# 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 （用于Python中的数值运算）
* SciPy （用于Python中的科学计算）
* Pandas （用于进行数据操纵）
* Matplotlib （用于执行可视化处理）

## 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

One of the common matrix operations is element-wise multiplication, where we will multiply one element of a matrix by an element of another matrix. The shape of the resultant matrix will be same as the input matrix, for example:

公共矩阵运算之一是逐元素乘法，其中我们将矩阵的一个元素乘以另一个矩阵的元素。 所得矩阵的形状将与输入矩阵相同，例如：

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

>>>A \* A

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

|  |
| --- |
| However, we can't perform the following operation, which will throw an error when executed:  但是，我们不能执行以下操作，这将在执行时抛出错误：  >>>A \* B  ----------------------------------------------------------  -----------------  ValueError Traceback (most recent call last)  <ipython-input-53-e2f71f566704> in <module>()  ----> 1 A\*B  ValueError: Operands could not be broadcast together with shapes (2,2) (4,).  ValueError：操作数不能与形状（2,2）（4，）一起广播。  Simply, the numbers of columns of the first operand have to match the number of rows in the second operand for matrix multiplication to work.  ValueError：操作数不能与形状（2,2）（4，）一起广播。  简单地说，第一操作数的列数必须与第二操作数中的行数匹配，以便矩阵乘法工作。 |

Let's do the dot product, which is the backbone of many optimization and algebraic operations. I still feel doing this in a traditional environment was not very efficient. Let's see how easy it is in NumPy, and how super-efficient it is in terms of memory.

让我们做点积，这是许多优化和代数运算的支柱。 我仍然觉得这样做在传统的环境下不是很有效率。 让我们看看NumPy有多么容易，它在内存方面是多么高效。

>>>np.dot(A, A)

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

We can do operations like add, subtract, and transpose, as shown in the following example:

我们可以进行加，减和转置等操作，如下例所示：

>>>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]])

The same transpose operations can be performed using an alternative operation, such as this:

可以使用替代操作执行相同的转置操作，例如：

>>>A.T

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

We can also cast these ndarrays into a matrix and perform matrix operations, as shown in the following example:

我们还可以将这些ndarrays转换为矩阵并执行矩阵操作，如以下示例所示：

>>>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]])

We can perform all sorts of complex matrix operations with NumPy, and they are pretty simple to use too! Please have a look at documentation for more information on NumPy.

我们可以用NumPy执行各种复杂的矩阵运算，他们也很简单使用！ 请查看有关NumPy的更多信息的文档。

Let's switch back to some of the common mathematics operations, such as min, max, mean, and standard deviation, for the given array elements. We have generated the normal distributed random numbers. Let's see how these things can be applied there:

让我们切换回一些常见的数学运算，例如给定数组元素的最小，最大，平均和标准偏差。 我们已经生成了正态分布随机数。 让我们看看这些东西可以在那里应用：

>>>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

This was an example demonstrating how NumPy can be used to perform simple mathematic and algebraic operations of finding out the mean and standard deviation of a set of numbers.

这是一个示例，演示如何使用NumPy来执行简单的数学和代数运算，找出一组数字的平均值和标准差。

#### Reshaping and stacking

In case of some of the numeric, algebraic operations we do need to change the shape of resultant matrix based on the input matrices. NumPy has some of the easiest ways of reshaping and stacking the matrix in whichever way you want.

