。请看搜索模式包含一个括号括 起来的子表达式；只有匹配该子表达式的材料才会被 refindall 返回。最后，程序用 used\_pos 和 legal\_pos 的补集构建非法词性的集合。

torn =re.compile(r"'font-size(['''

>>>document =open("dict.htm").read()

ls\_pos =used\_=psrlc\_u))

>>>print list(illegal\_pos)

['v.i','intrans']

这个简单的程序只是冰山一角。我们可以开发复杂的工具来检查文字处理器文 件的一致性，并报告错误，使字典保持器可以使用原来的文字处理器纠正原始

文件。

只要我们知道数据的正确格式，就可以编写其他程序将数据转换成不同格式。例 11-1中的程序使用 nltk.clean\_html) 剥离 HTML 标记，提取词和它们的发音，以“逗 号分隔值”(CSV) 格式生成输出。



**例11-1** **将** **Microsoft Word 创建的** **HTML 转换成** **CSV。**

def lexical\_data(html\_file):

SEP='\_ENTRY'

html =open(html\_file).read()

html =re.sub(r'<p',SEP +'<p',html)

text =nltk.clean\_html(html)

text =''.join(text.split())

for entry in text.split(SEP):

if entry.count(' ')>2:

yield entry.split('',3)

eol\_pda**(**tt.","wb"))

>>>import csv

**注意!**

**更多** **HTML 复杂的处理可以使用** [**http://www.crummy.com/software/**](http://www.crummy.com/software/)

BeautifulSoup/上的 Beautiful Soup 包。

**从电子表格和数据库中获取数据**

电子表格通常用于获取词表或范式。例如： 一个比较词表可以用电子表格创建，用排 表示每个同源组，每种语言一列(见 nltk.corpus.swadesh 和 www.rosettaproject.org)。 大多数电子表格软件可以将数据导出为 CSV 格式。正如我们将在下面看到的，使用 csv 模块Python程序可以很容易地访问它们。

有时词典存储在一个完全成熟的关系数据库中。经过适当的标准化，这些数据库可 以确保数据的有效性。例如：我们可以要求所有词性都来自指定的词汇，通过声明 词性字段为枚举类型或用一个外键引用一个单独的词性表。然而，关系模型需要提 前定义好数据(模式)结构，这与高度探索性的构造语言数据的主导方法相违背。 被认为是强制性和独特的字段往往是可选的、可重复的。只有当数据类型提前全都 知道时关系数据库才是适用的。如果不是，或者几乎所有的属性都是可选的或重复 的，关系数据库的做法就行不通了。

然而，当我们的目标只是简单地从数据库中提取内容时，完全可以将表格(或 SQL 查询结果)转换成 CSV 格式，并加载到我们的程序中。我们的程序可能会执行不 太容易用SQL 表示的语言学目的的查询，如：select all words that appear in example

['...','a','and','body','by','cease','condition','down','each', 'foot','lifting','mind','of','progress','setting','to']

\* · if len(defn\_word)>3)④

sentences for which no dictionary entry is provided。对于这个任务，我们需要从记录 中提取足够的信息，使它连同词条和例句能被唯一地识别。假设现在这个信息是在 CSV 文件dict.csv中。

"sleep","sli:p","v.i","a condition of body and mind .."

"walk","wo:k","v.intr","progress by lifting and setting down each foot .." "wake","weik","intrans","cease to sleep"

现在，我们可以表示此查询，如下所示。

>>>import csv

>>>lexicon =csv.reader(open('dict.csv'))

>>>pairs =[(lexeme,defn)for(lexeme, , ,defn)in lexicon]

>>>lexemes,defns =zip(\*pairs)

>>>defn\_words =set(w for defn in defns for w in defn.split())

>>>sorted(defn\_words.difference(lexemes))

然后，这些信息可以指导正在进行的工作以丰富词汇和更新关系数据库的内容。

**转换数据格式**

已标注语言数据很少以最方便的格式保存，往往需要进行各种格式转换。字符编码 之间的转换已经讨论过(见3.3节)。在这里，我们专注于数据结构。

最简单的情况，输入和输出格式是相同结构的。例如：我们可能要将词汇数据从 Toolbox 格式转换为 XML, 可以直接一次一个地转换词条(见11.4节)。数据结构 反映在所需程序的结构中： 一个 for 循环，每次循环处理一个词条。

另一种常见的情况，输出是输入的摘要形式，如： 一个倒置的文件索引。有必要在 内存中建立索引结构(见例4.8),然后把它以所需的格式写入一个文件。下面的例 子构造一个索引，映射字典定义的词汇到相应的每个词条“的语意?,已经对定义文 本分词③,并丢弃短词④。 一旦该索引建成，打开一个文件，然后遍历索引项，以所 需的格式输出行⑤。

>>>idx =nltk.Index((defn\_word,lexeme) ①

· for(lexeme,defn)in pairs②

for defn\_word in nltk.word\_tokenize(defn)③

>>>idx\_file=open("dict.idx","w")

>>>for word in sorted(idx):

·



idx\_words =','.join(idx[word])

idx\_line="8s:8s\n"&(word,idx\_words)⑤

idx\_file.write(idx\_line)

>

由此产生的文件dictidx 包含下面的行。(如果有更大的字典，我们希望找到每个索 引条目中列出的多个语意)。

body:sleep

cease: wake

condition:sleep

down: walk

each:walk

foot:walk

lifting:walk

mind:sleep

progress;walk

setting:walk

sleep:wake

在某些情况下，输入和输出数据都包括两个或两个以上的维度。例如：输入可能是 一组文件，每个都包含单词频率数据的单独列。所需的输出可能是一个两维表，其 中原来的列以行出现。在这种情况下，我们一次填补一列，填充内部的数据结构， 然后在写数据到输出文件中时， 一次读取一行。

最棘手的情况，源格式和目标格式覆盖的域略有不同，在它们之间的翻译过程中信 息不可避免地会丢失。例如：我们可以组合多个 Toolbox 文件创建一个CSV 文件， 其中包含了比较词表，丢失了输入文件中除Ix 字段以外所有的信息。如果后来修 改了 CSV 文件，将变化注入原有的 Toolbox 文件将是一个消耗体力的过程。这种 “往返”问题的部分解决方法是为每个语言对象关联明确的标识符，使用这些对象 的标识符。

**决定要包含的标注层**

发布的语料库中所包含的信息的丰富性差别很大。语料库最低限度通常会包含至少 一个声音或字形符号的序列。另一方面， 一个语料库可以包含大量的信息，如：句 法结构、形态、韵律、每个句子的语义、加上段落关系或对话行为的标注。标注的 这些额外的层可能正是有人执行一个特定的数据分析任务所需要的。例如：如果我

们可以搜索特定的句法结构，找到一个给定的语言模式就更容易；如果每个词都标 注了意义，为语言模式归类就更容易。这里提供一些常用的标注层。

分词：文本的书写形式不能明确地识别它的标识符。分词和规范化的版本作为常规 的正式版本的补充可能是一个非常方便的资源。

断句：正如我们在第3章中看到的，断句比它看上去的似乎更加困难。因此， 一些 语料库使用明确的标注来断句。

分段：段和其他结构元素(标题，章节等)可能会明确注明。

*词性：文档中的每个单词的词类。*

句法结构： 一个树状结构显示一个句子的组成结构。

浅层语义：命名实体和共指标注，语义角色标签。

*对话与段落：对话行为标记，修辞结构。*

然而，现有的语料库之间在如何表示标注上并没有多少一致性。两个大类的标注表 示应加以区别。**内联标注**通过插入带有标注信息的特殊符号或控制序列修改原始文 档。例如：为文档标注词性时，字符串 “fly” 可能被替换为字符串“fly/NN” 来表 示词 fly 在文中是名词。相比之下， **对峙标注不**修改原始文档，而是创建一个新的 文档，通过使用指针引用原始文档来增加标注信息。例如：这个新的文档可能包含 字符串“<token id=8 pos=NN'>”, 表示8号标识符是一个名词。

**标准和工具**

一个用途广泛的语料库需要支持广泛的格式。然而， NLP 研究的前沿需要各种新 定义的没有得到广泛支持的标注。 一般情况下，并没有广泛使用适当的创作、发布和 使用语言数据的工具。大多数项目都必须制定它们自己的一套工具，供内部使用，这 对缺乏必要资源的其他人没有任何帮助。此外，我们还没有一个可以胜任的普遍接受 的标准来表示语料库的结构和内容。没有这样的标准，就不可能有通用的工具 同时，没有可用的工具，适当的标准也不太可能被开发、使用和接受。

针对这种情况的方法就是开拓未来，开发一种通用的能充分表现捕获多种标注类型

(见11.8节的例子)的格式。NLP 的挑战是编写程序处理这种格式的泛化。例如， 如果编程任务涉及树状数据，文件格式允许任意有向图，那么必须验证输入数据检 查树状图的属性，如根源、连通性、无环性。如果输入文件包含其他层的标注，该 程序将需要知道数据加载时如何忽略它们，将树状数据保存到文件时不能否定或抹 杀这些层。

另一种方法是编写一个一次性的脚本来操纵语料格式；这样的脚本将许多 NLP 研 究人员的文件夹弄得乱七八糟。在语料格式解析工作应该只进行一次(每编程语言) 的前提下，NLTK 中的语料库阅读器是更系统的方法。

为取代集中在一种共同的格式，我们更希望开发一种共同的接口(参见nltk.corpus) 。 思考NLP 中语料类型treebanks 的情况。将短语结构树状图存储在一个文件中的方 法很多。我们可以使用嵌套的括号，或嵌套的XML 元素，或每行带有一个(child-id,” parent-id) 对的依赖符号，或一个 XML 版本的依赖符号等。然而，每种情况中的 逻辑结构几乎是相同的。很容易设计一种共同的接口，让应用程序员编写代码使用 如：children) 、leaves) 、depth() 等方法来访问树状数据。注意这种做法来自计算机 科学中已经接受的做法，即抽象数据类型、面向对象设计、三层结构(见图11-6)。 其中的最后一个——来自关系数据库领域——允许终端用户应用程序使用通用的 模型(“关系模型”)和通用的语言(SQL) 抽象出文件存储的特质，并允许新的文 件系统技术的出现，而不会干扰到终端用户的应用。以同样的方式， 一个通用的语 料库接口将应用程序从数据格式隔离。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方法  方法  方法 | 常见格式  方法  方法  格式  方法 | 常见的通过三层架构的界面 | | | |
| 方法 | 方法 | | 方法 |
| 界面 | | | |
| 格式 | | 格式 | |

**图11-6** **通用格式对比通用接口**

在此背景下，创建和发布一个新的语料库时，尽可能使用现有广泛使用的格式是权 宜之计。如果这样不可能，语料库可以带有一些软件——如 nltk.corpus 模块——支

持现有的接口方法。

**处理濒危语言时特别注意事项**

语言对科学和艺术的重要性体现在文化宝库包含在语言中。世界上大约7000种人 类语言中的每一种都是丰富的，在它独特的方面，在它口述的历史和创造的传说， 在它的文法结构和它变化的词汇及它们含义中的细微差别。受威胁残余文化中的词 能够区分具有科学家未知的治疗用途的植物亚种。当人们互相接触时，每个人都为 之前的语言提供一个独特的窗口，语言随着时间的推移而变化。世界许多地方，小 的语言变化从一个镇到另一个镇，累加起来在一个半小时的车程空间中成为一种完 全不同的语言。对于其惊人的复杂性和多样性，人类语言犹如丰富多彩的挂毯随着 时间和空间而伸展。

