103ron te .01n

T0 100ib

C(9:01L t:0

**第14章** **Python 与数据科学**

宽

**1 4 . 1 P y t h o n概** **述**

Python 是一种面向对象的解释型计算机程序设计语言，由荷兰人Guido van Rossum 于1989年发明。 Python 的第一个公开发行版于1991年发行。至2016年， Python 已经有 20多年的发展历史，成熟稳定。 Python 的主要参考实现是CPython, 它是一个由社区驱 动的自由软件，目前由Python 软件基金会管理。

Python 是一种功能强大的通用型语言，它包含了一组完善的、容易使用的标准库， 能够完成许多常见任务。它的语法简洁清晰，与其他语言不一样的一个重要特点是采用缩 进来定义语句块(若干语句构成一个语句块),也就是Python 的缩进是强制要求的。至于 缩进多少空格，不作硬性规定， 一般我们用4个空格即可(本章内容的讲述将不会纠缠于 一些概念比如语句和语句块以及语法结构，而是通过一些实例来掌握Python 语言)。

Python 支持命令式编程、面向对象程序设计、函数式编程、面向切面编程、泛型编 程多种编程范式。与 Ruby,Perl,TCL 等动态语言一样，Python 具备垃圾回收功能，能 够自动管理内存使用。

Python 虚拟机本身几乎可以在所有的操作系统中运行。因此，可以在几乎所有的操 作系统上运行Python 程序，包括Windows,Linux,Unix,Mac OS等。也可以使用诸如 py2exe,PyPy,PyInstaller 等工具，将Python 源代码转换成可以脱离Python 解释器执行 的程序。

Python 经常被当作脚本语言，用于处理系统管理任务和Web 编程，许多大型网站就 是用Python 开发的，比如YouTube.Instagram 等。得益于人们开发的各种数据预处理、 数据挖掘与机器学习、自然语言处理、数据可视化等软件库， Python 的应用领域得到了 扩展，被应用到各种数据分析场合。很多大公司和机构，包括Google,Yahoo,NASA

等，都大量地使用Python 。 第16章“数据科学案例”将介绍LIGO(Laser Interferome- ter Gravitational-Wave Observatory,激光干涉引力波天文台)的数据分析软件，就是用 Python 语言编写的。

目 前Python 是最流行的10种编程语言之 一 ，表14 - 1是TIOBE① 发布的编程语言流 行度排行榜，图14 - 1则是2002 — 2016年8月的TIOBE Programming Community Index (关于编程语言流行度 (Popularity) 的指标)。

**表14-1** **编程语言流行度排行榜**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nov 2016 | Nov 2015 | Change | Programiming Language | Ratings | Change |
| 1 | 1 |  | Java | 18.755% | -1.65% |
| 2 | 2 |  | C | 9.203% | 一7.94% |
| 3 | 3 |  | C++ | 5.415% | 一0.78% |
| 4 | 4 |  | C# | 3.659% | 一0.66% |
| 5 | 5 |  | Python | 3.567% | 一0.20% |
| 6 | 8 | 人 | Visual Basic.NET | 3.167% | +0.94% |
| 7 | 6 |  | PHP | 3.125% | 一0.12% |
| 8 | 7 | √ | JavaScript | 2.705% | +0.23% |
| 9 | 11 | 人 | Assembly language | 2.441% | +0.56% |
| 10 | 10 |  | Perl | 2.361% | +0.33% |
| 11 | 14 | 人 | Objective-C | 2.246% | +0.82% |
| 12 | 15 | 人 | Swift | 2.039% | 十0.80% |
| 13 | 48 | 众 | Go | 2.001% | +1.80% |
| 14 | 9 |  | Ruby | 1.978% | 一0.06% |
| 15 | 16 | 人 | Matlab | 1.967% | +0.78% |
| 16 | 12 |  | Delphi/Object Pascal | 1.950% | +0.27% |
| 17 | 13 | ︾ | Visual Basic | 1.923% | +0.24% |
| 18 | 37 | 众 | Groovy | 1.811% | +1.48% |
| 19 | 19 |  | R | 1.715% | +0.70% |
| 20 | 18 |  | PL/SQL | 1.512% | +0.48% |

TIOBE Programming Communiy Index

30



Rating(%)

一Jave 一C

一C++ 一C#

Python

一Visual Basic.NET PHP

一JavaScnipt

Assembly language Perl

**图14-** **12002** **—** **2016年8月的** **TIOBE** **Programming** **Community** **Index**

资料来源：[www.tiobe.com.](https://www.tiobe.com.)



①<http:/!>[www.tiobe.com/tiobe-index/.](https://www.tiobe.com/tiobe-index/.)



**14.2** **Python开发环境配置(Setup)**

为了使用Python 语言进行编程，首先需要把 Python 开发环境安装到电脑上。

截止到2016年， Python 有两个版本， 一个是2.7版，另一个是3.5版，这两个版本 是不兼容的。Python 正在朝着3.5版本进化。在进化过程中，大量的针对2.7版本的代码 需要移植才能在新的环境下运行。许多第三方库，暂时无法在3.5版本上运行。

为了保证我们写的程序能够使用大量数据预处理、数据分析、数据可视化等第三方 库，我们将使用2.7版Python 。 随着移植工作的推进，可以在Python3.5 上正确运行的第 三方库会越来越多，再考虑迁移也不迟。

安装 Python 的 方 法 有 两 种 ： 一 种 是 使 用 官 方 标 准 版 Python(<https://>

[www.python.org/](https://www.python.org/)); 另一种是使用第三方发布的集成版，比如 Anaconda(<https://>

[www.continuum.io/downloads](https://www.continuum.io/downloads))。 使用前者，用户需要自行安装所需的第三方库，使用后 者，常用的第三方库已经集成到安装包中，安装完成后即可使用。

我们这里将主要介绍在 Windows 环境下，使用Anaconda, 安装Python 的开发环境。 安装过程非常简单，用户需要到Anaconda 网址下载针对目标操作系统(32bit 或者64bit Windows) 的 Python2.7 安装包，然后运行安装包。可以按照缺省配置，把Anaconda 安 装到C:\Anaconda2 目录下。安装软件自动修改Windows 的环境变量PATH, 保证Py- thon 程序的正确运行。

Python 安装完成后，用户就会拥有Python 解释器(负责Python 程序的解释执行)、 一个命令行交互环境 (Python Shell)、通过网页进行交互式编程的环境 (Jupyter Note- book) 以及一个简单的集成开发环境 (Spyder) 。 如图14-2所示。

|  |  |
| --- | --- |
| **Anaconda2(64-bit)** ▶ | **O** **Anaconda** **Cloud**  ○ **Anaconda** **Navigator**  ■ **Anaconda** **Prompt** **Rn** **IPython**  Jupyter Notebook Jupyter QTConsole  Reset Spyder Settings  ●**Spyder** |
| CFCA  CodeLite Cygwin  D cygwin-X(32-bit) Eclipse  FreescreenSharing Java  7 Java Development Kit |
|  |

**图14-2** **Anaconda Python 开发环境**

在编程过程中，如果用户的目的是尝试执行一些代码片段，可以选用Jupyter Note- book 工具，如果用户的目的是编写执行复杂功能的程序，则建议使用集成开发环境 Spy- der 。它们的界面如图14-3和图14-4所示。

**jupyter** Untitled1 Last Checkpoint 10/25/2016(unsaved changes)

File Edit View Insert Cell Kemel Help



   N□c Code 口 CelToobar ◎



**In [1]:a=33**

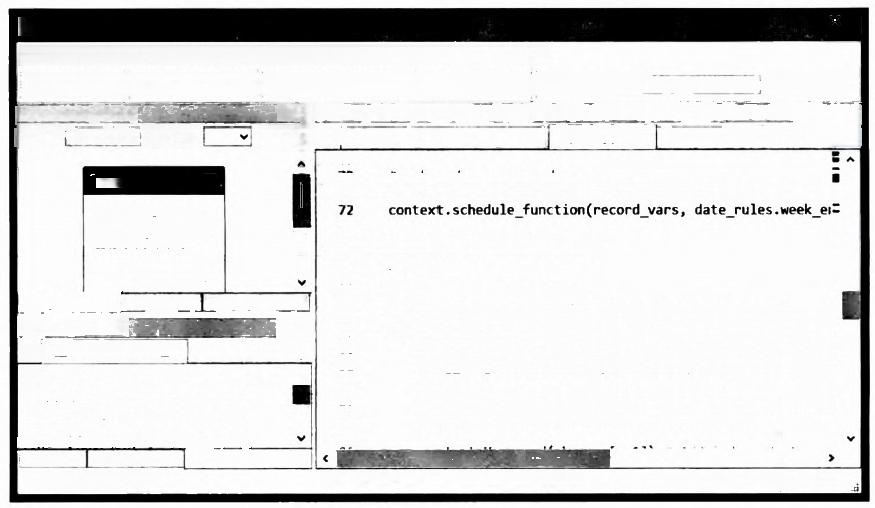
**print(a)**

33

 **In(** 1



**图** **1** **4** **-** **3** **Jupyter Notebook 交互式开发环境**



Eile Edit Search Source Run Debug Consoles Tools View Help

□ **日** **u** ▶ 画 合 Objeot inspector

Source Console Object[

69

Usage

71

Here you canger help of

any object by pressing C

in frontof it,either on the

Editor or the Console

Heip can also be shown

a let parenthesis nent to Object i … File …

**TPython console**

Ω ■ Console 1**/A**

81

**h**)**o**|de,Jun |Ana016, 4.1.1(64-^

11:07:13)[MSC v.150064 bit

**(AND64)** ]

Ivne “convrioht”."ccedits"or

Cons… History… IPython cons …

**Permissions:RN End-of-lines:CRLF Encoding:UTF-8-UESSED Line:85 Column:1 Memory:38** X

-C:Users\Adninistzator\Desktop\2016-09-10 algorithm trading备… ×

dualmovingaverage.py 口 ml.py 口 saverahoodata.pr 口写

#week\_end<->every\_day

context.schedule\_function(rebalance,date\_rules.week\_end

73

74 def handle\_data(context,data):

75 pass

7 def rebalance(context,data):

context.recent\_prices.append(data.current(context.sym,'t

if len(context.recent prices)==context.window\_length+2

short\_mavg =data.history(context.sym,'price',cont

long\_mavg =data.history(context.sym,'price',conte:

再 序 2 典 协 ■ 3 雄 + → |C:\Anaconda2 ↑

*list of I's and 's,1 when the price increc* =np.diff(context.recent\_prices)>8

x Bditor

日 E 3e

*#Make a*

changes

Utomatically ater wriung

罗 **■** 耳

82

83

84

**85|**

**Variable** …

78

79

80

70

**图** **1** **4** **-** **4** **Spyder 开发环境**

启 动Jupyter Notebook的 过 程 是 ， 点 击 Windows 开 始 菜 单 ， 找 到Anaconda2 程 序 组 ， 点 击 “ 运 行Jupyter Notebook” 快 捷 方 式 ， 等 待 一 会 儿 ， 浏 览 器 自 动 启 动 ， 展 示Jupyter Note- book 界面。选择 “New” 菜单，选择 “Python [Root]” 菜单项，新建Notebook 。然后用 户就可以在输入框中按照Python 的语法要求输入Python 代码，点击运行按钮N解释执行 代 码 . 即 可 立 即 观 察 到 执 行 的 结 果 。 如 果 有 语 法 错 误 ， 系 统 立 即 给 出 出 错 提 示 ， 方 便 判 断 和纠错 。

启动Spyder 开发环境的办法是，点击 Windows 开始菜单，找到 Anaconda2 程序组， 点 击 “ 运 行Spyder” 快 捷 方 式 。Spyder 启 动 以 后 ， 用 户 可 以 新 建 项 目 、 新 建 文 件 、 编 写 代码和调试代码 。



**14.3** **通过** **一** **系列实例学习Python**

**14.3.1** **变量/常量/注释**

变量是在程序的执行过程中可以改变的量，它可以是任意的数据类型。在Python 中 ， 变量无须事先定义其数据类型，给其进行赋值时，就确定了它的数据类型。

下面的代码把整数33赋予变量a, 那么变量a 具有整数类型。

|  |
| --- |
| a=33  print(a) |

变量需要有一个名称，称为变量名。变量名以英文和\_开头，后续字符可以是英文、 数字或者\_ 。比如 my\_book 是一个有效的变量名，9book 则不是一个有效的变量名。

表14- 2列出了Python 编程语言使用的保留字，用户不能给变量起这样的名字，或 者把这些保留字用作其他标识符(比如函数名)。

**表14-2** **Python的保留字**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| and | assert | break | class | continue | deff | del | elif | else | except |
| exec | finally | for | from | global | if | import | in | is | lambda |
| not | or | pass | print | raise | return | try | while | with | yield |

常量是在程序的执行过程中不能改变的量，包括整数、小数、字符串等常量。比如33 是一个整数常量，33.3则是一个小数常量，“I am a boy”是一个字符串常量。

下面的代码，分别用整数、小数、字符串常量对变量进行赋值。

|  |
| --- |
| a=35  b=76.7  c="I am a boy" |

Python 的注释以#号开头，#号之后一直到一行末尾的所有字符都是注释的一部分， Python 解释器将忽略它们，不予执行。下面的代码里 this is a comment 和 this is another comment 都是注释。

|  |
| --- |
| 井this is a comment  name="mark"#this is another comment |

**14.3.2** **数据类型**

Python 支持多种数据类型，当我们给变量赋值，就是把具体的数值和变量关联起来。 Python 支持的数据类型包括基本数据类型，如布尔值、整数、浮点数、字符串、复



数等，以及衍生数据类型，如列表、元组、字典、集合等。第三方软件库还提供二维表、 Panel 等对象类型，可以表示复杂的数据结构。

**1. 布尔值** **(bool)**

布尔值只有 True,False 两种值，要么是True, 要么是False 。在 Python 中，可以直 接用True,False 表示布尔值，也可以通过关系运算或者逻辑运算计算出来。下面的代码 中，通过关系运算3>2(为真，即True), 给变量b1赋值。

|  |
| --- |
| b1=3>2 print(b1)  type(b1)# 输出b1 的数据类型，也就是bool |

**2. 整数int**

Python 可以表示精度不限的整数(包括0和负整数)。在程序中，整数的表示方法和 数学上的写法是一致的，比如100,-100,0等。下面的代码中，给整数变量a,b,c 进 行了赋值。

|  |
| --- |
| a=100 b=-100  c=0xff  type(c)# 输出c 的数据类型，也就是int |

计算机使用二进制表示数值，二进制和十六进制有明确的对应关系。在某些场合使用 十六进制表示整数比较方便，十六进制用0x 前缀和若干0～9以及a～f 字符表示，比如 0xff00, 表示的10进制数为15×16³+15×16²+0×16¹+0×16°的数值。

**3. 浮点数** **(float)**

浮点数也就是小数。浮点数可以使用标准的写法，如3.23,3.24,一9.11等，也可 以使用科学计数法表示，比如3.23e3表示3.23×10³,即3230,1.2e-5 表示0.000012。 下面的代码中，分别给变量f1,f2 进行了浮点数赋值。

|  |
| --- |
| f1=3.23  f2=1.2e-5  type(f1)# 输出f1 的数据类型，也就是float |

**4.** **复数** **(complex)**

除了整数和小数，Python 还支持复数，复数包括实部和虚部，虚部带一个字符j 。下 面的代码，分别给变量cl 和 c2 进行了复数赋值。

|  |
| --- |
| c1=3+2.7j  c2=-5-9.3j  type(c1)# 输出c1 的数据类型，也就是complex |

**5.** **字符串** **(str)**

字符串是由若干字符组成的有序序列，字符串可以用单引号或者双引号括起来。如果 分别用三个双引号首尾括起来，可以用若干行字符串常量，给一个字符串变量赋值，换行 和空格都是字符串的一部分。下面代码中，分别给变量s1,s2,s3 和 s4 进行了字符串赋

值，并且通过下标寻访其元素(即字符),下标的各种使用方式，代码中给出了说明(注 意 Python 的下标是从0开始的)。

|  |
| --- |
| s1='small string'  s2="a much larger sting"  s3="""spanning multiple  lines"""  print(s3)  type(s1)# 输出s1 的数据类型，也就是str  str=Hello World!'  print str#打印整个字符串  print str[0]#打印0号下标字符  print str[2:5]#打印2、3、4号下标字符  print str[2:]#打印2号下标字符，以及后续字符 print str \*2#打印字符串两遍  print str+"TEST"#把str 和"TEST"连接起来，然后打印 |

**6.** **列表** **(list)**

列表是包含多种不同类型的元素的可以改变的有序序列，当然列表的元素也可以是同 样类型的，比如整数列表、小数列表等。列表的表示方法是用[]把元素包含起来，中 间用逗号隔开。下面的代码中，给列表list 进行了赋值，并且通过下标寻访其元素，下标 的各种使用方式，代码中给出了说明。

|  |
| --- |
| list=['abcd',786,2.23,'john',70.2] tinylist=[123,'john']  print list#打印 list 的所有元素  print list[0]#打印list 的0号下标元素 print list[1:3]#打印下标为1、2的元素 print list[2:]#打印下标>=2的元素  print list[-1]#打印倒数第一个元素  print list[:-1]#从下标为0的元素开始打印，打印到倒数第2个元素  print tinylist\*2#打印 tinylist 两遍  print list+tinylist#连接list 和tinylist, 然后打印 |

列表是一个可变的有序序列，可以往列表里追加元素(使用append 方法),也可以把 元素插入到指定的位置(使用insert 方法),比如索引号为1的位置。要删除列表末尾的 元素，用pop() 方法。要删除指定位置的元素，用 pop(i) 方 法 ，i 为索引号。要把某 个元素替换成别的元素，可以直接赋值给对应的索引位置。示例代码如下：



|  |
| --- |
| list=['abcd',786,2.23,'john',70.2] list.append('more')  print(list)  list.insert(1,785)  print(list)  list.pop()  print(list)  list.pop(2)  print(list) |

**7.** **使用列表对矩阵进行转置的实例**

列表的元素可以是一个列表，于是可以用列表构造二维数组。矩阵的转置是把矩阵的 行变成列，列变成行。

在下面的实例中，首先构造一个3行5列的数组a, 然后对其进行转置 (Transpose), 保存到 b 中。程序的逻辑是，为b 生成一个一个的空行，对b 的每个空行，从a 的适当位 置上取得元素，插入该空行，实现转置。

使用第三方库 Pandas 可以方便地实现矩阵的转置。本实例的最后给出了使用pandas 的 DataFrame 数据结构实现矩阵转置的代码。首先，在b 列表的基础上，创建一个 Dat- aFrame, 然后调用DataFrame 的 transpose 方法，对矩阵b 进行转置。

|  |
| --- |
| a=[[3,4,5,6,7],[13,14,15,16,17],[23,24,25,26,27]]#3 行5列的矩阵 b=[]  i=0  for new\_row in range(len(a[0])):  b.append([])  for new\_col in range(len(a)):  b[new\_row].append(a[new\_col][new\_row])  print a  print b  #这段代码执行的结果是  #[[3,4,5,6,7],[13,14,15,16,17],[23,24,25,26,27]]#a是一个3行5列的矩阵 #[[3,13,23],[4,14,24],[5,15,25],[6,16,26],[7,17,27]]#b 是 一 个5行3列 的矩阵  import pandas as pd df=pd.DataFrame(b)  print df.transpose() |

**8. 元组** **(tuple)**

元组也是一种有序序列，和列表非常相似。元组和列表的主要区别是，元组一旦初始 化以后，就不能修改了，它也没有append(),insert() 这样的方法。元组可以被看作只

读的 (read-only) 列表。定义一个tuple 时，就需要把元素都定下来。

由于元组的元素是不可变的，所以用户编写的代码更加安全，不会由于不小心修改了 某些元素，导致程序执行错误(如果需要修改数据，可以新建一个列表，然后把元素复制 过去)。编写Python 程序时，能用tuple 代替 list 就尽量用tuple。

从元组获取元素的方法和列表是一样的，可以通过下标来访问各个元素。示例代码 如 下 ：

|  |  |
| --- | --- |
| tuple=('abcd,786,2.23,'john',70.2)  tinytuple=(123,'john')  print tuple#打印 tuple 的所有元素  print tuple[0]#打印 tuple的0号下标元素  print tuple[1:3]#打印 tuple 的下标为1、2的元素  print tuple[2:]#打印下标>=2的元素  print tuple[-1]#打印倒数第一个元素  print tuple[:-1]#从下标为0的元素开始打印，打印到倒数第2个元素 print tinytuple\*2#打印tinytuple 两遍  print tuple+tinytuple#连接tuple 和 tinytuple, 然后打印  stu1=('john',20,'male',95.5,90.5,88) | |
| print(stu1[0])#  print(stu1[1])#  print(stu1[-1])# | 打印 tuple 的0号下标元素  打印 tuple 的1号下标元素 打印倒数第一个元素 |

**9. 字** **典** **(dict)**

Python 的字典实际上是一个哈希表 (Hash Table), 这个表包含一系列的键值对(<Key, Value>),键值对的 key和value可以是整数、小数、字符串或者布尔值等数据类型。Dict 具 有极快的查找速度。寻访一个dict 的某个key 所对应的元素，可以使用dict [key] 的方法， 也可以使用dict.get(key) 方法。要删除一个key, 用 pop(key) 方法，对应的 value会从 dict中删除。判断一个key是否在一个dict中，可以用key in dict来判断。

tinydict={'name':'john','code':6734,'dept':'sales','one':1,2:'two' print tinydict['one']#打印 key 'one'对应的value

print tinydict[2]#打印key 2对应的 value

print tinydict.get(2)#打印 key 2 对应的value print"code"in tinydict#判断 key 是否在字典中

print tinydict#打印整个字典

print tinydict.keys()#打印所有的key

print tinydict.values()#打印所有的value

**10.** **集合与冻结集合** **(set/frozenset)**

集合的概念和数学中集合的概念是一致的，它的元素是无序的，而且每个元素唯一， 也就是没有重复元素。两个set 可以做数学意义上的交集、并集等操作。集合的表示方法

是，用{}把元素包含起来，中间用逗号隔开。

通过 add(key) 方法可以添加元素到 set 中，可以重复添加，但由于元素是唯一的， 所以重复添加不会有效果。通过remove(key) 方法可以删除元素。集合是否包含某个成 员，可以用 key in set 来判断。

冻结集合 (frozenset) 是特殊的集合类型，它是不可变的。frozenset 一旦创建便不能 更改，没有 add,remove 方法。示例代码如下：

|  |
| --- |
| s1=set([1,2,3])  s2=set([2,3,4])  print s1&s2#交集  print s1|s2# 并集  s1.add(5)  s2.remove(3)  print s1&s2  print s1 |s2  print1 in s1#判断元素是否在集合中 fs1=frozenset([4.0,"string",True]) print(fs1) |

**14.3.3** **运算符及其优先级、表达式**

Python 的运算符包括算术运算符、关系运算符、逻辑运算符、集合运算符、对象运 算符等。表14-3列出了主要的运算符及其实例。

**表14-3** **运算符及其实例**

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 运算符类型 | 运算符描述 | 实例a=10,b=20 |
| 十 | 算术 | 加法运算 | a+b结果为30 |
| 一 | 算术 | 减法运算 | a—b结果为-10 |
| \* | 算术 | 乘法运算 | a\*b结果为200 |
| / | 算术 | 除法运算 | b/a结果为2  a/b结果为0 |
| % | 算术 | 取模运算(求余数) | b%a结果为0 |
| \*\* | 算术 | 幂运算 | 2\*\*3结果为8 |
| // | 算术 | 除法运算.截掉小数点后的有效数字(称为 Floor除法) | 9//2结果为4  9.0//2.0结果为4.0 |
| == | 关系 | 是否相等 | (a==b)为False |
| != | 关系 | 是否不相等 | (a!=b)为True |
| <> | 关系 | 是否不相等 | (a<>b)为True,结果和 !=一样 |
| > | 关系 | 是否大于 | (a>b)为False |
| < | 关系 | 是否小于 | (a<b)为True |