在一些数字，代数运算的情况下，我们需要基于输入矩阵改变结果矩阵的形状。 NumPy有一些最简单的方式来重塑和堆叠矩阵以任何你想要的方式。

>>>A

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

If we want a flat matrix, we just need to reshape it using NumPy's reshape() function:

如果我们想要一个平面矩阵，我们只需要使用NumPy的reshape（）函数重塑它：

>>>>(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]])

This kind of reshaping is required in many algebraic operations. To flatten the ndarray, we can use the flatten() function:

这种重塑在许多代数运算中是必需的。 要展平ndarray，我们可以使用flatten（）函数：

>>>A.flatten()

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

There is a function to repeat the same elements of the given array. We need to just specify the number of times we want the element to repeat. To repeat the ndarray, we can use the repeat() function:

有一个函数重复给定数组的相同元素。 我们需要指定我们希望元素重复的次数。 要重复ndarray，我们可以使用repeat（）函数：

>>>np.repeat(A, 2)

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

>>>>A

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

In the preceding example, each element is repeated twice in sequence. A similar function known as tile() is used for for repeating the matrix, and is shown here:

在前面的示例中，每个元素按顺序重复两次。 称为tile（）的类似函数用于重复矩阵，如下所示：

>>>np.tile(A, 4)

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

There are also ways to add a row or a column to the matrix. If we want to add a row, we use the concatenate() function shown here:

还有一些方法可以向矩阵中添加行或列。 如果我们想添加一行，我们使用concatenate（）函数：

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

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

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

This can also be achieved using the Vstack() function shown here:

这也可以使用Vstack（）函数来实现，如下所示：

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

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

Also, if you want to add a column, you can use the concatenate() function in the following manner:

此外，如果要添加列，可以按以下方式使用concatenate（）函数：

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

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

|  |
| --- |
| Alternatively, the hstack() function can be used to add columns. This is used very similarly to the vstack() function in the example shown above.  或者，hstack（）函数可用于添加列。 这与上面所示的示例中的vstack（）函数非常相似。 |

#### Random numbers

Random number generation is also used across many tasks involving NLP and machine learning tasks. Let's see how easy it is to get a random sample:

随机数生成也用于涉及NLP和机器学习任务的许多任务。 让我们看看获取随机样本有多么容易：

>>>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]])

There is one more function called random.randn(), which generates normally distributed random numbers in the given range. So, in the following example, we want random numbers between 2 and 5.

还有一个称为random.randn（）的函数，它在给定的范围内生成正态分布的随机数。 因此，在下面的示例中，我们需要2到5之间的随机数。

>>>>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]])

This is achieved by using the function random.randn(2,5).

这通过使用函数random.randn（2,5）来实现。

## SciPy

Scientific Python or SciPy is a framework built on top of NumPy and ndarray and was essentially developed for advanced scientific operations such as optimization, integration, algebraic operations, and Fourier transforms.

科学Python或SciPy是一个构建在NumPy和ndarray之上的框架，基本上是为高级科学操作（如优化，集成，代数运算和傅立叶变换）而开发的。

The concept was to efficiently use ndarrays to provide some of these common scientific algorithms in a memory-efficient manner. Because of NumPy and SciPy, we are in a state where we can focus on writing libraries such as scikit-learn and NLTK, which focus on domain-specific problems, while NumPy / SciPy do the heavy lifting for us. We will give you a brief overview of the data structures and common operations provided in SciPy. We get the details of some of the black-box libraries, such as scikit-learn and understand what goes on behind the scenes.

这个概念是有效地使用ndarrays以内存高效的方式提供一些这些常见的科学算法。 因为NumPy和SciPy，我们处于一个状态，我们可以专注于编写库，如scikit-learn和NLTK，这些库专注于特定领域的问题，而NumPy / SciPy为我们提供了很大的帮助。 我们将向您简要介绍SciPy中提供的数据结构和常见操作。 我们得到一些黑盒图书馆的细节，例如scikit-learn和理解幕后情况。

>>>import scipy as sp

This is how you import SciPy. I am using sp as an alias but you can import everything.

这是您如何导入SciPy。 我使用sp作为别名，但你可以导入一切。

Let's start with something we are more familiar with. Let's see how integration can be achieved here, using the quad() function.