然而，世界上大多数语言面临着灭绝。对此，许多语言学家都在努力工作，记录语 言，构建世界语言遗产的相关重要方面的丰富记录。在 NLP 的领域能为这方面的 努力提供什么帮助吗?开发标注器、分析器、命名实体识别等不是最优先的，通常 没有足够的数据来开发这样的工具。相反，最经常提出的是需要更好的工具来收集 和维护数据，特别是文本和词汇。

从表面看，开始收集濒危语言的文本应该是一件简单的事情。即使我们忽略了棘手 的问题，如：谁拥有文本，文本中包含的文化知识有关敏感性，转录仍然有很多明 显的实际问题。大多数语言缺乏标准的书写形式。当一种语言没有文学传统时，拼 写和标点符号的约定也没有得到很好的建立。因此，通常的做法是与文本收集一起 创建一个词典。当在文本中出现新词时不断更新词典。可以使用文字处理器(用于 文本)和电子表格(用于词典)来完成这项工作。更妙的是，SIL 的自由软件Toolbox 和 Fieldworks 对文本和词汇的创建集成提供了很好的支持。

当濒危语言的说话者学会自己输入文本时， 一个共同的障碍就是对正确拼写的 极度关注。词典大大有助于这一进程，但我们需要查找的方法不要假设有人能 确定任意一个词的引文形式。这个问题对具有复杂形态的包括前缀的语言可能 是很急迫的。这种情况下，应该使用语义范畴标注词项，并允许通过语义范畴 或注释查找。

可以通过相似的发音查找词项。下面是如何做到这一点的一个简单的演示。第一步

是确定易混淆的字母序列，映射复杂的版本到更简单的版本。我们还可以注意到： 辅音群中字母的相对顺序是拼写错误的一个来源，所以我们要将辅音字母顺序规 范化。

>>>mappings =[('ph','f'),('ght','t'),('^kn','n'),('qu','kw'),

('[aeiou]+','a'),(r'(.)\1',r'\1')]

>>>def signature(word):

. · piecrd =re.fie]p+l,d)

for patt,repl in mappings:

 return''.join(char for piece in pieces for char in sorted(piece))[:8] >>>signature('illefent')

'lfnt'

>>>signature('ebsekwieous')

'bskws'

>>>signature('nuculerr')

'nclr'

下一步，我们对词典中的所有词汇创建从特征到词汇的映射，并为给定的输入词找 到候选的修正(但我们必须先计算这个词的特征)。

>>>signatures=nltk.Index((signature(w),w)for w in nltk.corpus.words.words()) >>>signatures[signature('nuculerr')]

['anicular','inocular','nucellar','nuclear','unicolor','uniocular', 'unocular']

最后，我们应该按照与原词的相似程度对结果排序。通过函数rankO完成。唯一剩 下的函数提供给用户一个简单的接口。

>>>def rwtd(rs,t)wk:.rd\_ koer**d**],w),w)for w in wordlist)

>.>· >def rd)

… · if sig in signatures:

· return rank(word,signatures[sig])

. · else:

· return []

>>>fuzzy\_spell('illefent')

['olefiant','elephant','oliphant','elephanta']

>>>fuzzy\_spell('ebsekwieous')

['obsequious']

>>>fuzzy\_spell('nucular')

['nuclear','nucellar','anicular','inocular','unocular','unicolor', 'uniocular']

这仅仅是一个演示，其中一个简单的程序就可以方便地访问语言书写系统可能不规 范或语言的使用者可能拼写得不是很好的上下文中的词汇数据。其他简单的 NLP 在这个领域的应用包括：建立索引以方便对数据的访问，从文本中拾取词汇表，构 建词典时定位词语用法的例子，在知之甚少的数据中检测普遍或特殊模式，并在创 建的数据上使用各种语言的软件工具执行专门的验证。我们将在第11.5 节继续进 行介绍。

**11.4** **使用XML**

可扩展标记语言(The Extensible Markup Language,XML) 为设计特定领域的标记 语言提供了一个框架。它有时被用于表示已被标注的文本和词汇资源。不同于 HTML的标签是预定义的， XML 允许我们组建自己的标签。不同于数据库，XML 允许创建的数据而不必事先指定其结构，它允许有可选的、可重复的元素。在本节 中，我们简要回顾一下XML 的一些与表示语言数据有关的特征，并说明如何使用 Python 程序访问 XML文件中存储的数据。

**语言结构中使用** **XML**

由于其灵活性和可扩展性，XML 是表示语言结构的最佳选择。下面是一个简单的 词汇条目的例子。

(2) <entry>

<headword>whale</headword>

poofse larger cetacean mammals having a streamlined

body and breathing through a blowhole on the head</gloss>

</entry>

它由一系列括在尖括号中的XML 标记组成。每个开始标记，如<gloss>, 都匹配一 个结束标记，如</gloss>; 它们共同构成一个 XML 元素。上面的例子已经用空白 符排版好了，它也可以放在长长的一行中。处理XML 的方法通常不会注意空白符。 在结构良好的XML 中，在嵌套的同一级别中所有的开始标签必须有对应的结束标 记(即 XML文档必须是格式良好的树)。

XML 允许使用重复的元素，例如：添加另一个 gloss 字段，如我们在下面看到的。 我们将使用不同的空白符来强调：如何布局并不重要。

(3) <entry><headword>whale</headword><pos>noun</pos><gloss>any of the larger cetacean mammals having a streamlined body and breathing

through a blowhole on the head</gloss><gloss>a very large person; impressive in size or qualities</gloss></entry>

再进一步可能是使用外部标识符链接词汇到一些外部资源，如 WordNet 。(4) 中我 们组合 gloss 和一个同义集标识符到一个我们称之为 “sense” 的新元素。

(4) <entry>

<headword>whale</headword>

<pos>noun</pos>

<sense>

<gloss>any of the larger cetacean mammals having a streamlined body and breathing through a blowhole on the head</gloss>

<synset>whale.n.02</synset>

</sense>

</sense>

<gloss>a very large person;impressive in size or qualities</gloss>

<synset>giant.n.04</synset>

</sense>

</entry>

另外，我们可以使用**XML 属性表**示同义集标识符，不必使用(5)中的嵌套结构。

(5) <entry>

<headword>whale</headword>

hale.n.02">any of the larger cetacean mammals having

a streamlined body and breathing through a blowhole on the head<

/gloss><gloss synset="giant.n.04">a very large person;impressive in size or

qualities</gloss>

</entry>

这说明了 XML 的某些灵活性。如果它看上去似乎有点随意，这是因为它就是很 随意!遵循 XML 的规则，我们可以指定新的属性名称，任意地嵌套它们。我们 可以重复元素，省略它们，让它们每次的顺序都不同。我们可以指定字段，它的 出现取决于其他一些字段的值，例如：如果词性是动词，那么条目可以有 past\_tense 元素以保存动词的过去式，但如果词性是名词，就不允许有 past\_tense 元素。为 了在这种自由上加一些秩序，使用“架构 (schema)” 限制一个XML 文件的格式， 这是一种类似于上下文无关文法的声明。存在工具可以测试带有架构的 XML 的 有效性。

**XML的作用**

我们可以用XML 来表示许多种语言信息。然而，灵活性是要付出代价的。每次增 加复杂性，如：允许一个元素是可选的或重复的，对所有访问这些数据的程序都要 做出更多的工作。这也使得更难以检查数据的有效性，或使用一种XML 查询语言 来查询数据。

因此，使用XML 来表示语言结构并不能神奇地解决数据建模问题。我们仍然需要 解决如何结构化数据，然后使用架构定义结构，并编写程序读取和写入格式，以及 把它转换为其他格式。同样，我们仍然需要遵循一些有关数据规范化的标准原则。 这样做可以避免相同信息的重复复制，所以当只有一个副本变化时，不会导致数据 不一致。例如：交叉引用表示为<xref>headword</xref>将重复存储一些其他词条的 核心词，如果在其他位置的字符串的副本被修改，链接就会被打断。信息类型之间 存在的依赖关系需要建模，使我们不能创建没有根的元素。例如：如果 sense 的定 义不能作为词条独立存在，那么 sense 就要嵌套在 entry 元素中。多对多关系需要 从层次结构中抽象出来。例如：如果一个 word 可以有很多对应的 senses,一个sense 可以有几个对应的words, 而 words 和 senses 都必须作为 (word,sense) 对的列表 分别枚举。这种复杂的结构甚至可以分割成3个独立的XML 文件。

正如我们所看到的，虽然XML 提供了一个格式方便和用途广泛的工具，但它不是 能解决一切问题的灵丹妙药。

**ElementTree 接口**

Python的 ElementTree 模块提供了一种方便的方式用于访问存储在 XML文件中的 数据。ElementTree 是Python标准库(自从 Python 2.5) 的一部分，如果你在使用 Python 2.4,这也作为NLTK 的一部分提供。

我们将使用 XML 格式的莎士比亚戏剧集来说明 ElementTree 的使用方法。加载 XML文件并检查原始数据，首先在文件的顶部°,在那里我们看到一些XML 头和 一个名为 play.dtd的架构，接着是根元素PLAY。我们从Act1° 再次获得数据。(输 出中省略了一些空白行)

int rarcant\_file).read()

>>>merchant\_file=nltk.data.find('corpora/shakespeare/merchant.xml')

<?xml version="1.0"?>

<?xml-stylesheet type="text/css"href="shakes.css"?>

<!--<!DOCTYPE PLAY SYSTEM "play.dtd">-->

<PLAY>

<TITLE>The Merchant of Venice</TITLE>

>>>print raw[1850:2075] ②

<TITLE>ACT I</TITLE>

<SCENE><TITLE>SCENE I. Venice.A street.</TITLE>

<STAGEDIR>Enter ANTONIO,SALARINO,and SALANIO</STAGEDIR>

<SPEECH>

<SPEAKER>ANTONIO</SPEAKER>

<LINE>In sooth,I know not why I am so sad:</LINE>

刚刚访问了作为一个字符串的 XML 数据。正如我们所看到的，在Act 1开始处的 字符串包含 XML 标记： title 、scene 、stage directions 等。

下一步是作为结构化的XML 数据使用 ElementTree 处理文件的内容。我们正在处理 一个文件(一个多行字符串),并建立一个树状图，所以方法的名称是parse °并不奇 怪。变量 merchant 包含一个 XML元素 PLAY²。此元素有内部结构；我们可以使用 一个索引来得到它的第一个子元素， 一个 TITLE 元素。我们还可以看到该元素的文 本内容：戏剧的标题。要得到所有的子元素的列表，使用 getchildren)方法。

>>>from nltk.etree.ElementTree import ElementTree

>>>merchant =ElementTree().parse(merchant\_file) ①

>>>merchant

<Element PLAY at 22fa800>②

>>>merchant [0]

<Element TITLE at 22fa828>③

>>>merchant[0].text

'The Merchant of Venice'④

>>>merchant.getchildren()⑤

[<Element TITLE at 22fa828>,<Element PERSONAE at 22fa7bO>,<Element SCNDESCR at 2300170>,

<Element PLAYSUBT at 2300198>,<Element ACT at 2300le8>,<Element ACT at 234ec88>

<Element ACT at 23c87d8>,<Element ACT at 2439198>,<Element ACT at 24923c8>]

这部戏剧由标题、角色、场景描述、字幕和五幕组成。每一幕都有一个标题和一些 场景，每个场景由台词组成，台词由行组成，有4个层次嵌套的结构。让我们深入 到第四幕。

>>>merchant [-2][0].text

'ACT IV!



