# 第1章 自然语言处理简介

现在，让我们先从介绍自然语言处理（NLP）开始吧。众所周知，语言是人们日常生活的核心部分，任何与语言问题相关的工作都会显得非常有意思。希望这本书能带您领略到NLP的风采，激起学习NLP的兴趣。我们希望您能了解一下这一领域中那些令人惊叹的概念，并在工作中实际尝试一些具有挑战性的NLP应用。

在英语环境中，语言处理研究这一领域通常被简称为NLP。对语言有深入研究的人通常被叫做语言学家，而“计算机语言学家”这个专用名词则指的是将计算机科学应用于语言处理领域的人。因此从本质上来说，一个计算语言学家应该既有足够的语言理解能力，同时还可以用其计算机技能来模拟出语言的不同方面。虽然计算机语言学家们主要研究的是语言处理理论，但NLP无疑是对计算机语言学的具体应用。

NLP多数情况下指的是计算机上各种大同小异的语言处理应用，以及用NLP技术所构建的实际应用程序。在实践环境中，NLP与教孩子学语音的过程非常类似。其大多数任务（譬如单词、语句的理解，形成语法和结构都正确的语句等）对于人类而言都是非常自然的能力。但对于NLP来，其中有一些任务就必须要转向标识化处理、语块分解、词性标注，语法解析、机器翻译、语音识别这些领域的一部分，且这些任务大部分还仍是当前计算机界中最为棘手的挑战。在本书中，我将更侧重于讨论NLP的实用方面，因此我们会假设读者在NLP上已经有了一些背景知识。所以，读者最好在最低限度上对编程语言有一点了解，并对NLP和语言学保有一定的兴趣。

通过本章的介绍，我们希望读者能：

* 对NLP及其相关概念有个基本的了解。
* 完成Python和NLTK及其它库的安装。
* 编写一些非常基本的Python和NLTK代码片段。

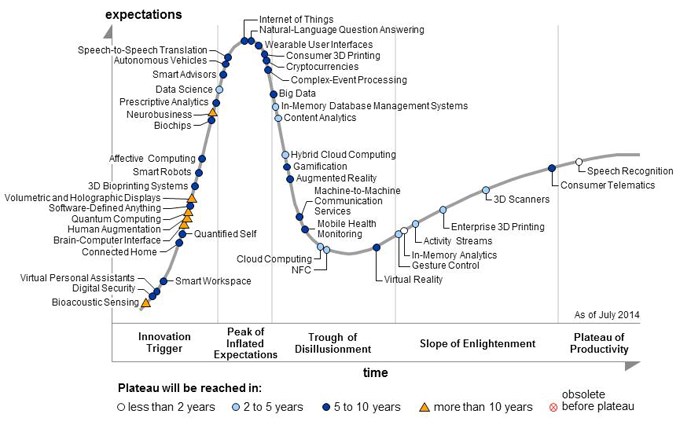
如果您从来没有接触过NLP这个概念词，我们在下面给您推荐了两本书，请花一些时间阅读一下其中的任何一本——只需要看看它们的初始几章即可。另外，您也应该快速浏览一下Wikipedia上与NLP相关的页面：

* 《Speech and Language Processing》，由Daniel Jurafsky与James H. Martin所著
* 《Statistical Natural Language Processing》，由Christopher D. Manning与Hinrich Schütze所著

## 为什么要学习NLP?

关于这个问题，我们可以先来看看Gartner公司新一轮的趋势报告，您可以很清晰地看到，NLP技术赫然高居榜首。目前，NLP已被认为是业界最为稀缺的技能之一。自大数据的概念问世之后，我们所面对的主要挑战是，业界需要越来越多不仅能处理结构化数据，同时也能处理半结构化或非结构化数据的人才。对于我们所产生出来的那些博客、微博，Facebook订阅，聊天信息，E-mail以及网络评论。各公司都在致力于收集所有不同种类的数据，以便建立更好的客户针对性，形成有意义的见解。而要想处理所有的这些非结构化数据源，我们就需要人们掌握一些NLP技能。

身处信息时代，我们甚至不能想象生活中没有Google会是什么样子。我们会因一些最基本的事情而用到Siri；我们会需要用垃圾过滤器来过滤垃圾邮件；我们会需要在自己的Word文档中用到拼写检查器等等。在现实世界中所要用到的NLP应用数不胜数。



