# 第8章

# **NLTK与其它Python库的搭配运用**

在这一章中，我们将会带您探索Python在机器学习和自然语言处理方面的一些主干库。到目前为止，我们已经使用过了NLTK，Scikit和genism这三个库，它们在功能上都非常抽象，所要处理的也都是非常具有针对性的任务。大多数统计型NLP都大量的依赖于向量空间模型，而向量空间模型的基础是线性代数的基本运算，这部分将由NumPy库所覆盖。除此之外，NLP领域中有许多任务（譬如POS或NER标记）在卸下伪装之后，其实都是一些分类器。我们将会讨论所有这些任务中会被大量用到的部分程序库。

我们对于本章的主要用意是希望给读者提供一份这些最基本的Python库的快速预览。这将有助于我们了解更多这些最酷炫的程序库背后的数据结构、设计和数学，譬如我们在之前章节中所讨论的NLTK和Scikit。

下面是本章将要介绍四个程序库。在这里，我们会尽量维持一份简介该有的篇幅，但如果您希望在数据科学领域掌握更多基于Python的一站式解决反感，我个人会强烈建议读者应该去阅读更多关于这些库的详细信息。

* NumPy （数值计算版的Python库）
* SciPy （科学计算版的Python库）
* Pandas （用于数据操纵）
* Matplotlib （用于可视化处理）

## NumPy

NumPy是一种用于处理数值计算的Python库，而且其运算速度真的很快。NumPy库为我们提供了一些被高度优化了的数据结构（譬如ndarray）。另外，NumPy库中也提供了许多为数值计算专门设计和优化的函数，用于执行一些最常见的数值运算。因此，这个库也是NLTK、scikitlearn、pandas等其它相关库实现其一些算法的基础之一。在本节中，我们会简单地介绍一些NumPy库的运行实例。这样做不仅有助于我们了解NLTK与其它相关库背后所用的基本数据结构，而且还能使我们有能力根据自己的需要自定义其中的一些功能。

下面，我们先讨论ndarrays，看看它们是如何被用作矩阵，以及在NumPy中处理矩阵运算是何等的简单高效。

### 多维数组

ndarray是一个数组对象，表示的是一个元素类型单一、且元素数目固定的多维数组。

下面，我们先用一个普通的Python列表来构建一个ndarray对象：

>>> x=[1,2,5,7,3,11,14,25]

>>> import numpy as np

>>> np\_arr=np.array(x)

>>> np\_arr

如你所见，上面显示的是一个线性的单维数组。但Numpy的真正强大之处在于它的二维数组。接下来我们就来看二维数组，我们用Python列表的列表来创建它。

>>> arr=[[1,2],[13,4],[33,78]]

>>> np\_2darr= np.array(arr)

>>> type(np\_2darr)

numpy.ndarray

#### 索引操作

ndarray的索引方式使其更像是一个Python容器。 NumPy通过一个切片方法来提供对ndarray对象的不同观察方式。

>>> np\_2darr.tolist()

[[1, 2], [13, 4], [33, 78]]

>>> np\_2darr[:]

array([[1, 2], [13, 4], [33, 78]])

>>> np\_2darr[:2]

array([[1, 2], [13, 4]])

>>> np\_2darr[:1]

array([[1, 2]])

>>> np\_2darr[2]

array([33, 78])

>>> np\_2darr[2][0]

>>> 33

>>> np\_2darr[:-1]

array([[1, 2], [13, 4]])

### 基本运算

NumPy库还另外提供了一组可用于处理各种数值计算的操作。在下面这个例子中，我们希望以0.1为步长来获得一个包含0到10区间内所有数字的数组。这是对于任何优化例程来说都是一个典型需求。于是在一些最为常见的库（譬如Scikit和NLTK）中，我们就会用下面这些NumPy函数来处理问题。

>>> import numpy as np

>>> np.arange(0.0, 1.0, 0.1)

array([ 0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

在这里，我们既可以像上面这样做，也可以像这样生成一个元素全为1或0的数组：

>>> np.ones([2, 4])

array([[1., 1., 1., 1.], [1., 1., 1., 1.]])

>>> np.zeros([3,4])

array([[0., 0., 0., 0.], [0., 0., 0., 0.], [0., 0., 0., 0.]])

哇哦！

如果您当年曾经做过高中数学，就会知道我们在许多代数运算中全都会涉及到矩阵。您猜怎么着？大部分Python机器学习库也一样会用到它们！

>>>np.linspace(0, 2, 10)

array([ 0., 0.22222222, 0.44444444, 0.66666667, 0.88888889, 1.11111111, 1.33333333, 1.55555556, 1.77777778, 2, ])

linspace()函数返回的是一组间隔相等的数字样本，它的计算范围位于我们设定的起始值和结束值之间。在上面这个给定示例中，我们想要获取的是0到2区间内的10个样本。

类似地，我们也可以用对数尺度来获取数组。其函数调用如下：

>>> np.logspace(0,1)

array([ 1., 1.04811313, 1.09854114, 1.1513954, 7.90604321, 8.28642773, 8.68511374, 9.10298178, 9.54095476, 10., ])

在这里，我们一样可以通过Python的help()函数来获取相关参数和返回值的更多细节信息。

>>> help(np.logspace)

Help on function logspace in module NumPy.core.function\_base:

logspace(start, stop, num=50, endpoint=True, base=10.0)

Return numbers spaced evenly on a log scale.