>>>merchant [-2][1]

<Element SCENE at 224cf80>

atV[-2e]n[1i]ourt of justice.'

>>>merchant [-2][1][54]

<Element SPEECH at 226ee40>

>>>merchant[-2][1][54][0]

<Element SPEAKER at 226ee90>

>>>merchant[-2][1][54][0].text

'PORTIA'

>>>merchant[-2][1][54][1]

<Element LINE at 226eeeO>

>>>merchant[-2][1][54][1].text

"The quality of mercy is not strain'd,"

**轮到你来：**

对语料库中包含的其他莎士比亚戏剧，如：《罗密欧与朱丽叶》或《麦克 白》,重复上述的一些方法。方法列表请参阅 nltk.corpus.shakespeare.

fileids0。

虽然我们可以通过这种方式访问整个树状图，但使用特定名称查找子元素会更加方 便。回想一下顶层的元素有几种类型。我们可以使用 merchant.findall(ACT) 遍历我 们感兴趣的类型(如：幕)。下面是一个这种特定标记在每一个级别的嵌套搜索的 例子。

>>>for i,act in enumerate(merchant.findall('ACT')):

for j,scene in enumerate(act.findall('SCENE')):

 for k,speech in enumerate(scene.findall('SPEECH')): · for line in speech.findall('LINE'):

· if 'music'in str(line.text):

k+1,line.text)

print "Act gd Scene sd Speech 8d:8s"%(i+1,j+1,

Act Act Act Act Act Act Act Act Act Act Act Act Act Act

3 Scene 2 Speech 9:Let music sound while he doth make his choice;

3 Scene 2 Speech 9:Fading in music:that the comparison

3 Scene 2 Speech 9:And what is music then?Then music is

5 Scene 1 Speech 23:And bring your music forth into the air.

5 Scene 1 Speech 23:Here will we sit and let the sounds of music

5 Scene 1 Speech 23:And draw her home with music.

5 Scene 1 Speech 24:I am never merry when I hear sweet music.

5 Scene 1 Speech 25:Or any air of music touch their ears,

5 Scene 1 Speech 25:By the sweet power of music:therefore the poet

5 Scene 1 Speech 25:But music for the time doth change his nature.

S SSpp::autmhb . rilusic.

5 Scene 1 Speech 29:It is your music,madam,of the house.

5 Scene 1 Speech 32:No better a musician than the wren.



不是沿着层次结构向下遍历每一级，而是寻找特定的嵌入元素。例如：让我们来看

看演员的顺序。我们可以使用频率分布看看谁最能说。

>>>speaker\_seq=[s.text for s inmerchant.findall('ACT/SCENE/SPEECH/SPEAKER')] >>>speaker\_freq =nltk.FreqDist(speaker\_seq)

>>>top5 =speaker\_freq.keys()[:5]

>>>top5

['PORTIA','SHYLOCK', 'BASSANIO', 'GRATIANO','ANTONIO']

我们也可以查看对话中谁跟着谁的模式。由于有23个演员，我们需要首先使用5.3 节中描述的方法将“词汇”减少到可处理的大小。

>>>mapping =nltk.defaultdict(lambda:'OTH')

>>>for s in top5:

mapping[s]=s[:4]

.



>>>speaker\_seq2 =[mapping[s]for s in speaker\_seq]

>>>cfd=nltk.ConditionalFreqDist(nltk.ibigrams(speaker\_seq2))

>>>cfd.tabulate()

ANTO BASS GRAT OTH PORT SHYL

ANTO 0 11 4 11 9 12

BASS 10 0 11 10 26 16

GRAT 6 8 0 19 9 5

OTH 8 16 18 153 52 25

PORT 7 23 13 53 0 21

SHYL 15 15 2 26 21 0

忽略153的条目，因为是前五位角色之间相互对话，最大的值表示Othello 和 Portia

的相互对话最多。

**使用** **ElementTree 访问** **Toolbox 数据**

在2.4节中，有一个访问Toolbox 数据的简单接口，Toolbox 数据是语言学家用来管 理数据的一种流行并行之有效的格式。这一节中，我们将讨论以 Toolbox 软件所不 支持的方式操纵 Toolbox 数据的各种技术。我们讨论的方法也可以应用到其他记录 结构化数据，而不必管实际的文件格式。

我们可以使用 toolbox.xml) 方法来访问 Toolbox 文件，将它加载到 ElementTree 对

象中。此文件包含巴布亚新几内亚罗托卡特语的词典。

>>>from nltk.corpus import toolbox

>>>lexicon =toolbox.xml('rotokas.dic')

语言数据管理 457

有两种方法可以访问lexicon 对象的内容：通过索引和通过路径。索引使用熟悉的 语法；lexicon[3]返回3号条目(实际上是从0算起的第4个条目),lexicon[3][0] 返回它的第一个字段。

>>>lexicon[3][0]

<Element lx at 77bd28>

>>>lexicon[3][0].tag

'lx'

>>>lexicon[3][0].text

'kaa'

第二种方式访问lexicon 对象的内容是使用路径。lexicon 是一系列 record 对象，其 中每个都包含一系列字段对象，如 Ix 和 ps。使用路径 record/lx, 我们可以很方便 地理解所有的语意。在这里，我们使用FindAll()函数来搜索路径record/lx的所有匹 配，并且访问该元素的文本内容，将其规范化为小写。

>>>[lexeme.text.lower()for lexeme in lexicon.findall('record/lx')]

['kaa','kaa','kaa','kaakaaro','kaakaaviko','kaakaavo','kaakaoko',

'kaakasi','kaakau','kaakauko','kaakito','kaakuupato',...,'kuvuto']

查看XML格式的 Toolbox数据。ElementTree 的 write()方法需要一个文件对象。通 常使用 Python 的内置在 open()函数创建。为了在屏幕上显示输出，我们可以使用 一个特殊的预定义的文件对象，称为stdout° (标准输出),它是在 Python 的 sys 模 块中定义的。

>>>import sys

>>>from nltk.etree.ElementTree import ElementTree

>>>tree =ElementTree(lexicon[3])

>>>tree.write(sys.stdout)①

<record>

<1x>kaa</1x>

<ps>N</ps>

<pt>MASC</pt>

oking banana</ge>

p>>itilong kukim</tkp>

<sf>FLORA</sf>

<dt>12/Aug/2005</dt>

<ex>Taeavi iria kaa isi kovopaueva kaparapasia.</ex>

<xp>Taeavi i bin planim gaden banana bilong kukim tasol long paia.</xp>

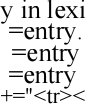
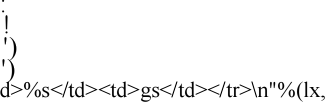
<xe>Taeavi planted banana in order to cook it.</xe>

</record>

**格式化条目**

我们可以使用在前一节看到的同样的想法生成 HTML 表格而不是纯文本。这有助 于将Toolbox 词汇发布到网络上。它可以产生 HTML元素<table> 、<t>(表格的行) 及<td> (表格数据)。

>>>html="<table>\n"

>>forl:t'gxs'

ps,g>html +="</table>"

>>>print html

<table>

<tr><td>kakae</td><td>???</td><td>small</td></tr>

<tr><td>kakae</td><td>CLASS</td><td>child</td></tr>

<tr><td>kakaevira</td><td>ADV</td><td>small-like</td></tr> <tr><td>kakapikoa</td><td>???</td><td>small</td></tr>

<tr><td>kakapikoto</td><td>N</td><td>newborn baby</td></tr>

<tr><td>kakapu</td><td>v</td><td>place in sling for purpose of carrying

d>kakapua</td><td>N</td><td>sling for lifting</td></tr>

<tr><td>kakara</td><td>N</td><td>arm band</td></tr>

<tr><td>Kakarapaia</td><td>N</td><td>village name</td></tr>

<tr><td>kakarau</td><td>N</td><td>frog</td></tr>

</table>

**11.5** **使用** **Toolbox 数据**

鉴于Toolbox 在语言学家中十分流行，我们将讨论一些使用 Toolbox 数据的进一步 的方法。很多在前面章节讲过的方法，如计数、建立频率分布、为同现制表，这些 都可以应用到 Toolbox 条目的内容上。例如：我们可以为每个条目计算字段的平均 个数。

>>>from nltk.corpus import toolbox

>>>lexicon =toolbox.xml('rotokas.dic')

>>>sum(len(entry)for entry in lexicon)/len(lexicon)

13.63 5955056179775

在本节中我们将讨论记录在语言学的背景下出现的但都不被Toolbox软件支持的两 个任务。

**为每个条目添加字段**

很容易添加一个自动从现有字段派生出的新的字段，同时也有助于搜索和分析。例 如：例11-2中我们定义了一个函数cvO, 将辅音和元音的字符串映射到相应的 CV 序列，即： kakapua 将映射到CVCVCVV。这种映射有4个步骤。首先，将字符串 转换为小写。其次将所有非字母字符[^a-z]用下划线代替。然后，将所有元音替换 为V。 最后，所有不是V 或下划线的必定是一个辅音，所以我们将它替换为C。 现 在，我们可以扫描词汇，在每个Ix字段后面添加一个新的cv 字段。例11-2演示了 如何在特定条目上添加字段；注意输出的最后一行表示新的cv 字段。

例11-2 为词汇条目添加新的cv 字段

def cv(s):

from nltk.etree.ElementTree import SubElement

s =s.lower()

s =re.sub(r'[^a-z]', s =re.sub(r'[aeiou]',

s=re.sub(r'[^V\_]',

return (s)

r'\_',s)

r'V',s)

r'C',s)

def add\_cv\_field(entry):

for lfe='lx':

ccvv\_\_fiilld.(efld (teenrty),'cv')

>>>lexicon =toolbox.xml('rotokas.dic')

>>>add\_cv\_field(lexicon[53])

>>>print nltk.to\_sfm\_string(lexicon[53])

\lx kaeviro

\ps V

**\**teAliftoff

\ge take off

\tkp go antap

\sc MOTION

\vx 1

\nt used to describe action of plane

\dt 03/Jun/2005

\ex Pita kaeviroroe kepa kekesia oa vuripierevo kiuvu.

\xp Pita i go antap na lukim haus win i bagarapim.