让我们从我们更熟悉的东西开始。 让我们看看如何使用quad（）函数在这里实现集成。

>>>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

If we integrate the x, it will be x2/2, which is 0.5. There are other scientific functions, such as these:

如果我们积分x，它将是x2 / 2，这是0.5。 还有其他科学功能，如：

* Interpolation (scipy.interpolate)
* Fourier transforms (scipy.fftpack)
* Signal processing (scipy.signal)
* 插值（scipy.interpolate）
* 傅立叶变换（scipy.fftpack）
* 信号处理（scipy.signal）

But we will focus on only linear algebra and optimization because these are more relevant in the context of machine learning and NLP.

但是我们将仅关注线性代数和优化，因为这些在机器学习和NLP的上下文中更相关。

### 线性代数

The linear algebra module contains a lot of matrix-related functions. Probably the best contribution of SciPy is sparse matrix (CSR matrix), which is used heavily in other packages for manipulation of matrices.

线性代数模块包含大量与矩阵相关的函数。 SciPy的最大贡献可能是稀疏矩阵（CSR矩阵），其在其他包中用于操纵矩阵。

SciPy provides one of the best ways of storing sparse matrices and doing data manipulation on them. It also provides some of the common operations, such as linear equation solving. It has a great way of solving eigenvalues and eigenvectors, matrix functions (for example, matrix exponentiation), and more complex operations such as decompositions (SVD). Some of these are the behind-the-scenes optimization in our ML routines. For example, SVD is the simplest form of LDA (topic modeling) that we used in Chapter 6, Text Classification.

SciPy提供了存储稀疏矩阵和对它们进行数据操作的最佳方法之一。 它还提供了一些常见的操作，如线性方程求解。 它具有解决特征值和特征向量，矩阵函数（例如，矩阵取幂）和更复杂的操作（例如分解（SVD））的伟大方式。 其中一些是我们的ML例程中的幕后优化。 例如，SVD是我们在第6章“文本分类”中使用的LDA（主题建模）的最简单形式。

The following is an example showing how the linear algebra module can be used:

以下是显示线性代数模块如何使用的示例：

>>>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)

|  |
| --- |
| Detailed documentation is available at  详细文档可从  IUUQ EPDTTDJQZPSHEPDTDJQZSFGFSFODFMJOBMHIUNM. |

### 特征值与特征向量

In some of the NLP and machine learning applications, we represent the documents as term document matrices. Eigenvalues and eigenvectors are typically calculated for many different mathematical formulations. Say A is our matrix, and there exists a vector v such that Av=λv.

在一些NLP和机器学习应用程序中，我们将文档表示为术语文档矩阵。 通常针对许多不同的数学公式计算特征值和特征向量。 说A是我们的矩阵，并且存在矢量v，使得Av =λv。

In this case, λ will be our eigenvalue and v will be our eigenvector. One of the most commonly used operation, the singular value decomposition (SVD)will require some calculus functionality. It's quite simple to achieve this in SciPy.

在这种情况下，λ将是我们的特征值，v将是我们的特征向量。 作为最常用的操作之一，奇异值分解（SVD）将需要一些微积分功能。 在SciPy中实现这个很简单。

>>>evals = LA.eigvals(A)

>>>evals

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

And eigen vectors are as follows:

特征向量如下：

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

We can perform other matrix operations, such as inverse, transpose, and determinant:

我们可以执行其他矩阵运算，如逆，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

In a real-world scenario, when we use a typical matrix, most of the elements of this matrix are zeroes. It is highly inefficient to go over all these non-zero elements for any matrix operation. As a solution to this kind of problem, a sparse matrix format has been introduced, with the simple idea of storing only non-zero items.

在现实世界的场景中，当我们使用典型的矩阵时，该矩阵的大多数元素是零。对于任何矩阵运算，忽略所有这些非零元素是非常低效的。作为这种问题的解决方案，已经引入了稀疏矩阵格式，具有仅存储非零项的简单想法。

A matrix in which most of the elements are non-zeroes is called a dense matrix, and the matrix in which most of the elements are zeroes is called a sparse matrix.