该图摘自http://www.gartner.com/newsroom/id/2819918

在这里，我们可以再列举一些令人惊叹的NLP应用实例，您虽然很可能已经用过，但未必知道它们是基于NLP技术来构建的：

* 拼写校正（MS Word/其它编辑器）
* 搜索引擎（Google、Bing、Yahoo!、WolframAlpha）
* 语音引擎（Siri、Google Voice）
* 垃圾邮件分类（所有电子邮件服务）
* 新闻订阅（Google、Yahoo!等）
* 机器翻译（Google翻译与其它类似服务）
* IBM Watson[[1]](#footnote-1)

构建上述这些应用都需要非常具体的技能，需要优秀的语言理解能力和能有效处理这些语言的工具。因此，这些不仅是各NLP最具优势领域的未来趋势，同时也是我们用NLP这种最独特技能所能创建的应用种类。

在实现上面提到的某些应用以及其它基本的NLP预处理时，我们有许多开源工具可用。这些工具有些是由相关组织在建立自己的NLP应用时开发的，而有些则纯粹属于开源项目。下面我们就来看一份NLP工具的小清单：

* GATE
* Mallet
* Open NLP
* UIMA
* Stanford toolkit
* Genism
* **Natural Language Tool Kit (NLTK)**

上述大多数工具都是用Java来编写的，在功能上也都很相似。其中的一些功能强大，提供了各种不同的NLP实用工具。但如果我们考虑到易用性和其对相关概念的解释度的话，NLTK的得分就非常高了。同时，NLTK也是一个非常易学的工具包，这得益于Python本身非常平缓的学习曲线（毕竟NLTK是用它编写的），人们学习起来会非常快。NLTK中收纳了绝大部分的NLP任务，它们都被实现得非常优雅，且易于使用。正是出于上述所有的这些原因，NLTK如今已成为了NLP社区最流行的库之一。

在这里，我们会假设所有读者已经对Python有所了解。如果您还不了解的话，我们会希望您先去学习一下Python。如今在互联网上可以找到大量的Python基本教程。同时能让您对该有语言进行一个快速概览的书籍也不在少数。当然，我们也会针对不同跟主题与您探讨Python中的一些特性。但就目前而言，只要您掌握了基本的Python知识，譬如列表、字符串、正则表达式以及基本的I/ O操作，就可以继续读下去了。

|  |
| --- |
| 您可以从下列任意一网站中获取Python安装包：  <https://www.python.org/downloads/>  <http://continuum.io/downloads>  <https://store.enthought.com/downloads/> |

这里我会推荐读者选用来自Anaconda或Canopy的Python发行版。因为这些发行版本身就具备了一些捆绑库，譬如SciPy、numpy、scikit等等，它们可用于数据分析及其它与NLP相关领域的应用。甚至，NLTK也会成为该发行版的一部分。

|  |
| --- |
| 请参照下面网址中的说明来安装NLTK与NLTK数据：  <http://www.nltk.org/install.html> |

下面，让我们来测试一下。

请在你们各自的操作系统中打开终端，并运行：

$ python

该命令应该会为您打开一个Python解释器：

Python 2.6.6 (r266:84292, Oct 15 2013, 07:32:41)

[GCC 4.4.7 20120313 (Red Hat 4.4.7-4)] on linux2

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

>>>

我希望您在这里会得到一个与上面情况类似的输出。当然，您也有可能会看到一个不太一样的输出，因此理想情况下，我们应该准备最新版本的Python（建议是2.7版）、GCC编译器，以及其它操作系统的细部安排。当然，我们知道Python目前最新的版本大约是3.0+，但对于其它任意的开源系统来说，我们应该保守地选择一个更稳定的版本，而不是贸然跳到最新版本。如果您已经将项目迁移到Python3.0+，那就务必请参阅下面链接中的说明。以便了解那些被添加的新特性：https://docs.python.org/3/whatsnew/3.4.html.

对于基于UNIX的系统，Python属于默认程序（无须任何设置）。而Windows用户则可以通过设置相关路径来使Python进入正常工作状态。您可以通过以下方式来确认NLTK是否已经被正确安装：

>>>import nltk

>>>print "Python and NLTK installed successfully"

Python and NLTK installed successfully

好了，我们可以准备出发了！

## 先从Python开始吧！

虽然，我们在这里并不打算对Python进行任何太过深入的探讨，但带您快速浏览一下Python的基础要点还是很有必要的。当然，为了观众着想，我们最好将这次基础性的快速回顾之旅控制在5分钟之内。在此期间，我们将会讨论到数据结构的基本知识，一些常用函数，以及我们在接下来几节中将会用到的Python通用结构。

|  |
| --- |
| 我强烈推荐您花两个小时看一下题为《Google Python class》的参考资料： <https://developers.google.com/edu/python>，那对我们来说应该算是个不错的开始。当然，您还得通过Python的官方网站https://www.python.org/来获取更多的教程及其它相关资源。 |