续前表

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 运算符类型 | 运算符描述 | 实例a=10 .b=20 |
| >= | 关系 | 是否大于或者等于 | (a>=b)为False |
| <= | 关系 | 是否小于或者等于 | (a<=b)为True |
| = | 赋值 | 赋值 | c=a+b将把a+b的值赋给c |
| 十= | 赋值 | 加法与赋值 | c十=a相当于c=c+a |
| 一= | 赋值 | 减法与赋值 | c - =a相当于c=c - a |
| \*= | 赋值 | 乘法与赋值 | c\*=a相当于c=c\*a |
| /= | 赋值 | 除法与赋值 | c/=a相当于c=c/a |
| %= | 赋值 | 取模与赋值 | c % = a 相 当 于 c = c % a |
| \*\*= | 赋值 | 幂运算与赋值 | C\*\*=a相当于c=c\*\*a |
| //= | 赋值 | Floor除法与赋值 | c//=a相当于c=c//a |
| and | 逻辑 | 两个操作数都为真，结果为真，否则为假 | 3>2 and 4>3为真 |
| or | 逻辑 | 两个操作数只要有一个为真，结果为真，否 则为假 | 3>2 or 3>4为真 |
| not | 逻辑 | 对操作数取反，真变成假，假变成真 | a=3  not(a>4)为真 |
| in | 集合 | 元素是否在集合里 | list=[1.2,3]  1 in list为True |
| not in | 集合 | 元素是否不在集合里 | list=[1,2.3]  4 not in list为True |
| is | 对象 | 是否为同一个对象，如果id(x)等于id(y), 那么结果为True | tuplel=(1,2,3) tuple2=tuplel  tuple2 is tuplel为True |
| is not | 对象 | 是否为不同对象，如果id(x)不等于id(y), 那么结果为True | tuplel=(1.2.3)  tuple2=(5,6.7)  tuple2 is tuplel为False |

除了上述运算符，还有按位运算符，列举如下(见表14-4)。这里假设a 的内部二进 制编码为00001010,b 的内部二进制编码为00010100。

**表14-4** **按位运算符**

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 运算符类型 | 运算符描述 |  |
| &. | 按位运算 | 按位与操作，即两个bit都为1结果才为1 否则为0 | (a &.b)的结果为00000000 |
| I | 按位运算 | 按位或操作，两个bit有一个为1,结果就为 1,否则为0 | (a|b)的结果为00011110 |
|  | 按位运算 | 按位异或操作，相同为假，不同为真，即 1^1=0,0^0=0,1^0=1,0^1=1 | (a^b)的结果为00011110 |
| ~ | 按位运算 | 按位取反，bit的值为1,取反的结果为0, 否则为1 | (～a)11110101 |
| << | 按位运算 | 操作数向左移动若干位，右边的bit填0 | a<<2的结果为00101000 |
| >> | 按位运算 | 操作数向右移动若干位，左边的bit填0 | a>>2的结果为00000010 |

运算符是有优先级的，比如在一个表达式中，既有加减法运算，又有乘除法运算，那 么在没有括号的情况下，先做乘除法运算，再做加减法运算，使用括号则改变运算执行的



顺序。比如 a+b\*c, 对该表达式进行计算时，先做b\*c, 再把结果和a 相加，得到最终 结果。如果把该表达式改成 (a+b)\*c, 那么对该表达式进行计算时，先做a+b, 再把 结果和c 相乘，得到最终结果。表14-5按照优先级从高到低，列出了所有的运算符。

**表14-5** **运算符的优先级(从高到低)**

|  |  |
| --- | --- |
| 运算符 | 优先级与说明 |
| \*\* | 幂运算 |
| ~,十， 一 | 取反、一目加、一目减 |
| \*,/,%,// | 乘法、除法、取模、Floor除法 |
| 十，一 | 加法、减法 |
| >>,<< | 左移、右移Bit |
| &. | 按位与操作 |
| 八，I | 按位进行异或操作、或操作 |
| <=,<,>,>= | 比较运算符 |
| <>,==,!= | 相等、不相等比较运算符 |
| 三 ， % = , / = , / / = , 一 = , 十 = , | = , & . <  三 ， \* = , \* \* = | 赋值运算符 |
| is,is not | 对象运算符 |
| in,not in | 集合运算符 |
| not,or,and | 逻辑运算符 |

在变量、常量、运算符的基础上，我们就可以构造表达式，对数据进行计算。表达式 是利用运算符把兼容的常量、变量拼接起来的式子，表达式是编写程序的基础。比如，我 们有两个整数类型的变量，那么就可以利用关系运算符，构造关系运算表达式a>b, 当 a 的值大于b 时，其值为真(True), 否则为假 (False)。还可以在此基础上，构造逻辑表 达式，比如 a>b and c>d, 那么当a>b 和c>d 同时为真时，该表达式的值为真。关系表 达式和逻辑表达式将用在分支程序结构中。

**14.3.4** **顺序、分支、循环程序结构**

程序的基本结构有三种，分别是顺序结构、分支结构和循环结构。

**1.** **顺序程序结构**

顺序结构是最简单的一种结构。解释程序执行顺序结构的Python 程序，将顺序地解 释执行各个语句，直到程序的末尾。比如，下面的程序，首先给两个变量赋值，然后交换 两个变量，最后打印出两个变量的值。

|  |
| --- |
| #顺序结构 a=3  b=4  t=a  a=b  b=t  print(a) print(b) |

**2.** **分支程序结构**

分支程序结构用于根据一定的条件进行判断(关系表达式、逻辑表达式),然后决定 程序的走向。它由if 语句、else 语句、elif 语句来构造，分支程序可以嵌套。我们通过实 例来了解if 语句的语法结构和功能。

分支程序结构的第一个实例如下，当a 的值大于b 时，对其值进行交换，最后先输出 a, 再输出b, 也就是按照从小到大的顺序输出b。

|  |
| --- |
| #分支结构 a=3  b=2  if a>b:  t=a  a=b  b=t# 这三个语句属于一个语句块，注意语句块的缩进 print(a)  print(b) |

分支程序结构的第二个实例如下，当a 的值大于b, 则打印 a greater than b, 否则打 印 a less than or equal to b。在这个实例里，条件为真时，我们要进行某种处理，条件为 假时，要进行另外一种处理。

|  |
| --- |
| a=3  b=2  if a>b:  print('a greater than b') else:  print('a less than or equal to b') |

分支程序结构的第三个实例如下，这个实例对不同区段的成绩进行A,B,C,D,E 级别的分档，需要使用if…elif…else 分支程序结构。

score=75

grade ='A'

if(score<60):

grade =E' elif(score<70):

**grade =D'**

elif(score<80):

grade ='℃ elif(score<90):

grade='B' else:

grade print(grade)

='A'

**3.** **循环程序结构**

可以用两个关键字while 和 for 来构造循环程序结构。循环程序结构的第一个例子是 一 个while 循环，首先给变量i 赋予初值1,然后通过while 循环判断它是否还在1～5之 间，然后把i 累加到变量sum 中，最后求出1+2+3+4+5的值。其中， i 为循环变量。

|  |
| --- |
| #循环结构  i=1  sum=0  while(i<=5):  sum =sum+i  i=i+1 print(sum) |

循环程序结构的第二个实例是一个 for 循环，首先创建一个列表，然后对列表长度 (为3)之上创建的一个有效下标范围[0,1,2]的每个下标，顺序访问列表的每个元素。 其中，index 为循环变量。

|  |
| --- |
| fruits=['banana','apple','mango']  for index in range(len(fruits)):#len(fruits)为3,range(…)为[0,1,2] print'Current fruit:’,fruits[index]  print"Good bye!" |

**4.** **数据结构与算法**

我们设计程序和编写程序是为了解决实际问题。程序的要素有两个： 一个是数据结构； 一个是算法。数据结构是对要解决的问题的一个建模，算法则是解决问题的一系列步骤。比 如，现在需要对全班的语文成绩进行排序，然后从高到低显示同学的名字和语文成绩。首 先，需要把每个同学的名字和语文成绩保存起来，可以使用两个列表 name\_list,score\_

list, 分别保存全班同学的名字和语文成绩，两个列表相同下标的元素对应某个同学的名字和 语文成绩，这就是数据结构。在此基础上，需要对整个问题进行分解，把大的问题(程序) 分解成一系列的小问题(模块)。把上述问题分解成两个小问题，分别是对全班的成绩进行 排序和显示全班同学的姓名和成绩。对于这两个小问题，需要进一步设计解决它们的一系列 的步骤(算法),比如对于全班成绩的排序问题，可以利用冒泡排序算法进行解决，对于显 示全班同学的姓名和成绩，需要的步骤则简单得多，可以通过一个for 循环，把各个同学的 姓名、成绩(各个下标对应的name\_list 、score\_list 列表的元素)打印出来。任何复杂的算 法都可以使用前文讲述的顺序、分支、循环等程序的基本结构来构造。

算法具有一系列的特点，包括：(1)确定性。算法的每个步骤都是确定的，不是含糊、 模棱两可的。(2)有穷性。算法应该包含有限的操作步骤而不是无限的。(3)有0个或者多 个输人。算法可以通过输入获得必要的外界信息。(4)有1个或者多个输出。算法的目的就 是求解，问题的解就是算法的输出。(5)有效性。算法的每个步骤都得到确定的结果。

**5.** **排序实例——冒泡排序**

排序是最基本的数据处理方法之一。比如，现在有一个班的学生成绩，希望从高到低 进行排序，看一看各个学生的排名情况。

排序算法有很多种，包括选择排序、插入排序、希尔排序、冒泡排序、快排序等，这 里介绍冒泡排序算法，其他排序算法的原理请参考相关资料。

我们通过下面的实例来理解冒泡排序的原理。列表的初始值为[81,83,82,86, 84,85],在第一趟，我们针对所有元素，从左到右，依次判断相邻两个元素的大小，如 果前面的元素大于后面的元素，就交换它们。第一趟完成以后，最大的元素将到达最末尾 的位置。第二趟，我们对整个列表靠近前面的n-1 个元素进行同样的操作 (n 为元素个 数)。冒泡排序用嵌套for 循环实现，代码如下：

|  |
| --- |
| list\_score=[81,83,82,86,84,85]  list\_name=['john','mary','lily','mark','June','April'] #冒泡排序  count=len(list\_score)  for i in range(0,count):  for jin range(i+1,count):  if list\_score[i]>list\_score[j]:  list\_score[i],list\_score[j]=list\_score[j].list\_score[i]  list\_name[i],list\_name[j]=list\_name[j],list\_name[i] for i in range(0,count):  print"%s:%d"%(list\_name[i],list\_score[i]) |

在这里，对list\_score [i],list\_score [j] 进行交换，使用list\_score [i],list\_ score [j]=list\_score [j],list\_score [i] 语句实现，显示了Python 语言的简洁性。

print"%s:%d"%(list\_name [i],list\_score [i]) 语句表示，按照“%s:%d” 的

格式串输出两个参数的值，分别是 list\_name [i] 和 list\_score [i],%s 表示输出字符

串，%d 表示输出十进制数值，当list\_name [i] 为 Mary,list\_score [i] 为83时，输出 的结果将是Mary:82。

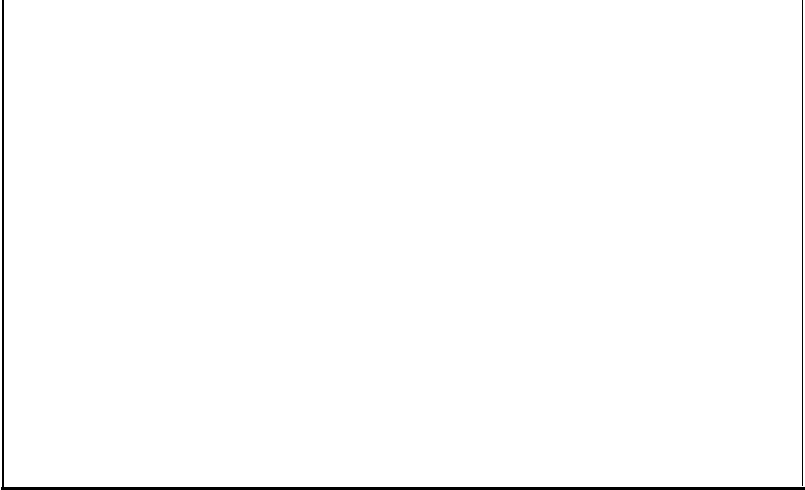
**6.** **查找实例——二分查找**

在一个列表里寻找一个元素，如果数据是没有顺序的，那么只能从头开始找，直到找 到该元素为止，或者已经查找完整个列表，没有找到。如果列表的元素是有序的(比如列 表包含数字，数字元素从小到大排列，或者列表包含字符串，字符串元素符合字典序等), 我们可以用二分查找算法，寻找元素。

二分查找算法的基本原理是，我们到有序序列的中间去查找该元素，如果找到，则停 止，否则，看中间这个元素的大小，如果它比目标元素小，那么到有序序列的右半序列去 查找，如果它比目标元素大，则到有序序列的左半序列去查找，直到找到目标元素，或者 查找了整个序列为止(找不到)。二分查找用while 循环实现，代码如下：

|  |
| --- |
| a=[1,2,3,4,5,6,7,8] target =6  found\_index=-1 low=0  high=len(a)-1 |





while low<=high:

mid=(low+high)//2

midVal=a[mid]

if midVal<target:

low=mid+1

elif midVal>target:

high=mid-1

else:

found\_index=mid

break

if(found\_index==-1):

print('not found') else:

print'%s:%d'%('found',found\_index)

**14.3.5** **函数、库函数的使用**

函数是从英文的function 直接翻译过来的。在 Python 语言里，函数是具有一定功能 的一段代码。对于经常用到的一些功能，比如打印输出变量的值，可以把实现这些功能的 代码组织成函数的形式，在需要这些功能时，直接调用函数即可，无须再写一遍类似的代 码，函数有利于程序的模块化设计风格的实现。

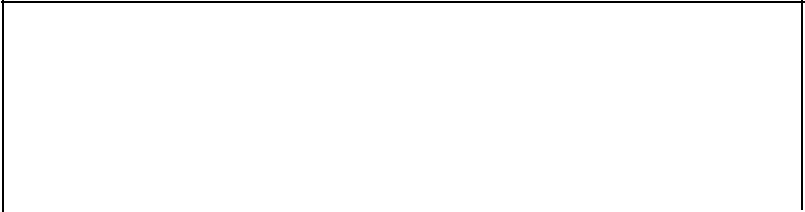
定义函数时，我们需要规定好函数接受什么样的参数，将返回什么样的值。对于调用 者来讲，只需了解这些信息就足够了，至于函数内部是如何实现对应的功能的，他无须关 心。下面通过一个实例展示如何定义一个函数。在上文中，二分查找代码是针对特定的列 表的某个元素的查找，如果把二分查找实现为一个函数，那么在需要对另外一个列表进行 元素查找时，调用函数即可。

定义一个函数的语法如下：

def 函数名(参数列表):

函数体(语句块)

下面的代码，展示了二分查找函数的定义以及对它的两次调用。从这个实例可以看 出，通过把一些公用的功能实现为一个函数，我们可以多次调用函数，实现更加复杂的功 能，代码则变得简洁多了。



def bin\_search(a,target): low=0

high=len(a)-1

while low<=high:

mid=(low+high)//2

midVal=a[mid]

|  |  |
| --- | --- |
| if midVal<target:  low =mid+1  elif midVal>target:  high=mid-1  else:  return mid  return-1  a=[1,2,3,4,5,6,7,8] target=6  found\_index=-1  found\_index =bin\_search(a,target)井调用bin\_search if(found\_index==-1):  print('not found') else:  print '%s:%d'%('found',found\_index)  a=[3,5,7,11,13,17,23,29] target=2  found\_index=-1  found\_index=bin\_search(a,target)# 调用bin\_search if(found\_index==-1):  print('not found') else:  print'%s:%d%('found',found\_index) | 函数  函数 |

在Python 中，在实现一个函数时，可以调用其他函数，甚至可以调用自身，即函数 可以递归调用。函数的递归调用，使得解决一些问题的代码变得简洁、易于理解。

采用函数的递归调用，计算n 的阶乘，其设计思路是：(1)如果n==0 或者n==1,

那么n 的阶乘为1。(2)如果知道了n-1 的阶乘，把它乘上n 就可以得到n 的阶乘，问题 的规模就缩小了一阶，也就是n 的阶乘的计算变成n-1 阶乘的计算，加上一个附加的步 骤(乘上n) 。由于0或者1的阶乘我们是很容易得到的，也就是问题规模缩小到1或者0 时，我们就可以解决它了。一旦低阶的问题得到解决，就可以一步步倒退回去，把各个更 高阶的问题解决掉。利用函数的递归调用，实现阶乘的代码如下：

|  |
| --- |
| def fractal(n):  if n==1 or n==0:  return 1  return n\*fractal(n-1)  print fractal(5) |

利用函数的递归调用，可以解决一个经典的问题，即汉诺塔 (Tower of Hanoi) 问题。 汉诺塔问题是一个古典的数学问题，问题的设定是，古代有一个梵塔，塔内有3个座，分别



是 A,B,C 座。开始时，A 座上有64个盘子，盘子大小不一，大的在下，小的在上。有 一个和尚，想把这64个盘子从A 座移动到C 座，但是有一些约束条件，即一次只能移动 一个盘，并且在移动的过程中，在3个座上都保持大盘在下、小盘在上。在移动的过程 中，可以利用B 座。

对于这个问题的解决思路是，如果只有1个盘子，那么直接把这个盘子从A 座移动到 C 座即可。如果现在有n 个盘子，可以把n-1 个盘子当作一个整体(问题规模缩小1), 利用C 座进行倒腾，把n-1 个盘子移动到 B 座 ( 利 用C 座);接着，直接把第n 个盘 (即最大那个盘)移动到C 座；最后，把已经倒腾到 B 座的n-1 个盘，利用A 座，倒腾 到 C 座。利用函数的递归调用，实现 Hanoi 塔问题求解的代码如下：

|  |
| --- |
| #coding =utf-8  def hanoi(n,x,y,z):  if n==1:  print(n,':',x,'-->',z)  else:  hanoi(n-1,x,z,Y)# 将前n-1 个盘子从x 移动到y 上  print(n,':',x,'-->',z)# 将最底下的最大的一个盘子从x 移动到z 上 hanoi(n-1,y,x,z)# 将 Y上的n-1 个盘子移动到z 上  n=int(input (请输入汉诺塔的层数：')  hanoi(n,'x¹,'y','z') |

**内置函数的使用**

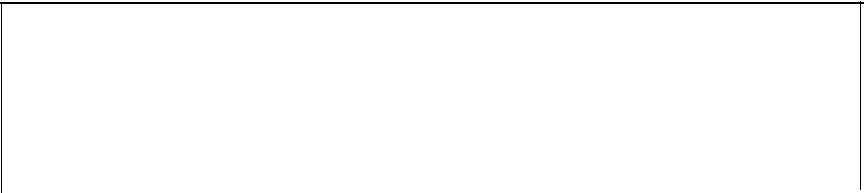
Python 解释器已经内置了若干函数，方便用户编程时调用。这些函数可以实现数学

运算、集合操作、逻辑判断、输入输出等功能。在这里，我们将介绍print函数和input 函 数 ，Python 其他内置函数及其使用方法，请参考 <https://docs.python.org/2/library/>

functions.html。

input 函数实现用户输入。input 函数可以带一个用户提示 (prompt), 也就是在用户 输入之前，先输出一个提示性的语句，告诉用户需要输入什么数据。如果用户的输入在语 法上是无效的，input 函数可能引发一个异常 (raise an expection)SyntaxError。关于异 常处理将在本节的末尾介绍。在上文的汉诺塔实例中，n=int(input (请输入汉诺塔的层 数：')语句，表示先打印提示性语句'请输入汉诺塔的层数：',然后等待用户输入，再把 用户输入转换为一个整数，赋给变量 n。

print 语句主要用于输出用户数据， 一般输出到屏幕上，即 Python 对象 sys.stdout 。 print 语句可以实现灵活的输出，这里我们仅仅举个例子，更精细的输出控制，请参考相 关文档。



a=3

b=4.5

c='young man'

print(a,b,c)井顺序输出a,b,c 的内容

print'%d,%f,%s\n'%(a,b,c)# 按照格式串'%d,%f,%s\n '输出a,b,c 的值

#%d 表示输出整数，%f 表示输出小数，%s 表示输出字符串，\n 表示换行

**14.3.6** **类和对象、对象的构造、对象摧毁、封装和继承、重写**

Python 是一种面向对象的编程语言，它通过封装机制，把数据和对数据的操作封装 成类，类的实例化则是一个个的对象。

比如，要对职员进行管理，需要登记他们的姓名、性别、年龄、薪水等信息，针对某 个职员可以显示他的这些信息。通过设计职员类，把上述属性管理起来，并且提供显示职 员信息的操作函数。职员类建立以后，可以生成职员类的实例，分别对应John,Mary 等 职员。示例代码如下：

|  |
| --- |
| class Employee(object):  def init (self,name):  self.name=\_name  def setName(self,\_name):  self.name=\_name def setSex(self,\_sex):  self.sex=\_sex  def setAge(self,\_age):  self.age=\_age  def setSalary(self,salary):  self.salary=\_salary  def show(self):  print 'name:',self.name,',sex:',self.sex,',age:',self.age,',sala-  ry:',self.salary  emp1=Employee('John')# 构造一个对象(Employee 类的实例)  emp1.setName('John')  emp1.setSex(Male')  emp1.setAge(33)  emp1.setSalary(9800.00)  emp1.show()  emp2=Employee(Mary')# 构造一个对象(Employee 类的实例)  emp2.setName(Mary')  emp2.setSex(Female')  emp2.setAge(35)  emp2.setSalary(9500.00)  emp2.show()  printEmployee. name # 显示类名 |

Python的每个类都有一些内置的属性，包括 dict  doc  name  module  bases 等。比如 name 属性值为类的名字，上述 Employee类的 name 属性为字符串Employee' (其他属性的含义，请参考相关资料)。

**1.** **构造函数**

一个类的构造函数负责对象的构造，构造函数的名称为 init **,它带一个** **self 参** 数以及其他参数，其中self 参数指向将要构造的对象，也就是self 是对象的引用。构造函数 的作用是对对象进行初始化。比如，上述代码中，Employee类的构造函数 init **,** **通** 过 name参数，给对象的name 属性进行赋值。对 init **函数的调用隐含在empl=** Employee('John') 和 emp2=Employee('Mary') 语句的调用过程中，这两个语句分别创建了 empl 对象和emp2 对象，它们的name 属性分别为John '和'Mary'。

**2.** **对象的摧毁和垃圾回收**

对于程序不再使用的对象， Python 周期性执行垃圾回收过程，自动删除这些对象， 以释放它们占用的内存空间。

**3.** **继承**

在定义类时，可以基于已有的类定义新的类，新的类(子类)和已有的类(父类)是 继承的关系。子类继承了父类的所有属性和方法(函数),还可以增加新的属性和方法。 比如，我们定义了一个新的类Manager, 它继承于Employee 类，但是增加了一个新的属 性 subsidy(特殊津贴),具体代码如下：

|  |
| --- |
| class Manager(Employee):#define child class  def setSubsidy(self,\_subsidy):  self.subsidy=\_subsidy  mgr1=Manager('tom')  mgr1.setName('tom')  mgr1.setSex(Male')  mgr1.setAge(31)  mgr1.setSalary(10100.00)  mgr1.setSubsidy(1000.00)  mgr1.show() |

**4.** **重写**

在上一个实例中，mgrl.show() 只显示了经理的姓名、性别、年龄和薪水，但是没有 显示特殊津贴。为此，必须为新的类 Manager 定义一个新的 show函数，这个函数和父类 Employee的show 函数同名 ·但是功能有些不一样，除了显示姓名、性别、年龄和薪水，还 显示特殊津贴。这种对父类的方法进行重新定义的机制，称为重写 (Override) 。代码如下：

|  |
| --- |
| class Manager(Employee):#define child class  def setSubsidy(self,\_subsidy):  self.subsidy=\_subsidy  def show(self):  print 'name:',self.name,'.sex:',self.sex,',age:',self.age,',sala-  ry:',self.salary.',subsidy:'.self.subsidy  mgr1=Manager('tom')  mgr1.setName('tom') |



|  |
| --- |
| mgr1.setSex(Male') mgr1.setAge(31)  mgr1.setSalary(10100.00) mgr1.setSubsidy(1000.00) mgr1.show() |

Python 的每个类都由 Python 语言和解释器提供了一些基本的方法(函数),这些函 数可以由用户重写。除了上文讲述的 init 方法之外，其他方法如表14-6所示。

**表14-6** **可以重写(override)的类方法**

|  |  |
| --- | --- |
| 序号 | 方法和说明 |
| 1 | init\_ (self [,args…])  类的构造函数，用户可以添加任意参数。调用方法：obj=className(args) |
| 2 | del (self)  析构函数，在删除对象时被调用，用于释放资源。调用方法：dell obj |
| 3 | repr (self)  把对象转换成字符串表示方式。用这个字符串直接调用eval函数，通常可以获得原来的对象。 调用方法：repr(obj) |
| 4 | str (self)  转换对象为一个字符串表示方式。调用方法： str(obj) |
| 5 | cmp (self,x)  比较两个对象。调用方法：cmp(obj,x) |