\xe Peter went to look at the house that the wind destroyed. \cv CVVCVCV

如果一个 Toolbox 文件正在不断更新，例11-2将需要多次运行。可以 将 add\_cv\_field)修改为修改现有条目的内容。使用这样的程序为数据 分析创建一个附加的文件比替换手工维护的源文件要安全。

**验证** **Toolbox** **词汇**

Toolbox 格式的许多词汇不符合任何特定的模式。有些条目可能包括额外的字段， 或以一种新的方式排序现有字段。手动检查成千上万的词汇条目是不可行的。我们 可以在FreqDist 的帮助下很容易地找出频率异常的字段序列。

in ltk.FreqDist(':'.join(field.tag for field in entry)for entry

>>>fd.items()

[('lx:ps:pt:ge:tkp:dt:ex:xp:xe',41),('lx:rt:ps:pt:ge:tkp:dt:ex:xp:xe',37),

('lx:rt:ps:pt:ge:tkp:dt:ex:xp:xe:ex:xp:xe',27),('lx:ps:pt:ge:tkp:nt: dt:ex:xp:xe',20),

...,('lx:alt:rt:ps:pt:ge:eng:eng:eng:tkp:tkp:dt:ex:xp:xe;ex:xp:xe:ex:

xp:xe',1)]

检查完高频字段序列后，我们可以为词汇条目设计一个无关上下文的文法。例11-3 中的文法使用了第8章中的 CFG 格式。这样的文法模型隐含了 Toolbox 条目的嵌 套结构，建立树状结构，树的叶子是单独的字段名。我们遍历条目并报告它们与文 法的一致性，如例11-3所示。那些被文法接受的在前面加一个“+””,不被文法 接受的在前面加一个“-”②。在开发这样一个文法的过程中，它可以帮助过滤掉 一些标签③。

例11-3 使用上下文无关文法验证 Toolbox 中的条目

grammar =nltk.parse\_cfg('''

S ->Head PS Glosses Comment Date Sem\_Field Examples

Head -> Lexeme Root

Lexeme ->"lx"

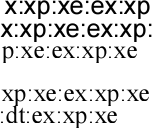
Root ->"rt"l

PS ->"ps"

Gloss ->"ge"l"tkp"l"eng"

Glosses ->Gloss Glosses l

eamte\_Fidt"->"sf"

----- :g:p::e:t:n::td:t:ett:ex:xp:xe

Examples ->Example Ex Pidgin Ex English Examples l

\_ \_

Example ->"ex"

Ex\_Pidgin ->"xp"

Ex\_English ->"xe"

Comment

' )

->"cmt"l"nt"|

def validate\_lexicon(grammar,lexicon,ignored\_tags):

rd\_parser =nltk.RecursiveDescentParser(grammar)

for entry in lexicon:

marker\_list=[field.tag for field in entry if field.tag not in

ignored\_tags]

if rd\_parser.nbest\_parse(marker\_list):

print "+",':'.join(marker\_list) ①

else:

print"-",':'.join(marker\_list)②

>>>lexicon =toolbox.xml('rotokas.dic')[10:20]

>>>ignored\_tags =['arg','dcsv','pt','vx'] ③

>>>validate\_lexicon(grammar,lexicon, ignored\_tags)

lx:rt:ps:ge:eng:eng:eng:ge:tkp:tkp:dt:cmt:ex:xp:xe;ex:xp:xe;ex:xp:xe:

ex:x-pl-l:e:ex:xp:xe

另一种方法是使用块分析器(见第7章),因为它能识别局部结构并报告已确定的 局部结构。例11-4 中我们为词汇条目建立一个块文法。这个程序输出的一个示例 如图11-7所示。

例11-4 为 Toolbox 词典分块：此块文法描述了一种中国方言的词汇条目结构

from nltk\_contrib import toolbox

grammar =rmmm

lexfunc:{<1f>(<lv><ln|le>\*)\*}

example: {<rf|xv><xn|xe>\*}

sense: {<sn><ps><pn|gv|dv|gn|gp|dn|rn|ge|delre>\*<example>\*<lexfunc>\*} record: {<1x><hm><sense>+<dt>}

n H #

>>>from nltk.etree.ElementTree import ElementTree

>>>db =toolbox.ToolboxData()

>>>db.open(nltk.data.find('corpora/toolbox/iu\_mien\_samp.db'))

>>>lexicon =db.parse(grammar,encoding='utf8')

>>>toolbox.data.indent(lexicon)

>>>tree =ElementTree(lexicon)

>>>output =open("iu\_mien\_samp.xml","w")

>>>tree.write(output,encoding='utf8')

>>>output.close()

|  |
| --- |
| **<record>**  <1x>ceuv jiax</1x>  **<hm** />  <sense>  <sn />  <ps>vobj</ps>  <dv>nzaeng jiax</av>  <ge>quarrel</ge>  >吵/>架</gn>  <gp>chao3 jia4</gp>  <dn>争吵</dn>  <example>  <xv>Ninh mbuo i hmuangv mv -jiex jiax.</xv>  <xe>That husband and wife have never quarrelled.</xe?  <xn>他们夫妻俩从来不吵架。</xn>  </example><example>  >>GHonvgev ge n e d.<v't>quarrel.</xe>  <xn>讲道理，别吵架 . </xn>  </example><lexfunc>  <1f />  <1v />  </lexfunc>  </sense><dt>18/Feb/2004</at>  </record> |

**图11-** **7** **一个词条的** **XML** **表示，对** **Toolbox** **记录的块分析结果**

**11.6** **使用** **OLAC** **元数据描述语言资源**

NLP 社区成员的一个共同需要是发现具有很高精度和召回率的语言资源。数字图 书馆社区目前已开发的解决方案包括元数据聚集。

**元数据是什么?**

元数据最简单的定义是“关于数据的结构化数据”。元数据是对象或资源的描述信 息，无论是物理的还是电子的。即使术语“元数据”本身是相对较新的，只要收集 的信息被组织起来，元数据下面隐含的意义一直在被使用。图书馆目录是一种行之

有效的元数据类型；它们已经作为资源管理和发现工具存在几十年了。元数据可以 由“手工”产生也可以使用软件自动生成。

都柏林核心元数据倡议 (Dublin Core Metadata Initiative) 于1995年开始开发信息 发现、共享和管理的约定。都柏林核心元数据元素表示一个广泛的、跨学科一致的 元素核心集合，这些元素核心集合有可能对资源发现有广泛帮助。都柏林核心由 15 个元数据元素组成，其中每个元素都是可选的和可重复的，它们是：标题、创 建者、主题、描述、发布者、参与者、日期、类型、格式、标识符、来源、语言、 关系、覆盖范围和版权。此元数据集可以用来描述数字或传统的格式中存放的资源。

开放档案倡议 (Open Archives Initiative,OAI) 提供了一个跨越数字化的学术资料 库的共同框架，而不考虑资源的类型，包括文档、资料、软件、录音、实物制品、 数码代替品等。每个库由一个网络访问服务器组成，为归档项目提供公共访问。每 个项目都有一个唯一的标识符，并与都柏林核心元数据记录(也可以是其他格式的 记录)关联。OAI 为元数据搜索服务定义了一个协议来“收获”资源库的内容。

**OLAC: 开放语言档案社区**

开放语言档案社区 (Open Language Archives Community,OLAC) 是正在创建的一 个世界性语言资源的虚拟图书馆的机构和个人的国际伙伴关系： (i) 制定目前最好 的关于语言资源的数字归档实施的共识； (ii) 开发、存储和访问这些资源的互操作 信息库和服务的网络。OLAC的主页在htp:/www.language-archives.org/。

OLAC元数据是描述语言资源的标准。通过限制某些元数据元素的值为使用受控词 表中的术语，确保跨库描述的统一性。OLAC 元数据可用于描述物理和数字格式的 数据和工具。OLAC 元数据扩展了都柏林核心元数据集(一个描述所有类型的资源 被广泛接受的标准)。对这个核心集， OLAC 添加了语言资源的基本属性，如：主 题语言和语言类型。下面是一个完整的OLAC 记录例子。

<?xml version="1.0"encoding="UTF-8"?>

<olac:olac xmlns:olac="<http://www.language-archives.org/OLAC/1.1/>"

xmlns="<http://purl.org/dc/elements/1.1/>"

xmlns:dcterms="<http://purl.org/dc/terms/>"

xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

xsi:schemaLocation="<http://www.language-archives.org/OLAC/1.1/>

<http://www.language-archives.org/OLAC/1.1/olac.xsd>">

<title>Agrammar of Kayardild.With comparative notes on Tangkic.</title>

<creator>Evans,Nicholas D.</creator>

<subject>Kayardild grammar</subject>

<subject xsi:type="olac:language"olac:code="gyd">Kayardild</subject> <language xsi:type="olac:language"olac:code="en">English</language> <description>Kayardild Grammar(ISBN 3110127954)</description>

:tac ler">Nicholas Evans

</contributor>

<format>hardcover,837 pages</format>

<relation>related to ISBN 0646119966</relation>

rage>stic-type"olac:code="language\_description"/>

<type xsi:type="dcterms:DCMIType">Text</type>

</olac:olac>

以XML格式发布语言档案的目录，这些记录会定期被使用OAI协议的OLAC服务“收 获”。除了这个软件基础设施外，OLAC 通过与语言资源社区广泛咨询的过程，已经证 明了它是一系列描述语言资源的最佳实践(如：参见<http://www.language-archives.org/> REC/bpr.html。)。

OLAC 网站上使用一个查询引擎可以搜索 OLAC 库。搜索“German lexicon”可以 发现以下资源。

· CALLHOME德语词典<http://www.language-archives.org/item/oai:www.ldc.upenn>. edu:LDC97L18

● MULTILEX 多语种词典 <http://www.language-archives.org/item/oai:elra.icp.inpg>. fr:M0001

● Slelex 西门子语音词汇 <http://www.language-archives.org/item/oai:elra.icp.inpg>. fr:S0048

搜索“Korean” 会发现新闻语料库、树库、词库、儿童语言语料库和行间注释文本。

它还可以发现软件，包括语法分析器和词法分析器。

可以看到上面的网址包括子字符串的形式： oai:www.ldc.upenn.edu:LDC97L18。这 是OAI 标识符，使用在ICANN(the Internet Corporation for Assigned Names and Numbers, 互联网名称和编号分配公司)注册的 URI 方案。这些标识符格式是： oai:archive:localid, 其中 oai 是 URI 方案的名称，archive 是一个档案标识符，如

www.ldc.upenn.edu,local\_id 是档案分配的资源标识符，如 LDC97L18。

给定OLAC 资源的 OAI 标识符，可以使用以下形式的URL 检索资源的完整的XML 记录： <http://www.language-archives.org/static-records/oai:archive:local>\_id。

**11.7** **小结**

· 大多数语料库中基本数据类型是已标注的文本和词汇。文本有时间结构，而词 汇有记录结构。

● 语料库的生命周期，包括数据收集、标注、质量控制及发布。发布后生命周期 仍然继续，因为语料库会在研究过程中被修改和丰富。

· 语料库开发包括捕捉语言使用的代表性的样本与使用任何一个来源或文体都

有足够的材料之间的平衡；增加变量的维度通常由于资源的限制而不可行。

● XML 提供了一种有用的语言数据的存储和交换格式，但解决普遍存在的数据 建模问题是没有捷径的。

· Toolbox 格式被广泛使用在语言记录项目中；可以编写程序来支持 Toolbox 文 件的维护，将它们转换成XML。

· 开放语言档案社区 (OLAC) 提供了用于记录和发现语言资源的基础设施。

**11.8** **深入阅读**

本章的其他材料发布在 <http://www.nltk.org/>, 包括网上免费资源的链接。

语言学语料库的首要来源是语言数据联盟 (Linguistic Data Consortium) 和欧洲语 言资源局 ( European Language Resources Agency), 两者都有广泛的在线目录。本 书中提到的主要语料库的细节也有介绍：美国国家语料库(Reppen,Ide &Suderman, 2005)、英国国家语料库(BNC,1999),Thesaurus Linguae Graecae(TLG,1999)、儿 童语言数据交换系统 (MacWhinney,1995) 和 TIMIT(Garofolo et al.,1986)。