### 列表

列表（list）是Python中最常用的数据结构之一。它们基本上相当于其它编程语言中的数组。下面，就让我们先从Python列表所提供的最重要那些功能开始吧。

我们可以在Python控制台中进行如下尝试：

>>> lst=[1,2,3,4]

>>> # mostly like arrays in typical languages

>>>print lst

[1, 2, 3, 4]

当然，Python列表也可以用更为灵活的索引来进行访问。下面我们再来看一个例子：

>>>print 'First element' +lst[0]

在这里，您会得到如下错误信息：

TypeError: cannot concatenate 'str' and 'int' objects

这是因为Python是一种解释型编程语言，它会在对其表达式进行计算的同时检查其中的变量类型。我们在声明这些变量时无须对其进行初始化和类型声明。在这里，我们的列表中所包含的是一些整数对象，它们不能被直接关联到这里的print函数上，后者只能接受一个String对象。出于这个原因，我们需要将该列表元素转换成字符串。这个过程也被我们称之为类型转换。

>>>print 'First element :' +str(lst[0])

>>>print 'last element :' +str(lst[-1])

>>>print 'first three elements :' +str(lst[0:2])

>>>print 'last three elements :'+str(lst[-3:])

First element :1

last element :4

first three elements :[1, 2,3]

last three elements :[2, 3, 4]

### 自助功能

如果您想要详细了解Python中各种不同的数据类型和函数，最好的方法就是调用其帮助函数，譬如help()和dir(lst)。

其中，我们可以通过dir(某Python对象)命令来列出指定Python对象中所有给定的属性。譬如说，如果我们像下面这样将一个列表对象传递给该函数，它就会列出所有我们可以用列表来做的很酷的事情：

>>>dir(lst)

>>>' , '.join(dir(lst))

'\_\_add\_\_ , \_\_class\_\_ , \_\_contains\_\_ , \_\_delattr\_\_ , \_\_delitem\_\_ , \_\_

delslice\_\_ , \_\_doc\_\_ , \_\_eq\_\_ , \_\_format\_\_ , \_\_ge\_\_ , \_\_getattribute\_\_

, \_\_getitem\_\_ , \_\_getslice\_\_ , \_\_gt\_\_ , \_\_hash\_\_ , \_\_iadd\_\_ , \_\_imul\_\_

, \_\_init\_\_ , \_\_iter\_\_ , \_\_le\_\_ , \_\_len\_\_ , \_\_lt\_\_ , \_\_mul\_\_ , \_\_ne\_\_ ,

\_\_new\_\_ , \_\_reduce\_\_ , \_\_reduce\_ex\_\_ , \_\_repr\_\_ , \_\_reversed\_\_ , \_\_rmul\_\_

, \_\_setattr\_\_ , \_\_setitem\_\_ , \_\_setslice\_\_ , \_\_sizeof\_\_ , \_\_str\_\_ , \_\_

subclasshook\_\_ , append , count , extend , index , insert , pop , remove

, reverse , sort'

而通过help(某Python对象)命令，我们可以得到给定Python对象的详细文档，以及该对象的几个具体用例，譬如：

>>> help(lst.index)

Help on built-in function index:

L.index(value, [start, [stop]]) -> integer -- return first index of value.

This function raises a ValueError if the value is not present.

基本上来说，由于help和dir这两个函数可以运用于任何Python数据类型之上，因此是一个很好的学习函数和其它对象细节的方式。而且，它还提供了一些基本的使用范例，这在大多数情况下都是非常有用的。

Python的字符串类型与其它语言非常类似，但字符串操作同时也是Python最主要的特性之一。即在Python中，处理字符串会是一件非常轻松的工作。即使是那些非常简单的操作，例如字符串的切割，您也会看到相较于Java/C中的大费周章，它们在Python中是多么的简单明了。

通过我们之前用过的help函数，我们可以得到任何Python对象及函数的帮助信息。下面，我们就再来看一些对字符串这种数据类型来说最为常见的操作：

* split()：一个能基于某些分隔符来对字符串进行切割的方法。如果您没有为其提供具体参数，它就会默认空格为其分隔符。

>>> mystring="Monty Python ! And the holy Grail ! \n"

>>> print mystring.split()

['Monty', 'Python', '!', 'and', 'the', 'holy', 'Grail', '!']