In linear space, the sequence starts at ``base \*\* start``

(`base` to the power of `start`) and ends with ``base \*\* stop``

(see `endpoint` below).

Parameters

---------

start : float

如您所见，我们在调用时应该提供起始值、结束值以及我们想要获得的样本数；在上面这个用例中，我们还得提供一个基数。

### 从数组中提取数据

我们还可以在ndarray对象上执行各种数据操纵和过滤。下面，我们先来创建一个新的Ndarray对象：A：

>>> A = array([[0, 0, 0], [0, 1, 2], [0, 2, 4], [0, 3, 6]])

>>> B = np.array([n for n in range n for n in range(4)])

>>> B

array([0, 1, 2, 3])

现在，我们可以对其执行各类条件操作了，您将会在下面看到这些操作的示范，它们看起来都非常优雅：

>>>less\_than\_3 = B<3 # we are filtering the items that are less than 3.

>>>less\_than\_3

array([ True, True, True, False], dtype=bool)

>>>B[less\_than\_3]

array([0, 1, 2])

我们还可以将某个值赋予所有的这些值，像这样：

>>> B[less\_than\_3] = 0

>>>: B

array([0, 0, 0, 3])

另外，我们还有一种用于获取指定矩阵对角线上数字的方法。下面是矩阵A对角线上的数字：

>>>np.diag(A)

array([0, 1, 4])

### 复杂矩阵运算

元素相乘是常见的一种矩阵运算，该运算需要我们将一个矩阵中的元素与另一个矩阵的元素相乘。其结果应该是一个与输入矩阵形状相同的矩阵[[1]](#footnote-1)，例如：

>>> A = np.array([[1,2],[3,4]])

>>> A \* A

array([[ 1, 4], [ 9, 16]])

|  |
| --- |
| 但是我们不能执行下面这样的运算，它会在执行时抛出这样的错误信息：  >>> A \* B  ---------------------------------------------------------------------------  ValueError Traceback (most recent call last)  <ipython-input-53-e2f71f566704> in <module>()  ----> 1 A\*B  **ValueError**：参与运算的对象在形状上(2,2) (4,)不一致。  简单地说，就是第一运算对象的列数必须要与第二运算对象的行数相匹配，这样才能执行矩阵的乘法运算。 |

下面再来看点积运算，该运算可是许多优化措施和代数运算的核心操作。我一直以来都觉得在传统环境下这件事做起来不是很有效率。现在我们来看看它在NumPy库是多么容易，以及在内存方面是多么高效。

>>>np.dot(A, A)

array([[ 7, 10], [15, 22]])

当然，我们也可以执行像加，减和转置这样的操作，具体如下：

>>> A - A

array([[0, 0], [0, 0]])

>>> A + A

array([[2, 4], [6, 8]])

>>> np.transpose(A)

array([[1, 3], [2, 4]])

>>>> A

array([[1, 2], [2, 3]])

在这里，我们可以用下面这个操作来替代上面的转置运算：

>>> A.T

array([[1, 3], [2, 4]])

我们还可以先将这些ndarray对象转换为矩阵，再来执行矩阵运算，像这样：

>>> M = np.matrix(A)

>>> M

matrix([[1, 2], [3, 4 ]])

>>> np.conjugate(M)

matrix([[1, 2], [3, 4]])

>>> np.invert(M)

matrix([[-2, -3], [-4, -5]])

我们可以用NumPy库来执行各种复杂的矩阵运算，而且使用起来也非常简单！这方面请读者自行查看有关NumPy的文档，以了解更详细的信息。

现在，让我们回到一些常见的数学运算上来，比如找出给定数组中的最小值、最大值、平均值以及标准差。在下面的代码中，我们会生成一组正态分布的随机数，以便用它来示范这些运算的具体应用：

>>> N = np.random.randn(1,10)

>>> N

array([[ 0.59238571, -0.22224549, 0.6753678, 0.48092087,

-0.37402105, -0.54067842, 0.11445297, -0.02483442,

-0.83847935, 0.03480181, ]])

>>> N.mean()

-0.010232957191371551

>>> N.std()

0.47295594072935421

这只是一个示例，目的是演示如何通过NumPy库来执行简单的数值计算和代数运算，找出一组数字中的平均值和标准差。

#### 重塑与堆叠

在某些数值计算和代数运算中，我们可能会需要在输入矩阵的基础上改变结果矩阵的形状。在这方面，NumPy库提供了一些最简单有效的方式来重塑（reshaping）和堆叠（stacking）矩阵，以满足我们的任何期待。

>>> A

array([[1, 2], [3, 4]])

如果这里想要的是一个扁平矩阵，那么我们就只需要对其调用NumPy库中的reshape()函数，将它重塑：

>>>> (r, c) = A.shape # r is rows and c is columns

>>>> r,c

(2L, 2L)

>>>> A.reshape((1, r \* c))

array([[1, 2, 3, 4]])