**14.3.7** **异常处理**

程序执行过程中可能发生异常情况，比如一个非零的整数除以0就会发生异常。我们 可以捕抓异常，然后打印提示信息，帮助用户了解到发生的情况，然后采取补救措施，比 如等待用户输入正确的数值、释放磁盘空间、连接到互联网等。

一般把有可能引发异常的代码放在一个try: 语句块，在try: 语句块之后，跟着一个 except: 语句及其语句块，该语句块的代码将对错误情况作出处理。具体的语法如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| try:  #在这里编写程序的处理逻辑     |  |  | | --- | --- | | except Exception I:  #如果发生异常Exception | I,就执行这个语句块 | |  |  | | except Exception II:  #如果发生异常Exception | II,就执行这个语句块 | |  |  |   else:  #如果没有异常发，就执行这个语句块 |



下面的代码，显示了如何捕抓整数除以0的异常。

|  |
| --- |
| try:  a=10  b=0  print a/b  except ZeroDivisionError:  print 'divide by zero error captures!' |

Python 的异常实际上是某个异常类的实例，比如，上述实例中的ZeroDivisionError 是一个异常类。我们也可以自己定义异常类，在程序发生特定错误时，发送异常 (raise Exception) 给外层程序捕抓以及处理。比如下面的实例，首先定义一个异常类Network- error, 它继承于RuntimeError 类。接着在程序的处理逻辑里，当程序判断找不到某个主 机，则引发一个异常，并且进行后续处理。

|  |
| --- |
| class Networkerror(RuntimeError):  def init (self,args):  self.args=\_args  try:  host\_not\_found=True  if(host\_not\_found):  raise Networkerror("host not found")  #如果没有异常，继续执行后续代码…… except Networkerror,e:  print e.args |

**14.3.8** **正则表达式**

正则表达式是一种用来匹配字符串的有力工具，它是一个特殊的字符序列，用于匹配 或者查找其他字符串里的子串。正则表达式可以用来判断一个字符串是不是日期、电子邮 件、邮政编码或者电话号码，帮助我们对用户输入数据进行合法性检验，也可以用来在一 个字符串里寻找这些实体类型。

下面的实例，用正则表达式匹配电话号码。正确的电话号码的模式是3个数字跟着 一个横杠，然后跟着3个数字，再跟着一个横杠，最后跟着4个数字，具体的模式是 ^(\d {3}) 一 ( \d {3}) 一 ( \d {4})$',^ 表示开始，$表示结束，\d 表示数字。

这个正则表达式经过编译以后，就可以用来匹配电话号码。

|  |
| --- |
| import re  #用正则表达式匹配电话号码  pattern\_phone =re.compile('^(\d{3})-(\d{3})-(\d{4})$') |

|  |
| --- |
| phone ='800-555-1212'  phonematch=pattern\_phone.match(phone) if phonematch:  print phonematch.group() else:  print "phone number is error!"  phone='800-555-1212-1234'  phonematch=pattern\_phone.match(phone) if phonematch:  print phonematch.group() else:  print "phone number is error!" |

re 模块提供了正则表达式的支持，为了使用正则表达式，我们首先需要导入这个模 块。在编译正则表达式或者使用正则表达式的过程中，如果发生错误，将引发 re.error 异常。

正则表达式的模式 (Pattern) 的定义用到很多特殊字符，能够匹配复杂的字符串 模式。①②

14.3.9 文 件 I/O (输入输出)

Python 可以通过库函数，对文件进行操作，包括打开文件、关闭文件、读文件、写 文件、添加数据等。在第三方库函数 Pandas 和 Scikit-learn 的介绍中，我们将给出具体的 实例。

14.4 第 三 方 库 和 实 例



人们在编写 Python 程序时，并不是什么都从头做起，而是有很多的第三方库可以使 用。第三方库一般以模块 (Module) 的方式组织，这是一种对代码进行组织的方式，把相 关的代码组织到一个模块里，易于理解和使用。

最简单的模块就是一个Python 源代码文件，在这个源文件里，可以定义类 (class)、 变量 (variable) 和函数 (function)。

为了使用第三方库，需要把模块导入到本文件，才能使用里面的功能。比如我们要使 用 Pandas 库的功能，可以通过在源代码里写上importpandas aspd 把这个库的相关类、函 数和变量导人到本文件，这里的pd 是该库的一个别名。我们也可以在一个 import 语句 里，把若干模块导入进来，比如 import module,module₂,…,modulen。



① <http://www.tutorialspoint.com/python/python_reg_expressions.htm.>

②<https://docs.python.org/2/howto/regex.hml.>



可以把一些通用功能写到一个源文件 (source file) 里，形成自己的模块，这些模块 也可以通过import 语句在其他的文件里导入使用。

对应的模块文件需要在操作系统的搜索路径中 (Search Path) 中，搜索路径指的是 Python 解释器可以在里面搜索模块文件的目录列表。在 Windows 操作系统上，可以通过 设定 PATH 系统环境变量，设置搜索路径。

**14.4.1 Pandas 介绍与实例**

**1.Pandas 介绍**

Pandas 是开源的 (BSD-licensed)Python 库，提供易于使用的数据结构和数据分析工 具。开发者对 Pandas 进行了优化，使得 Pandas 的执行速度得到了保证。 Pandas 基于 NumPy 库开发，和其他第三方库可以无缝地集成。Pandas 对时间序列分析提供了很好的 支持，它已经在金融数据分析领域得到广泛应用。

Pandas可以处理如下不同类型的数据，包括：(1)表格数据。表格的各个列可以具有不 同的类型，类似于SQL 数据库的表格或者Excel电子表格。(2)时间序列数据。Pandas 支持 有序和无序的 (Ordered/unordered) 时间序列数据的处理。时间序列数据无须是固定频率 (Fixed-Frequency) 的数据。(3)矩阵。 Pandas 支持异构数据类型的矩阵，可以设定行和列 的标签 (Label, 即行、列的名称)。(4)其他各类统计数据集 (Statistical Dataset)。

为了表示和分析现实世界中各类真实数据集， Pandas 提供了一系列基本模块 (Fun- damental Building Block), 即基本的数据结构。Pandas 支持的数据结构主要有：(1) Se- ries 。1 维数组，与NumPy 中的1维数组Array 类似，二者与Python 的数据结构List 也 很相近。区别在于List 中的元素可以是不同的数据类型，Array 和 Series 中则只允许存储 相同数据类型的元素，这样可以更有效地使用内存，提高运算效率。(2) Time Series。以 时间为索引的 Series。(3)DataFrame。2 维的表格型数据结构。可以将DataFrame 理解 为Series 的容器。(4)Panel 。3 维数组，可理解为DataFrame 的容器。

这些数据结构可以表示和处理金融、统计、社会科学以及众多的工程领域的大多数应 用场景用到的数据。

Pandas 的功能包括：(1)处理数据的缺失值 (Missing Data), 包括浮点数和非浮点 数的缺失值。(2)动态扩展性，用户可以插入或者删除 DataFrame 数据结构的列。(3)数 据对齐 (Data Alignment)。用户可以把数据对象对齐到标签 (Label), 或者由 Series, DataFrame 等数据结构自行对齐。(4)分组聚集功能 (Group by and Aggregation)。

(5)把其他 NumPy 等第三方库的数据结构转换成 DataFrame 的功能。(6)数据转换 (Transformation) 。(7) 数据集的合并 (Merging) 和连接 (Joining) 。(8) 从 CSV 文件、 Excel 文件数据库进行数据装载，把数据保存到HDF5 格式的文件. 以及从 HDF5 格式的 文件装载数据。(9)坐标轴的层次标签 (Hierarchical Labeling) 。(10) 数据透视表的旋 转 (Pivoting) 、 改变形状 (Reshaping) 。(11) 对大数据集进行基于标签的 (Label based) 数据切片 (Slicing) 、 提取子集 (Sub Setting)、建立和使用索引 (Fancy Indexing)。

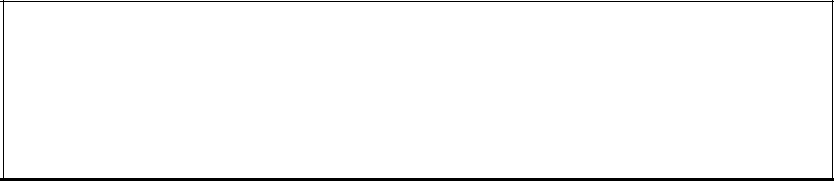
(12)Pandas 还提供面向时间序列数据处理的一些特殊功能，比如时间频率转换、移动窗 口上的统计值计算、移动窗口上的线性回归、序列的前移和后移 (Shifting and Lagging)、 生成数据范围比如生成时间范围 (Date Range Generation) 等。

**数据科学家对数据进行处理和分析，大致需要经过几个重要的阶段，分别是清洗和修改**

(Munging and Cleaning)、建模和分析、把分析结果按照易于可视化的方式或者表格方式进行 组织，Pandas 对这些任务都提供了支持。下面我们通过实例来学习Pandas 的功能。①

**2. 创建** **Series**

下面的实例，创建了一个Series, 然后把它打印出来，NaN 表示 Not a Number, 即它不 是一个有效数值。这个Series 的行标签为0,1,2,3,4,5等。



import pandas as pd import numpy as np

import matplotlib.pyplot as plt

s=pd.Series([1,3,5,np.nan,6,8])

print s

**3. 创建** **DataFrame**

对于DataFrame 来说，每一列的数据类型都是相同的，不同的列可以是不同的数据类 型。以SQL 数据库表格进行类比， DataFrame 中的每一行是一个记录，每一列则为一个 字段，即记录的一个属性。

下面的实例，首先创建了一个时间范围，然后基于这个时间范围创建了一个 Dat- aFrame,DataFrame 的行标签即刚刚创建的时间范围，列标签为A,B,C,D。

|  |
| --- |
| import pandas as pd import numpy as np  import matplotlib.pyplot as plt  dates=pd.date\_range('20130101',periods=6)  df =pd.DataFrame(np.random.randn(6,4),index =dates,columns=['A',B',C,D']) print df  df[E']=['one','one','two','three','four','three']# 增加一列，列名为E print df |

还可以以另外一种方式创建DataFrame, 输人的参数是一个字典，字典的每个键值对 (<Key,Value>) 的 Value 可以转换成一个 Series 。示例代码如下：

|  |
| --- |
| df2=pd.DataFrame({'A':1.,  B':pd.Timestamp('20130102'),  ℃:pd.Series(1,index=range(4),dtype ='float32'), D':np.array([3]\*4,dtype='int32'),  E':pd.Categorical(["test","train","test","train"]),  F':'foo'}) print df2 |

创建出来的DataFrame 的内容如下：



①<http://pandas.pydata.org/pandas-docs/stable/10min.html.>



|  |  |
| --- | --- |
| AB CDEF |  |
| 012013-01-0213test 112013-01-0213trainfoo | foo |
| 212013-01-0213test 312013-01-0213trainfoo | foo |

**4.** **查看数据和元信息**

head 和 tail 方法可以显示DataFrame 前 N 条和后N 条记录，N 为对应的参数，默认 值为5。通过index ( 行 ) 和columns (列)属性，可以获得DataFrame 的行和列的标签。 这也是了解数据内容和含义的重要步骤。 decribe 方法可以计算各个列的基本描述统计值， 包含计数、平均数、标准差、最大值、最小值及4分位差等。具体代码如下，需要和上个 实例配合，才能正确运行。通过dtypes 属性，可以获得各列的数据类型。

df.values属性则以列表的方式保存DataFrame 的具体数据。

|  |
| --- |
| print df.head()#前 5 行  print df.tail()#后 5 行  df.describe()# 描述信息  print df.index#行标签  print df.columns#列标签  print df.dtypes#各列的数据类型 print df.values#DataFrame的 值 |

**5.** **转置与排序**

对数据进行转置，就是把DataFrame 的行变成列，列变成行，比如原来的2维表是6 行3列，那么转置以后的2维表是3行6列。具体代码如下：



print df.T

可以对DataFrame 按照坐标轴进行排序。下面的代码，对DataFrame 按照第1个坐标 轴进行排序，就是沿着列方向，对各个列标签即Column Name进行排序。按照第0个坐 标轴进行排序，是沿着行方向，对行标签进行排序。

|  |
| --- |
| df =df.sort\_index(axis=1,ascending =False)#沿着列方向对col name 进行  排序  print df  #这段代码的执行结果是  #D CBA  #2013-01-01-1.135632-1.509059-0.2828630.469112 #2013-01-02-1.0442360.119209-0.1732151.212112 #2013-01-031.071804-0.494929-2.104569-0.861849 #2013-01-040.271860-1.039575-0.7067710.721555 #2013-01-05-1.0874010.2762320.567020-0.424972  #2013-01-060.524988-1.4784270.113648-0.673690 |



还可以基于某个列，对 DataFrame 进行排序。下面的代码对 DataFrame 根据B 属性 列进行排序。

|  |
| --- |
| df =df.sort\_values(by='B') print df |

6. 提取部分数据

可以通过列名，提取一个数据列，示例代码如下：

|  |
| --- |
| print df['A'] |

可以通过行号范围、行标签范围，提取若干数据行，示例代码如下。在这里需要注 意，0:3为下标范围，表示提取下标为0,1,2的行，'20130102':'20130104'为行标签范 围，表示提取20130102''20130103''20130104'三行。

|  |
| --- |
| print df[0:3]  print df['20130102':'20130104'] |

提取一个数据块的实例如下，该实例提取行标签范围和列标签列表对应的行列子集， 也可以通过行下标、列下标来提取行列子集。既然可以取得一个行列子集，便可以取得一 个单元格的值。示例代码如下：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | print | df.loc['20130102':'20130104',['A','B']] | | print | df.iloc[3:5,0:2]#row=3,4,col=0,1 | | print | df.iloc[[1,2,4],[0,2]]#row=1,2,4,col=0,2 | | print | df.iloc[1:3,:] #row=1,2,col=all | | print | df.iloc[:,1:3] #row=all,col=1,2 | | print | df.iloc[1,1]#提取一个单元(cell) 的值 | | print | df.iat[1,1]#提取一个单元(cell) 的值 | |

可以使用条件过滤，获取部分行数据。下面示例代码中，第一行语句，只把df 里的 A 列>0的行提取出来，第二行语句，把df2 的 E 列的值为'train'的行提取出来。

|  |
| --- |
| print df[df.A>0]  print df2[df2['E'].isin(['train'])] |

7. 设置单元格的值 (Setting)

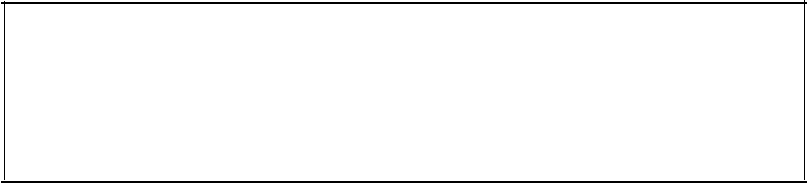
可以通过行标签和列标签或者通过行下标和列下标，对单元格的值进行设置，也可以 对整列进行设置。示例代码如下：

|  |
| --- |
| df.at[dates[0],'A']=-99# 通过行标签和列标签，设定单元格的值 print df.head()  df.iat[0,1]=3# 通过行下标和列下标，设定单元格的值  print df.head() |

|  |  |
| --- | --- |
| df.loc[:,D']=np.array([5]\*len(df))# 度，即有多少行  print df | 对整列进行设置，len(df) 表示df 的长 |

**8. 对缺失值的处理** **(Handle Missing Data)**

对于缺失值，我们可以把包含缺失值的整行数据从DataFrame 里剔除，或者把缺失值 替换成某个有意义的值。示例代码如下：

dfiat[0,2]=np

. .nan

df =df.dropna(how='any')#把包含缺失值的行删除 print df

print pd.isnull(df)

df.fillna(value=5)# 用5代替缺失值

**9.计算每列的均值**

通过DataFrame 的 mean 方法，可以计算每个数据列的均值。示例代码如下：

print df.mean()

**10.对每列运用一个函数**

可以对DataFrame 的每个数据列运用某个函数，比如把每列的最大值减去最小值，计 算出数据的极差 (Range) 等。示例代码如下：

|  |
| --- |
| df.apply(lambda x:x.max()-x.min()) |

**11.** **计算直方图**

直方图是数据集里各个取值的频率的图形表示。下面的示例代码，计算了s 序列的每 个值的频率。

|  |
| --- |
| s=pd.Series(np.random.randint(0,7,size=10)) 井生成10个随机数，值域为 (0,7)  print s.value\_counts() |

**12.** **字符串处理**

可以以向量化的处理方式(一次处理若干元素),对序列进行字符串处理，比如把所 有的字符串都变成小写形式。示例代码如下：

|  |
| --- |
| s=pd.Series(['A','B','C','Aaba','Baca',np.nan,'CABA','dog','cat']) s=s.str.lower()  print s |

**13.DataFrame的合并** **(Concatination)**

可以把若干DataFrame (模式相同)合并起来，构成一个大的DataFrame 。下面的实 例把一个DataFrame 横向切割成三个子集，然后用concat 进行合并，构成一个大的 Dat- aFrame, 其内容和原来的DataFrame 是一样的。示例代码如下：

df =pd.DataFrame(np.random.randn(10,4)) print df

pieces=[df[:3],df[3:7],df[7:]]

print pd.concat(pieces)

**14.DataFrame 的连接** **(Join)**

连接操作是根据一定的条件，把两个DataFrame 数据集的各行合起来构成目标 Dat- aFrame 的一行。示例代码如下。该实例把left 数据集和right 数据集的每一行，根据名称 为 key 的列的值相同的条件，合起来构成目标 DataFrame 的一行。

|  |
| --- |
| left=pd.DataFrame({key':['foo','bar'],'lval':[1,2]}) right=pd.DataFrame({key':['foo','bar'],'rval':[4,5]}) print left  print right  pd.merge(left,right,on ='key') |

**15. 添加数据行** **(Append)**

DataFrame 是一个可变的数据集，可以添加新的数据行。示例代码如下。该实例把一 个 DataFrame 的第三行切下来，然后添加到原来的 DataFrame 的末尾，构成新的Dat- aFrame。

|  |
| --- |
| df=pd.DataFrame(np.random.randn(8,4),columns=['A','B',C,D']) print df  s=df.iloc[3]  print s  df=df.append(s,ignore\_index=True) print df |

**16. 分组与聚集** **(grouping &aggregation**)

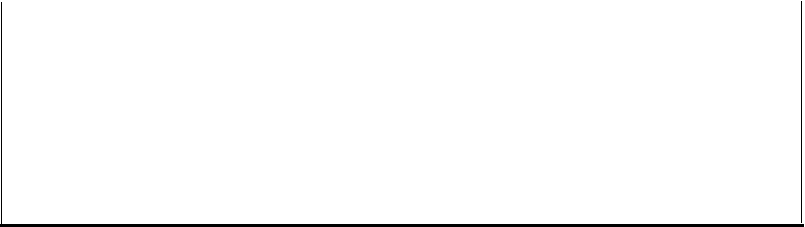
可以对DataFrame 进行分组和聚集。分组是根据某个列的值把所有的行分成一组一组 的。聚集则是进行求和、最小值、最大值、平均值等的计算。示例代码如下：

|  |
| --- |
| df =pd.DataFrame({'A':['foo','bar','foo','bar','foo',"bar','foo','foo'], 'B':['one','one','two','three','two','two','one','three'],  C':np.random.randn(8),  D':np.random.randn(8)}) print df  print df.groupby('A').sum()  print df.groupby(['A','B']).sum() |

**17.** **数据透视表**

可以为DataFrame 创建数据透视表。下面的示例代码，创建了以A 列 、B 列为行变 量，以 C 列为列变量，以D 列为单元格的值的数据透视表。





df =pd.DataFrame({'A':['one','one','two','three']\*3, B':['A','B',C]\*4,

℃':['foo','foo','foo','bar','bar','bar']\*2, D':np.random.randn(12),

E':np.random.randn(12)}) print df

pd.pivot\_table(df,values=D',index=['A','B'],columns=[℃])

这段代码建立了如下的数据透视表(见表14-7)。

**表14-7** **数据透视表**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | bar | foo |
| A | B |  |  |
| one | A | -1.760236 | 一0.799277 |
| B | 0.195817 | 1.083795 |
| C | 0.423455 | —0.838866 |
| two | A | 一0.429625 | NaN |
| B | NaN | -1.893984 |
| C | —1.306391 | NaN |
| three | A | NaN | 0.933793 |
| B | 0.274710 | NaN |
| C | NaN | 0.919669 |

数据透视表的目的是对数据进行汇总，有可能透视表的一个cell对应原来DataFrame 的多行数据，那么这个cell 应该取什么样的值，由pivot\_table 函数的aggfunc 属性决定。 比如当aggfunc=np.min, 那么进行聚集时，取其中最小值， aggfunc 还可以取最大值、 平均值、总和等。示例代码如下。该实例对多个数据行的某一列取平均值。

|  |
| --- |
| from collections import OrderedDict from pandas import DataFrame  import pandas as pd  import numpy as np  table=OrderedDict((  ("Item",['Item0','ItemO','Item0','Item1']), (CType',['Gold','Bronze','Gold','Silver']), ('USD',[1,2,3,4]),  (EU',[1.1,2.2,3.3,4.4])  ))  df =DataFrame(table) print df  pd.pivot\_table(df,index ='Item',columns ='CType',values ='USD',aggfunc = np.mean) |



d.stack()

DataFrame 的数据行和 pivot\_table 的 cell 的数据关系，可以通过图14-5查看。

|  |  |  |  |
| --- | --- | --- | --- |
| **Ix=Item** | **Bronże、** | **Gold** | **Silver** |
| Item0 | **2** | 2=mean(1,3) | NaN |
| Iteml | **NaN,** | NaN | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ix** | **Item** | **CType** | **USD** | **EU** |
| 0 | Item0 | Gold | 1 | 1 |
| 1 | Item0 | Bronze | 2- | 2 |
| 2 | Item0 | Bold | 3. | 3 |
| 3 | Iteml | Silver | 4- | 4 |

d.pivot\_table(index='Ttem',column='CType',values='USD',aggfunc=np.mean)

**图14-5** **pivot\_table实例及其数据的计算关系**

**18.** **重塑** **Reshape(stack** **&unstack)** **操作**

图14-6展示了堆叠/反堆叠操作的实例。最初的DataFrame 在行方向和列方向上有 多级索引 (Multi Indices)。对 一 个2维表进行透视表操作 (Pivoting), 本质上是 Dat- aFrame 的堆叠 (Stacking)/ 反堆叠操作的特例。

对 DataFrame 进行堆叠操作，它把列方向的最底层的索引 (Innermost Column In- dex) 转换成行方向最底层的索引 (Innermost Row Index), 当然相应的单元格数据也需 要做出改变。堆叠操作的反操作是反堆叠，它把行方向最底层的索引 (Innermost Row In- dex) 转换成列方向最底层的索引 (Innermost Column Index)。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | c0 | | c1 |
| c00 | c01 | c10 |
| r0 | r00 | (0,0) | (0,1) | (0,2) |
| r01 | (1,0) | (1,1) | (1,2) |



d.unstack()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | c0 | | | | c1 | |
| c00 | | c01 | | c10 | |
| r00 | r01 | r00 | r01 | r00 | r01 |
| r0 | (0,0) | (1,0) | (0,1) | (1,1) | (0,2) | (1,2) |

Unstacked(i.e.Broader)

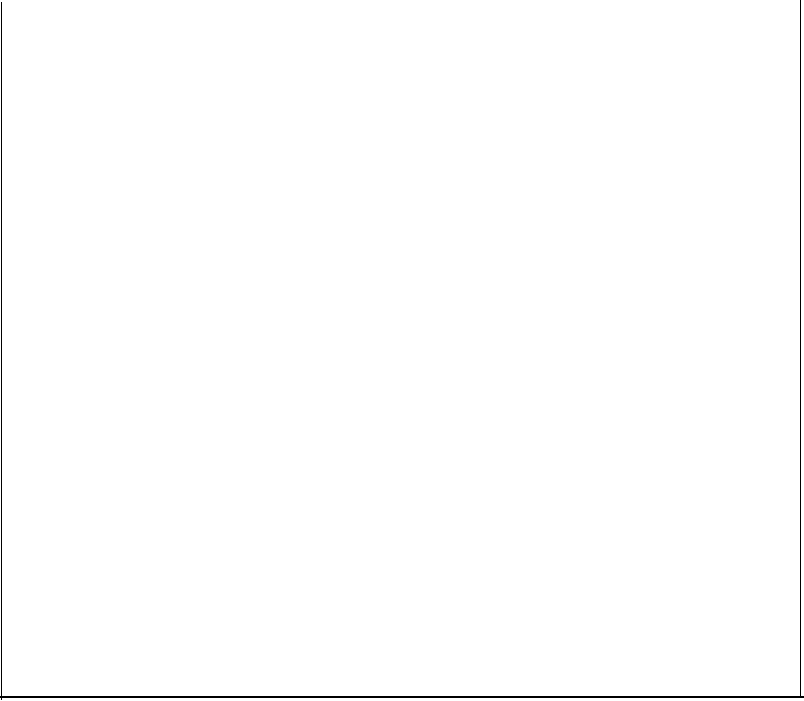
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | c0 | | **cl** |
| r0 | r00 | c00 | (0,0) | NaN |
| c01 | (0,1) | NaN |
| c10 | NaN | (0,2) |
| r01 | c00 | (1,0) | NaN |
| c01 | (1,1) | NaN |
| c10 | NaN | (1,2) |