* strip()：一个可以从字符串中删除其尾随空白符（譬如'\n'、'\n\r'）的方法：

>>> print mystring.strip()

>>>Monty Python ! and the holy Grail !

您会注意到“\ n”字符也被剥离了。另外，您也可以通过rstrip()和lstrip()来选择是剥离字符串左边还是右边的尾部空白符。

* upper()/lower()：我们可以用这些方法改变字符串中字母的大小写：

>>> print (mystring.upper()

>>> MONTY PYTHON !AND THE HOLY GRAIL !

* replace()：该方法可用于替换目标字符串中的某个子串：

>>> print mystring.replace('!','''''')

>>> Monty Python and the holy Grail

当然，字符串类型的函数可远不止这些。这里只是讨论了其中最常用的一些而已。

|  |
| --- |
| 您可以通过下面的链接了解更多字符串函数及其用例：  <https://docs.python.org/2/library/string.html> |

### 正则表达式

对NLP爱好者来说，正则表达式是另一个非常重要的技能。正则表达式（regular expression）是一种能对字符串进行有效匹配的模式。我们会大量使用这种模式，以求从大量凌乱的文本数据中提取出有意义的信息。下面，我们就来整体浏览一下您将会需要用到哪些正则表达式。其实，我这一生至今所用过的正则表达式无非也就是这些：

* （句点）：该表达式用于匹配除换行符\n外的任意单字符。
* \w：该表达式用于匹配某一字符或数字，相当于[a-z A-Z 0-9]
* \W（大写W）：该表达式用于匹配任意非单词性字符。
* \s（小写s）：用于匹配任意单个空白字符，包括换行、返回、制表等，相当于[\n\r\t\f]。
* \S：该表达式用于匹配单个任意非空白字符。
* \t：该表达式用于匹配制表符。
* \n：该表达式用于匹配换行符。
* \r：该表达用于匹配返回符。
* \d：该表达式用于匹配十进制数字，即[0-9]。
* ^：该表达式用于匹配相关字符串的开始位置。
* $：该表达式用于匹配相关字符串的结尾位置。
* \：该表达式用来抵消特殊字符的特殊性。譬如要匹配$符号，就在它前面加上\。

下面，我们来看一个用于查找东西的例子。在这里，MyString是我们要进行相关模式查找的目标字符串对象。字符串的子串搜索是re模块中最常见的用例之一。我们可以来看看它是如何实现的：

>>> # We have to import re module to use regular expression

>>> import re

>>> if re.search('Python',mystring):

>>> print "We found python "

>>> else:

>>> print "NO "

只要我们执行了以上代码，就会立即收到如下信息：

We found python

我们还可以使用更多正则表达式模式来进行查找。譬如说，findall()就是一个常被用于对字符串进行全部模式查找的函数。它会按照给定模式对字符串进行查找，并列出其中所有匹配的对象：

>>>import re

>>>print re.findall('!',mystring)

['!', '!']

如您所见，MyString中存在着两个“！”实例，findAll返回了这两个对象的列表。

### 字典

字典（dictionary）也是我们最常用到的一种数据结构。在其它编程语言中有时也被称之为**关联数组/存储**。字典是一种键值索引型的数据结构，其索引键可以是任意一种不可变类型的，譬如说字符串和数字都经常被用来充当索引键。

字典是一种被多种编程语言广泛用来实现诸多算法的、非常便利的数据结构。而且，Python中的字典结构还是所有的这些编程语言中最为优雅的哈希表实现之一。哈希表是一种操作起来非常容易的字典结构，其优势在于，您只需通过寥寥几段代码就可以用它建立起一个非常复杂的数据结构，而同样的任务在其它语言中可能就需要您花费很多时间，写很多代码。很显然，程序员们应该花更多时间专注于算法，而不是数据结构本身。

下面，我打算用字典结构中非常常见的一个用例来获取某段既定文本中各单词的出现频率分布。您可以看到，只需短短几行代码，我们就取得了各单词在文本中的出现频率。如果您再用任意其它语言来尝试一下相同的任务，就会明白Python是何等的奇妙：

>>># declare a dictionary

>>>word\_freq={}

>>>for tok in string.split():

>>> if tok in word\_freq:

>>> word\_freq [tok]+=1

>>> else:

>>> word\_freq [tok]=1

>>> print word\_freq

{'!': 2, 'and': 1, 'holy': 1, 'Python': 1, 'Grail': 1, 'the': 1, 'Monty':