计算语言学协会定期组织研讨会发布论文集，它的两个特别兴趣组是：SIGWAC 和

SIGANN。前者推动使用网络作为语料，去除 HTML 标记的 CLEANEVAL; 后者 鼓励对语言注解的互操作性的努力。(Croft,Metzler& Strohman,2009) 提供了网络 搜索器的扩展讨论。

(Buseman,Buseman &Early,1996) 提供 Toolbox 数据格式的全部细节，最新的发 布可以从 <http://www.sil.org/computing/toolbox/>免费下载。构建一个 Toolbox 词典的 过程指南参见<http://www.sil.org/computing/ddp/>。我们在 Toolbox上努力地将更多的 例子记录在(Bird,1999) 和(Robinson,Aumann&Bird,2007) 。(Bird&Simons,2003)

调查了语言数据管理的几十种其他工具。也请参阅关于文化遗产数据的语言技术的 LaTeCH研讨会的论文集。

还有很多优秀的XML 资源(如 <http://zvon.org/>) 和编写 Python 程序处理 XML 的 资源 <http://www.python.org/doc/lib/markup.html>。许多编辑器都有XML 模式。XML 格式的词汇信息包括 OLIF(<htp://www.olif.net/>) 和 LIFT(<htp://code.google.com/> p/lift-standard/)。

对于语言标注软件的调查，见 <http://www.ldc.upenn.edu/annotation/>的语言标注页。 不同注解最初的设计见(Thompso n&McKelvie, 1997)。语言标注的一个抽象的数 据模型称为“标注图”在 (Bird&Liberman,200 1) 中提出。语言描述的一个通用 本体 (GOLD) 记录在<http://www.linguistics-ontology.org/>中。

有关规划和建设语料库的指导，请参阅 (Meyer,2 002) 和 (Farghaly,20 03)。关于 标注者之间一致性得分的方法的更多细节，见(Artstein &Poesio,2008)和(Pevzner &Hearst,2002)。

Rotokas 数据由Stuart Robinson提供，勉方言(Iu Mien) 数据由GregAumann 提供。

有关开放语言档案社区的更多信息，请访问<http://www.language-archives.org/>, 或参 见 (Simons&Bird,2003)。

**11.9** **练习**

1.①在例11-2中新字段出现在条目底部。修改这个程序使它就在Ix 字段后面插入 新的子元素。(提示：使用 Element('cv')创建新的 cv 字段，分配给它一个文本值，

然后使用父元素的insert) 方法。)

2.①编写一个函数，从一个词汇条目删除指定的字段。(我们可以在把数据给别人 之前用它做些清洁，如：删除包含无关或不确定的内容的字段。)

3.①编写一段程序，扫描一个HTML 字典文件，找出具有非法词性字段的条目，

并报告每个条目的核心词。

4.①编写一段程序，找出所有出现少于10次的词性 (ps 字段)。或许是否有 打字错误?

5.①我们学习到增加cv字段的方法(见11.5节)。当修改的Ix字段的内容时，需 要保持这个字段始终最新。为这个程序编写一个版本，添加cv 字段，取代所有现 有的cv 字段。

6.①编写一个函数，添加一个新的字段syl, 计数一个词中的音节数。

7.①编写一个函数，显示一个词位的完整条目。当词位拼写错误时，它应该显示 拼写最相似的词位的条目。

8.①编写一个函数，从一个词典中找出最频繁的连续字段对(如： ps 后面往往是 pt) 。(这可以帮助我们发现一些词条的结构。)

9.①使用办公软件创建一个电子表格，每行包含一个词条，词条包括中心词、词 性和注释。以CSV格式保存电子表格。编写Python代码来读取CSV文件并以Toolbox 格式输出。使用Ix 表示中心词，ps 表示词性， gl 表示注释。

10.①在 nltk.Index 帮助下，索引莎士比亚的戏剧中的词。产生的数据结构允许按 单个词查找，如： music, 返回演出、场景和台词的引用链表，[(3,2,9),(5,1,23),…] 的形式，其中(3,2,9)表示第3场演出场景2台词9。

11.①构建一个条件频率分布记录《威尼斯商人》中每段台词的词长，以角色名字 为条件；如： cfd[PORTIA'][12]给出Portia的包含12个词的台词的数目。

12.①编写一个递归函数将任意NLTK 树状图转换为对应的 XML, 其中非终结符

不能表示成 XML元素，叶表示文本内容，如下所示。

<S

<NP>

<NNP>Pierre</NNP>

<NNP>Vinken</NNP>

</NP>

<COMMA>,</COMMA>

13.●获取 CSV 格式的比较词表，编写一段程序，输出相互之间至少有3个编辑 距离的同源词。

14. ● 建立一个出现在例句的词位的索引。假设对于一个给定条目的词位是w。然 后为这个条目添加一个单独的交叉引用字段xrf, 引用其他有例句包含w 的条目的 中心词。对所有条目进行该操作，结果保存为 Toolbox 格式文件。

**后记：语言的挑战**

自然语言带来了一些有趣的计算性挑战。我们已经在前面的章节探讨过许多这样的挑 战，包括分词、标注、分类、信息提取和建立句法和语义表示。你现在应该已经具备 了操作大型数据集来创建强大语言现象的模型，并将它们扩展到实际语言技术的组件 中。我们希望自然语言工具包 (NLTK) 已经做出了关于实用自然语言处理的令人振奋 的努力，并希望能有更广泛的受众。

尽管有了前面的讨论，语言带给我们的不仅仅是临时的计算性挑战。考虑下面的句子， 它们证明了语言的丰富性。

(1)Overhead the day drives level and grey,hiding the sun by a flight of grey spears. (William Faulkner,As I Lay Dying,1935)

(2)When using the toaster please ensure that the exhaust fan is tumed on.(sign in dormitory kitchen)

(3)Amiodarone weakly inhibited CYP2C9,CYP2D6,and CYP3A4-mediated activities

with Ki values of 45.1-271.6μM (Medline,PMID:10718780)

(4)Iraqi Head Seeks Arms (spoof news headline)

(5)The earnest prayer of a righteous man has great power and wonderful results.(James 5:16b)

(6)Twas brillig,and the slithy toves did gyre and gimble in the wabe (Lewis Carroll, Jabberwocky,1872)

(7)There are two ways to do this,AFAIK :smile:(Internet discussion archive)

语言丰富性的其他证据是以语言为工作中心的学科多样性。 一些常见的学科包括翻译、 文学评论、哲学、人类学和心理学。许多不常见的学科主要研究语言的使用，包括法

律、诠释学、辩论术、电话学、教育学、考古学、密码分析学及言语病理学。它们分 别应用不同的方法来收集现象、发展理论和测试假设。它们都有助于加深我们对语言 和表现在语言上的理解。

鉴于语言的复杂性和从不同的角度研究它的广泛兴趣，很显然，我们仅仅只触及了表 面。此外，NLP 的本身也有许多我们没有提到的重要方法和应用。

在后记中，我们将以更宽广的视角看待 NLP, 包括它的基础和你可能想要探索的进一 步方向。 一些主题还没有得到 NLTK 很好的支持，你也许可以通过为工具包贡献新的 软件和数据，来修正这些问题。

**语言处理与符号处理**

以计算方式处理自然语言的观念来自于一个研究项目，可以追溯到20世纪初，使用逻 辑重建数学推理，Frege 、Russell 、Wittgenstein 、Tarski 、Lambek 和 Carnap 的工作清楚 地表明了这一点。这项工作使得语言可以作为自动处理形式化系统的概念。之后的3 个开发奠定了自然语言处理的基础。第一个是**形式语言理论。**它定义了一种语言作为 能被一类自动机接受的字符串集合，如：上下文无关语言和下推自动机，并提供了计 算句法的基础。

第二个开发是**符号逻辑。**它提供了一个形式化方法，用以捕捉自然语言的选定方面，这 与表达逻辑证明有关。符号逻辑中的形式化演算提供一种语言的句法和推理规则，也有 可能提供了一套理论模型的规则解释；例子是命题逻辑和一阶逻辑。给定这样的演算和 明确的句法和语义，这样就可能通过将自然语言的表达翻译成形式化演算的表达式，联 系语义与自然语言的表达。例如：如果把John saw Mary 翻译为公式 saw(j,m), (或明 或暗地)将英语动词 saw 解释为二元关系，而John 和 Mary 表示个体元素。更多的一般 性表达式如：All birds fly 需要量词，在这个例子中是√,意思是“对所有的”:Vx(bird(x)

→fly(x))。逻辑的使用提供了技术性的机制来处理推理，而推理是语言理解的重要组成 部分。

另一个密切相关的开发是**组合原则，** 即一个复杂表达式的意思是由它的各个部分的意 思和它们的组合模式组成的(见第10章)。这一原则在句法和语义之间提供了有用的 对应关系，即一个复杂的表达式的含义可以递归地计算。考虑句子： It is not true that p, 其中p 是一个命题。我们可以表示这个句子的意思为not(p)。同样，我们可以表示John saw Mary 的意思为 saw(j,m) 。 现在，我们可以使用上述信息递归地计算It is not true that John saw Mary 的表示，得到 not(saw(j,m))。

刚刚简要介绍的方法都有一个前提：自然语言计算关键依赖于操纵符号表示的规则。 NLP发展的一个特定时期，特别是在20世纪80年代，这个前提为语言学家和 NLP 从 业人员提供了一个共同的起点，产生了一种被称为基于归一(基于特征)的形式化文 法家族(见第9章),以及可以在 Prolog 编程语言上实现 NLP 应用。虽然基于文法的 自然语言处理仍然是一个重要的研究领域，但由于种种因素，在过去的15～20年它已 经有些黯然失色了。 一个显著的影响因素来自于自动语音识别。虽然早期的语音处理 采用一种类似于基于规则的音韵处理的模型，典型的例子如：Sound Pattern of English (Chomsky&Halle,1968), 但结果远远不能够解决像实时地识别实际讲话这样困难的问 题。相比之下，包含来自于大量语音数据的学习模式的系统明显更准确、更高效和更 稳健。此外，语言社区发现为测试常见数据性能的定量测量建立的共享资源有助于建 立更好的系统。最终，大部分的NLP 社区接受了面向**数据密集型**的语言处理技术，与 此同时越来越多地使用机器学习技术并以评价为主导的方法。

**当代哲学划分**

在上一节中描述的自然语言处理的两种方法对比，与在西方哲学的启蒙时期出现的关 **于理性主义与经验主义和现实主义与理想主义**的早期形而上学的辩论有关。这些辩论 出现在反对当时正统思想(认为一切知识的来源是神的启示)的背景下，在17世纪和 18世纪期间，哲学家认为人类理性或感官经验优先于启示。笛卡尔和莱布尼兹及其他 人采取了理性的立场，声称所有的真理来源于人类思想，从出生起我们的脑海中就植入的 “天赋观念”的存在。例如：他们认为欧几里德几何原理是使用人的理性制定的，而不是 超自然的启示或感官体验的结果。相比之下，洛克和其他人采取了经验主义的观点，认为 我们的知识的主要来源是我们的感官经验，人类理性在反应这些经验上起次要作用。这一 立场经常引用的证据是伽利略的发现——基于对行星运动的仔细观察——太阳系是以太 阳为中心，而不是地球为中心。在语言学的背景下，本次辩论导致以下问题：人类语言经 验与我们先天的“语言能力”各自在多大程度上作为我们的语言知识的基础?在 NLP 中这个问题表现为在计算模型构建中语料库数据与语言学反省之间的优先级如何?