Stacked(i.e.taller)

**图14-6** **DataFrame的stack 和unstack操作**

这个实例的行方向和列方向都包含二级索引。堆叠操作使得DataFrame 变高了，因为 堆叠操作把数据在更少的列和更多的行上“叠加”起来。反堆叠操作使得DataFrame 变矮 了、变宽了。





from collections import OrderedDict from pandas import DataFrame

import pandas as pd

import numpy as np

#行方向的多级索引(row multi index)

row\_idx\_arr=list(zip(['r0','r0'],['r-00','r-01']))#r0有两个子节点r-00,r-01 row\_idx=pd.MultiIndex.from\_tuples(row\_idx\_arr)

#列方向的多级索引(column multi index)

col\_idx\_arr=list(zip(['c0','c0','c1'],['c-00','c-01','c-10']))#c0 有两个子节

点，c1 有一个子节点

col\_idx=pd.MultiIndex.from\_tuples(col\_idx\_arr)

井创建DataFrame,2 行 3 列

d=DataFrame(np.arange(6).reshape(2,3),index =row\_idx,columns =col\_ idx)

d=d.applymap(lambda x:(x//3,x%3))

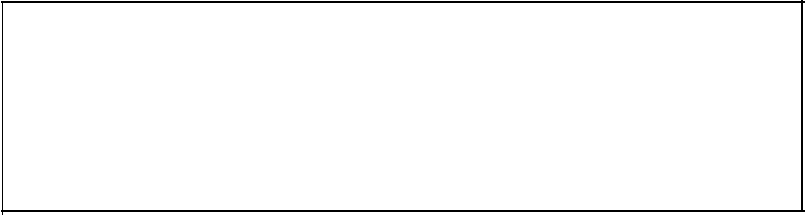
s=d.stack()#Stack print s

u=d.unstack()#Unstack print

Pandas 还允许进行中间层次索引的堆叠和反堆叠操作。比如，在上述实例中，可以 对最顶层的索引进行堆叠和反堆叠操作。最典型的用例还是在最底层的索引层次上进行这 些操作，易于理解。

**19. 时间序列** **(Time Series) 数据处理**

Pandas 提供了 resample 函数，对时间序列数据进行频率转换 (Frequency Conver- sion) 和重新采样 (Resample) 。 示例代码如下。该实例把秒级采样的数据，进行“每5 分钟”的重新采样，每五分钟里的秒级数据，以求和的方式进行汇总。



rng=pd.date\_range('1/1/2012',periods=10,freq='S')# 频率为秒 ts=pd.Series(np.random.randint(0,500,len(rng)),index=rng)

print ts

进行“每5分钟”的重新采样

#ts=ts.resample('5min',how print ts

ts=ts.resample('5Min').sum()#

='sum')

时间序列数据的时间戳，可以改变时区设定。比如，由世界标准时间 (Coordinated Universal Time,UTC) 转换成美国东部的时间。

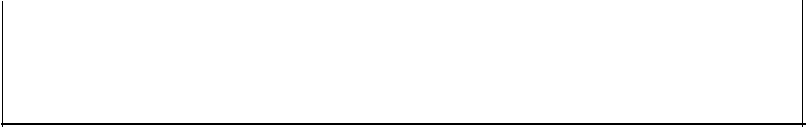
|  |
| --- |
| #Time zone representation  rng=pd.date\_range('3/6/201200:00',periods=5,freq=D')# 频率为天 ts=pd.Series(np.random.randn(len(rng)),index=rng)  print ts  ts\_utc =ts.tz\_localize('UTC') print ts\_utc  ts\_new=ts\_utc.tz\_convert('US/Eastern') print ts\_new |

时间序列数据可以分为时期 (Period) 序列和时点 (Point) 序列。每个月末的外汇储 备总额就是一个时点序列。每个月的出口额则是一个时期序列。Pandas 提供函数，在这 两类时间序列之间进行转换。示例代码如下。该实例把时点序列转换成时期序列，再转换 成时点序列。

|  |
| --- |
| #Converting between time span representations  rng=pd.date\_range('1/1/2012',periods=5,freq="M')# 频率为月 ts =pd.Series(np.random.randn(len(rng)),index=rng)  print ts  ps =ts.to\_period() print ps  print ps.to\_timestamp() |

在时点序列和时期序列之间进行转换，可以方便地实现一些算术运算。比如下面的实 例中，把频率为季度的时间序列数据，转换成每个季度最末一个月的第一天的上午9点的 时点序列数据。

|  |
| --- |
| prng=pd.period\_range('1990Q1','2000Q4',freq='Q-NOV')  ts =pd.Series(np.random.randn(len(prng)),index=prng)  print ts.head()  ts.index=(prng.asfreq(M','e')+1).asfreq('H','s')+9  #季度最末尾一个月第一天上午9点，M 表示Month,e 表示 end,H 表示Hour,s 表 示start  print ts.head()  #结果为  #1990Q1-0.277671  #1990Q20.163338  #1990Q3-0.315763  #1990Q4-1.527297  #1991Q1-1.632511  #1990-03-0109:00-0.277671 |



#1990-06-0109:000.163338

#1990-09-0109:00-0.315763

#1990-12-0109:00-1.527297

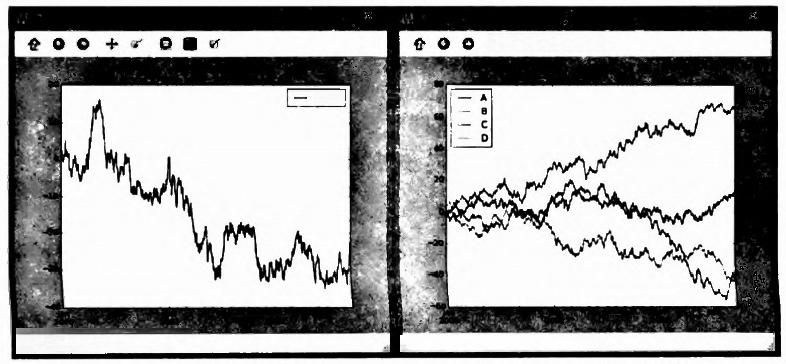
#1991-03-0109:00-1.632511

**20.** **绘图**

可以使用matplotlib 对 Pandas 数据进行可视化。下面展示了两个实例，第一个实例 显示了一个Series, 该序列是从2000年1月1日开始1000天的随机数序列，第二个实例 显示了一个DataFrame, 它使用的行标签和第一个实例的行标签是一样的。

|  |
| --- |
| #可视化Series  import matplotlib.pyplot as plt  ts=pd.Series(np.random.randn(1000),index=pd.date\_range('1/1/2000',pe- riods =1000))  plt.figure();  ts =ts.cumsum()#cumulative sum ts.plot()  plt.legend(loc ='best')  plt.show()  #可视化DataFrame  df=pd.DataFrame(np.random.randn(1000,4),index =ts.index,columns =['A', B','C,D'])  df=df.cumsum()#cumulative sum plt.figure();  df.plot();  plt.legend(loc=best')  plt.show() |

上述两个实例的可视化结果如图14-7所示。



十 r0 ■☑

None

三200101:11 y=5.727848 t=2002-05-01 y=73.132911

(a)Series 的可视化结果 (b)DataFrame的可视化结果

**图14-7** **Pandas的Series数据和** **DataFrame数据的可视化结果**



**21.** **文件读/写**

利用Pandas 提供的函数，可以把数据保存到文件中，也可以从文件中读取数据。

Pandas 支持的数据文件格式包括CSV,HDF5,Excel 等。示例代码如下：

|  |
| --- |
| #读写文件  #Writing to&read from a csv file df.to\_csv('foo.csv')  df=pd.read\_csv('foo.csv')  print df.head()  #Writing to&read from a HDF5 Store df.to\_hdf('foo.h5','df')  df=pd.read\_hdf('foo.h5','df')  print df.head()  #Writing to &read from an excel file  df.to\_excel('foo.xlsx',sheet\_name ='Sheet1')  df =pd.read\_excel('foo.xlsx','Sheet1',index\_col=None,na\_values=['NA']) print df.head() |

**14.4.2 Scikit-learn 介绍与实例**

**1.Scikit-learn 介绍**

机器学习是在若干样本数据上学习到一个模型，然后用这个模型对新数据进行预测。 如果每个样本都包含多个元素(数字),则称为多维数据或者多变量数据 (Multi-Dimen- sional/MultivariateData),每个变量称为一个特征 (Feature)。

Scikit-learn 是面向Python 的机器学习软件包。Scikit-learn 软件包支持主流的有监督 机器学习 (Supervised Learning) 方法、无监督机器学习 (Unsupervised Learning) 方法。 支持众多的方法是Scikit-learn 得到广泛应用的原因之一。

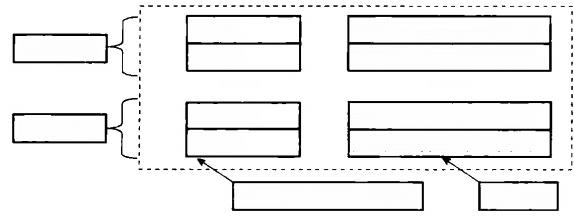
有监督的机器学习方法包括通用的线性模型 (Generalized Linear Model,比如线性回 归 (Linear Regression))、支持向量机 (Support Vector Machine,SVM)、决策树 (Deci- sion Tree)、贝叶斯方法 (Bayesian Method) 等。无监督的机器学习方法包括聚类 (Clus- tering) 、 因子分析 (Factor Analysis)、主成分分析 (Principal Component Analysis)、无 监督神经网络 (Unsupervised Neural Network) 等。样本数据(输人部分、输出部分)以 及训练测试样本划分见图14-8。

下面通过实例讲述如何使用Scikit-learn 进行机器学习模型的训练和预测。

**2.** **装载数据集、训练分类模型、预测**

Scikit-learn 软件包自带一些标准数据集，用于测试目的。比如iris 和 digits 数据集， 用于分类 (Classification, 预测一个分类),Boston house prices 数据集用于回归分析 (Regression, 预测一个价格)。分类是给事物分配一个类别，比如给一张图像分配一个 “狗”或者“不是狗”的分类标签。回归是针对新的输入数据，输出一个值(实数值),比





因变量的值(类别/值)

因变量的值(类别/值)

·

样本n-1 因变量的值(类别/值)

样本n 因变量的值(类别/值)

特征向量，即自变量x 因变量

训练样本

测试样本

样本1 样本2

**图14-8** **样本数据(输入部分、输出部分)以及训练/测试样本划分**

如利用历史价格数据，预测一个未来的价格。

下面的实例利用支持向量机分类模型 (Support Vector Classification) 实现手写体识 别，主要的步骤包括：(1)装载digits 数据集；(2)训练支持向量机分类模型；(3)使用 这个模型对某个手写的数字(最末尾一个样本)进行预测；(4)把最末尾一个样本对应的 图像显示出来。

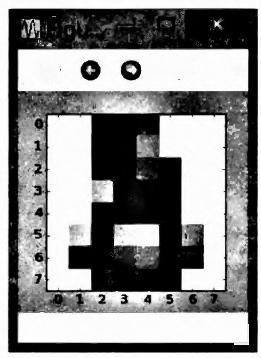
|  |
| --- |
| from sklearn import datasets#(1)装载digits 数据集 iris =datasets.load\_iris()  digits =datasets.load\_digits()  print digits.data  print digits.target  #(2)训练SVC 模型  from sklearn import svm  clf=svm.SVC(gamma=0.001,C=100.)  clf.fit(digits.data[:-1],digits.target[:-1])  #(3)用模型进行预测， -1表示倒数第一个样本 print clf.predict(digits.data[-1:])  import pickle  s=pickle.dumps(clf)  clf2=pickle.loads(s)  print clf2.predict(digits.data[-1:])  from sklearn.externals import joblib joblib.dump(clf,'filename.pkl')  clf3=joblib.load('filename.pkl')  print clf3.predict(digits.data[-1:])  #(4)以图像方式，显示倒数第一个样本 import matplotlib.pyplot as plt  plt.figure(1.figsize=(3,3))  plt.imshow(digits.images[-1],cmap=plt.cm.gray\_r,interpolation ='nearest') plt.show() |



一个数据集包含数据以及描述这些数据的元信息。数据保存在数据集的.data 成员属 性中，包含n 个样本。在有监督学习问题中，响应变量 (Response Variable,即因变量) 保存在 .target 成员属性中。

比如在digits 数据集中，digits.data 成员使得我们可以存取每个样本(特征向量，样本的每 一列表示一个特征); digits.target 则给出这个数据集中每个样本对应的真正的数字(0,1, 2,…,9等10个数字中的某个数字)。比如对应某个数字图像，里面的手写体数字是“8”。

对于digits 数据集，机器学习的任务是训练一个手写体识别的模型。当给出一个新的 图像 (Image) 时，模型能够预测该图像对应哪个数字。在训练样本中，我们在一系列的 图像以及对应的分类标签(也就是对应到数字0,1,2, …,9等10个类别)中，试图训 练出一个预测器 (Estimator), 它能够针对某个没有见过的图像，判断到底是哪个类别， 即对应哪个数字。机器学习模型看到的图像见图14-9。



合 +

出

**图14-9** **机器学习模型看到的图像(它的预测分类是数字8)**

在 Scikit-learn中，一个分类预测器 (Estimator) 实现了两个方法fit(X,y) 和 pre- dict(T), 前者进行训练，后者进行预测。对模型进行训练时，需要把训练数据集(包括 样本X, 因变量y) 作为参数，递给fit 函数。在这个实例中，把倒数第一个图像之前的 所有图像以及对应的分类标签都用上了，使用了[:-1]的数组下标范围指定训练数据 集，即X 为digits.data [:-1],y为 digits.target [:-1]。

上述实例使用了sklearn.svm.SVC 预测器，它实现了支持向量机分类模型。该预测 器的构造函数需要指定两个模型参数，即gamma=0.001 和 C=100 (这些参数的作用请 参考第5章“数据的深度分析”)。

**3.** **模型持久化**

可以把训练好的模型使用Python 内置的持久化格式 (Built-in Persistence Model, 称 为 pickle 格式)保存起来。在上述实例中，如下代码片段，首先把训练好的模型持久化， 然后马上装载，再利用它进行预测。

|  |
| --- |
| import pickle  s=pickle.dumps(clf)  clf2=pickle.loads(s)  print clf2.predict(digits.data[-1:]) |



对模型的持久化和装载，也可以用joblib 的 dump 和 load 方法实现。示例代码如下：

|  |
| --- |
| from sklearn.externals import joblib joblib.dump(clf,'filename.pkl')  clf3=joblib.load('filename.pkl')  print clf3.predict(digits.data[-1:]) |

**4.** **模型的再训练**

如果多次调用模型的fit 函数，Scikit-learn将学习新的模型并且覆盖原来建立的模型。 下面的代码片段，首先把SVC 模型的核函数设定为线性核函数，然后进行学习并且预测。 接着把模型的核函数设定为RBF 函 数 (Radial Basis Function, 即径向基函数),然后进行 学习并且预测。

|  |
| --- |
| clf.set\_params(kermel='linear').fit(digits.data[:-1],digits.target[:1]) print clf.predict(digits.data[-1:])  clf.set\_params(kernel='rbf').fit(digits.data[:-1],digits.target[:-1]) print clf.predict(digits.data[-1:]) |

**5.** **装载数据集、训练回归模型、预测**

上述代码是一个利用机器学习进行分类的实例。我们接着给出一个利用机器学习训练 回归模型进行预测的实例。

这个实例的目的是训练一个线性回归模型，利用给定的特征值，预测房屋的价格 (Housing Price)。① 在代码中，(1)首先装载Boston house prices数据集，并且对各个数 据列进行了重命名；(2)调用.DESCR方法，显示数据集的元信息；(3)创建线性回归 模型 (Linear Regression), 并且用整个数据集进行训练，然后计算均方误差 (Mean Squared Error,MSE);(4) 对数据集进行划分，划分成训练数据集和测试数据集，然 后分别对两个数据集计算均方误差；(5)最后计算训练数据集和测试数据集的残差 (Residual)。

Scikit-learn 支持若干交叉验证的方法。交叉验证 (Cross Validation,CV) 是验证预 测器的预测性能(预测器在新数据上的预测准确率， Accuracy of Supervised Models on Unseen Data) 的一种统计分析方法。其基本思想是将原始数据 (Dataset) 进行分组， 一 部分作为训练集 (Train Set),另一部分作为验证集 (Validation Set)。首先用训练集对分 类器进行训练，再利用验证集来测试训练得到的模型 (Model), 以此来作为评价分类器的 性能指标。

主要的CV 方法包括：(1) Hold-(ut Method。将原始数据随机分为两组， 一组作为 训练集， 一组作为验证集。利用训练集训练分类器，然后利用验证集验证模型，记录最后 的分类准确率。②该方法原理简单，只需随机把原始数据分为两组即可。但是，该方法没 有体现交叉的思想，在测试集上分类准确率的高低，与原始数据的分组有很大的关系，因 此，这种方法得到的结果并不具有说服力。(2) K-fold Cross Validation(K-CV)。将原始 数据分成K 组 (K≥2, 一般是均匀划分),将每个数据子集分别当做一次验证集，其余的

①<http://www.bigdataexaniner.com/2016/03/31/how-to-run-linearregression-in-python-scikit-learn.>

② 对于分类器，评价其分类准确率，对于回归模型.则评价其预测误差。



K-1 个数据子集作为训练集，这样会得到K 个模型，用这K 个模型在验证集上的分类准 确率的平均数，作为此 K-CV 下分类器的性能指标。K-CV 可以有效地避免过拟合以及欠 拟合状况的发生，最后得到的结果比较具有说服力。(3) Leave-One-Out Cross Validation (LOO-CV) 。 假设原始数据有 N 个样本，每个样本单独作为验证集.其余的N—1 个样本 作为训练集。LOO-CV 会得到 N 个模型，用这N 个模型在验证集上分类准确率的平均 数，作为此LOO-CV 下分类器的性能指标。

相对于K-CV,LO(-CV 具有两个优势：首先，每一回合中几乎所有的样本都用于训 练模型，因此最接近原始样本的分布，这样评估所得的结果比较可靠；其次，实验过程中 没有随机因素会影响实验数据，确保实验过程是可以复制的。LOO-CV 的缺点则是计算成 本高。需要建立的模型数量与原始数据样本数量相同，当原始数据样本数量相当多时，效 率低下，可以考虑利用并行计算减少LO(-CV 需要的时间。

为了检验线性回归模型的拟合优度，用 Im.score() 计算判定系数(是回归平方和 (SSR) 在总变差 (SST) 中所占的比重，请参考第5章“数据的深度分析”的相关内容)。 判定系数描述了总变差 (SST) 中有多大的比例是由模型来解释的，也就是预测值和实际 值是否足够接近。这个系数越接近1,模型的拟合优度越好。

具体的代码如下，代码的关键语句已经增加了注释。

|  |
| --- |
| import numpy as np import pandas as pd  import scipy.stats as stats  import matplotlib.pyplot as plt import sklearn  from sklearn.datasets import load\_boston  #(1)装载数据集，显示元信息 boston=load\_boston()  print boston.keys()  print boston.feature\_names print boston.DESCR  #创建DataFrame  bos =pd.DataFrame(boston.data)  print bos.head()  #设定各个属性列的名称  bos.columns =boston.feature\_names print bos.head()  #增加价格PRICE’列  print boston.target[:5] bos[PRICE']=boston.target print bos.head() |

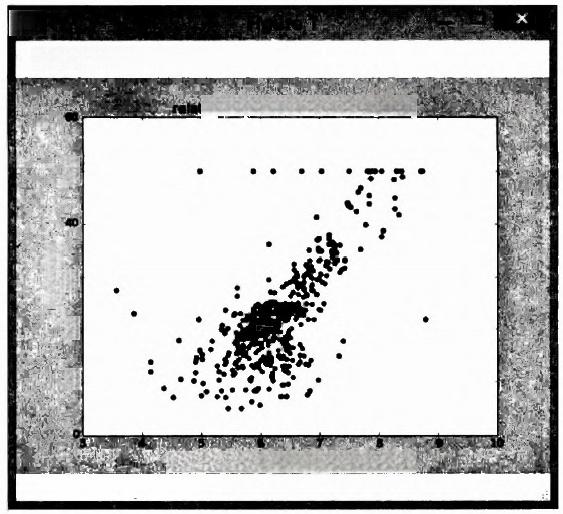
|  |
| --- |
| #(2)用整个数据集，进行线性回归模型训练  from sklearn.linear\_model import LinearRegression X=bos.drop(PRICE',axis=1)  lm=LinearRegression()  lm.fit(X,bos.PRICE)  print'model score',lm.score(X,bos.PRICE) #(3)显示各个变量的系数  pd.DataFrame(zip(X.columns,1m.coef\_),columns=['features','estimated coe- eficients'])  #(4)显示RM变量和PRICE变量(因变量)的散点图 plt.scatter(bos.RM,bos.PRICE)  plt.xlabel('average number of rooms per dwelling(RM)')  plt.ylabel('house price')  plt.title('relationship between RM and price')  plt.show()  print 'plot 1 shown'  #使用已经训练出来的线性回归模型，进行预测 lm.predict(X)[0:5]  #比较实际价格和预测价格的散点图  plt.scatter(bos.PRICE,1m.predict(X))  plt.xlabel('prices:$Y\_i$')  plt.ylabel('predicted prices:$\hat{Y}\_i$')  plt.title('prices vs predicted prices:$Y\_i$vs $\hat{Y}\_i$')  plt.show()  print 'plot 2 shown' #计算均方误差  mseFull=np.mean((bos.PRICe-1m.predict(X))\*\*2) print mseFull  #手工(manual)划分训练数据集、测试数据集  X\_train=x[:-50]  X\_test=x[-50:]  Y\_train=bos.PRICE[:-50] Y\_test=bos.PRICE[-50:] print X\_train.shape  print X\_test.shape  print Y\_train.shape.  print Y\_test.shape |

|  |
| --- |
| #手工方式划分训练数据集和测试数据集，不是一个好办法  #因为有可能划分出来的训练数据集代表性差#比如只考虑到廉价的房子，而没 有考虑到昂贵的房子  #模型的泛化能力就差了  #利用train\_test\_split函数，自动(automatic)划分训练数据集和测试数据集 from sklearn.cross\_validation import train\_test\_split  X\_train,X\_test,Y\_train,Y\_test =train\_test\_split(X,bos.PRICE,test\_size = 0.33,random\_state=5)  print X\_train.shape print X\_test.shape print Y\_train.shape print Y\_test.shape  #在训练数据集上训练模型，然后在训练数据集上和测试数据集上进行预测 #并且显示训练数据集上、测试数据集上的均方误差  lm=LinearRegression()  lm.fit(X\_train,Y\_train)  pred\_train=1m.predict(X\_train)  pred\_test=1m.predict(X\_test)  print"Fit a model X\_train,and calculate MSE with X\_train,Y\_train:", np.mean((Y\_train-1m.predict(X\_train))\*\*2)  print "Fit a model X\_train,and calculate MSE with X\_test,Y\_test:",np.mean ((Y\_test-1m.predict(X\_test))\*\*2)  #显示训练数据集上和测试数据集上的残差(Residuals)  plt.scatter(1m.predict(X\_train),1m.predict(X\_train)-Y\_train,c='b',s= 40,alpha=0.5)  plt.scatter(lm.predict(X\_test),1m.predict(X\_test)-Y\_test,c='g',s=40, alpha=0.5)  plt.hlines(y=0,xmin=0,xmax=50)  plt.title(Residual plot using training(blue)and test(green)data') plt.ylabel('Residuals')  plt.show()  print 'plot 3 shown' |

图14-10是执行上述代码以后生成的关于自变量RM 和因变量House Price关系的散点 图。从图中可以看出，在RM 变量和House Price变量之间有很强的相关关系 (Correlation)。