1}

### 编写函数

和其它编程语言一样，Python也有自己的函数编写方式。在Python中，函数的定义通常会从关键字def开始，后面紧跟着相应的函数名和括号()。而所有类似于其它编程语言中的那些参数和参数类型的声明都会被放在该括号内。然后，其实际代码部分将会从冒号（:）后面开始，代码的初始行通常会是一个文档字符串（注释），接着是代码的主体部分，最后我们会以一个return语句来结束整个函数。下面我们来看个实例，这个函数实例wordfreq的开头是关键字def，它没有参数，最后以一个return语句作为结束。[[2]](#footnote-2)

>>> import sys

>>> def wordfreq (mystring):

>>> '''

>>> Function to generated the frequency distribution of the given text

>>> '''

>>> print mystring

>>> word\_freq={}

>>> for tok in mystring.split():

>>> if tok in word\_freq:

>>> word\_freq [tok]+=1

>>> else:

>>> word\_freq [tok]=1

>>> print word\_freq

>>> def main():

>>> str="This is my fist python program"

>>> wordfreq(str)

>>> if \_\_name\_\_ == '\_\_main\_\_':

>>> main()

如您所见，其代码主体与上一节中所写的完全相同，只不过我们这回以函数的形式使这段代码具备了可重用性和可读性。当然，用解释器风格来编写Python代码的做法也很常见，但从大型程序的编写实践来说，使用函数/类和某种成熟的编程范式是一个更佳的做法。而且，我们也希望用户能早日编写并运行他的第一个Python程序。对此，您需要按照以下步骤来进行：

1. 用您喜欢的文本编辑器创建一个空的Python文件mywordfreq.py。
2. 将上面的代码写入或复制到该文件中。
3. 在操作系统中打开命令行终端。
4. 在该终端中执行以下命令：  
    $ python mywordfreq,py "This is my fist python program !!"
5. 最后，您应该会得到以下输出：  
    {'This': 1, 'is': 1, 'python': 1, 'fist': 1, 'program': 1, 'my':1}

现在，相信您对Python所提供一些常见的数据结构有了一个非常基本的了解。您已经编写出一个完整的Python程序，并成功地执行了它。在我看来，这些Python导引知识已经足以让您面对本书最初这几章的挑战了。

|  |
| --- |
| 您还可以通过下面网站中的一些Python教程了解更多相关的Python命令：  <https://wiki.python.org/moin/BeginnersGuide> |

## 向NLTK迈进

尽管在这里，我们并不打算深入探讨自然语言处理理论，但也理应尽快让您实际接触一下NLTK。因此，我打算开始介绍一些NLTK的基本用例，这是一个很好的机会，您可以先为今后做类似事情做一些准备。下面，我们会从一个Python程序员们习惯的处理方式切入，演示如何用NLTK将该方式转换成一个更为高效、可靠、简洁的解决方案。

我们先来看一个纯文本分析的例子。这个例子是我们要从python官方主页上摘取部分内容。

>>> import urllib2

>>> # urllib2 is use to download the html content of the web link

>>> response = urllib2.urlopen('http://python.org/')

>>> # You can read the entire content of a file using read() method

>>> html = response.read()

>>> print len(html)

47020

目前，我们还没有得到任何关于该URL所讨论话题的线索，所以接下来，我们要先做一次探索性数据分析（EDA）。通常对于一段文本域而言，EDA可能包含了多重含义，但这里只会涉及到其中的一个简单用例，即该文档的主体术语类型。其主题是什么？它们的出现频率如何？整个分析过程还会或多或少地涉及到一些预处理层面的步骤。我们会试着先用纯Python的方式来实现它，然后改用NLTK再将其实现一次。

我们先要清理掉其中的html标签。一种可行的做法是只选取其中的标记，包括数字和字符。如果之前有在工作中使用过正则表达式，您应该可以轻松地将这些html字符串转换成一个标记列表：

>>> # Regular expression based split the string

>>> tokens = [tok for tok in html.split()]

>>> print "Total no of tokens :"+ str(len(tokens))

>>> # First 100 tokens

>>> print tokens[0:100]

Total no of tokens :2860

['<!doctype', 'html>', '<!--[if', 'lt', 'IE', '7]>', '<html', 'class="no-

js', 'ie6', 'lt-ie7', 'lt-ie8', 'lt-ie9">', '<![endif]-->', '<!--[if',

'IE', '7]>', '<html', 'class="no-js', 'ie7', 'lt-ie8', 'lt-ie9">',

'<![endif]-->', ''type="text/css"', 'media="not', 'print,', 'braille,'

...]