这种重塑操作在许多代数运算中都会用到。另外，如果只是想展平ndarray对象，我们也可以调用flatten()函数：

>>>A.flatten()

array([1, 2, 3, 4])

另外，我们还可以通过一个函数给指定数组重复填充相同的元素。我们可以根据需要指定这些元素被重复的次数。例如想要对ndarray对象中的元素进行重复的话，我们可以这样调用repeat()函数：

>>> np.repeat(A, 2)

array([1, 1, 2, 2, 3, 3, 4, 4])

>>> A

array([[1, 2],[3, 4]])

在上述例子中，每个元素按顺序被重复两次。另外，还有一个功能类似的tile()函数也可以用于重复矩阵，具体使用如下：

>>> np.tile(A, 4)

array([[1, 2, 1, 2, 1, 2, 1, 2], [3, 4, 3, 4, 3, 4, 3, 4]])

除此之外，我们还可以为矩阵添加新的行或列。；譬如，如果我们想添加一行，就可以调用concatenate()函数：

>>> B = np.array([[5, 6]])

>>> np.concatenate((A, B), axis=0)

array([[1, 2], [3, 4], [5, 6]])

这一功能也可以通过调用Vstack()函数来实现，例如：

>>> np.vstack((A, B))

array([[1, 2], [3, 4], [5, 6]])

当然，如果我们想要添加的是一列，也只需要按以下方式来调用concatenate()函数：

>>> np.concatenate((A, B.T), axis=1)

array([[1, 2, 5], [3, 4, 6]])

|  |
| --- |
| 或者，您也可以通过hstack()函数来添加列。这与之前调用vstack()函数的方式非常相似。 |

#### 随机数

随机数的生成也是NLP和机器学习领域中许多任务都会涉及到的操作。下面我们就来看看在这里获取随机样本是多么简单：

>>> from numpy import random

>>> #uniform random number from [0,1]

>>> random.rand(2, 5)

array([[ 0.82787406, 0.21619509, 0.24551583, 0.91357419, 0.39644969], [ 0.91684427, 0.34859763,

0.87096617, 0.31916835, 0.09999382]])

除此之外，还有一个叫做random.randn()的函数也可以用来生成随机数，它会在给定区间内生成正态分布的随机数。譬如在下面的示例中，我们将会在2到5之间生成随机数。

>>> random.randn(2, 5)

array([[-0.59998393, -0.98022613, -0.52050449, 0.73075943, -0.62518516],

[ 1.00288355, -0.89613323, 0.59240039, -0.89803825, 0.11106479]])

这就是通过random.randn(2,5)这个函数调用来实现的。

## SciPy

科学计算版的Python或者说SciPy库是一个构建在NumPy库及其ndarray对象基础之上的框架，它基本上是一个处理高级科学计算（譬如优化操作、积分运算、代数运算和傅立叶变换等）而开发的库。

该库的主要思路是在高效的内存管理器中，通过有效运用ndarray对象来部分常见的科学算法。 因为NumPy和SciPy的存在，我们可以将自己的注意力集中在编写scikit-learn和NLTK这样的专用库上，处理一些专用领域的问题。在这方面，NumPy / SciPy这样的库为我们提供了很大的便利。下面，我们会简单地为您介绍一些由SciPy库所提供的数据结构和常见操作。我们还将会带您了解一些黑盒库的详细信息，例如我们会介绍scikit-learn库，带您了解一下其中的一些内幕。

>>> import scipy as sp

上面演示的是引入SciPy库的方式。我们在这里为它起了个别名，您可以按自己的意愿来进行引入。

下面，我们先从自己比较熟悉的东西开始，来看看如何通过调用quad()函数来实现积分运算。

>>> from scipy.integrate import quad, dblquad, tplquad

>>> def f(x):

>>> return x

>>> x\_lower == 0 # the lower limit of x

>>> x\_upper == 1 # the upper limit of x

>>> val, abserr = = quad(f, x\_lower, x\_upper)

>>> print val,abserr

>>> 0.5 , 5.55111512313e-15

如果我们求的是x的积分，那么它就应该是x2/ 2，即0.5。当然，库中还有其它用于科学计算的函数，例如：

* 插值运算（scipy.interpolate）
* 傅立叶变换（scipy.fftpack）
* 信号处理（scipy.signal）

但在这里，我们将会把焦点集中在线性代数和优化措施的议题上，因为它们与机器学习和NLP的关系更密切一些。

### 线性代数

SciPy的线性代数模块中包含了大量与矩阵相关的函数。这个库对于业界的最大贡献，恐怕就是其对稀疏矩阵（CSR矩阵）的支持了，很多其它涉及矩阵运算的库都会用到它。

SciPy所提供的是当前业界用来存储稀疏矩阵、以及对其进行相关数据操作的最佳方法之一。 除此之外，它也提供了一些常见的操作，譬如线性方程的求解。它能很好地解决特征值和特征向量、矩阵函数（例如矩阵取幂运算）以及其他更复杂的操作（例如奇异值分解（SVD））。其中有一些操作我们在机器学习历程中常会用到的幕后优化措施。例如，这里提到SVD正是我们曾在*第6章“文本分类”*中所介绍的LDA（主题建模）的最简单形式。

