**NLP\_chapter2**

**第二章 获得文本语料和词汇资源**

**2.1 获取文本语料库**

**古腾堡语料库**

>>> nltk.corpus.gutenberg.fileids() 获取该语料库中的文件标识符

可以直接定义变量进行简单的语言处理：

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

若想使用第一章中的语言处理函数，需作如下定义：

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

另一种定义方式：

>>> from nltk.corpus import gutenberg

>>> gutenberg.fileids()

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

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

通过循环遍历前面列出的 gutenberg 文件标识符链表相应的 fileid，然后计算统计每个文本:

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

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

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)

raw() 函数给我们没有进行过任何语言学处理的文件的内容。因此，例如:len(gutenberg.raw(' blake-poems.txt')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。

**网络和聊天文本**

网络文本集合：

>>> from nltk.corpus import webtext

即时消息聊天会话语料库：

>>> from nltk.corpus import nps\_chat

**布朗语料库**

>>> from nltk.corpus import brown

布朗语料库是一个研究文体之间的系统性差异 ——一种叫做文体学的语言学研究——

很方便的资源，按照文体分类

统计不同文体中的情态动词的频率分布：

>>> 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)

**路透社语料库**

包含 10,788 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。其中fileid 为“test/14826”的文档属于测试组

与布朗语料库不同，路透社语料库的类别是有互相重叠的

**在函数名的后面加链表[x:x]以获取函数指定位置的部分**

**就职演说语料库**

是 55 个文本的集合，每个文本都是一个总统的演说。具有时间维度

>>> from nltk.corpus import inaugural

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，我们使用 f

ileid[:4]提取前四个字符。

>>> [fileid[:4] for fileid in inaugural.fileids()]

**标注文本语料库**

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。

**在其他语言的语料库**

NLTK包含包含多国语言语料库

udhr，是超过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

其中中文版本的fileid为‘Chinese\_Mandarin-GB2312’

Python的plot函数无法显示汉字（用raw函数提取）作为图像的坐标轴

**文本语料库的结构**

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，文本集有一个时间结构，新闻集合是最常见的例子。

NLTK中定义的基本语料库函数:

fileids() 语料库中的文件

fileids([categori es]) 这些分类对应的语料库中的文件

categories() 语料库中的分类

categories([fileids]) 这些文件对应的语料库中的分类

raw() 语料库的原始内容

raw(fileids=[f1,f2, f3] ) 指定文件的原始内容

raw(categories=[c1,c2]) 指定分类的原始内容

words() 整个语料库中的词汇

words( fileids=[f1, f2, f3]) 指定文件中的词汇

words( categories=[c1, c2] ) 指定分类中的词汇

sents() 整个语料库中的句子

sents (fileids=[f1,f2, f3] ) 指定文件中的句子

sents(categories=[c1,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

**载入自己的语料库**

使用NLTK中的 PlaintextCorpusReader 帮助下载入它们

将变量 corpus\_root 的值设置为文件所在目录

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fileids的模式，如:'[abc]/.\*\.txt',注意文件格式的前面一定要加.\*\,如'.\*\.txt’

>>> from nltk.corpus import PlaintextCorpusReader

>>> corpus\_root = '/usr/share/dict' �

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

或者可以用另一个函数访问

>>> from nltk.corpus import BracketParseCorpusReader

两者使用方法和功能比较相似，而BracketParseCorpusReader更适合已解析过的语料库，PlainTextCorpusReader更适合文本文件

**2.2 条件频率分布**

当语料文本被分为几类(文体、主题、作者等)时，我们可以计算每个类别独立的频率分布。这将允许我们研究类别之间的系统性差异。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。

**条件和事件**

条件频率分布处理的是配对序列，每个事件需要关联一个条件

**按文体计数词汇**

ConditionalFreqDist函数需要配对序列作为对象，所以在使用时需先定义一个配对序列，再对序列中的两个链表分别赋值，如：

>>> cfd = nltk.ConditionalFreqDist(

(genre,word)

for genre in brown.categories()

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

也可以先定义好配对序列，直接对函数应用，如：

>>> cfd = nltk.ConditionalFreqDist(genre\_word)

其中genre\_word是已经定义好了的配对序列

它的一些简单应用与FreqDist函数基本相同，区别仅在于多了一个条件

可以输入变量的名称来检查它，以及确认它有几个条件：

>>> cfd

>>> cfd.conditions()

每一个条件都只是一个频率分布

**绘制分布图和分布表**

计数特定演讲中出现的词的次数：

>>> cfd = nltk.ConditionalFreqDist(

(target,fileid[:4])

for fileid in inaugural.fileids()

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

for w in inaugural.words(fileid)