如您所见，上面列出了我们在处理文本内容时用不到的HTML标签和其它多余字符。当然，这个任务还有个更为简洁的版本：

>>> import re

>>> # using the split function

>>> #https://docs.python.org/2/library/re.html

>>> tokens = re.split('\W+',html)

>>> print len(tokens)

>>> print tokens[0:100]

5787

['', 'doctype', 'html', 'if', 'lt', 'IE', '7', 'html', 'class', 'no',

'js', 'ie6', 'lt', 'ie7', 'lt', 'ie8', 'lt', 'ie9', 'endif', 'if',

'IE', '7', 'html', 'class', 'no', 'js', 'ie7', 'lt', 'ie8', 'lt', 'ie9',

'endif', 'if', 'IE', '8', 'msapplication', 'tooltip', 'content', 'The',

'official', 'home', 'of', 'the', 'Python', 'Programming', 'Language',

'meta', 'name', 'apple' ...]

这样看上去已经简洁多了吧？但其实它还可以更简洁一点。在这里，我们所做的努力是尽可能地去除干扰，但那些被清理的HTML标记还会如雨后春笋般地冒出来，而且我们可能也会想以单词长度为标准，删除掉具有某一长度的单词——譬如说移除像7、8这样的元素，因为在目前情况下，这些都只是些干扰词。现在，我们要做的不是用NLTK来重复相同的任务，完成这些预处理步骤。因为所有的清理工作都可以通过调用clean\_html()函数[[3]](#footnote-3)来完成：

>>> import nltk

>>> # http://www.nltk.org/api/nltk.html#nltk.util.clean\_html

>>> clean = nltk.clean\_html(html)

>>> # clean will have entire string removing all the html noise

>>> tokens = [tok for tok in clean.split()]

>>> print tokens[:100]

['Welcome', 'to', 'Python.org', 'Skip', 'to', 'content', '&#9660;',

'Close', 'Python', 'PSF', 'Docs', 'PyPI', 'Jobs', 'Community', '&#9650;',

'The', 'Python', 'Network', '&equiv;', 'Menu', 'Arts', 'Business' ...]

很酷吧？而且，这无疑让我们的代码更简洁易行了。

下面再来看看如何获得这些术语的频率分布。当然，我们还是要从纯Python的方式做起，之后再告诉您NLTK的方式。

>>> import operator

>>> freq\_dis={}

>>> for tok in tokens:

>>> if tok in freq\_dis:

>>> freq\_dis[tok]+=1

>>> else:

>>> freq\_dis[tok]=1

>>> # We want to sort this dictionary on values ( freq in this case )

>>> sorted\_freq\_dist= sorted(freq\_dis.items(), key=operator.itemgetter(1), reverse=True)

>>> print sorted\_freq\_dist[:25]

[('Python', 55), ('>>>', 23), ('and', 21), ('to', 18), (',', 18), ('the',

14), ('of', 13), ('for', 12), ('a', 11), ('Events', 11), ('News', 11),

('is', 10), ('2014-', 10), ('More', 9), ('#', 9), ('3', 9), ('=', 8),

('in', 8), ('with', 8), ('Community', 7), ('The', 7), ('Docs', 6),

('Software', 6), (':', 6), ('3:', 5), ('that', 5), ('sum', 5)]

由于目标是Python的官方主页，Python和（>>>）解释器符号自然就成了最常用的术语，这也符合该网站给人的感觉。

当然，这个任务还有一个更好用，也更有效的方法，即调用NLTK中的FreqDist()函数。在此，我们可以来看看调用后前相同代码的比对：

>>> import nltk

>>> Freq\_dist\_nltk=nltk.FreqDist(tokens)

>>> print Freq\_dist\_nltk

>>> for k,v in Freq\_dist\_nltk.items():

>>> print str(k)+':'+str(v)

<FreqDist: 'Python': 55, '>>>': 23, 'and': 21, ',': 18, 'to': 18, 'the':