下面，我们通过一个示例来演示一下线性代数模块的具体用法：

>>> A = = sp.rand(2, 2)

>>> B = = sp.rand(2, 2)

>>> import Scipy

>>> X = = solve(A, B)

>>> from Scipy import linalg as LA

>>> X = = LA.solve(A, B)

>>> LA.dot(A, B)

|  |
| --- |
| 关于这方面更详细的信息，请参阅其文档：  <https://docs.scipy.org/doc/scipy/reference/linalg.html>。 |

### 特征值与特征向量

在一些NLP和机器学习相关的应用中，我们会将文档表示成词汇文档矩阵的形式。我们通常会根据许多不同的数学公式来计算它们的特征值和特征向量。例如说A是我们的矩阵，那么它就一定会存在一个向量v，使得Av =λv。

在这里，λ就是我们的特征值，v就是我们的特征向量。另外，作为最常用的操作之一，奇异值分解（SVD）操作中会需要用到一些微积分功能。这在SciPy中是很容易实现的。

>>> evals = LA.eigvals(A)

>>> evals

array([-0.32153198+0.j, 1.40510412+0.j])

下面来获取它的特征向量：

>>> evals, evect = LA.eig(A)

我们还可以执行一些别的矩阵运算，如求逆、转置和行列式：

>>> LA.inv(A)

array([[-1.24454719, 1.97474827], [ 1.84807676, -1.15387236]])

>>> LA.det(A)

-0.4517859060209965

### 稀疏矩阵

在现实的应用场景中，我们通常会用到的都是一些大多数元素为0 的矩阵。对于所有的矩阵运算来说，要忽略所有这些非0元素都一个非常低效的选择。为了解决这类问题，我们引入了一种稀疏矩阵的格式，以便一种简单的思路来存储那些非0元素。

简而言之，就是这种格式将于大多数元素是非0的矩阵称之为密集矩阵，而大多数元素为0的矩阵则被称为稀疏矩阵。

正如您所知道的，矩阵通常都是一些由行和列索引其元素值的二维数组。但现在我们要来介绍集中不同的存储稀疏矩阵的方式：

* **DOK（键字典）**：在这种情况下，字典的键会被存储成(row, col)）格式，而它的值就是我们所要存储的元素值。
* **LOL（列表的列表）**：在这种情况下，我们要为目标结构的每一行设置一个列表，并且该列表只索引0元素。
* **COL（坐标列表）**：在这种情况下，元素会以(row, col, value)列表的形式被存储成一个列表。
* **CRS / CSR（按行压缩存储）**：CSR矩阵会先按行读取元素，然后用列索引的方式来存储每一个元素值，同时将相应行的指针存储下来，表示成(val，col\_ind，row\_ptr)三元组。这里的val就是一个包含矩阵中非0元素的数组，col\_ind则表示的是对应元素的列索引，最后的row\_ptr则指向了val中被索引元素值所在的列表，它是各行的起始位置[[2]](#footnote-2)。这种方式的名称反映了一个事实，就是它事实上是对COO存储格式进行了压缩。这种格式能有效地处理算术运算、列切片操作、矩阵向量的乘积这些问题。

|  |
| --- |
| 更详细的信息请参阅：  <http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.sparse.csr_matrix.html> |

* **CSC（稀疏列）[[3]](#footnote-3)**：这种方式与CSR基本相同，只不过这里的值要先按列读取，然后用行索引存储每个元素，并存储列指针。换句话说，CSC的表示形式是(val, row\_ind, col\_ptr)。

|  |
| --- |
| 这部分内容也可以查看文档：  <http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.sparse.csc_matrix.html> |

下面，我们就来亲自实践一下CSR矩阵的操作。假设我们现在有一个稀疏矩阵A：

>>> from scipy import sparse as s

>>> A = array([[1,0,0],[0,2,0],[0,0,3]])

>>> A

array([[1, 0, 0], [0, 2, 0], [0, 0, 3]])

>>> from scipy import sparse as sp

>>> C = = sp.csr\_matrix(A);

>>> C

<3x3 sparse matrix of type '<type 'NumPy.int32'>'

with 3 stored elements in Compressed Sparse Row format>

如您所见，这个CSR矩阵中只存储了三个元素。下面再来看看它实际存储了什么：

>>> C.toarray()

array([[1, 0, 0], [0, 2, 0], [0, 0, 3]])

>>> C \* C.todense()

matrix([[1, 0, 0], [0, 4, 0], [0, 0, 9]])

正如我们所期待的那样。CSR矩阵实际上并没有略过所有的0，我们用它可以得到与原矩阵一样的计算结果。

>>> dot(C, C).todense()

### 优化措施

我们读者要明白一件事，即每当我们要在后台构建一个分类器或标注器时，所做的一切都为了执行某种优化历程。下面我们来对SciPy库中所提供的函数做一些基本的了解。我们从求给定多项式的最小值开始，下面代码所演示的就是SciPy库在这方面的一个优化例程。

>>> def f(x):

>>> returnx return x\*\*2-4

>>> optimize.fmin\_bfgs(f,0)

Optimization terminated successfully.