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

这里的条件是文本内的内容而不是文本的分类所以需作一些特别的处理，这段代码为文件 1865-Lincoln.txt 中每 个小写形式以america开头的词——如:Americans——产生一个配对('america', '1865')。

在 plot()和 tabulate()方法中，我们可以使用 conditions= parameter 来选择指定哪些条件显示。如果我们忽略它，所有条件都会显示。同样，我们可以使用 samples= p arameter 来限制要显示的样本。

通常，使用链表推导作为一个函数的参数时，可以忽略掉方括号

**使用双连词生成随机文本**

此程序获得《创世记》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随机产生文本：

def generate\_model(cfdist, word, num=15): for i in range(num):

print word,

word = cfdist[word].max()

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

cfd = nltk.ConditionalFreqDist(bigrams) �

**NLTK 中的条件频率分布的常用方法和习惯用法**

cfdist= ConditionalFreqDist(pairs) 从配对链表中创建条件频率分布

cfdist.conditions() 将条件按字母排序

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) 指定样本和条件限制下制表

cfdist.plot() 为条件频率分布绘图

cfdist.plot(samples, conditions) 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

**2.4 词典资源**

词典或者词典资源是一个词和/或短语以及一些相关信息的集合，词典资源附属于文本，通常在文本的帮助下创建和丰富。

一个词项包括词目(也叫词条)以及其他附加信息，两个不同的词拼写相同被称为同音异义词。

NLTK 中的一些词典资源：

**词汇列表语料库**

NLTK包括一些仅仅包含词汇列表的语料库。

>>> from nltk.corpus import words

使用difference()函数来寻找两个链表间不同的词汇，如：

A.difference(B) 寻找并打印只在A出现不在B中出现的词汇

还有一个停用词语料库，就是那些高频词汇，如:the，to

>>> from nltk.corpus import stopwords

字母拼词谜题问题字母不重复出现问题的解决方法：

FreqDist 比较法允许我们检查每个字母在候选词中的频率是否小于或等于相应的字母在拼词谜题中的频率，即先定义一个变量包含题目中所给出的所有字母且每个字母仅出现一次：

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

设置一个条件为两语料库的FreqDist函数相比较：

… nltk.FreqDist(w) <= puzzle\_letters]

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。

>>> names = nltk.corpus.names

**发音的词典**

一个稍微丰富的词典资源是一个表格(或电子表格)，在每一行中含有一个词加一些性质。NLTK 中包括美国英语的 CMU 发音词典，它是为语音合成器使用而设计的。

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

对每一个词，这个词典资源提供语音的代码——不同的声音不同的标签——叫做音素。

CMU发音词典中的符号是从 Arpabet 来的

每个条目由两部分组成，单词字符串和发音链表

对于可分的变量，可以对几个变量同时赋值等于该变量，但变量的数目必须严格等于该变量含有的可拆分变量的数目，如：

For word,pron in entries

Ph1,ph2,ph3 = pron

可以有几种方法来拼读同一个读音，而有些发音与书写之间不匹配

寻找最后一个发音为‘M’但以‘n’结尾的词：

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

寻找以‘N’发音开头但首字母不是n的词缀：

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

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:

>>> def stress(pron):

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

这里用了双层嵌套循环，第一层循环将每一个单独的音素提取出来赋值给临时变量phone，第二层循环将音素字符串中的数字提取出来赋值给变量char，受限于entries变量的结构，仅用一层循环无法实现该功能

可以使用条件频率分布来帮助我们找到词汇的最小受限集合,这里主要起到分类作用，频率没有多大意义，输出需要做一些特殊处理。下例为找到所 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，仅输出含有十个以上词汇的分组的一部分词汇：

>>> p3 = [(pron[0]+'-'+pron[2],word)

for (word,pron) in entries

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

>>> cfd = nltk.ConditionalFreqDist(p3)

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

if len(cfd[template])>10:

words = sorted(cfd[template])

wordstring=' '.join(words)

print(template,wordstring[:70]+'...')