14, 'of': 13, 'for': 12, 'Events': 11, 'News': 11, ...>

Python:55

>>>:23

and:21

,:18

to:18

the:14

of:13

for:12

Events:11

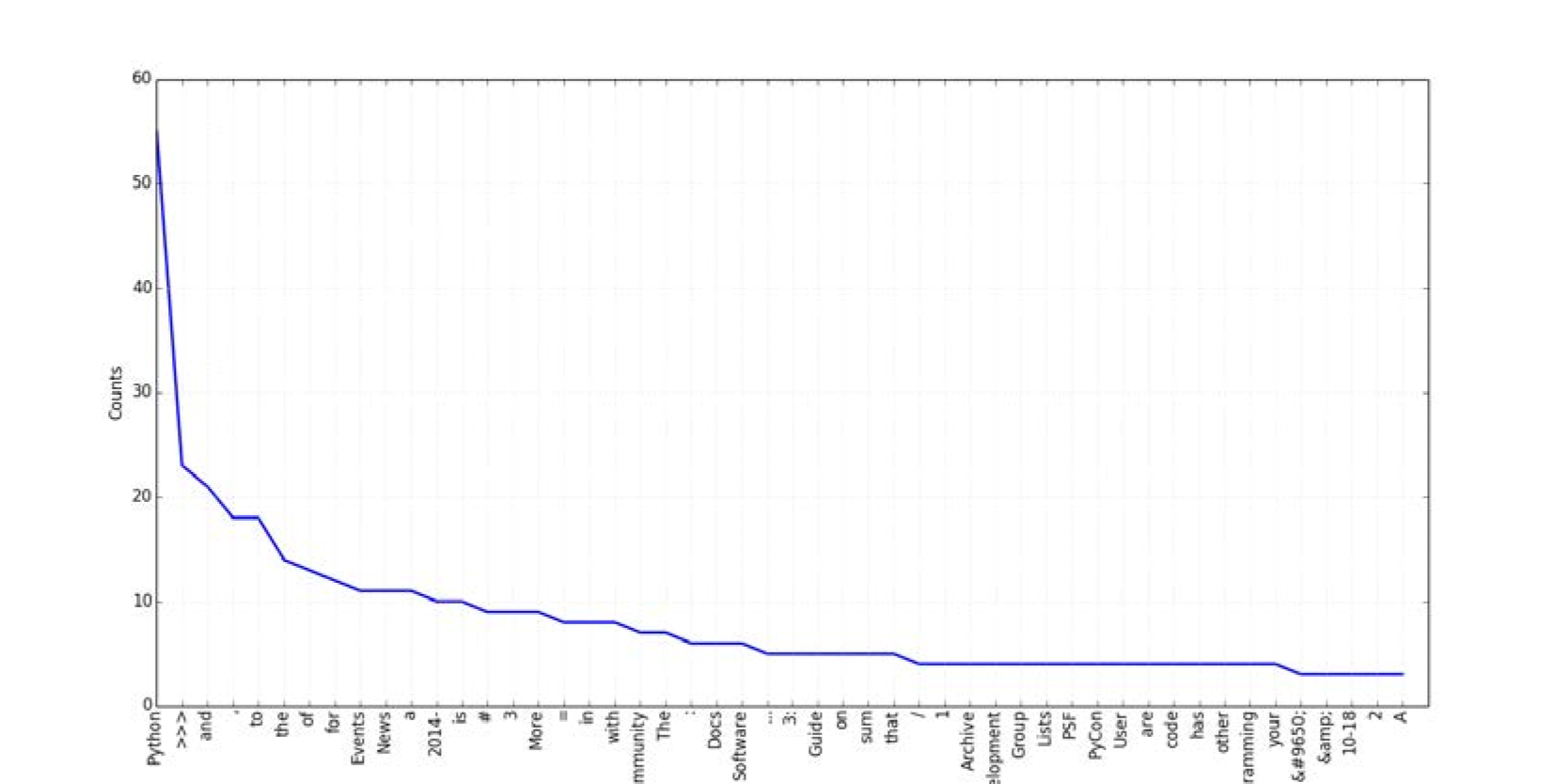
News:11

|  |
| --- |
| **下载示例代码**  您在<http://www.packtpub.com>中登录您的账户，从中下载到您所购买的、由Packt出版的所有书籍的示例代码。如果您在别处购得此书，也可以在<http://www.packtpub.com/support>上注册相关文件，我们会用E-mail将其直接发送给您。 |

现在，让我们来做一些更时髦的事。我们来绘制这样一张图：

>>> Freq\_dist\_nltk.plot(50, cumulative=False)