Current function value: -4.000000

Iterations: 0

Function evaluations: 3

Gradient evaluations: 1

array([0])

这里的第一个参数是我们想用来求最小值的函数，而第二个参数则是我们对该最小值的初始猜测。在这个例子中，我们已经知道了最小值为0。如果读者还想获取更多详细信息，也可以在这里调用help()函数：

>>> help(optimize.fmin\_bfgs)

Help on function fmin\_bfgs in module Scipy.optimize.optimize:

fmin\_bfgs(f, x0, fprime=None, args=(), gtol=1e-05, norm=inf,

epsilon=1.4901161193847656e-08, maxiter=None, full\_output=0, disp=1,

retall=0, callback=None)

Minimize a function using the BFGS algorithm.

Parameters

---------

f : callable f(x,\*args)

Objective function to be minimized.

x0 : ndarray

Initial guess.

>>> from scipy import optimize

optimize.fsolve(f, 0.2)

array([ 0.46943096])

>>> def f1 def f1(x,y):

>>> return x \*\* 2+ y \*\* 2 - 4

>>> optimize.fsolve(f1, 0, 0)

array([ 0.])

总而言之，我们现在应该对于SciPy库中最基本的那些数据结构应该有了充足的知识准备，并了解其中的一些最常用的优化技术。我们的目的并不只是要鼓励您去运行一些与机器学习或自然语言处理相关的应用，而是希望您要超越自己所用的这些ML算法所在的数学语境，去看看那些源代码，并试着去理解它们。

具体的实现不仅有助于对算法的理解，而且还能让我们在优化/自定义的过程中去实现自己的需要。

## pandas

下面我们来讨论一下pandas库，这也是Python中最令人兴奋的库之一，尤其是对于那些喜欢喜欢R语言的人来说，想要在Python中以更向量化的方式来操作数据，就非它莫属了。我们会在这一部分章节中专门介绍一下pandas库，讨论一些pandas框架下的基本数据操作和相关处理。

### 读取数据

我们先来看所有数据分析问题中的一个最重要的任务：如何解析CSV /其他文件中的数据。

|  |
| --- |
| 我们在这里使用的是以下两份文件：  <https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data>  <https://archive.ics.uci.edu/ml/machine-learningdatabases/iris/iris.names>  当然，读者也可以自行选择其它CSV文件。 |

首先，我们要将上述链接中的数据文件下载到本地存储起来，并将其加载到pandas数据框架中，具体如下：

>>> import pandas as pd

>>> # Please provide the absolute path of the input file

>>> data = pd.read\_csv("PATH\\iris.data.txt",header=0")

>>> data.head()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 0 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 1 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 2 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

在这里，我们要将CSV文件读取并存储到DataFrame中。在读取CSV文件时，我们目前可以有许多选项可用。其中要面对的一个问题就是：我们所读取第一行数据在DataFrame中会被当作是标题（header）；所以如果我们要设置真正的标题，就需要将其header选项设置为None，然后再将列表名作为其中的列名来传递给DataFrame。当然，如果我们的CSV文件中已经有了完美形式的标题，那就不需要担心pandas库中的标题问题了，因为在默认情况下，它会始终假设第一行数据为标题。之前代码中开头的0实际上就被当成了标题行的行号。

所以下面我们要为之前所用的同一批数据添加一些标题：

>>> data = pd.read\_csv("PATH\\iris.data.txt", names=["sepal length",

"sepal width", "petal length", "petal width", "Cat"], header=None)

>>> data.head()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | sepal length | sepal width | petal length | petal width | Cat |
| 0 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 2 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |

这样一来，我们就为这个框架创建了一系列临时列名，当然，如果文件本身第一行就是其标题的话，我们可以撤下前面的标题选项，这样pandas库就会将其检测到的文件首行当作标题。除此之外，作为常见选项的还有Sep/Delimiter，其主要用于指定列的分隔符。而在读取和清理数据的具体方式上，我们也至少有20个不同的优化选项可用，例如删除Na、删除空白行以及对特定列进行索引等。下面，我们来看一下其在不同文件类型上个的表现：

* read\_csv：读取CSV文件。
* read\_excel：读取XLS文件。
* read\_hdf：读取HDFS文件。
* read\_sql：读取SQL文件。
* read\_json：读取JSON文件。

当然，这些动作都可以在*第2章“文本的歧义及其清理”*中所讨论的那些不同解析方法中找到替代方案。另外，我们在写文件时也有相同数量的选项可用。

接下来我们要来看看pandas框架具体有多大的能力。如果您是个R程序员，想必会喜欢看到类似R语言中的摘要和标题选项。.