图14-11是执行上述代码以后生成的对预测值和实际价格进行比较的散点图。从图 中可以看出，当房屋的实际价格较高时，预测的误差较大。

houe pice



2

合 ◎ + 原 ■

lanship betwaen RM and prce

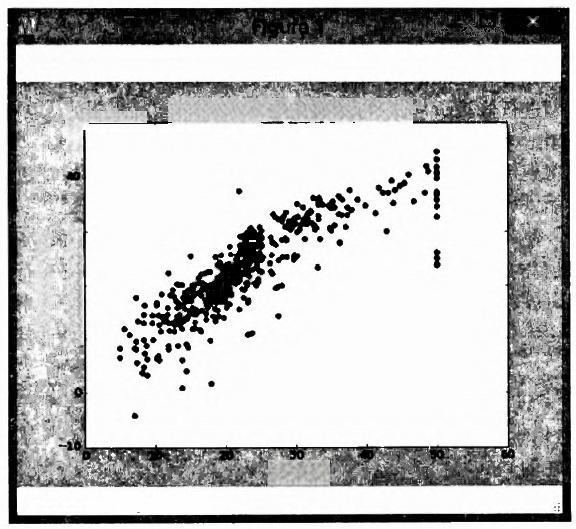
30

10

oe nbro llho

**x=6.00605 y=31.2658**

**图14-10** **关于自变量RM变量和因变量House Price关系的散点图**

**+** **G ☑**

**合** **◎**

plkcsv prodicted presst

50

叹

prc

6

prdicted

piy

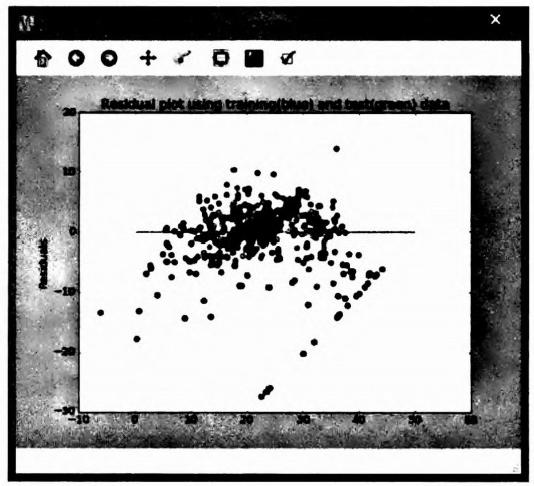
**y=13.6709**

**x=39.6774**

**图14-** **11** **对预测值和实际价格进行比较的散点图**

图14- 12是执行上述代码以后生成的训练数据集上和测试数据集上的残差图。绘制 残差图 (Residual Plot) 可以以图形的方式观察到预测值和实际值之间的误差。如果模型 对数据的拟合程度很好，那么预测值和实际值之间的残差将随机地散落在纵坐标为0的直 线上边和下边，表示预测值偏高或者偏低，但是离实际值不远。





**图14-12** **残差图**

说明：深灰色圆点表示训练数据集上的残差，浅灰色圆点表示测试数据集上的残差。

如果散点图的点离开纵坐标为0的直线较远，表示模型有可能没有捕抓到自变量和因 变量的关系，预测效果不好，误差太大。原因可能是某两个自变量之间存在相互作用 (Interaction between 2 Variables), 或者有可能数据是和时间相关的 (Time Dependent)。 这时候需要对模型进行修正。

**6.** **数据规范化、特征的选取、多种算法的比较、算法参数的优化**

下面的实例将展示如何对数据进行预处理(规范化)、如何对特征进行选择、如何比 较不同算法的性能、如何对算法的参数进行优化等。

这个实例①,使用从 UCI Machine Learning Repository 下载的 pima-indians-diabe- tes.data (根据患者的若干指标，判断患者是否患有糖尿病)数据集进行实验。具体代码 如下，关键的代码段和语句，已经做了必要的注释，方便读者阅读和理解。

|  |
| --- |
| import numpy as np import urllib  #(1)装载数据集  井数据集所在的URL  url="<http://archive.ics.uci.edu/ml/machine-learning-databases/pima-in>- dians-diabetes/pima-indians-diabetes.data"  #下载文件  raw\_data=urllib.urlopen(url)  井装载文本文件，格式为CSV,间隔符为逗号，装载为numpy matrix |



① 详 情 请 参 考 <https://kukuruku.co/post/introduction-to-machine-learning-with-python-andscikit-learn/.>

dataset=np.loadtxt(raw\_data,delimiter=",")

#分开X和 y(分类标签) X=dataset[:,0:8]

y=dataset[:,8]

print X[0:5,]

print y[0:5,]

print 'load data done.'

#(2)规范化与标准化

from sklearn import preprocessing

井对数据进行规范化normalized

normalized\_X=preprocessing.normalize(X)

井对数据进行标准化standardized

standardized\_X=preprocessing.scale(X)

print 'normalize&standardize data done.'

#(3)特征选择

from sklearn import metrics

from sklearn.ensemble import ExtraTreesClassifier model=ExtraTreesClassifier()

model.fit(X,y)

井显示各个特征的相对重要性(relative importance) print 'feature\_importances'

print(model.feature\_importances\_)

井特征选择

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import LogisticRegression model=LogisticRegression()

#创建Recursive Feature Elimination模型，选择3个特征 rfe=RFE(model,3)

rfe=rfe.fit(X,y)

#报告support 和 ranking

print 'feature support&ranking' print(rfe.support\_)

print(rfe.ranking\_)

#(4)训练LogisticRegression,用它进行预测，评价该模型 from sklearn import metrics

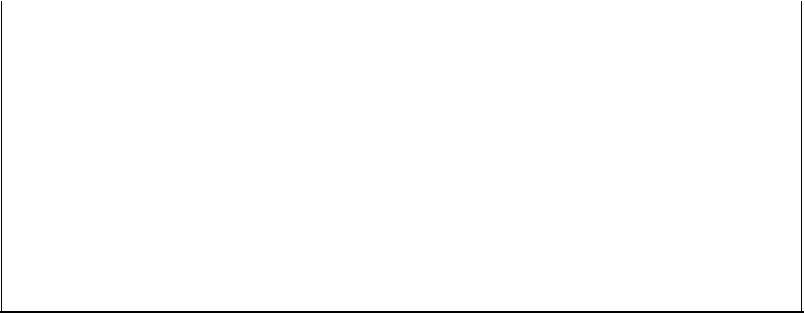
from sklearn.linear\_model import LogisticRegression model=LogisticRegression()

model.fit(X,y)

|  |
| --- |
| print(model)  #利用模型进行预测  expected=y  predicted=model.predict(X)  #显示模型的分类效果报告，以及模型的混淆矩阵  print(metrics.classification\_report(expected,predicted)) print(metrics.confusion\_matrix(expected,predicted))  #训练GaussianNB,用它进行预测，评价该模型 from sklearn import metrics  from sklearn.naive\_bayes import GaussianNB model=GaussianNB()  model.fit(X,y)  print(model)  #利用模型进行预测  expected=y  predicted=model.predict(X)  井显示模型的分类效果报告，以及模型的混淆矩阵  print(metrics.classification\_report(expected,predicted)) print(metrics.confusion\_matrix(expected,predicted))  # 训 练KNeighborsClassifier, 用它进行预测，评价该模型 from sklearn import metrics  from sklearn.neighbors import KNeighborsClassifier #fit a k-nearest neighbor model to the data  model=KNeighborsClassifier()  model.fit(X,y)  print(model)  #利用模型进行预测  expected=y  predicted=model.predict(X)  #显示模型的分类效果报告，以及模型的混淆矩阵  print(metrics.classification\_report(expected,predicted)) print(metrics.confusion\_matrix(expected,predicted))  #训练DecisionTreeClassifier, 用它进行预测，评价该模型 from sklearn import metrics  from sklearn.tree import DecisionTreeClassifier #fit a CART model to the data  model=DecisionTreeClassifier()  model.fit(X,y)  print(model) |



|  |
| --- |
| #利用模型进行预测  expected=y  predicted=model.predict(X)  #显示模型的分类效果报告，以及模型的混淆矩阵  print(metrics.classification\_report(expected,predicted))  print(metrics.confusion\_matrix(expected,predicted))  #训练Support Vector Classifier,用它进行预测，评价该模型  from sklearn import metrics from sklearn.svm import SVC #fit a SVM model to the data model=SVC()  model.fit(X,y)  print(model)  #利用模型进行预测 expected=y  predicted=model.predict(X)  #显示模型的分类效果报告，以及模型的混淆矩阵  print(metrics.classification\_report(expected,predicted))  print(metrics.confusion\_matrix(expected,predicted))  #(5)使用GridSearchCV,优化Ridge模型参数 import numpy as np  from sklearn.linear\_model import Ridge  from sklearn.grid\_search import GridSearchCV #prepare a range of alpha values to test  alphas =np.array([1,0.1,0.01,0.001,0.0001,0])  #使用给定的alpha值列表，创建和拟合脊回归模型(ridge regression model) model=Ridge()  grid=GridSearchCV(estimator =model,param\_grid=dict(alpha=alphas)) grid.fit(X,y)  print(grid)  #显示grid search的结果 print(grid.best\_score\_)  print(grid.best\_estimator\_.alpha)  #使用RandomizedSearchCV.优化Ridge模型参数 import numpy as np  from scipy.stats import uniform as sp\_rand from sklearn.linear\_model import Ridge  from sklearn.grid search import RandomizedSearchCV  #prepare a uniform distribution to sample for the alpha parameter |



param\_grid={'alpha':sp\_rand()

#使用随机产生的alpha 值，创建和拟合脊回归模型(ridge regression model) model=Ridge()

rsearch=RandomizedSearchCV(estimator=model,param\_distributions=param\_ grid,n\_iter=100)

rsearch.fit(X,y)

print(rsearch)

#显示 random parameter search的结果 print(rsearch.best\_score\_)

print(rsearch best\_estimator\_.alpha)

对代码的说明如下：(1)首先，数据需要被加载到内存中才能进行后续操作。该实例 使用NumPy 的 loadtxt 函数来加载CSV 格式的数据。(2)规范化 (Normalization) 是把 向量进行单位向量化。标准化 (Standardization) 是把数据缩放成均值为0、具有单位方 差的数据(下面将给出示例代码和运行结果)。(3)设计机器学习模型的一项重要工作是 特征工程，即对特征进行选择。特征工程有时候靠设计者的直觉和领域专业知识。在已经 给定一系列特征的情况下，Scikit-learn 提供若干方法来对其进行选择。其中，递归特征消 除算法 (Recursive Feature Elimination,RFE) 是其中一个特征选择算法。(4)创建更加 准确的预测器，需要对算法的参数进行优化，也就是选择合适的预测器参数 (Choosing Correct Parameters)。幸运的是，Scikit-learn 提供了很多函数来帮助解决这个问题，包括 RandomizedSearchCV 和GridSearchCV 方法，上述实例中分别用这两个函数对 Ridge 模型 的 alpha 参数进行了优化。(5)对模型的评价， 一个是打印模型的分类效果报告 (classifi- cation\_report), 一个是打印模型的混淆矩阵 (confusion\_matrix)。下面通过实例来解释 如何解读分类效果报告和模型的混淆矩阵。

模型的分类效果报告显示了模型在类别0和类别1上的分类准确率、召回率、 F1- 得 分以及支持度，如表14-8所示。这几个概念的含义，请参考第7章“文本分析”及第5 章“数据的深度分析”。

**表14-8** **模型的分类效果报告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类别 | Precision | Recall | F1-score | Support |
| 0 | 0.79 | 0.90 | 0.84 | 500 |
| 1 | 0.74 | 0.55 | 0.63 | 268 |

模型的混淆矩阵显示4个计数，分别表示实际值为类别0而模型预测值为类别0、实 际值为类别0而模型预测值为类别1(预测错误，假阴性 (False Negative))、实际值为类 别1而模型预测值为类别0(预测错误，假阳性 (False Positive)),以及实际值为类别1 而模型预测值为类别1的样本数量，如表14-9所示。

**表14-9** **模型的混淆矩阵**

|  |  |  |
| --- | --- | --- |
| 实际值 | 预测值 | |
| 0 | 10 |
| 0④ | 448 | 52 |
| 10 | 121 | 147 |



下面是对数据进行规范化和标准化的示例代码。

|  |
| --- |
| Y=[[3,15,7,13,11],  [15,3,13,11,7],  [9,5,7,8,6]]  import pandas as pd X=pd.DataFrame(Y)  #print X  from sklearn import preprocessing  #对数据进行规范化normalized  normalized\_X=preprocessing.normalize(X) print 'normalized\_X result'  print normalized\_X  #对数据进行标准化standardized  standardized\_X=preprocessing.scale(X) print 'standardized\_X result'  print standardized\_X |

上述代码的输出结果如下：

|  |
| --- |
| normalized\_X result#把向量进行单位向量化  [[0.12532680.626633990.29242920.543082790.45953159]  [0.626633990.12532680.543082790.459531590.2924292]  [0.563601860.313112150.4383570.500979430.37573457]]  standardized\_X result#把数据缩放成均值为0、具有单位方差的数据 [[-1.224744871.3970014-0.707106781.135549951.38873015]  [1.22474487-0.889000891.414213560.16222142-0.46291005] [0.-0.50800051-0.70710678-1.29777137-0.9258201]] |

数据预处理，除了对数据进行规范化、标准化之外，还可以对数据进行二值化处理和 数据分桶处理 (Binning), 示例代码和运行结果如下。二值化是把数据进行如下的转换， 大于阈值的转换成1,小于阈值的转换成0。

|  |
| --- |
| #数据的二值化处理  from sklearn import preprocessing  X=[[1.,-1.,2.], [2.,0.,0.], [0.,1.,-1.]]  binarizer=preprocessing.Binarizer().fit(X)#fit does nothing print binarizer  print binarizer.transform(X) |

|  |
| --- |
| #结果为  #[[1.0.1.] #[1.0.0.] #[0.1.0.]] |

数据分桶是把一个值域范围内的数据分成若干区段。下面的实例把0～100的分数划 分成(0,25),(25,50),(50,75),(75,100)等几个分数段，并且赋予其'Low''Okay' Good'Great '等类别标签。

|  |
| --- |
| #数据的分桶处理(Binning) import pandas as pd  #创建 一个DataFrame  raw\_data ={'regiment':['Nighthawks','Nighthawks','Nighthawks','Nighthawks',  Dragoons',Dragoons','Dragoons',Dragoons','Scouts','Scouts','Scouts','Scouts', 'company':['1st','1st','2nd,'2nd,'lst','1st','2nd,2nd,'lst','1st','2nd,'2nd'],  'name':['Miller','Jacobson','Ali','Milner','Cooze','Jacon','Ryaner','Sone',  Sloan',Piger',Riani','Ali'],  'preTestScore':[4,24,31,2,3,4,24,31,2,3,2,3],  'postTestScore':[25,94,57,62,70,25,94,57.62,70,62,70]}# 表格数据  df =pd.DataFrame(raw\_data,columns =['regiment','company','name','pre- TestScore','postTestScore'])# 给出各列的名称  print(df)  bins=[0,25,50,75,100]# 各个桶的上下界  group\_names=['Low','Okay','Good','Great']# 各个桶的类别标签  categories =pd.cut(df['postTestScore'],bins,labels =group\_names)# 分 桶 操作  df['categories']=pd.cut(df['postTestScore'],bins,labels=group\_names)# 增 加 categories 字 段  df['scoresBinned']=pd.cut(df['postTestScore'],bins) 井增加 scoresBinned 字段  print(categories)  pd.value\_counts(df['categories'])# 显示 categories 字段的直方图  print(df) |

除了对数据进行预处理之外，还可以对类别标签(也就是各个分类)进行预处理 (Label Preprocessing) 。LabelEncoder 把类别标签编码为0～ n-1 的数值，n 是类别标签 的数量。示例代码如下，关键代码做了注释。

第一个实例使用LabelEncoder 对数值型标签进行规范化。

|  |
| --- |
| from sklearn import preprocessing  le=preprocessing.LabelEncoder()  le.fit([1,2,2,6])#1 映射到0,2映射到1,6映射到2 print le.classes\_#输出array([1,2,6])  print le.transform([1,1,2,6])#输出 array([0,0,1,2])  print le.inverse\_transform([0,0,1,2])#输出array([1,1,2,6]) |

第二个实例使用LabelEncoder对非数值型标签进行规范化。

|  |
| --- |
| le=preprocessing.LabelEncoder()  le.fit(["paris","paris","tokyo","amsterdam"])#'amsterdam '映射到0, paris'映射到1,'tokyo'映射到2  list(le.classes\_)# 输出['amsterdam','paris','tokyo']  le.transform(["tokyo","tokyo","paris"])# 输出array([2,2,1])  list(le.inverse\_transform([2,2,1]))# 输出['tokyo','tokyo','paris'] |

Scikit-learn 库提供了若干二值分类 (Binary Classification) 算法。我们可以使用One-vs- All 方法，对这些分类器进行扩展，使之能够进行多值分类 (Multi-Class Classification)。

(ne vs All方法是在学习阶段，针对k 个类的每个类别，训练一个二值分类器。为了 进行二值分类器的训练，需要把多值分类标签 (Multi-Class Label) 转换成二值分类标签 (Binary Label),表示样本属于某类，或者不属于某类。 LabelBinarizer的 transform方法 用于此阶段。在预测阶段，基于若干分类器的预测结果，根据哪个预测的置信度最高，对 样本进行分类。LabelBinarizer 的 inverse\_transform 方法用于此阶段。 LabelBinarizer 的 示例代码如下，关键语句进行了注释。

|  |
| --- |
| from sklearn import preprocessing lb=preprocessing.LabelBinarizer()  lb.fit([1,2,6,4,2])# 输出 LabelBinarizer(neg\_label=0,pos\_label=1, sparse\_output =False)  lb.classes\_# 输出array([1,2,4,6]), 表示4个类  print lb.transform([1,6])#输出 array([[1,0,0,0],[0,0,0,1]]),一 个2维矩 阵，表示第一个样本属于第一个类别，第二个样本属于第四个类别  #如果类别只有两个(K=2), 那么 tranform的结果是一个列向量column vector,  而不是一个2维矩阵  lb=preprocessing.LabelBinarizer()  print lb.fit\_transform('yes','no','no',['yes'])# 输出 array([[1],[0],[0], [1]]),是一个列向量 |

**14.4.3 深度学习库Keras (基于Tensorflow,Theano)**

Keras 是基于Theano 的一个深度学习软件框架(目前也支持 Tensorflow 作为后端)。 它用Python 语言编写，在设计上参考了Torch 深度学习库，是一个高度模块化的深度学 习库，支持GPU 和CPU 。 用户可以选择安装 Theano 或者Tensorflow 作为后端，再装上 Keras, 修改其配置文件，指定所使用的后端。

Keras 文档齐全，接口非常直观，因此很容易学习。它本质上是在Theano 和 Tensor- flow 上增加了一个接口层，用户可以采用搭积木的方式建立深度学习模型。 Keras 的社区 和论坛非常活跃，版本更新非常快。Keras 的主要缺点是它不如原生的Theano 和 Tensor- flow 库执行得那么快。

Keras 的主要模块包括：(1)(ptimizers, 是优化模块，提供了通用的优化方法，包括随 机梯度下降法 (Stochastic Gradient Descent,SGD); (2)Objectives, 是目标函数模块；

(3)Activations, 是激活函数模块；(4) Initializations, 是参数初始化模块；(5) Layers 模块 则包含了构造深度神经网络的基本构件(层);(6) Preprocessing,是预处理模块；(7) Mod- els 模块把各种基本组件组合起来，组成一个深度神经网络。

下面通过两个实例①,展示利用Keras 进行深度学习模型构建的方便性。这两个实例， 一个是使用多层感知机 (Multi-Layer Perceptrons) 实现手写体识别 (MNIST 数据集), 整个神经网络有2个隐藏层，每层有512个神经元，前后层之间采用全连接。另外一个实 例使用相同的数据集，但是使用了卷积神经网络 (Convolved Neural Network,CNN), 整个神经网络由两个卷积层、1个子采样层以及1个全连接的隐藏层构成。具体代码如下， 代码做了注释，方便阅读和理解。

|  |
| --- |
| #使用MLP进行手写体识别  #"Trains a simple deep NN on the MNIST dataset  #Gets to 98.40%test accuracy after 20 epochs  #(there is a lot of margin for parameter tuning) #2 seconds per epoch on a K520 GPU.  #"  from future\_ import print\_function #导人keras库  import keras  from keras.datasets import mnist  from keras.models import Sequential  from keras.layers import Dense,Dropout from keras.optimizers import RMSprop |



①<https://github.com/fchollet/keras/tree/master/examples.>

|  |
| --- |
| #设置一些参数 batch\_size =128 num\_classes =10 epochs =1  #(1)装载数据，切分成训练数据集、测试数据集  #the data,shuffled and split between train and test sets (x\_train,y\_train),(x\_test,y\_test)=mnist.load\_data()  #转换成784列(784个特征)  x\_train=x\_train.reshape(60000,784)  x\_test=x\_test.reshape(10000,784)  x\_train=x\_train.astype('float32')  x\_test =x\_test.astype('float32')  #把像素灰度转换成[0,1]之间的浮点数 x\_train/=255  x\_test/=255  print(x\_train.shape[0],'train samples')  print(x\_test.shape[0],'test samples')  #convert class vectors to binary class matrices y\_train=keras.utils.to\_categorical(y\_train,num\_classes)  Y\_test =keras.utils.to\_categorical(y\_test,num\_classes)  print(y\_train)  print(x\_train)  #(2)建立机器学习模型  model=Sequential() 井建立顺序模型，即前向反馈神经网络  model.add(Dense(512,activation ='relu',input\_shape=(784,)))#普通神经网 络层，512个神经元  井输人为784个特征构成的向量  model.add(Dropout(0.2))# 利用Dropout 技术，避免过拟合  model.add(Dense(512,activation ='relu'))#普通神经网络层，512个神经元  model.add(Dropout(0.2))#利用Dropout技术，避免过拟合  model.add(Dense(10,activation ='softmax')#输出层，10个分类 model.summary()  #编译  model.compile(loss='categorical\_crossentropy', |

|  |
| --- |
| optimizer=RMSprop(),  metrics=['accuracy'])  #(3)训练  history=model.fit(x\_train,y\_train,  batch\_size =batch\_size,epochs =epochs,  verbose =2,validation\_data=(x\_test,y\_test))  #(4)评估  score =model.evaluate(x\_test,y\_test,verbose=0)  print(Test loss:',score[0])  print(Test accuracy:',score[1])  #(5)预测  one\_sample=x\_test[-1] print(one\_sample.shape)  one\_sample=one\_sample.reshape(1,784)  print(one\_sample.shape)  #显示一个样本  image\_sample=one\_sample.reshape(28,28)  print(image\_sample.shape)  import matplotlib.pyplot as plt plt.figure(1,figsize=(3,3))  plt.imshow(image\_sample,cmap =plt.cm.gray\_r,interpolation=hearest') plt.show()  井显示预测结果  predicted=model.predict(one\_sample)  predicted\_class =model.predict\_classes(one\_sample,verbose=0)  print(predicted)  print(predicted\_class) |

|  |
| --- |
| #使用CNN进行手写体识别  #"Trains a simple convnet on the MNIST dataset #Gets to 99.25%test accuracy after 12 epochs  #(there is still a lot of margin for parameter tuning) #16 seconds per epoch on a GRID K520 GPU.  #"  from future\_import print\_function #导人keras库 |

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense,Dropout,Flatten from keras.layers import Conv2D,MaxPooling2D

from keras import backend as K

#设置一些参数 batch\_size=128 num\_classes=10 epochs =1

#input image dimensions img\_rows,img\_cols=28,28

#(1)装载数据，切分成训练数据集、测试数据集

#the data,shuffled and split between train and test sets (x\_train,y\_train),(x\_test,y\_test)=mnist.load\_data()

if K.image\_data\_format()=='channels\_first':

x\_train=x\_train.reshape(x\_train.shape[0],1,img\_rows,img\_cols)

x\_test=x\_test.reshape(x\_test.shape[0],1,img\_rows,img\_cols) input\_shape =(1,img\_rows,img\_cols)

else:

x\_train=x\_train.reshape(x\_train.shape[0],img\_rows,img\_cols,1)

x\_test =x\_test.reshape(x\_test.shape[0],img\_rows,img\_cols,1) input\_shape =(img\_rows,img\_cols,1)

x\_train=x\_train.astype('float32')

x\_test =x\_test.astype('float32')

#把像素灰度转换成[0,1]之间的浮点数 x\_train/=255

x\_test/=255

print('x\_train shape:',x\_train.shape)

print(x\_train.shape[0],'train samples')

print(x\_test.shape[0],'test samples')

#convert class vectors to binary class matrices y\_train=keras.utils.to\_categorical(y\_train,num\_classes)

y\_test =keras.utils.to\_categorical(y\_test,num\_classes)