其中大多数元素是非零的矩阵被称为密集矩阵，并且其中大多数元素是零的矩阵被称为稀疏矩阵。

A matrix is typically a 2D array with an index of row and column will provide the value of the element. Now there are different ways in which we can store sparse matrices:

矩阵通常是具有行和列的索引的2D数组，将提供元素的值。现在有不同的方法来存储稀疏矩阵：

* **DOK (Dictionary of keys)**: Here, we store the dictionary with keys in the format (row, col) and the values are stored as dictionary values.
* **LOL (list of list)**: Here, we provide one list per row, with only an index of the non-zero elements.
* **COL (Coordinate list)**: Here, a list (row, col, value) is stored as a list.
* **CRS/CSR (Compressed row Storage)**: A CSR matrix reads values first by column; a row index is stored for each value, and column pointers are stored (val, row\_ind, col\_ptr). Here, val is an array of the non-zero values of the matrix, row\_ind represents the row indices corresponding to the values, and col\_ptr is the list of val indexes where each column starts. The name is based on the fact that column index information is compressed relative to the COO format. This format is efficient for arithmetic operations, column slicing, and matrix-vector products.
* DOK（键的字典）：这里，我们用格式（row，col）存储字典，并将值存储为字典值。
* LOL（列表列表）：这里，我们为每行提供一个列表，只有非零元素的索引。
* COL（坐标列表）：这里，列表（行，列，值）存储为列表。
* CRS / CSR（压缩行存储）：CSR矩阵首先按列读取值;为每个值存储行索引，并且存储列指针（val，row\_ind，col\_ptr）。这里，val是矩阵的非零值的数组，row\_ind表示对应于值的行索引，col\_ptr是每个列开始的val索引的列表。该名称基于列索引信息相对于COO格式进行压缩的事实。此格式对于算术运算，列切片和矩阵向量乘积有效。

|  |
| --- |
| See IUUQEPDTTDJQZPSHEPDTDJQZ SFGFSFODFHFOFSBUFETDJQZTQBSTFDTS@NBUSJYIUNM for more information. |

* CSC (sparse column): This is similar to CSR, except that the values are read first by column; a row index is stored for each value, and column pointers are stored. In otherwords, CSC is (val, row\_ind, col\_ptr).
* CSC（稀疏列）：这类似于CSR，除了值是首先读取列; 为每个值存储行索引，并且存储列指针。 换句话说，CSC是（val，row\_ind，col\_ptr）。

|  |
| --- |
| Have a look at the documentation at:  IUUQEPDTTDJQZPSHEPDTDJQZSFGFSFODF |

Let's have some hands-on experience with CSR matrix manipulation. We have a sparse matrix A:

让我们有一些CSR矩阵操作的实践经验。 我们有一个稀疏矩阵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>

If you read very carefully, the CSR matrix stored just three elements. Let's see what it stored:

如果你仔细阅读，CSR矩阵只存储三个元素。 让我们看看它存储了什么：

>>>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]])

This is exactly what we are looking for. Without going over all the zeroes, we still got the same results with the CSR matrix.

这正是我们正在寻找的。 没有越过所有的零，我们仍然得到了与CSR矩阵相同的结果。

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

### 优化

I hope you understand that every time we have built a classifier or a tagger in the background, all these are some sort of optimization routine. Let's have some basic understanding about the function provided in SciPy. We will start with getting a minima of the given polynomial. Let's jump to one of the example snippets of the optimization routine provided by SciPy.

我希望你明白，每次我们在后台构建一个分类器或一个标签，所有这些都是一些优化程序。 让我们对SciPy中提供的函数有一些基本的了解。 我们将从获得给定多项式的最小值开始。 让我们跳到SciPy提供的优化例程的示例片段之一。

>>>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])

Here, the first argument is the function you want the minima of, and the second is the initial guess for the minima. In this example, we already knew that zero will be the minima. To get more details, use the function help(), as shown here:

这里，第一个参数是你想要的最小值的函数，第二个参数是最小值的初始猜测。 在这个例子中，我们已经知道零将是最小值。 要获得更多详细信息，请使用函数help（），如下所示：

>>>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.])