>>> data.describe()

这里的describe()函数将会为您提供一份关于每一列以及相关唯一值的简要信息。

>>> sepal\_len\_cnt=data['sepal length'].value\_counts()

>>> sepal\_len\_cnt

5.0 10

6.3 9

6.7 8

5.7 8

5.1 8

dtype: int64

>>>data['Iris-setosa'].value\_counts()

Iris-versicolor 50

Iris-virginica 50

Iris-setosa 48

dtype: int64

同样是为了满足R语言的爱好者，我们现在处理向量也可以通过这样的东西来查看列中的每个值：

>>>data['Iris-setosa'] == 'Iris-setosa'

0 True

1 True

147 False

148 False

Name: Iris-setosa, Length: 149, dtype: bool

现在我们还可以对DataFrame进行过滤。例如这里的setosa只能匹配与Iris-setosa相关的条目。

>>> sntsosa=data[data['Cat'] == 'Iris-setosa']

>>> sntsosa[:5]

这其实就是我们典型的SQL分组函数。其它聚合函数的情况也差不多是这样。

|  |
| --- |
| 您可以通过下面链接来看看道琼斯的数据：  https://archive.ics.uci.edu/ml/machine-learningdatabases/00312/ |

### 数列

Pandas库中也有一种按日期索引的简洁方式，并支持日后按照各种时间顺序对其进行数据分析。这样做最大的优点就是只要我们按日期对数据进行了索引，那些最令人最痛苦的日期操作对我们来说就是一个命令的事。下面，我们就来具体看看序列数据的处理，例如这里有几个股票的股价数据，以及其每周开盘价和收盘价的变化。[[4]](#footnote-4)

>>> import pandas as pd

>>> stockdata = pd.read\_csv("dow\_jones\_index.data",parse\_dates=['date'], index\_col=['date'],

nrows=100)

>>> stockdata.head()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date | Quarter | stock | Open | High | Low | close | volume | percent\_ change\_ price |
| 01/07/2011 | 1 | AA | $15.82 | $16.72 | $15.78 | $16.42 | 239655616 | 3.79267 |
| 01/14/2011 | 1 | AA | $16.71 | $16.71 | $15.64 | $15.97 | 242963398 | -4.42849 |
| 01/21/2011 | 1 | AA | $16.19 | $16.38 | $15.60 | $15.79 | 138428495 | -2.47066 |

>>> max(stockdata['volume'])

1453438639

>>> max(stockdata['percent\_change\_price'])

7.6217399999999991

>>> stockdata.index

<class 'pandas.tseries.index.DatetimeIndex'>

[2011-01-07, ..., 2011-01-28]

Length: 100, Freq: None, Timezone: None

>>> stockdata.index.day

array([ 7, 14, 21, 28, 4, 11, 18, 25, 4, 11, 18, 25, 7, 14, 21, 28, 4,11, 18, 25, 4, 11, 18, 25,

7, 14, 21, 28, 4])

上述命令给出的是每个日期的日信息。[[5]](#footnote-5)

>>> stockdata.index.month

上述命令将会按月列出不同的值。

>>> stockdata.index.year

上述命令将会按年列出不同的值。

我们也可以通过一个叫做resample的函数按照自己的想法对数据进行聚合。它的选项包括sum、mean、median、min和max。

>>> import numpy as np

>>> stockdata.resample('M', how=np.sum)

### 列转换

假设我们现在想要过滤某些列或添加一列数据，就可以用刚才提供的列之列表来充当轴1参数来实现它。我们可以像下面这样来删除数据框架中的某列数据：

>>> stockdata.drop(["percent\_change\_volume\_over\_last\_wk"],axis=1)

下面我们来过滤掉一些自己不想要的列，并且将工作限制在某一组列中。为此，我们可以像这样新建一个DataFrame：

>>> stockdata\_new = pd.DataFrame(stockdata, columns=["stock","open","high"

,"low","close","volume"])

>>> stockdata\_new.head()

我们还可以执行一些类似于R语言中的列操作，譬如说重命名列。另外，我们也可以做这样的事：

>>> stockdata["previous\_weeks\_volume"] = 0

这会将目标列中的所有值都更改为0，我们可以有条件地执行这样的操作，并在其中创建一些派生变量。

### 噪声数据

通常情况下，数据科学家每天的日常生活都是从数据清洗开始。这项任务包括移除噪音、清除一些不想要的文件、确保日期格式的正确、忽略干扰性记录、并处理丢失的值。总之，数据清理所占用的时间段往往是最大的，其它活动则并非如此。。

在实际的应用场景中，数据在大多数情况下都是混乱的，我们必须处理其中的缺失值、空值、Na以及其它格式问题。因此，任何处理数据的程序库的其中一个主要功能就是要能处理上述这些问题并能以有效的方式解决它们。对于这其中的一些问题，pandas库倒也颇提供了一些令人惊叹的功能。

>>> stockdata.head()

>>> stockdata.dropna().head(2)

通过上面的命令，我们清除掉了目标数据中所有的Na。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date | quarter | Stock | Open | High | Low | close | volume | percent\_ change\_ price |
| 01/14/2011 | 1 | AA | $16.71 | $16.71 | $15.64 | $15.97 | 242963398 | -4.42849 |
| 01/21/2011 | 1 | AA | $16.19 | $16.38 | $15.60 | $15.79 | 138428495 | -2.47066 |
| 01/28/2011 | 1 | AA | $15.87 | $16.63 | $15.82 | $16.13 | 151379173 | 1.63831 |