#(2)建立机器学习模型

model=Sequential()# 建立顺序模型，即前向反馈神经网络

model.add(Conv2D(32,kernel\_size=(3,3),

activation ='relu',

input\_shape =input\_shape))#2维卷积层

#输入为一幅图像，包含img\_rows\*img\_cols 个像素

model.add(Conv2D(64,(3,3),activation ='relu'))#2维卷积层

model.add(MaxPooling2D(pool\_size=(2,2)))# 子采样层

model.add(Dropout(0.25))# 利 用Dropout 技术，避免过拟合

model.add(Flatten())# 把输入数据压扁，即把多维向量变成1维向量

model.add(Dense(128,activation ='relu'))#普通神经网络层，128个神经元

model.add(Dropout(0.5))# 利 用Dropout 技术，避免过拟合

model.add(Dense(num\_classes,activation ='softmax'))#输出层，10个分类

#编译

model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=keras.optimizers.Adadelta(),

metrics=['accuracy'])

#(3)训练

model.fit(x\_train,y\_train,

batch\_size =batch\_size,epochs =epochs,

verbose=2,validation\_data=(x\_test,y\_test))

#(4)评估

score =model.evaluate(x\_test,y\_test,verbose =0)

print("Test loss:',score[0])

print("Test accuracy:',score[1])

#(5)预测

one\_sample=x\_test[-1] print(one\_sample.shape)

one\_sample=one\_sample.reshape(1,28,28,1)

print(one\_sample.shape)

#显示一个样本

image\_sample=one\_sample.reshape(28,28)

print(image\_sample.shape)



import matplotlib.pyplot as plt plt.figure(1,figsize=(3,3))

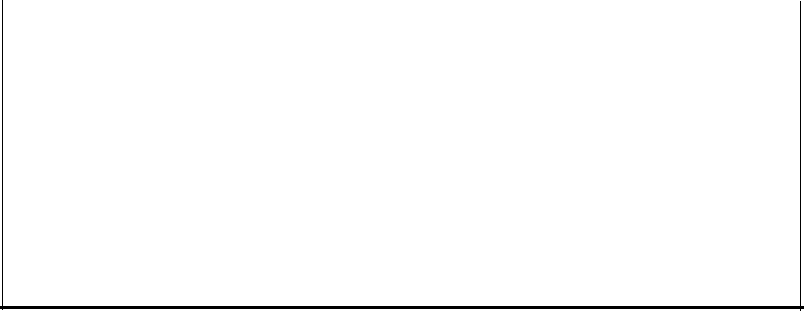
plt.imshow(image\_sample,cmap=plt.cm.gray\_r,interpolation=hearest') plt.show()

#显示预测结果

predicted=model.predict(one\_sample)

predicted\_class =model.predict\_classes(one\_sample,verbose=0)

print(predicted)

print(predicted\_class)

**14.4.4 Matplotlib 介绍与实例**

**1.Matplotlib 介绍**

Matplotlib 是一个2D 绘图库。用户可以使用Matplotlib 生成高质量(达到出版精度， Publication Quality) 的图形，并且可以以多样化的格式进行输出，比如以Postscript的格 式输出，然后包含在 Tex 文件中，进而生成PDF 文档。Matplotlib 是用Python 语言编写 的，它使用了NumPy 及其他函数库，并且经过优化，即便对于比较大的数组，进行图形 化绘制，其性能也是可以接受的。用户可以在Python 程序中调用Matplotlib的函数及其 功能，也可以通过 Python 命令行(shell) 或者IPython web接口，交互式地调用 Mat- plotlib 的函数及其功能。

Matplotlib 提供了面向对象的编程接口 (Object Oriented Interface), 用户只需编写寥 寥几句代码，就可以对数据进行图形绘制，看到数据的可视化效果。对于高级用户 (Power User) 来讲，Matplotlib 提供完全的定制能力，包括设定线型、字体属性、坐标 轴属性等。Matplotlib 提供了众多的图形类型，供用户选择，包括柱状图 (Bar Chart)、 误差图 (Error Chart)、散点图 (Scatter Plot)、功率谱图 (Power Spectra)、直方图 (Histogram) 等。

下面通过一系列的实例，展示Matplotlib 的绘图能力。

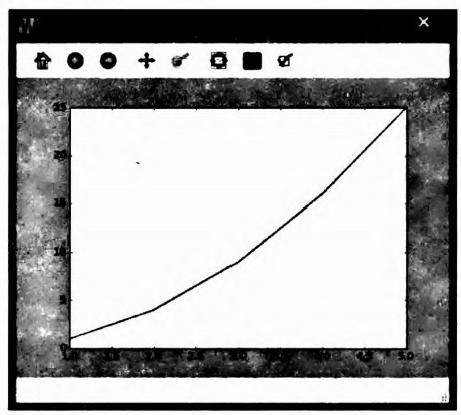
**2. 折线图** **(Line plot)**

折线图用分段折线，展示一组x 值和y 值之间的关系。示例代码如下：

|  |
| --- |
| #绘制折线图  import numpy as np  import matplotlib.pyplot as plt  x=[1,2,3,4,5]# 创建x 数组(列表)  y=[1,4,9,16,25]# 创建 y 数组(列表),和x 的元素一—对应 plt.plot(x,y)# 绘制折线图  plt.show()# 显示绘制结果 |

折线图见图14-13。





**图14-13** **折线图**

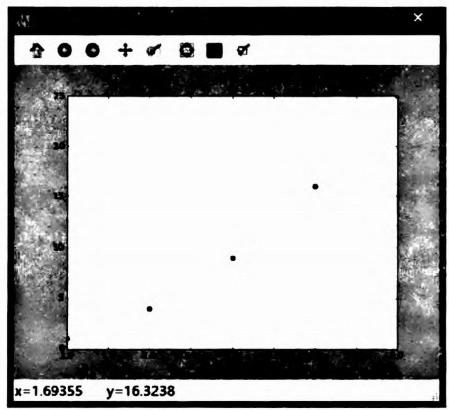
**3. 散点图** **(Scatter plot)**

把上述实例中的pl.plot(x,y) 改成pl.plot(x,y,'o'), 即可绘制散点图。通过散点

图，我们可以看到数据的总体趋势。代码如下：

|  |
| --- |
| #绘制散点图  import numpy as np  import matplotlib.pyplot as plt  x=[1,2,3,4,5]# 创建x 数组(列表)  y=[1,4,9,16,25]# 创建y 数组(列表),和x 的元素一一对应 plt.plot(x,y,'o')# 绘制散点  plt.show()# 显示绘制结果 |

散点图见图14-14。



**图14-14** **散点图**

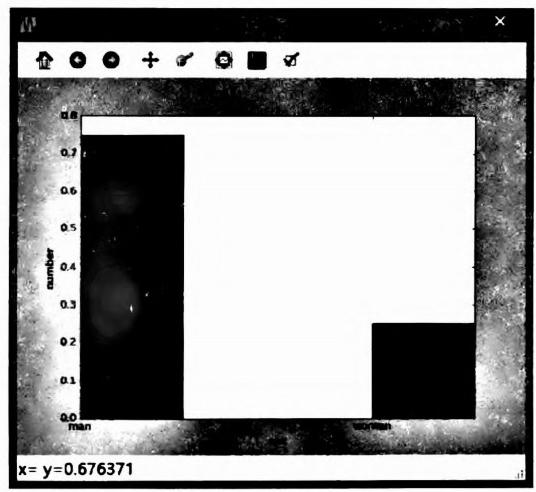
**4. 柱状图与饼图** **(bar chart &pie chart)**

柱状图和饼图都是观察数据的相对大小关系的常用图形。下面是柱状图和饼图的示例 代码，代码中增加了必要的注释，读者可以很方便地理解代码的功能。

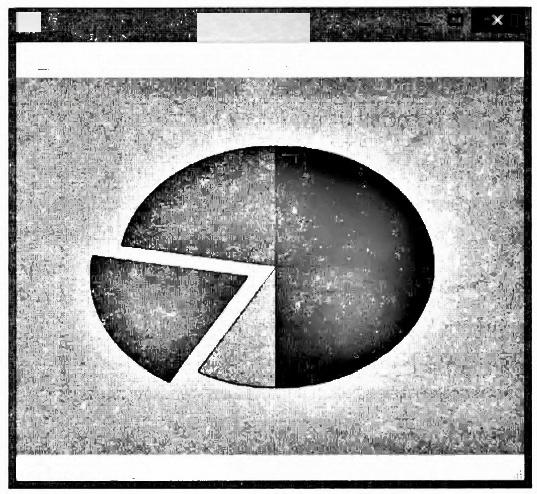
|  |
| --- |
| #绘制柱状图import matplotlib.pyplot as plt  plt.xlabel('sex')# 设 定x 轴的标签  plt.ylabel('number')# 设定y 轴的标签  plt.xticks((0,1),('man','woman'))# 设定x 轴各个刻度的标签  plt.bar(left=(0,1),height =(0.75,0.25),width =0.35,color ="red")#绘 制柱状图  #颜色为红色，柱状的高度分别为0.75和0.25 plt.show()# 显示柱状图 |

|  |
| --- |
| #绘制饼图  import numpy as np  import matplotlib.pyplot as plt  data=np.random.randint(1,11,5)# 利用numpy 生成5个随机的整数，值域(1,11) #x=np.arange(len(data))# 建立x 数组(列表)0,1,2,3,4  plt.pie(data,explode=[0,0,.2,0,0])# 绘制饼图，其中第三个饼突出  plt.show()# 显示饼图 |

柱状图和饼图见图14-15和图14-16。



**图14-15** **柱状图**

奶

igure1

食 **◎** **○** **+**

**x=0.766129 y=0.837289**

**图14-** **16** **饼图**

**5. 多个序列** **(multi series)**

在一个图形上，可以对多个序列(列表)进行对齐和显示。下面的实例分别在一个图 形上显示多个折线以及多个柱状序列。这两个实例都是对上述实例稍加修改实现的。

多个折线实例如下：

|  |
| --- |
| #多条折线  import numpy as np  import matplotlib.pyplot as plt  x=[1,2,3,4,5]# 创建x 数组(列表)  y1=[1,4,9,16,25]# 创建y1 数组(列表),和x 的元素一一对应 y2=[2,3,6,17,21]# 创建y2 数组(列表),和x 的元素一一对应  plt.plot(x,y1)# 绘制一条折线 plt.plot(x,y2)井绘制另一条折线 plt.show()# 显示绘制结果 |

多个柱状序列实例如下：

#多个柱状序列

import numpy as np

import matplotlib.pyplot as plt

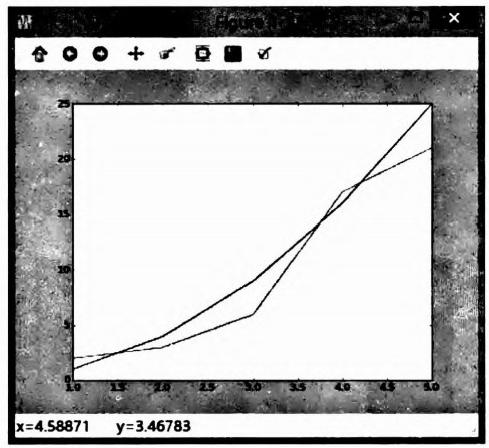
x1=[0,2,4,6,8]

data1=np.random.randint(1,11,5)# 利 用numpy 生成5个随机的整数，值域(1,11)

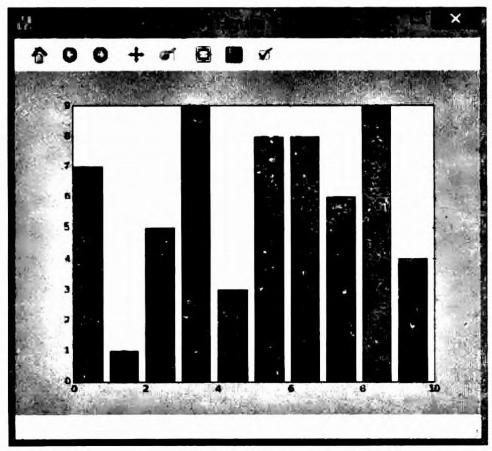
x2=[1,3,5,7,9]# 注意x1和x2 的各个值是不一样的

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | data2=np.random.randint(1,11,5)# | 利 用numpy生成5个随机的整数，值域(1,11) | | plt.bar(x1,data1,color="green")# | 绘制柱状序列 | | plt.bar(x2,data2,color="blue")# | 绘制另一个柱状序列 | | plt.show()# 显示柱状图 |  | |

图形结果见图14- 17和图14- 18。



**图14-17** **多条折线**

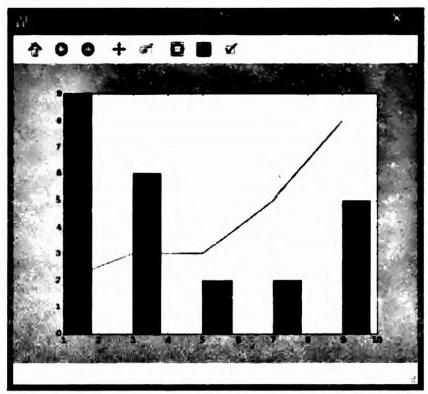


**图14-18** **多个柱状序列**

我们甚至可以把折线图与柱状图混合起来。

|  |  |
| --- | --- |
| #折线图与柱状图混合 import numpy as np  import matplotlib.pyplot as plt  x1=[1,3,5,7,9]  data1=np.random.randint(1,11,5)# 11)  x2=[1,3,5,7,9]  data2=np.random.randint(1,11,5)# 11)  plt.plot(x1,data1,color="green")# plt.bar(x2,data2,color="blue")# plt.show()# 显示绘制结果 | 利用numpy生成5个随机的整数，值域[1,  利 用numpy生成5个随机的整数，值域[1,  绘制折线  绘制柱状序列 |

图形结果见图14-19。

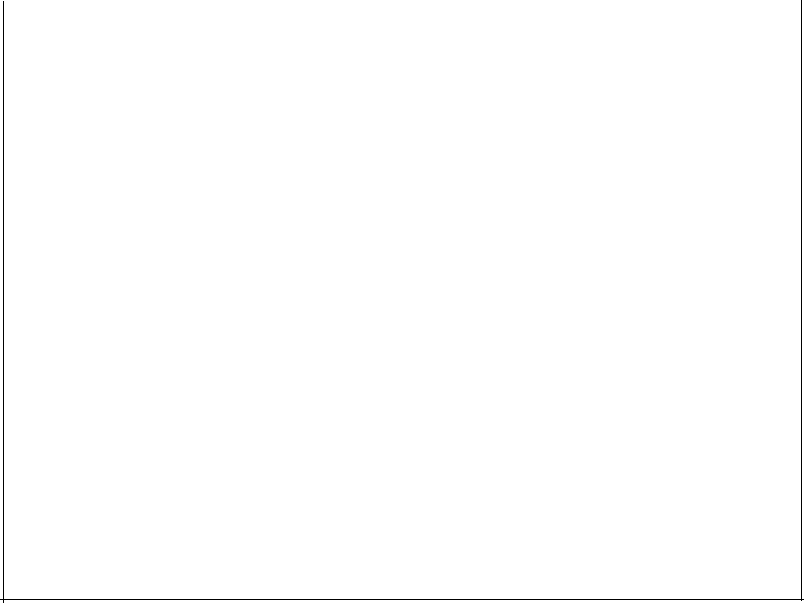


**图14-19** **折线图与柱状图混合**

**6.** **设置图形的属性**

我们可以根据实际需要对图形的一些属性进行设定，包括线条颜色、线型、字体、标注 (Marker) 的样式、图例、坐标轴的刻度、坐标轴的标题、图形的标题等。示例代码如下：

|  |
| --- |
| #设定图形属性  import numpy as np  import matplotlib.pyplot as plt  import matplotlib  font={'family':'Vera Sans', 'weight':'bold,  'size':14}# 字体属性 |



matplotlib.rc('font',\*\*font)# 使 用font 字体

x1=[1,2,3,4,5]

yl=[1,4,9,16,25] x2=[1,2,3,4,5]

y2=[2,5,11,12,17]

plot1=plt.plot(x1,y1,'r--\*¹,label='red 线型为--,即虚线，星型Marker

line')#图形为折线图，颜色为red,

circles')#图形为散点图，颜色

plot2=plt.plot(x2,y2,'go',label='green

为green

plt.title(Plot of y vs.x',\*\*font)#图的标题，使用font 字体

plt.xticks((1,2,3,4,5),('one','two','three','four','five'))# 设置x 轴的刻度 plt.xlabel('x axis',\*\*font)#设置x 轴标题，使用font 字体

plt.ylabel('y axis',\*\*font)#设置y 轴标题，使用font 字体

plt.xlim(0.0,9.0)# 设置x 轴坐标上下界

plt.ylim(0.0,30.)# 设置y 轴坐标上下界

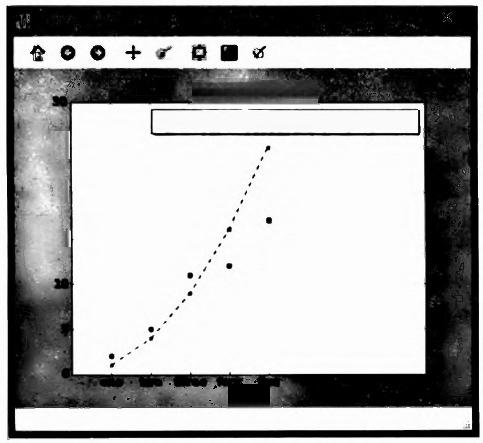
plt.legend(loc='best',shadow=True,ncol=2)# 显示图例

.

plt show()

Legend 的 loc 参数表示图例放置的位置，可以取的值有 best,upper right,upper left,center,lower left,lower right等，分别表示最佳位置、右上、左上、中间、左下、 右下等位置。

如果在当前figure 里，调用plot 时已经指定了label, 那么调用plt.legend() 时，就 不需要指定label 了。上述代码的执行结果如图14-20所示。

*ntoy*

- red line ··groen circles

20

15

y

**图14-20** **设定图形属性**



**7.** **多子图**

使用subplot() 可以快速绘制包含多个子图的图形，它的调用形式为 subplot(num-

Rows,numCols,plotNum) 。subplot 将整个绘图区域等分为 numRows 行 ×numCols 列 个子区域(子图),然后按照从左到右，从上到下的顺序对每个子图进行编号，左上角的 子图的编号为1,其他子图的编号依次为2,3, … 。如果 numRows,numCols 和 plot- Num 这三个数都小于10,可以把它们缩写为一个整数，例如 subplot(323) 。subplot

(323)和subplot(3,2,3) 可以引用相同的子图，即3行2列子图的第3个子图。

下面通过两个实例展示如何在一个图形里显示多个子图。第一个实例分别显示了直方 图和箱图，箱图显示了数据的最大值 (max) 、 最小值 (min) 、 下四分位点(25% percen- tile) 、 上四分位点(75% percentile) 以及中位数 (median,50%percentile) 。 具体代码 如下(已经加了注释):

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  data=np.random.randn(100)# 生成随机数  fig,(ax1,ax2)=plt.subplots(1.2,figsize=(8,4))# 建立一行两列，高度为4、 宽度为8的带子图的图形，返回 fig,ax1 子图1,ax2 子图2进行后续操作  ax1.hist(data)# 建立直方图子图  ax2.boxplot(data)# 建立箱图子图  plt.show()# 显示包含子图的图形 |

第二个实例则在图形上安排上下两个子图，两个子图使用了同样的横坐标的刻度。具 体代码如下：

import numpy as np

import matplotlib.pyplot as plt

plt.figure()# 创建图形

ax1=plt.subplot(211)# 创建子图1,即2行1列的1号子图

ax2=plt.subplot(212)# 创建子图2,即2行1列的2号子图 x=[1,2,3,4,5]

datal=np.random.randint(1.11,5)# 11)

利用numpy生成5个随机的整数，值域[1,

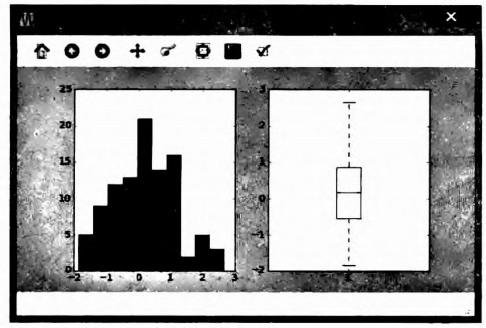
利用numpy生成5个随机的整数，值域[1,

data2=np.random.randint(1.11,5)# 11)

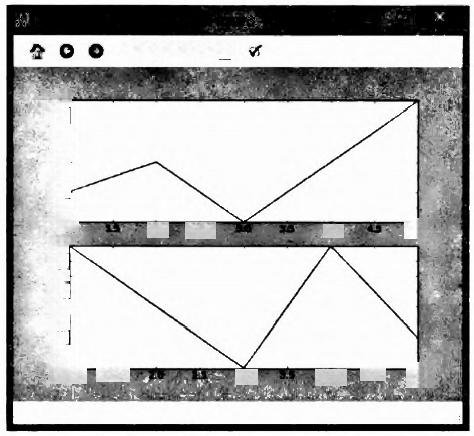
ax1.plot(x,data1) 井绘制折线图

ax2.plot(x,data2)# 绘制折线图 plt.show()# 显示绘制结果

**图形结果见图14-21和图14-22。**



**图14-21** **两个子图(左右)**



+ ◎

200广 5

0

B5

B0

75 to 65

20 215 40 0

70

**6**50

55

50

45

40

35

。 5

**图14-22** **两个子图(上下)**

**8. 股票价格** **K 线** **图**

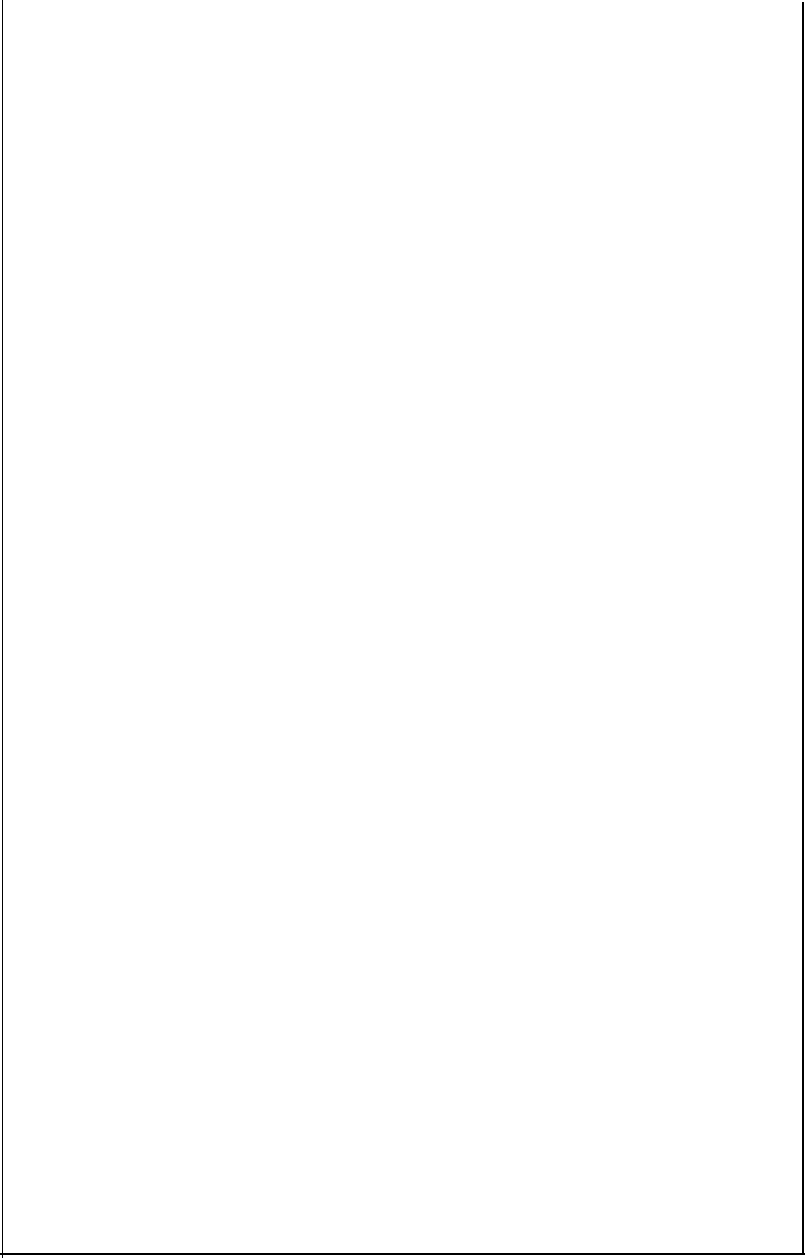
下面的实例①,从yahoo finance 上下载intel 的历史股票价格数据，然后绘制图形。 图形分两个子图，其中一个绘制K 线图(价格子图),另外一个绘制交易量。该段程序还 计算15天和30天的移动平均，绘制在价格子图 (sub plot) 上。具体代码如下，关键语 句做了必要的注释。

|  |  |  |
| --- | --- | --- |
| import matplotlib.pyplot import numpy as np | as plt |  |
| from matplotlib.dates | import | DateFormatter,WeekdayLocator |