To summarize, we now have enough knowledge about SciPy's most basic data structures, and some of the most common optimization techniques. The intention was to motivate you to not just run black-box machine learning or natural language processing, but to go beyond that and get the mathematical context about the ML algorithms you are using and also have a look at the source code and try to understand it.

总而言之，我们现在有足够的知识关于SciPy的最基本的数据结构，以及一些最常见的优化技术。 目的是激励你不仅仅运行黑盒机器学习或自然语言处理，而是要超越这一点，获得你所使用的ML算法的数学上下文，并且还要看看源代码，并尝试理解 它。

Implementing this will not just help your understanding about the algorithm, but also allow you to optimize/customize the implementation to your need.

实现这不仅仅帮助你理解的算法，而且还允许你优化/自定义实现到你的需要。

## pandas

Let's talk about pandas, which is one of the most exciting Python libraries, especially for people who love R and want to play around with the data in a more vectorized manner. We will devote this part of the chapter only to pandas; we will discuss some basic data manipulation and handling in pandas frames.

让我们来谈谈熊猫，这是最令人兴奋的Python库之一，特别是对于那些喜欢R，想要以更向量化的方式玩数据的人。 我们将本章的这一部分只用于熊猫。 我们将讨论一些基本的数据操作和在熊猫框架中的处理。

### 读取数据

Let's start with one of the most important tasks in any data analysis to parse the data from a CSV/other file.

让我们从任何数据分析中的一个最重要的任务开始，解析CSV /其他文件中的数据。

|  |
| --- |
| I am using IUUQTBSDIJWFJDTVDJFEVNMNBDIJOF MFBSOJOHEBUBCBTFTBEVMUBEVMUEBUB  https://archive.ics.uci.edu/ml/machine-learningdatabases/iris/iris.names  Feel free to use any other CSV file. |

To begin, please download the data to your local storage from the preceding links, and load it into a pandas data-frame, as shown here:

首先，请从上述链接将数据下载到本地存储，并将其加载到pandas数据框中，如下所示：

>>>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 |

This will read a CSV file and store it in a DataFrame. Now, there are many options you have while reading a CSV file. One of the problems is that we read the first line of the data in this DataFrame as a header; to use the actual header, we need to set the option header to None, and pass a list of names as column names. If we already have the header in perfect form in the CSV, we don't need to worry about the header as pandas, by default, assumes the first line to be the header. The header 0 in the preceding code is actually the row number that will be treated as the header.

这将读取CSV文件并将其存储在DataFrame中。 现在，您在阅读CSV文件时有很多选项。 其中一个问题是我们读取这个DataFrame中的数据的第一行作为头; 要使用实际的头，我们需要将选项头设置为None，并将名称列表作为列名传递。 如果我们已经在CSV中具有完美形式的头，我们不需要担心头像pandas，默认情况下，假设第一行是头。 前面代码中的头0实际上是将被视为头的行号。

So let's use the same data, and add the header into the frame:

所以让我们使用相同的数据，并将标题添加到框架中：

>>>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 |

This has created temporary column names for the frame so that, in case you have headers in the file as a first row, you can drop the header option, and pandas will detect the first row of the file as the header. The other common options are Sep/ Delimiter, where you want to specify the delimiter used to separate the columns. There are at least 20 different options available, which can be used to optimize the way we read and cleanse our data, for example removing Na's, removing blank lines, and indexing based on the specific column. Please have a look at the different type of files:

这已经为框架创建了临时列名称，因此，如果文件中的标题作为第一行，您可以删除标题选项，而pandas将检测文件的第一行作为标题。 其他常见选项是Sep / Delimiter，其中要指定用于分隔列的分隔符。 至少有20个不同的选项可用于优化我们读取和清理数据的方式，例如删除Na，删除空白行以及基于特定列的索引。 请看看不同类型的文件：

* read\_csv: reading a CSV file.
* read\_excel: reading a XLS file.
* read\_hdf: reading a HDFS file.
* read\_sql: reading a SQL file.
* read\_json: reading a JSON file.