另外，你们应该也注意到一些值之前有个$符，它会给我们的数字操作带来一些困难。下面我们要来摆脱这一麻烦，因为不这样做，它会干扰我们的结果（例如可能会导致$ 43.86不是这里的最值）。

>>> import numpy

>>> stockdata\_new.open.describe()

count 100

unique 99

top $43.86

freq 2

Name: open, dtype: object

我们还可以针对两列数据执行一些操作，并从中导出一个新变量：

>>> stockdata\_new.open = stockdata\_new.open.str.replace('$', '').convert\_

objects(convert\_numeric=True)

>>> stockdata\_new.close = stockdata\_new.close.str.replace('$', '').

convert\_objects(convert\_numeric=True)

>>> (stockdata\_new.close - stockdata\_new.open).convert\_objects(convert\_

numeric=True)

>>> stockdata\_new.open.describe()

count 100.000000

mean 51.286800

std 32.154889

min 13.710000

25% 17.705000

50% 46.040000

75% 72.527500

max 106.900000

Name: open, dtype: float64

我们也可以执行一些算术运算，并为此创建新的变量。

>>> stockdata\_new['newopen'] = stockdata\_new.open.apply(lambda x: 0.8 \* x)

>>> stockdata\_new.newopen.head(5)

另外，我们可以用下面这种方式来过滤某列值中的数据。例如，下面我们来从上面的股票值中过滤出其中一个公司的数据集。

>>> stockAA = stockdata\_new.query('stock=="AA"')

>>> stockAA.head()

总而言之，我们在这一节中介绍了一系列pandas库中与数据读取、清理、操纵以及聚合的实用函数。下一节我们来看看如何利用这些数据框架产生针对这些数据的可视化图表。

## matplotlib

matplotlib是Python语言环境中一个非常受欢迎的可视化库。接下来，我们将为您介绍一些最常用的可视化应用。首先我们要导入这个库：

>>> import matplotlib

>>> import matplotlib.pyplot as plt

>>> import numpy

下面，我们要利用运行在道琼斯指数上的数据集来执行一些可视化操作。之前，我们已经拥有了公司“AA”的股票数据。下面我们要为一家新公司CSCO再创建一个数据框架，然后围绕着它进行一些绘制工作：

>>> stockCSCO = stockdata\_new.query('stock=="CSCO"')

>>> stockCSCO.head()

>>> from matplotlib import figure

>>> plt.figure()

>>> plt.scatter(stockdata\_new.index.date,stockdata\_new.volume)

>>> plt.xlabel('day') # added the name of the x axis

>>> plt.ylabel('stock close value') # add label to y-axis

>>> plt.title('title') # add the title to your graph

>>> plt.savefig("matplot1.jpg") # savefig in local

我们还可以将上面绘制的图保存为JPEG / PNG文件，只需要调用savefig()函数，例如：

>>> plt.savefig("matplot1.jpg")

### 子图绘制

子图绘制法是我们布局整体绘制的最佳方式。这就像在一块画布上工作，我们可以在上面不会只有一轮绘制，而是多轮绘制的叠加。在下例中，我们将会进行四轮绘制。我们会通过参数numrow、numcol来定义画布，并在下一个参数指定绘制的编号。

>>> plt.subplot(2, 2, 1)

>>> plt.plot(stockAA.index.weekofyear, stockAA.open, 'r--')

>>> plt.subplot(2, 2, 2)

>>> plt.plot(stockCSCO.index.weekofyear, stockCSCO.open, 'g-\*')

>>> plt.subplot(2, 2, 3)

>>> plt.plot(stockAA.index.weekofyear, stockAA.open, 'g--')

>>> plt.subplot(2, 2, 4)

>>> plt.plot(stockCSCO.index.weekofyear, stockCSCO.open, 'r-\*')

>>> plt.subplot(2, 2, 3)

>>> plt.plot(x, y, 'g--')

>>> plt.subplot(2, 2, 4)

>>> plt.plot(x, y, 'r-\*')

（图）

我们也可以用一些更优雅的做法来进行多次绘制！

>>> fig, axes = plt.subplots(nrows=1, ncols=2)

>>> for ax in axes:

>>> ax.plot(x, y, 'r')

>>> ax.set\_xlabel('x')

>>> ax.set\_ylabel('y')

>>> ax.set\_title('title');

如你所见，我们可以有很多种编写代码的方式，这更像是在通用的Python环境中，我们按自身所想的方式在处理不同方面的绘制问题。

### 添加坐标轴

We can add an axis to the figure by using addaxis(). By adding an axis to the figure, we can define our own drawing area. addaxis() takes the following arguments:

我们可以通过使用addaxis（）为图添加一个轴。 通过向图中添加轴，我们可以定义我们自己的绘图区域。 addaxis（）使用以下参数：

\*rect\* [\*left\*, \*bottom\*, \*width\*, \*height\*]

>>> fig = plt.figure()

>>> axes = fig.add\_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range 0 to 1)

>>> axes.plot(x, y, 'r')

Let' plot some of the most commonly used type of plots. The great thing is that most of the parameters, such as title and label, still work in the same way. Only the kind of plot will change.