还有一个问题，在现实主义和理想主义之间的辩论中被奉若神明的是理论结构的形而 上学的地位。康德主张现象应与我们可以体验的表现及不能直接被认识的“事情本身” 之间相互区别开来。语言现实主义者会认为如**名词短语这**样的理论建构是一个现实世 界的实体，是人类看法和理由的独立存在，它实际导致了观测到的语言现象。另一方 面，语言理想主义者认为名词短语及如语义表示这样更抽象的结构本质上无法观察到， 只是担任着有用的虚构的角色。语言学家编写理论的方式往往与现实主义的立场相违

背，而NLP 从业人员持中立立场，或倾向于理想主义立场。因此，在NLP 中，只要一 个理论的抽象得到了有用的结果就可以了，而不管这个结果是否揭示了人类语言处理 任何光辉。

这些问题今天仍然存在，表现为符号与统计方法、深层与浅层处理、二元与梯度分 类，以及科学与工程目标之间的区别。然而，这样的反差现在已经非常细微，辩论 不再像从前那样是两极分化了。事实上，大多数的讨论 大部分的进展 都包 含“平衡协调”。例如： 一种中间立场是假设人类天生被赋予基于类比和记忆的学 习方法(弱理性主义),并使用这些方法确定他们在感官语言经验(经验主义)中 有意义的模式。

整本书中，我们已经看到了这种方法的很多例子。每次语料统计指导上下文无关文法 产生式的选择时，统计方法就会给出符号模型，即“文法工程”。每当使用基于规则的 方法创建的语料被用作特征来源训练统计语言模型时，符号方法都会给出统计语言模 型，即“文法推理”。这个过程是封闭的。

**NLTK** **的路线图**

自然语言工具包是在不断发展的，随着人们贡献的代码而不断扩大。NLP 和语言学的 一些领域(还)没有得到 NLTK 很好的支持。有关这本书的出版之后的开发新闻，请 查阅 <http://www.nltk.org/>。特别鼓励对以下方面的研究。

*音韵学和形态学：*

研究声音模式和文字结构的计算方法，通常使用有限状态机工具包。如不规 则现象如词形变化和非拼接形态，很难用我们学习的字符串处理方法进行处 理。该技术面临的挑战不仅仅是连接 NLTK 到一个高性能的有限状态机工具 包，而且要避免词典数据的重复，以及链接形态分析器和语法分析器所需形 态学特征。

*高性能模块：*

某 些 NLP 任务的计算量太大，使纯 Python 无法实现。然而，在某些情况下， 耗时只出现在训练模型期间，而不是使用它们标注输入的期间。NLTK 中的 程序包系统以一种简便的方式发布训练好的模型，即使是使用不能随意分 布的语料库训练的模型。替代方法是开发高性能的机器学习工具的 Python 接口，或通过使用如 MapReduce 的并行编程技术扩展 Python 的能力。

*词汇语义学：*

这是一个充满活力的领域，目前的研究大多围绕词典、本体、多词表达式等的 继承模型，大都在现在的 NLTK 的范围之外。目标是从丰富的外部存储获得词 汇信息，以支持词义消歧、解析和语义解释等任务。

*自然语言生成：*

从含义的内在表示生产连贯的文本是NLP 的重要组成部分。用于NLP 的基于 归一的方法已经在NLTK 中开发，在这一领域还能做出更大的贡献。

*语言实地调查：*

语言学家面临的一个重大挑战是记录数以千计的濒危语言，这项工作将产生大 量异构且快速变化的数据。更多实地调查数据格式，包括行间的文本格式和词 汇交换格式，可以在NLTK 中得到支持，用以帮助语言学家维护和分析这些数 据，释放它们，使它们能在数据提炼中花费尽可能多的时间。

*其他语言：*

对英语以外的语言的 NLP 改进支持包括两方面的工作：获准发布更多 NLTK 中收集的语料库；编写特定语言的 HOWTO 文件发布到 http://www.nltk. org/howto, 说明在 NLTK 中的使用，讨论语言相关的 NLP 问题，包括字符编 码、分词、形态。特定语言专业的 NLP 研究人员可以安排翻译这本书，并在 NLTK 的网站上保存一个副本；这将不仅仅是翻译讨论的内容，而要使用目标 语言的数据提供等效的可行的例子，这是一项繁琐的工作。

*NLTK-Contrib.*

许多 NLTK 中的核心组件都由NLP 社区成员贡献，它们最初被安置在 NLTK 中的 “Contrib” 包的 nltk\_contrib 中。对添加到这个包中的软件的唯一要求 是它必须用 Python 编写，与 NLP 有关，并给予与NLTK 中其他软件一样的开 源许可。不完善的软件也是值得欢迎的，随着时间的推移可能会被 NLP 社区 的其他成员改进。

*教材：*

从NLTK 开发的最初起，教材一直伴随着软件逐渐扩大用了填补这本书，再加

上大量的网上材料。我们希望教师们能够提供这些材料包括：幻灯片、习题集、 解答集、有关主题更详细的理解，并通知作者，这样我们就可以在 <http://www.nltk.org/> 上做链接。具有特殊价值的材料使NLP 成为计算机科学和 语言学系的本科主流课程，或者使NLP 出现在二级本科课程中，包括语言、 文学、计算机科学及信息技术课程中的计算内容。

*只是一个工具包：*

在序言中已经指出， NLTK 是工具包，而不是系统。在 NLTK 、Python、其他 Python 库、外部NLP 的工具和格式的接口集成中会有很多问题需要解决。

**Envoi …**

语言学家有时会被问到会说多少种语言，其实这一领域实际上关注的是语言间共享的 抽象结构的研究，是一种比学说尽可能多的语言更深刻更难以捉摸的研究。同样的， 计算机科学家有时会被问到他们懂多少种编程语言，其实计算机科学实际上关注的是 能在任何编程语言中实施的数据结构和算法的研究，是一种比争取学习尽可能多的编 程语言更深刻更难以捉摸的研究。

这本书涵盖了有关自然语言处理领域的许多主题。大多数的例子都使用了 Python 和英 语。不过，如果读者得出的结论：NLP 是关于如何编写 Python程序操纵的英文文本， 或者更广泛的，关于如何编写程序(以任何一种编程语言)处理(任何一种自然语言) 的文本，这就很难解决了。我们选择 Python 和英语是权宜之计，仅此而已。即使我们 关注编程本身也只是一种解决问题的手段：作为一种了解表示和操纵语言标注文本的 集合的数据结构和算法的方式，作为一种方法来建立新的语言技术，更好地服务于信 息社会的需求，并最终作为对人类语言丰富性的深层次的理解方法。

**参考文献**

[Abney,1989]Steven P.Abney.A computational model of human parsing.Journal of Psycholinguistic Research,18:129-144,1989.

[Abney,1991]Steven P.Abney.Parsing by chunks.In Robert C.Berwick,Steven P. Abney,and Carol Tenny,editors,Principle-Based Parsing:Computation and Psycho- linguistics,volume 44 of Studies in Linguistics and Philosophy.Kluwer Academic Pub- lishers,Dordrecht,1991.

[Abney,1996a]Steven Abney.Part-of-speech tagging and partial parsing.In Ken Church,Steve Young,and Gerrit Bloothooft,editors,Corpus-Based Methods in Lan- guage and Speech.Kluwer Academic Publishers,Dordrecht,1996.

[Abney,1996b]Steven Abney.Statistical methods and linguistics.In Judith Klavans

and Philip Resnik,editors,The Balancing Act:Combining Symbolic and Statistical Ap-

*proaches to Language.MIT Press,1996.*

[Abney,2008]Steven Abney.Semisupervised Learning for Computational Linguistics. Chapman and Hall,2008.

[Agirre and Edmonds,2007]Eneko Agirre and Philip Edmonds.Word Sense Disam-

*biguation:Algorithms and Applications.Springer,2007.*

[Alpaydin,2004]Ethem Alpaydin.Introduction to Machine Learning.MIT Press,2004.

[Ananiadou and McNaught,2006]Sophia Ananiadou and John McNaught,editors.

*Text Mining for Biology and Biomedicine.Artech House,2006.*

[Androutsopoulos et al.,1995]lon Androutsopoulos,Graeme Ritchie,and Peter Tha- nisch.Natural language interfaces to databases—an introduction.Journal of Natural Language Engineering,1:29-81,1995.

[Artstein and Poesio,2008]Ron Artstein and Massimo Poesio.Inter-coder agreement for computational linguistics.Computational Linguistics,pages 555-596,2008.

[Baayen,2008]Harald Baayen.Analyzing Linguistic Data:A Practical Introduction to Statistics Using R.Cambridge University Press,2008.

[Bachenko and Fitzpatrick,1990]J.Bachenko and E.Fitzpatrick.A computational grammar of discourse-neutral prosodic phrasing in English.Computational Linguis- tics,16:155-170,1990.

[Baldwin &Kim,2010]Timothy Baldwin and Su Nam Kim.Multiword Expressions. in Nitin Indurkhya and Fred J.Damerau,editors,Handbook of Natural Language Pro- cessing,second edition.Morgan and Claypool,2010.

[Beazley,2006]David M.Beazley.Python Essential Reference.Developer's Library. Sams Publishing,third edition,2006.

[Biber et al.,1998]Douglas Biber,Susan Conrad,and Randi Reppen.Corpus Linguis-

*tics:Investigating Language Structure and Use.Cambridge University Press,1998.*

[Bird,1999]Steven Bird.Multidimensional exploration of online linguistic field data.

In Pius Tamanji,Masako Hirotani,and Nancy Hall,editors,Proceedings of the 29th

*Annual Meeting of the Northeast Linguistics Society,pages 33-47.GLSA,University of*

Massachussetts at Amherst,1999.

[Bird and Liberman,2001]Steven Bird and Mark Liberman.A formal framework for linguistic annotation.Speech Communication,33:23-60,2001.

[Bird and Simons,2003]Steven Bird and Gary Simons.Seven dimensions of portability for language documentation and description.Language,79:557-582,2003.

[Blackburn and Bos,2005]Patrick Blackburn and Johan Bos.Representation and In-

*ference for Natural Language:A First Course in Computational Semantics.CSLIPubli-*

cations,Stanford,CA,2005.

[BNC,1999]BNC.British National Corpus,1999.[<http://info.ox.ac.uk/bnc/>].

[Brent and Cartwright,1995]Michael Brent and Timothy Cartwright.Distributional regularity and phonotactic constraints are useful for segmentation.In Michael Brent,

*editor,Computational Approaches to Language Acquisition.MIT Press,1995.*

[Bresnan and Hay,2006]Joan Bresnan and Jennifer Hay.Gradient grammar:An effect of animacy on the syntax of give in New Zealand and American English.Lingua 118: 254-59,2008.

[Budanitsky and Hirst,2006]Alexander Budanitsky and Graeme Hirst.Evaluating wordnet-based measures of lexical semantic relatedness.Computational Linguistics, 32:13-48,2006.