可以通过查找特定词汇来访问词典，使用 Python 的词典数据结构。通过指定词典的名字后面跟一个包含在方括号里的关键字(例如:词 fire)来查词典：

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

>>> prondict['fire']

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]]

这里[w]后面加[0]的原因是natural和language都有两个发音，只要求显示一个即可

**比较词表**

NLTK中包含了所谓的斯瓦迪士核心词列表(Swadesh wordlists)，几种语言中约 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

我们可以通过在 entries()方法中指定一个语言链表来访问多语言中的同源词。更进一步，我们可以把它转换成一个简单的词典

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

>>> translate = dict(fr2en)

>>> translate['chien']

由于这里定义的是法语在前英语在后，所以只能检索法语翻译成英语，反之会报错，即只能**单向翻译**

通过添加其他源语言，我们可以让这个简单的翻译器更为有用。可以通过添加的映射更新原来的翻译词典。

>>> de2en = swadesh.entries(['de', 'en']) # German-English

>>> es2en = swadesh.entries(['es', 'en']) # Spanish-English

>>> translate.update(dict(de2en))

>>> translate.update(dict(es2en))

**2.5 WordNet**

WordNet 是面向语义的英语词典，类似与传统辞典，但具有更丰富的结构。

**意义与同义词**

寻找词的同义词集：

>>> from nltk.corpus import wordnet as wn

>>> wn.synsets('motorcar')

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

car.n.01 被称为 synset 或“**同义词集**”，意义相同的词(或“词条”)的集合:

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

同义词集也有一些一 般的定义和例句:

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

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

虽然定义帮助人们了解一个同义词集的本意，同义词集中的词往往对我们的程序更有用。为了消除歧义，我们将这些词标注为 car.n.01.automobile，car.n.01.motorcar等。 这种同义词集和词的配对叫做**词条**。我们可以得到指定同义词集的所有词条：

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

查找特定的词条：

>>> wn.lemma('car.n.01.automobile')

得到一个词条对应的同义词集：

>>> wn.lemma('car.n.01.automobile').synset()

也可以得到一个词条的“名字”:

>>> wn.lemma('car.n.01.automobile').name()

以上语句的时候需注意synset和lemma语句的单复数形式

Python3.0版本区别于2.0版本，以上函数的末尾必须加（）

寻找词的所有词条：

>>> wn.lemmas('car')

输出单词dish的所有含义：

>>> for dish in wn.lemmas('dish'):

print(dish.synset().definition())

**WordNet 的层次结构**

WordNet的同义词集对应于抽象的概念，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。一些很一般的概念被称为**独一无二的根同义词集**。

WordNet使在概念之间漫游变的容易。我们可以看到一个宽泛的概念的更加具体(直接)的概念——**下位词**。

>>> motorcar = wn.synset('car.n.01')

>>> types\_of\_motorcar = motorcar.**hyponyms()**

注意下位词的概念是相对于同义词集而言的

我们也可以通过访问**上位词**来浏览层次结构。有些词有多条路径，因为它们可以归类在一个以上的分类中。

>>> motorcar.hypernyms()

>>> paths = motorcar.hypernym\_paths() #上位词路径，即层次结构

>>> [synset.name() for synset in paths[0]]

>>> [synset.name() for synset in paths[1]]

我们可以用如下方式得到一个最一般的上位(或根上位)同义词集:

>>> motorcar.root\_hypernyms()

NLTK 中有便捷的图形化 WordNet浏览器:nltk.app.wordnet()

**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。这个关系定位上下为 “是一个”层次。WordNet网络另一个重要的漫游方式是从物品到它们的部件(部分)或到它们被包含其中的东西(整体)。

例如:一棵树的部分是它的树干，树冠等;这些都是 part\_meronyms():

>>> wn.synset('tree.n.01').part\_meronyms()

一棵树的实质是包括心材和边材组成的，即 substance\_meronyms():

>>> wn.synset('tree.n.01').substance\_meronyms()

树木的集合形成了一个森林，即 member\_holonyms():

>>> wn.synset('tree.n.01').member\_holonyms()

在synsets函数里可以指定单词的词性，如：

wn.synsets('mint', wn.NOUN) #mint的名词性同义词集

动词之间也有关系。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

词有多个蕴涵：

>>> wn.synset('walk.v.01').entailments()

词条之间的一些词汇关系，如:**反义词**:

>>> wn.lemma('supply.n.02.supply').antonyms()

可以使用 dir()查看词汇关系和同义词集上定义的其它方法。

**语义相似度**

如果两个同义 词集共用一个非常具体的上位词——在上位词层次结构中处于**较低层的上位词**——它们一 定有密切的联系。

查找两同义词集的最底层共同上位词：

>>>'synset'.lowest\_common\_hypernyms('synset')

通过查找每个同义词集深度量化它的具体程度：

'synset'.min\_depth()

WordNet同义词集的集合上定义了类似的函数能够深入的观察,即判别两同义词集间的语义相似度。例如:path\_similarityassigns 是基于上位词层次结构中相互连接的概念之间的最短路径在 0-1 范围的打分(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。相似程度越低，该返回值就越小：

'synset'.path\_similarity('synset')

可以输入 help(wn)获得更多其他相似性度量方法的信息。NLT K 还包括 VerbNet，一个连接到 WordNet的动词的层次结构的词典。