让我们绘制一些最常用的图形类型。 伟大的事情是，大多数参数，如标题和标签，仍然以相同的方式工作。 只有那种情节会改变。

If you want to add an x label, a y label, and a title with the axis; the commands are as follows:

如果要添加x标签，y标签和带轴的标题; 命令如下：

>>> fig = plt.figure()

>>> ax = fig.add\_axes([0.1, 0.1, 0.8, 0.8])

>>> ax.plot(stockAA.index.weekofyear,stockAA.open,label="AA")

>>> ax.plot(stockAA.index.weekofyear,stockCSCO.open,label="CSCO")

>>> ax.set\_xlabel('weekofyear')

>>> ax.set\_ylabel('stock value')

>>> ax.set\_title('Weekly change in stock price')

>>> ax.legend(loc=2); # upper left corner

>>> plt.savefig("matplot3.jpg")

Try writing the preceding code and observe the output!

尝试编写前面的代码并观察输出！

（图）

### 散点图绘制

One of the simplest forms of plotting is to plot the y-axis point for different x-axis values. In the following example, we have tried to capture the variation of the stock price weekly in a scatter plot:

绘制的最简单形式之一是绘制不同x轴值的y轴点。 在以下示例中，我们尝试捕捉散点图中每周的股价变化：

>>> import matplotlib.pyplot as plt

>>> plt.scatter(stockAA.index.weekofyear,stockAA.open)

>>> plt.savefig("matplot4.jpg")

>>> plt.close()

### 条形图绘制

Intuitively, the distribution of the y axis is shown against the x axis in the following bar chart. In the following example, we have used a bar plot to display data on a graph.

直观地，在下面的条形图中相对于x轴示出了y轴的分布。 在以下示例中，我们使用条形图在图形上显示数据。

>>> n = 12

>>> X = np.arange(n)

>>> Y1 = np.random.uniform(0.5, 1.0, n)

>>> Y2 = np.random.uniform(0.5, 1.0, n)

>>> plt.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')

>>> plt.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')

### 3D绘图

We can also build some spectacular 3D visualizations in matplotlib. The following example shows how one can create a 3D plot using matplotlib:

我们还可以在matplotlib中构建一些壮观的3D可视化。 以下示例显示如何使用matplotlib创建3D图：

>>> from mpl\_toolkits.mplot3d import Axes3D

>>> fig = plt.figure()

>>> ax = Axes3D(fig)

>>> X = np.arange(-4, 4, 0.25)

>>> Y = np.arange(-4, 4, 0.25)

>>> X, Y = np.meshgrid(X, Y)

>>> R = np.sqrt(X\*\*2+ + Y\*\*2)

>>> Z = np.sin(R)

>>> ax.plot\_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot')

## 外部参考资料

I like to encourage readers to go over some of the following links for more details about the individual libraries, and for more resources:

我想鼓励读者阅读以下链接，了解有关各个库的更多详细信息，以及更多资源：

* <http://www.NumPy.org/>
* <http://www.Scipy.org/>
* <http://pandas.pydata.org/>
* <http://matplotlib.org/>

## 本章小结

This chapter was a brief summary of some of the most fundamental libraries of Python that do a lot of heavy lifting for us when we deal with text and other data. NumPy helps us in dealing with numeric operations and the kind of data structure required for some of these. SciPy has many scientific operations that are used in various Python libraries. We learned how to use these functions and data structures.

本章是对一些最基本的Python库的简要总结，当我们处理文本和其他数据时，它们为我们做了很多重大的工作。 NumPy帮助我们处理数字操作和其中一些所需的数据结构类型。 SciPy具有许多在各种Python库中使用的科学操作。 我们学习了如何使用这些函数和数据结构。

We have also touched upon pandas, which is a very efficient library for data manipulation, and has been getting a lot of mileage in recent times. Finally, we gave you a quick view of one of Python's most commonly used visualization libraries, matplotlib.

我们还碰到了熊猫，这是一个非常有效的数据操作库，并且在最近的时间已经获得了很多里程。 最后，我们快速浏览了Python最常用的可视化库matplotlib。

In the next chapter, we will focus on social media. We will see how to capture data from some of the common social networks and produce meaningful insights around social media.

在下一章中，我们将关注社交媒体。 我们将看到如何从一些常见的社交网络捕获数据，并对社交媒体产生有意义的洞察。

1. 译者注：原文如此，其实矩阵乘法运算的结果举证在形状上应该是行数等于第一输入矩阵，列数等于第二输入矩阵。在这里是两个相同的矩阵相乘，自然在形状上也相同。 [↑](#footnote-ref-1)
2. 译者注：原文在这里似乎是将CSR描绘成了CSC，然后又在后面的CSC重复了一遍，译者在这里做了自行更正。 [↑](#footnote-ref-2)
3. 译者注：这种方式与CSR相对应，因此也叫做**按列压缩存储**。 [↑](#footnote-ref-3)
4. 译者注：在这里，为了让表格与代码保持一致，就不对表头部分进行翻译了，下同。 [↑](#footnote-ref-4)
5. 译者注：此处原文似乎与代码对不上，如果按照原文day of the week for each date，上面调用的应该是stockdata.index.dayofweek。 [↑](#footnote-ref-5)