[Burton-Roberts,1997]Noel Burton-Roberts.Analysing Sentences.Longman,1997.

[Buseman et al.,1996]Alan Buseman,Karen Buseman,and Rod Early.The Linguist's

*Shoebox:Integrated Data Management and Analysis for the Field Linguist. WaxhawNC:*

SIL,1996.

[Carpenter,1992]Bob Carpenter.The Logic of Typed Feature Structures.Cambridge University Press,1992.

[Carpenter,1997]Bob Carpenter.Type-Logical Semantics.MIT Press,1997.

[Chierchia and McConnell-Ginet,1990]Gennaro Chierchia and Sally MeConnell-Gi-

*net.Meaning and Grammar:An Introduction to Meaning.MIT Press,Cambridge,MA,*

1990.

[Chomsky,1965]Noam Chomsky.Aspects of the Theory of Syntax.MIT Press,Cam- bridge,MA,1965.

[Chomsky,1970]Noam Chomsky.Remarks on nominalization.In R.Jacobs and P. Rosenbaum,editors,Readings in English Transformational Grammar.Blaisdell,Wal- tham,MA,1970.

[Chomsky and Halle,1968]Noam Chomsky and Morris Halle.The Sound Pattern of English.New York:Harper and Row,1968.

[Church and Patil,1982]Kenneth Church and Ramesh Patil.Coping with syntactic ambiguity or how to put the block in the box on the table.American Journal of Com- putational Linguistics,8:139-149,1982.

[Cohen and Hunter,2004]K.Bretonnel Cohen and Lawrence Hunter.Natural lan-

guage processing and systems biology.In Werner Dubitzky and Francisco Azuaje,ed-

*itors,Artificial Intelligence Methods and Tools for Systems Biology,page 147-174*

Springer Verlag,2004.

[Cole,1997]Ronald Cole,editor.Survey of the State of the Art in Human Language Technology.Studies in Natural Language Processing.Cambridge University Press, 1997.

[Copestake,2002]Ann Copestake.Implementing Typed Feature Structure Grammars. CSLI Publications,Stanford,CA,2002.

[Corbett,2006]Greville G.Corbett.Agreement.Cambridge University Press,2006.

[Croft et al.,2009]Bruce Croft,Donald Metzler,and Trevor Strohman.Search Engines: Information Retrieval in Practice.Addison Wesley,2009.

[Daelemans and van den Bosch,2005]Walter Daelemans and Antal van den Bosch.

*Memory-Based Language Processing.Cambridge University Press,2005.*

[Dagan et al.,2006]Ido Dagan,Oren Glickman,and Bernardo Magnini.The PASCAL recognising textual entailment challenge.In J.Quinonero-Candela,I.Dagan,B.Mag- nini,and F.d'Alché Buc,editors,Machine Learning Challenges,volume 3944 of Lecture Notes in Computer Science,pages 177-190.Springer,2006.

[Dale et al.,2000]Robert Dale,Hermann Moisl,and Harold Somers,editors.Handbook of Natural Language Processing.Marcel Dekker,2000.

[Dalrymple,2001]Mary Dalrymple.Lexical Functional Grammar,volume 34 of Syntax and Semantics.Academic Press,New York,2001.

[Dalrymple et al.,1999]Mary Dalrymple,V.Gupta,John Lamping,and V.Saraswat.

Relating resource-based semantics to categorial semantics.In Mary Dalrymple,editor,

*Semantics and Syntax in Lexical Functional Grammar:The Resource Logic Approach,*

pages 261-280.MIT Press,Cambridge,MA,1999.

[Dowty et al.,1981]David R.Dowty,Robert E.Wall,and Stanley Peters.Introduction to Montague Semantics.Kluwer Academic Publishers,1981.

[Earley,1970]Jay Earley.An efficient context-free parsing algorithm.Communications

*of the Association for Computing Machinery,13:94-102,1970.*

[Emele and Zajac,1990]Martin C.Emele and Rémi Zajac.Typed unification gram- mars.In Proceedings of the 13th Conference on Computational Linguistics,pages 293- 298.Association for Computational Linguistics,Morristown,NJ,1990.

[Farghaly,2003]Ali Farghaly,editor.Handbook for Language Engineers.CSLI Publi- cations,Stanford,CA,2003.

[Feldman and Sanger,2007]Ronen Feldman and James Sanger.The Text Mining

*Handbook:Advanced Approaches in Analyzing Unstructured Data.Cambridge Univer-*

sity Press,2007.

[Fellbaum,1998]Christiane Fellbaum,editor.WordNet:An Electronic Lexical Data-

*base.MIT Press,1998.http://wordnet.princeton.edul.*

[Finegan,2007]Edward Finegan.Language:Its Structure and Use.Wadsworth,Fifth edition,2007,

[Forsyth and Martell,2007]Eric N.Forsyth and Craig H.Martell.Lexical and discourse analysis of online chat dialog.In Proceedings of the First IEEE International Conference on Semantic Computing,pages 19-26,2007.

[Friedl,2002]Jeffrey E.F.Friedl.Mastering Regular Expressions.O'Reilly,second ed- ition,2002

[Gamut,199la]L.T.F.Gamut.Intensional Logic and Logical Grammar,volume 2 of

*Logic,Language and Meaning.University of Chicago Press,Chicago,1991.*

[Gamut,1991b]L.T.F.Gamut.Introduction to Logic,volume 1 of Logic,Language and Meaning.University of Chicago Press,1991.

[Garofolo et al.,1986]John S.Garofolo,Lori F.Lamel,William M.Fisher,Jonathon G.Fiscus,David S.Pallett,and Nancy L.Dahlgren.The DARPA TIMIT Acoustic-

*Phonetic Continuous Speech Corpus CDROM.NIST,1986.*

[Gazdaretal.,1985]Gerald Gazdar,Ewan Klein,Geoffrey Pullum,and Ivan Sag(1985).

*Generalized Phrase Structure Grammar.Basil Blackwell,1985.*

[Gomes et al.,2006]Bruce Gomes,William Hayes,and Raf Podowski.Text mining. In Darryl Leon and Scott Markel,editors,In Silico Technologies in Drug Target Identi- fication and Validation,Taylor &Francis,2006.

[Gries,2009]Stefan Gries.Quantitative Corpus Linguistics with R:A Practical Intro- duction.Routledge,2009.

[Guzdial,2005]Mark Guzdial.Introduction to Computing and Programming in Python: A Multimedia Approach.Prentice Hall,2005.

[Harel,2004]David Harel.Algorithmics:The Spirit of Computing.Addison Wesley, 2004.

[Hastie et al.,2009]Trevor Hastie,Robert Tibshirani,and Jerome Friedman.The El-

*ements of Statistical Learning:Data Mining,Inference,and Prediction.Springer,second*

edition,2009.

[Hearst,1992]Marti Hearst.Automatic acquisition of hyponyms from large text cor-

*pora.In Proceedings of the 14th Conference on Computational Linguistics (COLING),*

pages 539-545,1992.

[Heim and Kratzer,1998]Irene Heim and Angelika Kratzer.Semantics in Generative

*Grammar.Blackwell,1998.*

[Hirschman et al.,2005]Lynette Hirschman,Alexander Yeh,Christian Blaschke,and Alfonso Valencia.Overview of BioCreAtlvE:critical assessment of information extrac tion for biology.BMC Bioinformatics,6,May 2005.Supplement 1.

[Hodges,1977]Wilfred Hodges.Logic.Penguin Books,Harmondsworth,1977.

[Huddleston and Pullum,2002]Rodney D.Huddleston and Geoffrey K.Pullum.The

*Cambridge Grammar of the English Language.Cambridge University Press,2002.*

[Hunt and Thomas,2000]Andrew Hunt and David Thomas.The Pragmatic Program- mer:From Journeyman to Master.Addison Wesley,2000.

[Indurkhya and Damerau,2010]Nitin Indurkhya and Fred Damerau,editors.Hand- book of Natural Language Processing.CRC Press,Taylor and Francis Group,second edition,2010.

[Jackendoff,1977]Ray Jackendoff.X-Syntax:a Study of Phrase Strucure.Number 2 in Linguistic Inquiry Monograph.MIT Press,Cambridge,MA,1977.

[Johnson,1988]Mark Johnson.Attribute Value Logic and Theory of Grammar.CSLI Lecture Notes Series.University of Chicago Press,1988.

[Jurafsky and Martin,2008]Daniel Jurafsky and James H.Martin.Speech and Language Processing.Prentice Hall,second edition,2008.

[Kamp and Reyle,1993]Hans Kamp and Uwe Reyle.From Discourse to the Lexicon:

*Introduction to Modeltheoretic Semantics of Natural Language,Formal Logic and Dis-*

course Representation Theory.Kluwer Academic Publishers,1993.

[Kaplan,1989]Ronald Kaplan.The formal architecture of lexical-functional grammar. In Chu-Ren Huang and Keh-Jiann Chen,editors,Proceedings of ROCLING II,pages 1-18.CSLI,1989.Reprinted in Dalrymple,Kaplan,Maxwell,and Zaenen(eds),Formal

*Issues in Lexical-Functional Grammar,pages 7-27.CSLI Publications,Stanford,CA,*

1995.

[Kaplan and Bresnan,1982]Ronald Kaplan and Joan Bresnan.Lexical-functional

grammar:A formal system for grammatical representation.In Joan Bresnan,editor,

*The Mental Representation of Grammatical Relations,pages 173-281.MIT Press,Cam-*

bridge,MA,1982.

[Kasper and Rounds,1986]Robert T.Kasper and William C.Rounds.A logical se- mantics for feature structures.In Proceedings of the 24th Annual Meeting of the Asso- ciation for Computational Linguistics,pages 257-266.Association for Computational Linguistics,1986.

[Kathol,1999]Andreas Kathol.Agreement and the syntax-morphology interface in HPSG.In Robert D.Levine and Georgia M.Green,editors,Studies in Contemporary

*Phrase Structure Grammar,pages 223-274.Cambridge University Press,1999.*

[Kay,1985]Martin Kay.Unification in grammar.In Verónica Dahl and Patrick Saint-

*Dizier,editors,Natural Language Understanding and Logic Programming,pages 233-*

240.North-Holland,1985.Proceedings ofthe First International Workshop on Natural Language Understanding and Logic Programming.

[Kiss and Strunk,2006]Tibor Kiss and Jan Strunk.Unsupervised multilingual sentence boundary detection.Computational Linguistics,32:485-525,2006.

[Kiusalaas,2005]Jaan Kiusalaas.Numerical Methods in Engineering with Python.Cam- bridge University Press,2005.

[Klein and Manning,2003]Dan Klein and Christopher D.Manning.A'parsing:Fast exact viterbi parse selection.In Proceedings of HLT-NAACL 03,2003.

[Knuth,2006]Donald E.Knuth.The Art of Computer Programming,Volume 4:Gen- erating All Trees.Addison Wesley,2006.

[Lappin,1996]Shalom Lappin,editor.The Handbook of Contemporary Semantic Theory.Blackwell Publishers,Oxford,1996.

[Larson and Segal,1995]Richard Larson and Gabriel Segal.Knowledge of Meaning:An Introduction to Semantic Theory.MIT Press,Cambridge,MA,1995.

[Levin,1993]Beth Levin.English Verb Classes and Alternations.University of Chicago Press,1993.