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。

These can be the substitutes for all the different parsing methods we discussed in Chapter 2, Text Wrangling and Cleansing. The same numbers of options are available to write files too.

这些可以是我们在第2章“文本串列和清理”中讨论的所有不同解析方法的替代。 相同数量的选项也可用于写入文件。

Now let's see the power of pandas frames. If you are an R programmer, you would love to see the summary and header option we have in R.

现在让我们看看熊猫框架的力量。 如果你是一个R程序员，你会喜欢看到摘要和标题选项我们在R.

>>>data.describe()

The describe() function will give you a brief summary of each column and the unique values.

describe（）函数将为您提供每个列和唯一值的简要摘要。

>>>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

Again for R lovers, we are now dealing with vectors, so that we can look for each value of the column by using something like this:

再次对于R的恋人，我们现在处理向量，所以我们可以通过使用这样的东西寻找列的每个值：

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

0 True

1 True

147 False

148 False

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

Now we can filter the DataFrame in place. Here the setosa will have only entries related to Iris-setosa.

现在我们可以过滤DataFrame了。 这里setosa将只有与虹膜setosa有关的条目。

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

>>>sntsosa[:5]

This is our typical SQL Group By function. We have all kinds of aggregate functions as well.

这是我们典型的SQL Group By函数。 我们也有各种聚合函数。

|  |
| --- |
| You can browse through the following link to look at Dow Jones data:  https://archive.ics.uci.edu/ml/machine-learningdatabases/00312/ |

### 数列

Pandas also have a neat way of indexing by date, and then using the frame for all sorts of time series kind of analysis. The best part is that once we have indexed the data by date some of the most painful operations on the dates will be a command away from us. Let's take a look at series data, such as stock price data for a few stocks, and how the values of the opening and closing stock change weekly.

熊猫也有一个整齐的方式按日期索引，然后使用框架进行各种时间序列分析。 最好的部分是，一旦我们按日期索引数据，一些最痛苦的日期操作将是一个命令远离我们。 让我们来看看系列数据，例如几个股票的股票价格数据，以及开仓和收盘股票的价格每周变化。

>>>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])

The preceding command gives the day of the week for each date.

>>>stockdata.index.month

The preceding command lists different values by month.

>>>stockdata.index.year

The preceding command lists different values by year.

You can aggregate the data using a resample with whatever aggregation you want. It could be sum, mean, median, min, or max.

您可以使用任何您想要的聚合重新采样来聚合数据。 它可以是sum，mean，median，min或max。

>>>import numpy as np

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

### 列转换

Say we want to filter out columns or to add a column. We can achieve this by just by providing a list of columns as an argument to axis 1. We can drop the columns from a data frame like this:

假设我们要过滤出列或添加列。 我们可以通过提供列的列表作为轴1的参数来实现这一点。我们可以从数据框中删除列，如下所示：

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

Let's filter out some of the unwanted columns, and work with a limited set of columns. We can create a new DataFrame like this:

让我们过滤掉一些不需要的列，并使用有限的一组列。 我们可以这样创建一个新的DataFrame：

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

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

>>>stockdata\_new.head()

We can also run R-like operations on the columns. Say I want to rename the columns. I can do something like this:

我们还可以在列上运行类似R的操作。 说我想重命名列。 我可以做这样的事情：

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

This will change all the values in the column to 0. We can do it conditionally and create derived variables in place.

这将把列中的所有值都更改为0.我们可以有条件地创建派生变量。

### 噪声数据

A typical day in the life of a data scientist starts with data cleaning. Removing noise, cleaning unwanted files, making sure that date formats are correct, ignoring noisy records, and dealing with missing values. Typically, the biggest chunk of time is spent on data cleansing rather than on any other activity.