>>> # below is the plot for the frequency distributions



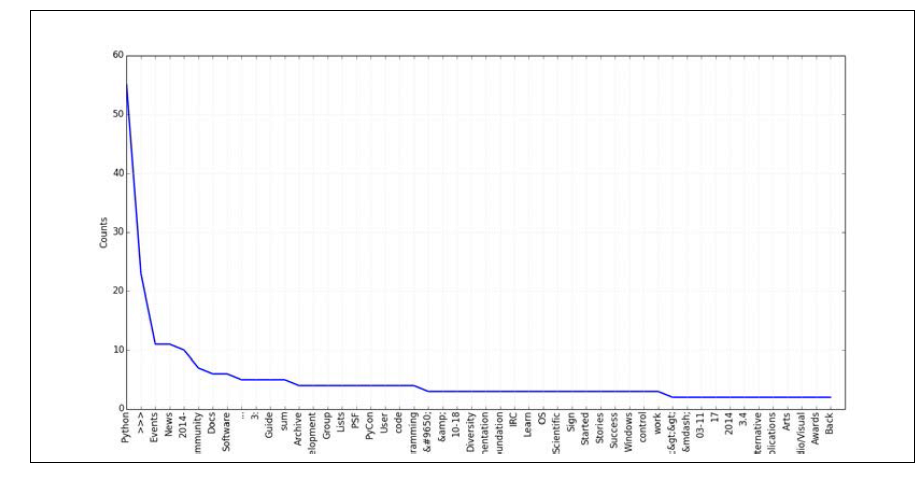
在上图中，我们可以看到累积频率的即时增长，在某些点上曲线会进入一条长长的尾巴。所以其中依然存在着一些干扰，有些类似于the、of、for以及=这样的词都是些无用词，这些词有一个专用术语：停用词。譬如the、a、an这样的词也都属于停用词。由于冠词代词在大多数文档中都是普遍存在的，因而无助于信息的识别。在大多数NLP及信息检索任务中，人们通常都会先删除掉这些停用词。下面，让我们再次回到之前运行的那个例子中：

>>> stopwords=[word.strip().lower() for word in open("PATH/english.stop.txt")]

>>> clean\_tokens=[tok for tok in tokens if len(tok.lower())>1 and (tok.lower() not in stopwords)]

>>> Freq\_dist\_nltk=nltk.FreqDist(clean\_tokens)

>>> Freq\_dist\_nltk.plot(50, cumulative=False)



|  |
| --- |
| 如果想知道关于词云的更多信息，请访问http://www.wordle.net/advanced 。 |

现在，代码看起来简洁多了吧！在完成这么多事后，您可以去Wordle网站上将其频率分布以CSV形式显示出来，应该会得到下面这张词云图：



## 该让您练练手了

* 请在不同的URL上尝试相同的练习。
* 并试着绘制出相应的单词云。

## 本章小结

总而言之，本章致力于为自然语言处理这一领域提供一份简介。虽然，本书假定读者在NLP领域，以及使用Python编程方面具有一定的背景知识，但我们也提供了一份与Python和NLP相关的快速入门。我们带您安装了所有在NLTK工作中将会用到的程序。另外，我们还通过几行简单的代码给您演示了NLTK的使用思路。我们提供的是一个了不起的词云实例，这是在大量非结构化文本中进行可视化处理的一种好方法，同时也是文本分析领域中相当流行的一种运用。我们的目标是要围绕着NLTK建构起所需要的一切，并让Python在我们的系统上顺利地工作。为此，您也应该要能编写并运行基本的Python程序。除此之外，我也希望读者能亲身感受一下NLTK库的魅力，自行构建出一个基本能实际运行的、涉及云词的小型应用程序。只要读者能顺利地产生出云词，我们就认为自己功德圆满了。

在接下来的几章中，我们将更为详细地了解Python这门语言，及其与处理自然语言相关的特性。另外，我们还将探讨一些基本的NLP预处理步骤，并了解一些与NLP相关的基本概念。

1. 译者注：IBM最新研制的人工智能系统Watson，它的运算更快，记忆力也更好，能读懂一些人类语言中的暗喻和双关。 [↑](#footnote-ref-1)
2. 译者注：原文如此，但正如你所见，代码中并没有return语句。且在没有返回值的情况下，python的函数是不必以return结束的。 [↑](#footnote-ref-2)
3. 译者注：最新版的NLTK已经取消了这个函数，并鼓励用户使用BeautifulSoup的get\_text()函数，因此对于：  
   clean = nltk.clean\_html(html)  
   我们应该将其改成（当然，在此之前还必须导入bs4库中的BeautifulSoup模块）：  
   soup = BeautifulSoup(html,"lxml")  
   clean = soup.get\_text() [↑](#footnote-ref-3)