[Levitin,2004]Anany Levitin.The Design and Analysis of Algorithms.Addison Wesley, 2004.

[Lutz and Ascher,2003]Mark Lutz and David Ascher.Learning Python.O'Reilly,sec- ond edition,2003.

[MacWhinney,1995]Brian MacWhinney.The CHILDES Project:Tools for Analyzing

Talk.Mahwah,NJ:Lawrence Erlbaum,second edition,1995.[http://childes.psy.cmu

*.edu/].*

[Madnani,2007]Nitin Madnani.Getting started on natural language processing with Python.ACM Crossroads,13(4),2007.

[Manning,2003]Christopher Manning.Probabilistic syntax.In Probabilistic Linguis- tics,pages 289-341.MIT Press,Cambridge,MA,2003.

[Manning and Schütze,1999]Christopher Manning and Hinrich Schütze.Foundations

*of Statistical Natural Language Processing.MIT Press,Cambridge,MA,1999.*

[Manning et al.,2008]Christopher Manning,Prabhakar Raghavan,and Hinrich Schü- tze.Introduction to Information Retrieval.Cambridge University Press,2008.

[McCawley,1998]James McCawley.The Syntactic Phenomena of English.University of Chicago Press,1998.

[McConnell,2004]Steve McConnell.Code Complete:A Practical Handbook ofSoftware Construction.Microsoft Press,2004.

[McCune,2008]William McCune.Prover9:Automated theorem prover for first-order and equational logic,2008.

[McEnery,2006]Anthony McEnery.Corpus-Based Language Studies:An Advanced Resource Book.Routledge,2006.

[Melamed,2001]Dan Melamed.Empirical Methods for Exploiting Parallel Texts.MIT Press,2001

[Mertz,2003]David Mertz.Text Processing in Python.Addison-Wesley,Boston,MA, 2003.

[Meyer,2002]Charles Meyer.English Corpus Linguistics:An Introduction.Cambridge University Press,2002.

[Miller and Charles,1998]George Miller and Walter Charles.Contextual correlates of semantic similarity.Language and Cognitive Processes,6:1-28,1998.

[Mitkov,2002a]Ruslan Mitkov.Anaphora Resolution.Longman,2002.

[Mitkov,2002b]Ruslan Mitkov,editor.Oxford Handbook of Computational Linguis- tics.Oxford University Press,2002.

[Müller,2002]Stefan Muller.Complex Predicates:Verbal Complexes,Resultative Con- structions,and Particle Verbs in German.Number 13 in Studies in Constraint-Based

Lexicalism.Center for the Study of Language and Information,Stanford,2002.http://

*www.dfki.del~stefan/Publcomplex.html.*

[Nerbonne et al.,1994]John Nerbonne,Klaus Netter,and Carl Pollard.German in

*Head-Driven Phrase Structure Grammar.CSLI Publications,Stanford,CA,1994.*

[Nespor and Vogel,1986]Marina Nespor and Irene Vogel.Prosodic Phonology.Num- ber 28 in Studies in Generative Grammar.Foris Publications,Dordrecht,1986.

[Nivre et al.,2006]J.Nivre,J.Hall,and J.Nilsson.Maltparser:A data-driven parser- generator for dependency parsing.In Proceedings of LREC,pages 2216-2219,2006.

[Niyogi,2006]Partha Niyogi.The Computational Nature of Language Learning and Evolution.MIT Press,2006.

[O'Grady et al,2004]William O'Grady,John Archibald,Mark Aronoff,and Janie Rees-Miller.Contemporary Linguistics:An Introduction.St.Martin's Press,fifth edition, 2004.

[OSU,2007]OSU,editor.Language Files:Materials for an Introduction to Language and Linguistics.Ohio State University Press,tenth edition,2007.

[Partee,1995]Barbara Partee.Lexical semantics and compositionality.In L.R.Gleit- man and M.Liberman,editors,An Invitation to Cognitive Science:Language,volume 1,pages 311-360.MIT Press,1995.

[Pasca,2003]Marius Pasca.Open-Domain Question Answering from Large Text Col- lections.CSLI Publications,Stanford,CA,2003.

[Pevzner and Hearst,2002]L.Pevzner and M.Hearst.A critique and improvement of an evaluation metric for text segmentation.Computational Linguistics,28:19-36,2002.

[Pullum,2005]Geoffrey K.Pullum.Fossilized prejudices about “however”,2005.

[Radford,1988]Andrew Radford.Transformational Grammar:An Introduction.Cam- bridge University Press,1988.

[Ramshaw and Marcus,1995]Lance A.Ramshaw and Mitchell P.Marcus.Text chunk- ing using transformation-based learning.In Proceedings of the Third ACL Workshop on Very Large Corpora,pages 82-94,1995.

[Reppen et al.,2005]Randi Reppen,Nancy Ide,and Keith Suderman.American Na tional Corpus.Linguistic Data Consortium,2005.

[Robinson et al.,2007]Stuart Robinson,Greg Aumann,and Steven Bird.Managing

fieldwork data with toolbox and the natural language toolkit.Language Documentation

*and Conservation,1:44-57,2007.*

[Sag and Wasow,1999]Ivan A.Sag and Thomas Wasow.Syntactic Theory:A Formal

*Introduction.CSLI Publications,Stanford,CA,1999.*

[Sampson and McCarthy,2005]Geoffrey Sampson and Diana McCarthy.Corpus Lin- guistics:Readings in a Widening Discipline.Continuum,2005.

[Scott and Tribble,2006]Mike Scott and Christopher Tribble.Textual Patterns:Key

*Words and Corpus Analysis in Language Education.John Benjamins,2006.*

[Segaran,2007]Toby Segaran.Collective Intelligence.O'Reilly Media,2007.

[Shatkay and Feldman,2004]Hagit Shatkay and R.Feldman.Mining the biomedical literature in the genomic era:An overview.Journal of Computational Biology,10:821- 855,2004.

[Shieber,1986]Stuart M.Shieber.An Introduction to Unification-Based Approaches to Grammar,volume 4 of CSLI Lecture Notes Series.CSLI Publications,Stanford,CA, 1986.

[Shieber etal.,1983]Stuart Shieber,Hans Uszkoreit,Fernando Pereira,Jane Robinson, and Mabry Tyson.The formalism and implementation of PATR-II.In Barbara J.Grosz and Mark Stickel,editors,Research on Interactive Acquisition and Use of Knowledge,

techreport 4,pages 39-79.SRI International,Menlo Park,CA,November 1983.(http:

*//www.eecs.harvard.edu/shieber/Biblio/Papers/Shieber-83-FIP.pdf)*

[Simons and Bird,2003]Gary Simons and Steven Bird.The Open Language Archives Community:An infrastructure for distributed archiving oflanguage resources.Literary and Linguistic Computing,18:117-128,2003.

[Sproat et al.,2001]Richard Sproat,Alan Black,Stanley Chen,Shankar Kumar,Mari

Ostendorf,and Christopher Richards.Normalization of non-standard words.Com-

*puter Speech and Language,15:287-333,2001.*

[Strunk and White,1999]William Strunk and E.B.White.The Elements of Style.Bos- ton,Allyn and Bacon,1999.

[Thompson and McKelvie,1997]Henry S.Thompson and David McKelvie.Hyperlink

semantics for standoff markup of read-only documents.In SGML Europe '97,1997.

[*http://www.Itg.ed.ac.uk/*](http://www.Itg.ed.ac.uk/)*~ht/sgmleu97.html.*

[TLG,1999]TLG.Thesaurus Linguae Graecae,1999.

[Turing,1950]Alan M.Turing.Computing machinery and intelligence.Mind,59(236): 433-460,1950.

[van Benthem and ter Meulen,1997]Johan van Benthem and Alice ter Meulen,editors.

*Handbook of Logic and Language.MIT Press,Cambridge,MA,1997.*

[van Rossum and Drake,2006a]Guido van Rossum and Fred L.Drake.An Introduction to Python—The Python Tutorial.Network Theory Ltd,Bristol,2006.

[van Rossum and Drake,2006b]Guido van Rossum and Fred L.Drake.The Python Language Reference Manual.Network Theory Ltd,Bristol,2006.

[Warren and Pereira,1982]David H.D.Warren and Fernando C.N.Pereira.An effi-

cient easily adaptable system for interpreting natural language queries.American Jour-

*nal of Computational Linguistics,8(3-4):110-122,1982.*

[Wechsler and Zlatic,2003]Stephen Mark Wechsler and Larisa Zlatic.The Many Faces of Agreement.Stanford Monographs in Linguistics.CSLI Publications,Stanford,CA, 2003.

[Weiss et al.,2004]Sholom Weiss,Nitin Indurkhya,Tong Zhang,and Fred Damerau. Text Mining:Predictive Methods for Analyzing Unstructured Information.Springer, 2004.

[Woods et al.,1986]Anthony Woods,Paul Fletcher,and Arthur Hughes.Statistics in Language Studies.Cambridge University Press,1986.

[Zhao and Zobel,2007]Y.Zhao and J.Zobel.Search with style:Authorship attribution in classic literature.In Proceedings of the Thirtieth Australasian Computer Science Con- ference.Association for Computing Machinery,2007.

Programming/Python

**O'REILLY°**

**Python** **自然语言处理**

从输入法联想提示 (predictive text) 、E-mail过滤到 自动文本汇总、机器翻译，大量的语言相关的技术都 离不开自然语言处理的支持，而这本书提供了自然语 言处理的入门指南。通过本书，你将学到如何编写能

处理大量非结构化文本的Python 程序。你可以访问具有丰富标注 的、涵盖了语言学各种数据结构的数据集，而且你将学习分析书 面文档的内容和结构的主要算法。

通过大量的例子和练习，本书将帮助你：

**·从非结构化文本中提取信息，无论是猜测主题还是识别“命** **名实体”;**

**·分析文本的语言学结构，包括语法和语义分析；**

**·访问流行的语言学数据集，包括WordNet** **和treebanks;**

**·整合从语言学到人工智能等多个领域的技术。**

通过使用Python和自然语言工具包 (NTLK) 的开源库，本书将帮 助你增加自然语言处理的实际经验。如果你对开发Web应用、分 析多语言的新闻源或记录濒危语言感兴趣，或者只是想要从程序 员的视角看人类语言如何运作，你将发现本书不仅有趣而且极其 有用。

**“少有的一本书，用如此清晰的方** **法、如此优美整洁的代码处理** **一个如此复杂的计算机问题……** **这是一本从中可以学习自然语** **言处理的书。’**

——Ken Getz MCW Technologies高级顾问

Steven Bird是墨尔本大学计算 机科学和软件工程系副教授， 宾夕法尼亚大学语言学数据联 盟高级研究助理。

Ewan Klein是爱丁堡大学信息 学院语言技术教授。

Edward Loper是毕业于宾夕法 尼亚大学专注于机器学习的自 然语言处理方向的博士，现在 在波士顿的BBN Technologies担 任研究员。

**oreilly.com.cn**

封面设计： Karen Montgomery, 张健

OReilly Media,Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆(不包含中国香港、澳门特别行政区和中国台湾地区)销售发行

This Authorized Edition for sale only in the territory of People's Republic of China

(excluding Hong Kong,Macao and Taiwan)

分类建议：计算机/程序设计/Python ISBN 978-7-115-33368-1

人民邮电出版社网址： www.ptpress.com.cn 定价：89.00元