在数据科学家的生活中的典型的一天从数据清洗开始。 删除噪音，清除不需要的文件，确保日期格式正确，忽略嘈杂的记录，并处理丢失的值。 通常，最大的时间段用于数据清理，而不是任何其他活动。

In a real-world scenario, the data is messy in most cases, and we have to deal with missing values, null values, Na's, and other formatting issues. So one of the major features of any data library is to deal with all these problems and address them in an efficient way. pandas provide some amazing features to deal with some of these problems.

在现实世界的场景中，在大多数情况下，数据是混乱的，我们必须处理缺失值，空值，Na和其他格式问题。 因此，任何数据库的主要特征之一是处理所有这些问题并以有效的方式解决它们。 熊猫提供一些惊人的功能来处理这些问题的一些。

>>>stockdata.head()

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

Using the preceding command we get rid of all the Na's from our data.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

You also noticed that we have a $ symbol in front of the value, which makes the numeric operation hard. Let's get rid of that, as it will give us noisy results otherwise (for example. $43.86 is not among the top values here).

你还注意到我们在值前面有一个$符号，这使得数字操作困难。 让我们摆脱它，因为它会给我们嘈杂的结果，否则（例如$ 43.86不是这里的顶级值）。

>>>import numpy

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

count 100

unique 99

top $43.86

freq 2

Name: open, dtype: object

We can perform some operations on two columns, and derive a new variable out of this:

我们可以对两列执行一些操作，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

We can also perform some arithmetic operations, and create new variables out of it.

我们还可以执行一些算术运算，并从中创建新的变量。

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

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

We can filter the data on the value of a column in this way too. For example, let's filter out a dataset for one of the companies among all those that we have the stock values for.

我们可以用这种方式过滤列的值的数据。 例如，让我们过滤出一个数据集，其中一个公司，我们有股票价值。

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

>>>stockAA.head()

To summarize, we have seen some useful functions related to data reading, cleaning, manipulation, and aggregation in this section of pandas. In the next section, will try to use some of these data frames to generate visualization out of this data.

总而言之，我们已经看到了一些与大熊猫这一节中的数据读取，清理，操作和聚合相关的有用函数。 在下一节中，将尝试使用这些数据框架中的一些来生成可视化数据。

## matplotlib

matplotlib is a very popular visualization library written in Python. We will cover some of the most commonly used visualizations. Let's start by importing the library:

matplotlib是一个非常受欢迎的可视化库用Python编写。 我们将介绍一些最常用的可视化。 让我们从导入库开始：

>>>import matplotlib

>>>import matplotlib.pyplot as plt

>>>import numpy

We will use the same running data set from the Dow Jones index for some of the visualizations now. We already have stock data for company "AA". Let's make one more frame for a new company CSCO, and plot some of these:

我们将使用与道琼斯指数相同的运行数据集来进行一些可视化。 我们已经有公司“AA”的股票数据。 让我们为一家新公司CSCO再做一个框架，并绘制一些：

>>>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

You can also save the figure as a JPEG/PNG file. This can be done using the savefig() function shown here:

您还可以将图保存为JPEG / PNG文件。 这可以使用savefig（）函数完成，如下所示：

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

### 子图绘制

Subplot is the best way to layout your plots. This works as a canvas, where we can add not just one plot but multiple plots. In this example, we have tried to put four plots with the parameters numrow, numcol which will define the canvas and the next argument in the plot number.

子图是布局你的地块的最好方法。 这作为一个画布，在那里我们可以添加不只是一个情节，但多个图。 在这个例子中，我们试图用参数numrow，numcol来放置四个图，它将定义画布，并且在图中的下一个参数。

>>>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）

We can do something more elegant for plotting many plots at one go!

我们可以做一些更优雅的东西来绘制很多地块！

>>>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');

As you case see, there are ways to code a lot more like in typical Python to handle different aspects of the plots you want to achieve.

正如你看到的，有很多方法来编写代码，就像在典型的Python中处理你想实现的地块的不同方面。

### 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。