① 本 实 例 参 考 了candlestick 实 例 ( 见 <http://matplotlib.org/examples/pylab_examples/finance_demo.html)>

和 moving average实 例 ( 见 <http://matplotlib.org/examples/pylab_examples/finance_work2.html>), 对它们进行了 整合，即把移动平均指标绘制在K 线 图 上 。

|  |
| --- |
| from matplotlib.dates import DayLocator,MONDAY,YEARLY,MonthLocator  from matplotlib.finance import quotes\_historical\_yahoo\_ohlc,candlestick\_ohlc  from matplotlib.finance import plot\_day\_summary\_oclh  def moving\_average(x,n,type='simple'):#定义移动平均的计算方法    compute an n period moving average.  type is 'simple'|'exponential'  材目  x=np.asarray(x)  if type =='simple':  weights =np.ones(n)  else:  weights =np.exp(np.linspace(-1.,0.,n))  weights/=weights.sum()  a=np.convolve(x,weights,mode='full')[:len(x)]  a[:n]=a[n]  return a  #设置plt 的字体格式  plt.rcParams['font.sans-serif']=['SimHei'] plt.rcParams['axes.unicode\_minus']=False  # 从Yahoo Finance 获取Intel 的股票价格  # 井 井 井 井 井 # # # # # # # # # # # # # 井 井 # 井 # # 井 # # # # 井 井 井 # # # # #这段代码quotes\_historical\_yahoo\_ohlc 运行不了  # 井 # # # # # 井 井 井 # # # # # # 井 # # # # # # # # # # # # # # # # # # # # #ticker='INIC'  #datel=(2015,1,1)# 起始日期，格式为(年，月，日)元组  #date2=(2016,1,1)# 结束日期，格式为(年，月，日)元组  #quotes =quotes\_historical\_yahoo\_ohlc(ticker,date1,date2)  井if len(quotes)==0:  # raise SystemExit  #print quotes[:5]  # 井 # # # # 井 # # # # # # 井 # # # # 井 # # # # # 井 # # # # # 井 井 井 井 # 井 #  # # 井 # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #  # 井 # # # # # 井 井 井 # # # # # # # 井 井 # # # # 井 # 井 # # # # # # 井 井 # # # # #用这段代码代替  # 参 考<http://blog.csdn.net/u014281392/article/details/73611624> |

# # # # 井 # # # # # # # # # # # # 井 # # # # # # # 井 # # # # # # # # # # # import tushare as ts

井 8matplotlib inline

quotes\_pre =ts.get\_k\_data('002739','2017-01-01')

quotes\_pre.info()

print(quotes\_pre[:3])

from matplotlib.pylab import date2num import datetime

def date\_to\_num(dates):

num\_time=[]

for date in dates:

date\_time=datetime.datetime.strptime(date,'%Y-%m-%d')

num\_date =date2num(date\_time)

num\_time.append(num\_date) return num\_time

#dataframe 转换为二维数组

quotes =quotes\_pre.as\_matrix()

num\_time =date\_to\_num(quotes[:,0]) quotes[:,0]=num\_time

# 日 期 ， 开 盘 ， 收盘， 最 高 ， 最低，成交量， 代码

print(quotes[:3])

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # 井 # # # # # #

# # # # # # # # # # # # # # # # # # 井 # # # # # # # # # 井 # # # # # 井 井 #

#刻度信息

monthdays =MonthLocator()#WeekdayLocator(MONDAY)#主要刻度

alldays =DayLocator() #次要刻度

#weekFormatter=DateFormatter('%b%d') 井如：Jan 12

monthdaysFormatter=DateFormatter('%m-8d-%Y') #如：2-29-2015

alldaysFormatter=DateFormatter('%d') #如：12

#提取价格数据里面的日期 dates=[]

for i in range(len(quotes)):

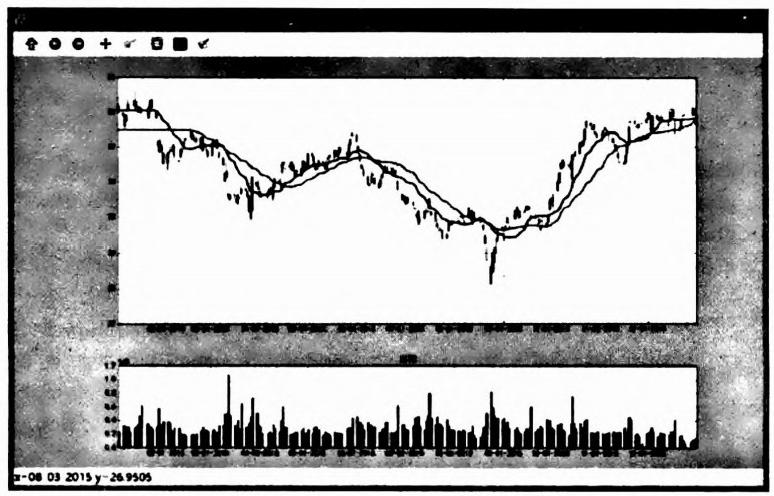
dates.append(quotes[i][0])

#提取价格数据里面的收盘价

|  |
| --- |
| prices=[]  for i in range(len(quotes)):  prices.append(quotes[i][4])  #计算15、30天移动平均  ma15=moving\_average(prices,15,type='simple')  ma30=moving\_average(prices,30,type='simple')  #提取价格数据里面的交易量 volumes=[]  for i in range(len(quotes)):  volumes.append(quotes[i][5])  print(volumes[:5])  #创建figure  #fig,(ax1,ax2)=plt.subplots(2,1)#,figsize=(8,18)) fig=plt.figure()  rect1=[0.14,0.35,0.77,0.6]#矩形区域的[左，下，宽，高](全部是0～1之间 的数，表示比例)  rect2=[0.14,0.05,0.77,0.2] ax1=plt.axes(rect1)  ax2=plt.axes(rect2)  fig.subplots\_adjust(bottom=0.2)  ax1.xaxis.set\_major\_locator(monthdays)  ax1.xaxis.set\_minor\_locator(alldays)  ax1.xaxis.set\_major\_formatter(monthdaysFormatter)  #ax.xaxis.set\_minor\_formatter(alldaysFormatter) #显示k线图  #plot\_day\_summary\_oclh(ax,quotes,ticksize=3)  candlestick\_ohlc(ax1,quotes,width=0.6,colorup='r',colordown='g') ax1.xaxis\_date()  ax1.autoscale\_view()  ax1.grid(True) #只为debug用 print(ma15)  print(ma30)  print('dates len',len(dates)) |

|  |
| --- |
| print('prices len',len(prices))  print('quotes len',len(quotes))  print('ma15 len',len(ma15))  print('ma30 len',len(ma30))  #在K线图上显示移动平均  linema15,=ax1.plot(dates,ma15,color='blue',1w=2,label=MA(20)')  linema30,=ax1.plot(dates,ma30,color='red,1w=2,label=MA(200)')  #在子图2上显示交易量  ax2.xaxis.set\_major\_locator(monthdays)  ax2.xaxis.set\_minor\_locator(alldays)  ax2.xaxis.set\_major\_formatter(monthdaysFormatter)  ax2.bar(dates,volumes,color="blue")#range(len(quotes))  ax2.xaxis\_date()#ax2.xaxis\_date()  ax2.autoscale\_view()  #plt.setp(plt.gca().get\_xticklabels(),rotation =45,horizontalalign- ment='right')  plt.title('002739')  plt.show() |

K 线图和移动平均见图14-23。



**图14-23** **K线图与移动平均**



**9.Latex 公式**

Matlplotlib 对 LaTex 提供一定的支持。在Matplotlib 里，可以使用LaTex 的命令来 编辑公式，只需要在字符串前面加一个 “r” 即可。下面的实例显示了正态分布的概率密 度函数的图形，并且在标题中，显示正态分布的概率密度函数。

|  |
| --- |
| import matplotlib.pyplot as plt import numpy as np  import matplotlib.mlab as mlab  import math  mu=0  variance=1  sigma =math.sqrt(variance)  x=np.linspace(-3,3,100)  plt.plot(x,mlab.normpdf(x,mu,sigma))  # 输 入latex 公式作为图形的title, 即在标题中显示正态分布的密度公式 plt.title(r'$f(x)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)~2}{2\ sigma^2}}$'①  plt.show() |

**10.** **读/写数据文件**

可以借助 Numpy 提供的函数读写文本文件，结合Matplotlib 进行图形绘制。示例代 码如下：

|  |
| --- |
| #从文本文件读数据，然后进行图形绘制 import numpy as np  import matplotlib.pyplot as plt  data=np.loadtxt('test.txt')#Use numpy to load test.txt into a 2-D array  plt.plot(data[:,0],data[:,1],'ro')# 第一列为x, 第二列为y, 绘制散点图 plt.xlabel('x')  plt.ylabel('y')  plt.xlim(0.0,10.)  plt.show()  #把数据写入文本文件 import numpy as np  x=np.arange(0.0,10.,1.)# 创建x 序列，范围0.0-10.0,间隔为1.0  y=x\*x# 创建y 序列，y的每个元素为对应的x 元素的平方 |



①关于如何编辑Latex 公式，请参考相关资料。



|  |
| --- |
| print 'x=',x  print 'y=',y  file=open('testwrite.txt',w')# 打开文本文件，准备写入，w表示write  for iin range(len(x)):  line=str(x[i])+'\t'+str(y[i])+'\n'# 创建一个文本行，str 函数把数值转  换成字符串，\t 表示制表符，\n 表示换行符 file.write(line)  file.close() |

**11. 导出图像文件/PDF 文件**

我们可以把生成的图形导出到图像文件或者 PDF 文件。示例代码如下，这个实例生 成一个柱状图，然后写入 temp.png 和 temp.pdf。

|  |
| --- |
| #导出文件  import matplotlib.pyplot as plt  fig=plt.figure()  plt.xlabel('sex')# 设 定x 轴的标签  plt.ylabel('number')# 设 定y 轴的标签  plt.xticks((0,1),('man','woman'))# 设 定x 轴各个刻度的标签  plt.bar(left=(0,1),height=(0.75,0.25),width=0.35,color="red")# 绘 制柱状图  #颜色为红色，柱状的高度分别为0.75和0.25 fig.savefig('temp.png',dpi=fig.dpi)  fig.savefig('temp.pdf',dpi=fig.dpi)  plt.show()# 显示柱状图 |

更多的实例，请读者参考 Matplotlib 的实例网址。① Matplotlib 还可以对地图进行绘 制，可以通过网址②访问相关实例。

**14.4.5 NetworkX 介绍与实例**

**1.NetworkX介绍**

NetworkX 是 用Python 语言开发的一个软件包，用于复杂网络分析，包括创建网络、 对网络进行操作、研究网络的结构及其动态演化机制 (Dynamics) 等 。NetworkX 是 一 款 免费的软件，遵循 BSDLicense 版权协议，用户可以对其进行修改和再发布 (Redis- tribute) 。NetworkX 项目建立于2002年。第一版由 Aric Hagberg,Dan Schult 和 Pieter

①<https://matplotlib.org/examples/index.html.>

②<https://pythonprogramming.net/geographical-plotting-basemap-tutorial/.>

Swart 在2002年和2003年设计和实现。 NetworkX 的第一个公开发行版于2005年发布。

NetworkX 内置了常用的图与复杂网络分析算法。使用NetworkX, 用户可以很方便 地存取网络文件格式、生成各种随机模型和经典模型的网络、分析网络结构、仿真建模、 对网络进行可视化等。NetworkX 可以应用于社交网络分析、生物网络分析、基础设施网 络分析等领域。

2. 创建无向图及其可视化

下面的实例创建了一个无向图，然后对其进行可视化，可视化结果如图14-24所示。 代码做了必要的注释。

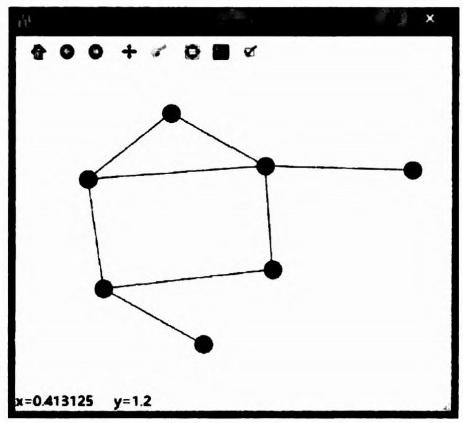


图14-24 一个无向图

|  |
| --- |
| import networkx as nx  import matplotlib.pyplot as plt  G=nx.Graph()# 创建图  G.add\_node("a")# 增加节点a  G.add\_nodes\_from(["b","c"])# 增加节点a、b  G.add\_edge(1,2)# 增加1 - 2边 edge=("d","e")  G.add\_edge(\*edge)# 增加d-e边 edge=("a","b")  G.add\_edge(\*edge)# 增加 a-b 边  G.add\_edges\_from([("a","c"),("c","d"),("a",1),(1,"d"),("a",2)])# 从列 表里，增加一系列的边  print("Nodes of graph:")  print(G.nodes())# 显示所有节点 |



print("Edges of graph:")

print(G.edges())井显示所有边

nx.draw(G,node\_size=512,with\_labels=True)# 绘制无向图 plt.savefig("simple\_path.png")# 存储为 PNG 格式文件

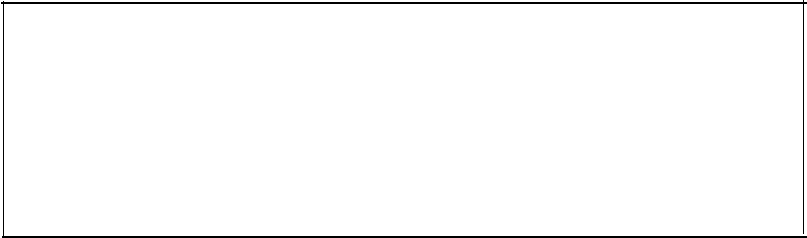
plt.show()# 显示无向图

**3.** **创建有向图**

建立有向图的方法和建立无向图是类似的，只需将 G=nx.Graph() 改 为 G=

nx.DiGraph() 即可。需要注意的是，这时候1-2边和2-1边是不同的边。

有向图和无向图可以互相转化，分别使用to\_undirected 方法和 to\_directed 方法即 可。示例代码如下：



G=nx.Graph()#or MultiGraph,etc G.add\_path([0,1])

H=G.to\_directed()

H.edges()# 输出[(0,1),(1,0)]

G2=H.to\_undirected()

G2.edges()# 输出[(0,1)]

**4.** **创建有权图**

不管是无向图还是有向图，它的边都可以带一个权重，称为有权图。有权图的边通过 add\_weighted\_edges\_from 函数进行增加，比如G.add\_weighted\_edges\_from([(0,

1,3.0),(1,2,7.5)]),表示增加了0- 1和1-2两条边，权重分别是3.0和7.5。

**5.网络统计指标、网络建模以及网络分析**

NetworkX 提供了网络统计指标计算、复杂网络建模方法(随机网络、小世界网络、 无标度网络等)、网络可视化方法以及常用的图论经典算法，包括宽度优先搜索 (Breath First Search,BFS)、深度优先搜索 (Deepth First Search,DFS)、最短路径、最小生成 树、最大流等算法。在线手册提供了详细的帮助文档和必要的实例。①

Network 提供度、度的分布、集聚系数、直径、平均最短路径、中心性等的计算方 法。下面的实例，首先生成一个 BA(Barabasi Albert) 无标度网络，计算度的直方图， 然后以图形的方式，显示度的分布情况，如图14-25所示。

|  |
| --- |
| import networkx as nx  G=nx.random\_graphs.barabasi\_albert\_graph(1000,3)# 生成一个n=1000,m= 3 的BA 无标度网络  print(G.degree())# 显示所有节点的度  print(nx.degree\_histogram(G)) 井显示图中所有节点的度分布序列(从1至最大 度的出现频次) |



①<http://networkx.readthedocs.io/en/latest/reference/algorithms.html.>

import matplotlib.pyplot as plt

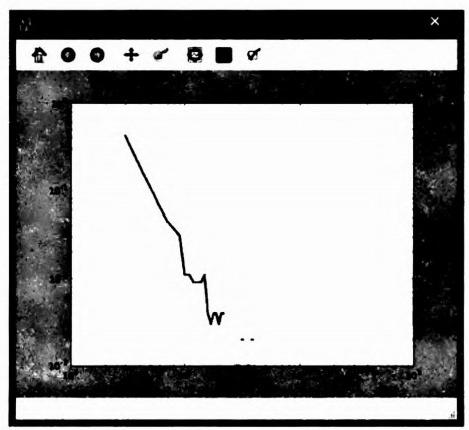
degree=nx.degree\_histogram(G)# 返回图中所有节点的度分布序列 x=range(len(degree))# 生 成x 轴序列，从1到最大度

y=[z/float(sum(degree))for z in degree]#将频数转换为频率

plt.loglog(x,y,color=“blue”,linewidth=2)# 在双对数(loglog) 坐标轴上，

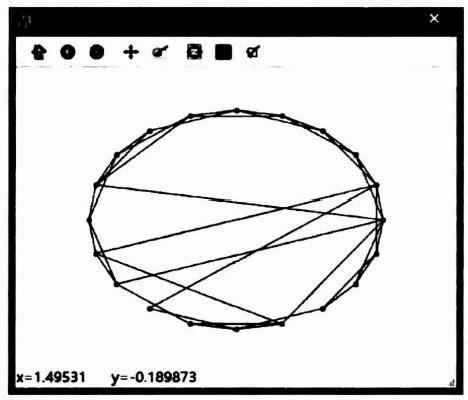
绘制度分布曲线

plt.show()# 显示绘制结果

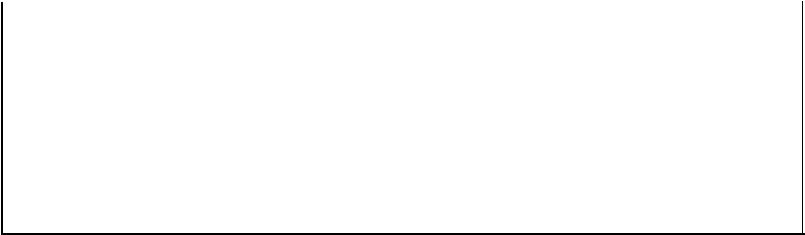


**图14-25** **度的分布**

NetworkX 提供了4种常见的网络建模方法，分别是规则图、 ER 随机图、WS 小世界 网络和 BA 无标度网络等。下面的实例，首先生成一个WS 小世界网络，然后显示这个网 络，如图14-26所示。



**图14-26** **小世界网络实例**



import networkx as nx

import matplotlib.pyplot as plt

WS=nx.random\_graphs.watts\_strogatz\_graph(20,4,0.3)# 生成包含20个节 点、每个节点4个近邻、随机化重连概率为0.3的小世界网络

pos=nx.circular\_layout(WS)# 采用了circular 布局方式

nx.draw(WS,pos,with\_labels=False,node\_size=30)# 绘制该小世界网络 plt.show()# 显示绘制结果

在可视化方面，NetworkX 可以对节点大小、颜色、标签、透明度等属性进行定制， 并且可以运用多种布局方式，对显示结果进行控制。

**6.** **最短路径实例**

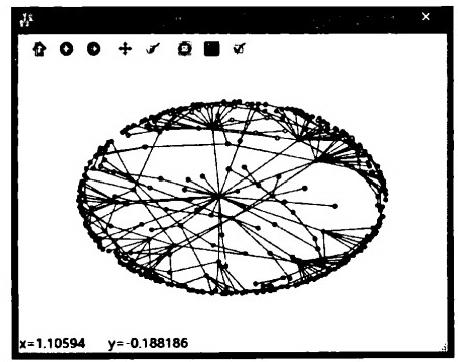
如下的代码针对上述无向图(本节的第一个实例),计算图中任意两个节点之间的最 短路径，并且把节点2和节点'd之间的最短路径给显示出来。

|  |
| --- |
| path=nx.all\_pairs\_shortest\_path(G)# 调用多源最短路径算法，计算图G所有 节点间的最短路径  print path[2]['d]#节点2、'd'之间的最短路径序列，即[2,1,'d] |

**7.** **社区发现**

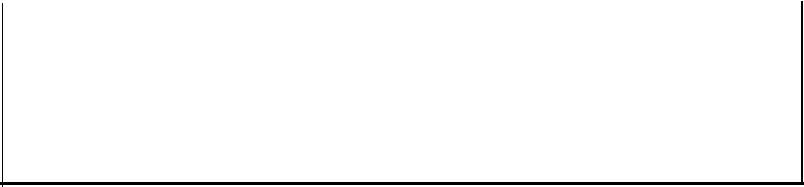
为了实现社区检测，需要先安装 Louvain 算法的 Python 库，下载地址为<https://> bitbucket.org/taynaud/python-louvain。

下面的实例，首先创建一个随机网络，然后用best\_partition 函数进行社区检测，最 后用不同的颜色进行渲染。社区检测结果见图14-27。



**图14-27** **杜区检测结果**

|  |
| --- |
| import networkx as nx import community  import matplotlib.pyplot as plt  G=nx.random\_graphs.powerlaw\_cluster\_graph(30,1,4) |

part=community.best\_partition(G)# 社区检测

values=[part.get(node)for node in G.nodes()]# 提取每个节点的社区 ID

nx draw\_spring(G,cmap=plt.get\_cmap('jet'),node\_color =values,node\_size=80, with\_labels=False)# 按照不同颜色绘制节点，指定colormap 为 jet

. 显示绘制结果

plt show()#

**14.4.6 NLTK 介绍与实例**

**1.NLTK** **介绍**

NLTK(Natural Language Toolkit) 是面向自然语言处理的开源软件库，用户可以基 于NLTK, 使用Python 语言编写自然语言处理程序。目前，NLTK 可以运行在 Win- dows,Mac OS X以 及Linux 等操作系统上。

NLTK 提供了一系列库函数，完成文本处理的分词 (Tokenization) 、 词干提取 (Stemming) 、 语法树分析以及词性标注 (Parsing &.Tagging)、命名实体识别 (NER) 、 短语识别 (Chunking, 即 Phrase Recognition) 以及文本分类等 (Classification) 。 同时， NLTK 提供了方便的编程接口，可以存取超过50个语料库 (Corpora) 和文本资源，比如 WordNet 等。

**2.** **句子分割与分词、中文分词**

使用NLTK 可以对文本进行句子分割以及对每个句子进行分词。下面给出了一个实 例及其运行结果。

|  |
| --- |
| import nltk  text ='the github is an open source software hosting site.We can open new project on it.'  sens=nltk.sent\_tokenize(text)# 切割文本为句子 print sens  words=[]  for sent in sens:  words.append(nltk.word\_tokenize(sent))# 把每个句子切割成单词，加入列表 print words  #这段代码的运行结果为  #['the github is an open source software hosting site.',we can open new pro- ject on it.'  #[['the','github','is','an','open','source','software','hosting','site','. '门，['we', 'can','open','new','project','on','it',.']] |

中文的分词和英文有所区别，英文的分词把各个单词区分开即可，但是中文的分词不 是把各个汉字区分开就可以，而是需要根据语义，把每个中文词汇切割开来。比如“大学 老师”应该切割成两个词“大学”和“老师”而不是4个汉字。在Python 下较好的中文 分词工具包括JieBa 中文分词系统①和 ICTCLAS 分词系统②等。

**3.** **词性标注**

进行词性标注，要用到 nltk.pos\_tag(tokens) 函数，其中 tokens 是句子分词后的

结果。词性标注是句子级的标注。下面给出了一个实例及其运行结果。

|  |
| --- |
| tags=[]  for tokens in words:  tags.append(nltk.pos\_tag(tokens)) print tags  #上述代码运行结果如下  #[[('the',DI'),('github','NN'),('is','VBZ'),('an',DT'),('open','JJ'),('source','NN'),  ('software',NN),('hosting',VBG'),('site','NN'),('.','.'),[('we','PRP'),('can','MD'), ('open',VB'),('hew,'JJ'),('project',NN),('on','TN),('it',PRP'),['.'∵:')] |

**4.** **命名实体识别**

通过调用nltk.ne\_chunk 函数，可以进行命名实体识别。需要把参数tags, 即句子词

性标注的结果，传递给 ne\_chunk 函数。示例代码如下：

|  |
| --- |
| import nltk  text =Deng Xiao-Ping was the leader of China in the year of 1980.' tokens =nltk.word\_tokenize(text)#分词  tags=nltk.pos\_tag(tokens)# 词性标注  print tags  ners=nltk.ne\_chunk(tags)# 利用词性标注结果，进行命名实体识别(NER) print'%s---%s'%(str(ners),str(ners.label()))  #这段代码的运行结果为 #(S  #(GPE Deng/NNP)#Deng Xiao-Ping应该被识别一个PERSON #Xiao-Ping/NN# 这里识别错了  #was/VBD  #the/DT  #leader/NN #of/IN  #(GPE China/NNP)#正确识别China 为GPE(General Purpose Entity) |



① <https://github.com/fxsjy/jieba.>

②<http://ictclas.nlpir.org/downloads.>

|  |
| --- |
| 井in/IN  井the/DT  #year/NN  #of/IN  #1980/CD  #./.)---S |

**5.** **分析语法树**

NLTK 工具包中有一个用于自然语言语法分析函数接口库，通过函数接口调用Stanford Parser 实现语法分析。在使用Standford Parser 之前，需要进行安装和配置，具体如下：

(1)首先下载 Stanford Parser的最新版本，下载地址为 <http://nlp.stanford.edu!> software/lex-parser.shtml#Download。

(2)接着把下载的 Stanford Parser 压缩包解压缩。然后从 stanford-parser-3.7.0-mod- els.jar 压缩文件中，在/edu/stanford/nlp/models/lexparser/ 路径下，找到englishPCFG.ser.gz (这是一个已经训练好的模型),并且解压缩到D:/jars (可以使用其他路径)。

(3)然后把 stanford-parser.jar,stanford-parser-3.7.0-javadoc.jar,stanford-parser-

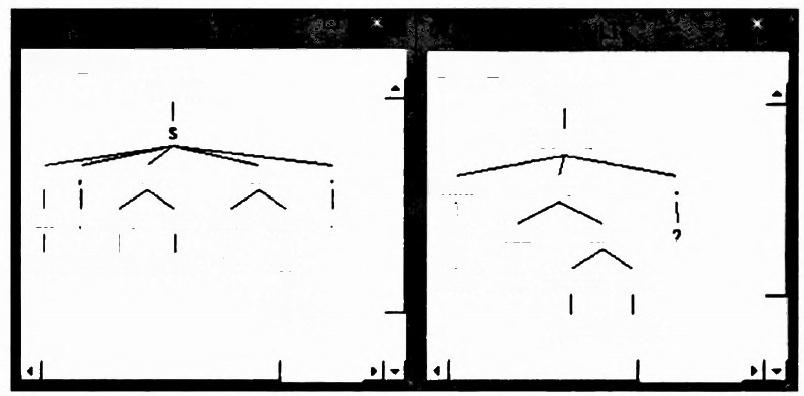
3.7.0-models.jar,stanford-parser-3.7.0-sources.jar 等文件拷贝到D:/jars 目录下。

安装和配置Stanford Parser 之后，就可以编程实现语法树分析，示例代码如下，代码 的关键段落和语句进行了注释。

|  |
| --- |
| import os  from nltk.parse import stanford  井添加STANEORD\_PARSER、STANFORD\_MODELS 环境变量 os.environ['STANFORD\_PARSER']=D:/jars/stanford-parser.jar'  os.environ['STANFORD\_MODELS']=D:/jars/stanford-parser-3.7.0-models.jar'  #添加JAVAHOME 环境变量  java\_path="C:/Program Files/Java/jdk1.8.0\_ 101/bin/java.exe"  os.environ['JAVAHOME']=java\_path  #分析语法树  parser=stanford.StanfordParser(model\_path="D:/jars/englishPCFG.ser.  gz")  sentences =parser.raw\_parse\_sents(("Hello,My name is Melroy."."What is yourname?"))  print sentences  #绘制语法树  for line in sentences:  for sentence in line:  sentence.draw() |



上述代码的执行结果如图14-28所示。

**File Zoom**

ROOT

MNTJ NP VP

UH PRPS NMN VBZ ADJP

I I I

HeWo My name is JJ

Metroy

Eile Zoom

ROOT

SBARQ

**SQ**

WHNP

w

What

VBZ NP

! PRPS MN

**your name**

**图14-28** **NLTK 利用Stanford Parser进行语法分析的结果**

注：语法树各个节点的标记及其具体含义，请参考 <https://www.ling.upenn.edu/courses/Fall_2003/ling001/> penn\_treebank\_pos.html。

**6.** **词干提取**

NLTK 支持词干提取，下面的示例代码，利用PorterStemmer 对'complications '进行 词干提取，得到的结果是'complic'。

|  |
| --- |
| from nltk.stem.porter import \*  PorterStemmer().stem\_word('complications') #输出'complic' |

**7. 文档相似度TF-IDF**

下面是一个计算文档间相似度的实例。①首先，定义 tokenize 函数，使用NLTK 的 tokenizer, 提取每篇文档的单词 (Token) 。 接着定义 stem\_tokens 函数，使用NLTK 的 Porter Stemmer 对文档的单词进行词干提取 (Stemming)。

由于the,and,of,that,in 等停止词 (Stop Word) 不包含太多的语义信息 (Uninform-

ative), 构 造TfidfVectorizer对象时，使用英文停止词过滤器，对文档的停止词进行过滤。

接下来，将使用TF-IDF 机制进行文档的索引和检索。 NLTK 软件包没有提供 TF- IDF 计算函数，因此，我们使用Scikit-learn 库。我们使用的文档数据集是20newsgroups, 包含来自20个新闻组 (Newsgroup) 的大约20000个文档。

在经过词干提取、停止词过滤的干净文本上 (Cleaned Up Text), 为每个文档计算 TF-IDF 表示，然后可以进行检索 (Searching) 、 计算文档相似度以及进行聚类和分类等。

TfidfVectorizer 对象的fit\_transform 方法，把文档加到模型中，即该文档参与IDF 的计算，并计算文档的TF-IDF 表示；transform方法，则不会把文档加入模型，仅仅对 输入的文档进行转换，计算该文档的TF-IDF 的表示。

本实例的主要目的是进行文档相似度计算和检索。在这里，使用向量的点积 (Dot

①<http://www.cs.duke.edu/courses/spring14/compsci290/assignments/lab02.html.>

Product. 用向量的夹角余弦计算文档相似度，向量夹角余弦的计算用到点积)来计算文 档之间的相似度。第一个文档(下标为0)的 TF-IDF 表示和整个文档集的每个文档的 TF-IDF 表示，进行点积运算，得出第一个文档和所有文档的相似度列表。

使用argsort 函数，并且在结果集里进行数组切片，找出最相似的5个文档 (Top 5 Similar Docuemtns) (最相关的文档，具有最高的余弦相似度 (Cosine Similarity), 在从 小到大的数组里，排在数组的末尾)。

用户可以输入新的文档(或者称为查询),通过调用TfidfVectorizer对象的transform 方法，就可以计算各个单词的 TF-IDF 值。然后把输入文档的 TF-IDF 向量和文档集的 TF-IDF 向量进行点积，计算余弦相似度，就可以检索出 (Search) 最相似的文档。具体 的代码如下，关键语句进行了注释。

|  |
| --- |
| import string import nltk  from nltk.stem.porter import PorterStemmer  from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.datasets import fetch\_20newsgroups  stemmer=PorterStemmer()#PorterStemmer,用于提取词干  #提取词干  def stem\_tokens(tokens,stemmer):  stemmed=[]  for item in tokens:  stemmed.append(stemmer.stem(item)) return stemmed  #分词  def tokenize(text):  tokens =nltk.word\_tokenize(text)  stems =stem\_tokens(tokens,stemmer)  return stems  #(1)提取20 newsgroups数据集，对文档集进行TF-IDF向量化  tfidfVectorizer=TfidfVectorizer()#no stemmer,no stop words  #tfidfVectorizer =TfidfVectorizer(tokenizer =tokenize,stop\_words = 'english')  twenty=fetch\_20newsgroups()  tfidf =tfidfVectorizer.fit\_transform(twenty.data)  print tfidf  print tfidf[0:1]  #(2)计算0号文档和所有文档的余弦相似度 |



|  |
| --- |
| from sklearn.metrics.pairwise import linear\_kernel  cosine\_similarities=linear\_kernel(tfidf[0:1],tfidf).flatten()#计算点积 print cosine\_similarities  #(3)寻找和0号文档最相似的4个文档的编号  related\_docs\_indices =cosine\_similarities.argsort()[:-5:-1]  #[:-5:-1]表示按照逆序从开始到-5,无间隔连续取-1、-2、-3、-4编号的 各个元素  print related\_docs\_indices  print twenty.data[0]#打印0号文档  #(4)用一个查询 query,查询文档集  query='this sentence has unseen text such as computer science but also Uni- versity of Maryland College Park'  response=tfidfVectorizer.transform([query])# 对查询进行 transform print response  feature\_names =tfidfVectorizer.get\_feature\_names()  for col in response.nonzero()[1]:  print feature\_names[col],'-',response[0,col]#打印查询的feature 和TF-IDF  井寻找和该查询(query) 最相似的4个文档  cosine\_similarities=linear\_kernel(response,tfidf).flatten()  related\_docs\_indices =cosine\_similarities.argsort()[:-5:-1]#寻找和该 查询(query)最相似的4个文档的编号，其中一个文档的id 为5496  print related\_docs\_indices print twenty.data[5496] |

**8.** **情感分析**

使用NLTK 可以对文本进行情感分析，确定其情感倾向。下面给出的实例①,对一系 列的句子进行情感倾向分析，然后打印出得分。为了运行这个实例，首先需要安装 twy- thon 软件包以及 vaderSentiment 软件包。在命令行输入(在Windows 平台上，运行Ana- conda2(64bit) 的 Anaconda Prompt 命令) pip install twython 命令以及 pip install vad- erSentiment 命令即可。此外，还需要在Jupyter Notebook里运行 nltk.download(), 并 且选择名称为 vader\_lexicon 的 model 进行安装。

from nltk.sentiment.vader import SentimentIntensityAnalyzer

#(1)句子列表

sentences=["VADER is smart,handsome,and funny.",#正面情感





①<https://github.com/cjhutto/vaderSentiment.>

|  |
| --- |
| "VADER is smart,handsome,and funny!",#用感叹号加重正面情感  ]  "VADER is very smart,handsome,and funny.",#用very(booster word)加重正面 情感  "VADER is VERY SMART,handsame,and EUNNY.",#用全大写的单词强调正面情感  "VADER is VERY SMART,handsome,and FUNNY!!!",#正常单词和全大写的单词的 组合  "VADER is VERY SMART,really handsome,and INCREDIBLY FUNNY!!!",#使用very (booster word)和感叹号，把情感倾向推向最大值  "The book was good.",#正面情感  "The book was kind of good.",#正面情感  "The plot was good,but the characters are uncompelling and the dialog is not great.",#正面情感，混合了负面情感  "A really bad,horrible book.",#使用very修饰的负面情感语句  "At least it isn't a horrible book.",#否定之否定 ":)and:D",#情感符号(emoticon)  "",#空语句  "Today sux",#负面情感的俚语(slang)  "Today sux!",#负面情感的俚语(slang),用感叹号强调  "Today SUX!",#负面情感的俚语(slang),用全大写的单词强调  "Today kinda sux!But I′ll get by,lol"#混合情感语句，带有俚语，带对照的连接 词"but"  #一个段落，若干句子  paragraph="It was one of the worst movies I've seen,despite good reviews.\ Unbelievably bad acting!!Poor direction.VERY poor production.\  The movie was bad.Very bad movie.VERY bad movie.VERY BAD movie.VERY BAD mov- ie!"  from nltk import tokenize  lines\_list=tokenize.sent\_tokenize(paragraph)#把段落切割成句子  sentences.extend(lines\_list)#把新句子添加到句子列表(sentences)里  #句子列表  tricky\_sentences=[  "Most automated sentiment analysis tools are shit.",#负面情感 "VADER sentiment analysis is the shit.",#负面情感  "Sentiment analysis has never been good.",#负面情感  "Sentiment analysis with VADER has never been this good.",#正面情感 "Warren Beatty has never been so entertaining.",#正面情感 |

|  |
| --- |
| "I won't say that the movie is astounding and I wouldn't claim that\ the movie is too banal either.",#中性情感  ]  "Ilike to hate Michael Bay filns,but I couldn't fault this one",#正面情感 "It's one thing to watch an Uwe Boll film,but another thing entirely\  to pay for it"。#负面情感  "The movie was too good",#正面情感  "This movie was actually neither that funny,nor super witty.",#负面情感  "This movie doesn't care about cleverness,wit or any other kind of intelligent humor.",#负面情感  "Those who find ugly meanings in beautiful things are corrupt without \ being charming.",#负面情感  "There are slow and repetitive parts,BUT it has just enough spice to\ keep it interesting.”,#正面情感  "The script is not fantastic,but the acting is decent and the cinematography\ is EXCELIENT!",#正面情感  "Roger Dodger is one of the most compelling variations on this theme.",#正 面情感  "Roger Dodger is one of the least compelling variations on this theme.",#负 面情感  "Roger Dodger is at least compelling as a variation on the theme.",#正面情感 "they fall in love with the product",#正面情感  "but then it breaks",#负面情感  "usually around the time the 90 day warranty expires",#负面情感  "the twin towers collapsed today",#负面情感  "However,Mr.Carter solemnly argues,his client carried out the kidnapping\ under orders and in the''least offensive way possible.'"#负面情感  sentences.extend(tricky\_sentences)#把新句子添加到句子列表(sentences)里  #(2)创建分类器  sid=SentimentIntensityAnalyzer()  #(3)对句子进行情感分析  for sentence in sentences:  print(sentence)  ss=sid.polarity\_scores(sentence)  for k in sorted(ss):  print k,ss[k]#打印正面/中性/负面情感倾向得分，以及复合得分 print |

在上述实例中，使用已经训练好的分类器 (SentimentIntensityAnalyzer) 对句子的情

感倾向进行分类。

对于特定的应用场合，可以自行训练分类器。需要用户给出正例 (Positive) 和反例 (Negative) 训练数据。在这些数据集中，抽取单词以及Bi-Gram (相邻两个单词构成一个Bi- Gram), 然后对模型进行训练。可以使用的模型有决策树 (Decision Tree) 分类器，朴素贝叶 斯 (Naive Bayes) 分类器、最大熵 (Maximum Entropy) 分类器，以及支持向量机 (Support Vector Machine,SVM) 分类器等。文献①给出了另一个实例，读者可以自行参考。

**9.** **文本分类实例——垃圾邮件分类**

文本分类是文本处理的主要任务之一。文本分类有很多应用 .比如垃圾邮件分类、把 客户请求分发给某个客户代表、风格分类 (Genre Classification)、情感分析等。

下面我们给出一个简单的垃圾邮件分类器 (Spam Filter)。② 这是一个二值分类问题， 即对输入的文档，系统需要给出是垃圾邮件(分类标签为Spam), 或者不是垃圾邮件(分 类标签为 Ham) 的分类结果。

这个实例使用的数据集来自Enron-Spam③ 和 SpamAssassin Public Corpus④。用户可 以从其网址下载，并解压缩到某个目录底下。

垃圾邮件分类的主要步骤如下：

(1)装载数据集，转换成可以使用的格式。对于垃圾邮件分类器来讲，它希望得到这 样一个2维的数组，第一列包含邮件的文本 (Email Body),第二列包含该邮件的分类标 签 (Spam 或者Ham)。

为此需要编写一个函数，把某个目录底下的(包括子目录底下的)所有邮件都读出 来，剔除邮件的头部 (Header), 只保留邮件体 (Body)。 由于邮件头和邮件体之间有一 个空行，只需忽略空行之前的内容，就能够提取每个邮件的邮件体。

**在代码中，使用的编码方式为** **latin-1。由于文档集里的某些邮件不是Unicode 编码格** **式的，使用latin-1 编码格式进行解码，能够尽最大可能对邮件进行正确解码。**

build\_data\_frame 函数调用read\_files 函数，把某个目录下及其子目录下的所有邮 件读取出来，构造一个 DataFrame. 第一列是每个邮件的邮件体，第二列是该邮件的分类 标签。每个邮件的行标签是 email的文件名。 Pandas 支持通过 append 方法，通过不断添 加新的数据，创建一个更大的DataFrame。

(2)从文本里提取特征值 (Feature), 以便分类器能够进行学习。为了对文档进行分类， 需要首先从文档提取特征值。对于文本分类问题，我们可以用单词的计数 (Word Count) 作 为特征。创建一个CountVectorizer 实例，并且调用其fit\_transform方法，该方法从文档集 里学习到 一 个词汇表 (Vocabulary of the Corpus),以及提取各个单词的计数。

(3)训练一个分类器。在本实例中，使用Naive Bayes 分类器实现邮件分类。我们初 始化了一个 MultinomialNB 实例，并且使用fit 函数，根据输入的特征向量和目标类别向 量，进行训练。

接着，对训练好的模型进行测试 . 即使用 一 些新数据进行预测，看看它能否进行正确 分类 。



①[http://blog.chapagain com np/machine-learning sentiment-analysis text-classification-using-pythor-nltk/.](http://blog.chapagaincomnp/machine-learningsentiment-analysistext-classification-using-pythor-nltk/.)

②<http://zacstewart.com/2015/04/28/document-classification-with-scikit-learn.html.>

③<http://www2.aueb.gr/users/ion/data/enron-spam/.>

④<http://spamassassin.apache.org/old/publiccorpus/.>

本实例中，使用Scikit-learn 软件包提供的pipeline机制，把数据预处理、特征抽取、预 测器等组件和步骤整合成一个对象。对这个对象，首先进行训练，然后利用它进行预测。

(4)通过交叉验证 (Cross-Validation) 评估分类器的准确度。本实例中使用了pipe- line, 把上述分离的操作步骤组合起来。可以使用pipeline, 利用数据集对模型进行训练， 然后利用模型进行预测。

交叉验证是对模型进行评估的主要方法.常用的交叉验证是 K-Fold 交叉验证。使用 Scikit-learn函数，可以很方便地实现 K-Fold 交叉验证。

对于模型的准确度常用的指标是F1-Score (关于该指标的含义，请参考第7章“文本 分析”和第5章“数据的深度分析”)。此外，还可以使用混淆矩阵评价一个模型的优劣。 混淆矩阵可以查看分类器对每个类别的分类效果。比如下面的混合矩阵，表示对于55326 个样本，总共有178个假 Spam (本来是Ham, 被判断成 Spam) 和3473个假Ham 邮件 (本来是Spam. 被判断成Ham)。

|  |  |  |
| --- | --- | --- |
|  | 预测为Ham | 预测为Spam |
| 实际为Ham | 21660 | 178 |
| 实际为Spam | 3473 | 30015 |

本实例的具体代码如下①:

|  |
| --- |
| #open a new jupyter notebook,copy the code and run  import os  import io  import numpy  from pandas import DataFrame  from sklearn.feature\_extraction.text import CountVectorizer from sklearn.naive\_bayes import MultinomialNB  from sklearn.pipeline import Pipeline  from sklearn.cross\_val idation import KFold  from sklearn.metrics import confusion\_matrix,f1\_score  **NEWLINE='\n'**  HAM='ham'  SPAM='spam'  #文件所在目录  SOURCES=[  ('data/spam',SPAM),  ('data/easy\_ham,HAM),  ('data/hard\_ham',HAM), |



①注意，该实例使用了Scikit-learn 库，由于这个实例是一个文本分类实例，所以我们把它和 NLTK 的文本处理 实例放在一块儿。



('data/beck-s',HAM),

('data/farmer-d',HAM),

('data/kaminski-v',HAM),

('data/kitchen-l',HAM),

('data/lokay-m',HAM),

('data/williams-w3',HAM),

('data/BG',SPAM),

('data/GP',SPAM),

('data/SH,SPAM)

]

SKIP\_FILES={'cmds'}

#读取文件

def read\_files(path):

for root,dir\_names,file\_names in os.walk(path):

for path in dir\_names:

read\_files(os.path.join(root,path)) for file\_name in file\_names:

if file\_name not in SKIP\_FILES:

file\_path=os.path.join(root,file\_name) if os.path.isfile(file\_path):

past\_header,lines=False,[]

f=io.open(file\_path,encoding="latin-1") for line in f:

if past\_header:

lines.append(line)

elif line ==NEWLINE:

past\_header=True f.close()

content=NEWLINE.join(lines) yield file\_path,content

#从文本文件，创建DataFrame

def build\_data\_frame(path,classification):

rows=[]

index=[]

for file\_name,text in read\_files(path):

rows.append({'text':text,'class':classification})

index.append(file\_name)

data\_frame=DataFrame(rows,index=index)

return data\_frame



|  |
| --- |
| #(1)创建空的DataFrame  data =DataFrame({'text':[],'class':[]})  #调用build\_data\_frame,往DataFrame 添加数据 for path,classification in SOURCES:  data =data.append(build\_data\_frame(path,classification))  data =data.reindex(numpy.random.permutation(data.index))  #(2)建立数据处理流水线，包含CountVectorizer 和 MultinomialNB pipeline =Pipeline([  ('count\_vectorizer',CountVectorizer(ngram\_range =(1,2))), ('classifier',MultinomialNB())  ])  #(3)创建K-Fold交叉验证  k\_fold=KFold(n=len(data),n\_folds =6) scores=[]  confusion=numpy.array([[0,0],[0,0]])  for train\_indices,test\_indices in k\_fold:  train\_text =data.iloc[train\_indices]['**text'** ].values  train\_y=data.iloc[train\_indices]['class'].values.astype(str) test\_text =data.iloc[test\_indices]['text'].values  test\_y=data.iloc[test\_indices]['class'].values.astype(str)  pipeline.fit(train\_text,train\_y)  predictions =pipeline.predict(test\_text)  confusion+=confusion\_matrix(test\_y,predictions)  score =f1\_score(test\_y,predictions,pos\_label=SPAM)  scores.append(score)  #(4)打印分类效果  print("Total emails classified:',len(data))  print('Score:',sum(scores)/len(scores))  print('Confusion matrix:')  print(confusion) |

**10.** **提高垃圾邮件分类准确度的一些考虑①**

为了提高垃圾邮件分类的准确度可以采用如下措施：(1)从邮件里提取更多的特征 值。比如，不仅使用单词计数 (Word Count), 而且可以使用N-Gram 计数。单词计数是

① 为了深人了解如何使用Scikit-learn实现文本分类，读者还可以参考 Scikit-learn 教程 (tutorial),<http://sci>- kit-learn.org/stable/tutorial/text\_analytics/working\_with\_text\_data.html。

一种 “Bag of Words” 计数，它把单词放到一个“袋子”里，然后对每个单词进行计数， 并没有考虑到单词之间的顺序关系。N-Gram 则把挨在一块儿的n 个单词构成一系列的n- Gram, 考虑到了单词的顺序关系，常用的n-Gram 有2-Gram 和3-Gram 。(2) 尝试其他不 同类型的特征值。单词计数对于长文档来说有不利的一面。比如， 一个包含6个单词的垃 圾邮件和一篇感人的5页的长邮件，但是包含了6个疑似垃圾邮件的单词( “Spammy” Word), 它们有可能得到相同的“垃圾邮件”概率得分(“Spamminess”Probability), 显 然，这是不合理的。为了解决这个问题，我们可以使用词项频率 TF(Term Frequency), 代替 Word Count 。TF 考虑到了文档长度的影响，它表达的意思是一个文档里某个词项所 占的比例。此外，某些单词在整个文档集中是一个普遍的单词，其权重应该降低，可以使 用 IDF(Inverse Document Frequency) 对 TF 进行调整(请参考第7章“文本分析”)。

(3)优化Naive Bayes 分类器的参数。我们可以使用诸如Grid Search 的策略进行分类器参 数的优化。(4)尝试使用其他分类器。比如，我们可以使用Bernouli Naive Bayes 分类器。 该分类器使用一个阈值把数值型特征转化成布尔值。 Bernouli Naive Bayes 分类器的表现 和上述分类器略有不同，它使得更多的Spam 成为漏网之鱼。(5)为了提高垃圾邮件分类 的性能，可以考虑使用集成学习(Ensen ble Learning) 的技术，把多个分类器整合起来， 提高分类效果。

**14.5** **思考题**

(1)Python 语言的特点。

(2)Python 的 tuple,list,dict 数据结构及其使用。

(3)Python 的异常处理。

(4)Python 的面向对象编程。

(5)Pandas 库及其实例。

(6)Scikit-learn 库及其实例。

(7)Theano,Keras 库及其实例。

(8)Matplotlib 库及其实例。

(9)NetworkX 库及其实例。

(10)NLTK 库及其实例。