NumPy

Table of Contents

1 NumPy简介

NumPy (https://en.wikipedia.org/wiki/NumPy) (Numeric Python) is the fundamental package for scientific computing with Python.

NumPy是一个运行速度非常快的数学库，主要用于数组计算。它可以让你在Python中使用向量和数学矩阵，以及许多用C语言实现的底层函数，你还可以体验到很好的运行效率。

参考：
http://www.numpy.org/ (http://www.numpy.org/)
https://docs.scipy.org/doc/numpy-dev/user/quickstart.html (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html)
NumPy Reference (https://docs.scipy.org/doc/numpy-dev/reference/index.html#reference)
http://old.sebug.net/paper/books/scipydoc/numpy_intro.html (http://old.sebug.net/paper/books/scipydoc/numpy_intro.html)

2 NumPy基础：ndarray

NumPy中定义的最重要的对象是称为 ndarray (https://docs.scipy.org/doc/numpy-dev/reference

/arrays.ndarray.html) 的N维数组类型，它是相同类型元素的集合。

下面是创建2维数组（大小为2x3）和3维数组（大小为3x3x3）的简单例子：

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> type(x)
<type 'numpy.ndarray'>
>>> y = np.array([[[ 0,  1,  2],
                   [ 3,  4,  5],
                   [ 6,  7,  8]],
                  [[ 9, 10, 11],
                   [12, 13, 14],
                   [15, 16, 17]],
                  [[18, 19, 20],
                   [21, 22, 23],
                   [24, 25, 26]]])
>>> y.shape
(3, 3, 3)
```

2.1 ndarray常见属性

表 1 是ndarray的常见属性。

Table 1: ndarray常见属性

ndarray属性	描述
ndim	数组维度。前面例子中x.ndim=2
shape	形状，即多少行和列。前面例子中x.shape=(2, 3)
size	元素个数。前面例子中x.size=6
dtype	元素类型。前面例子中x.dtype=dtype('int64')
itemsize	每一个条目所占的字节。前面例子中x.itemsize=8
nbytes	所有元素占的字节。前面例子中x.nbytes=48

2.1.1 ndarray内存布局

ndarray数组在内存中是连续内存块。有两种策略来组织多维数据：一种是column-major order（Fortran和Matlab采用这种策略），另一种是row-major order（C语言中采用这种策略）。如图 1 （摘自Guide to NumPy, 2.3.1 Contiguous Memory Layout）所示。

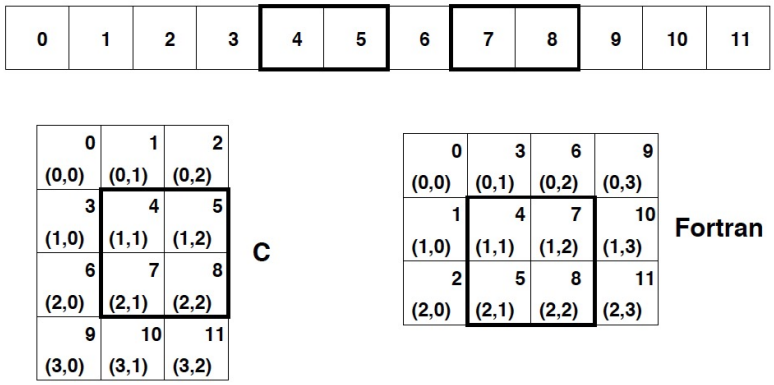

Figure 1: Options for memory layout of a 2-dimensional array

ndarray既支持**C**风格的内存布局，也支持**Fortran**风格的内存布局。 默认为C风格，进行转置操作后变为Fortran风格的内存布局。

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> x.flags
  C_CONTIGUOUS : True           # 表明x使用的是C风格的内存布局
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  UPDATEIFCOPY : False
>>> y = x.transpose()
>>> y.flags
  C_CONTIGUOUS : False
  F_CONTIGUOUS : True           # 表明y使用的是Fortran风格的内存布局
  OWNDATA : False
  WRITEABLE : True
  ALIGNED : True
  UPDATEIFCOPY : False
```

对于一维数组，显然既是C风格，又是Fortran风格的内存布局。

```
>>> x = np.array([0, 1, 2])     # 一维数组，显然既是C风格，又是Fortran风格的内存布局
>>> x.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  UPDATEIFCOPY : False
```

2.2 创建ndarray

表 2 是创建ndarray的一些函数。

Table 2: 创建ndarray的函数

创建**ndarray**的函数	说明
array (https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html#numpy.array)	创建ndarray
empty (https://docs.scipy.org/doc/numpy/reference/generated/numpy.empty.html#numpy.empty)	创建未初始化的ndarray
zeros (https://docs.scipy.org/doc/numpy/reference/generated/numpy.zeros.html#numpy.zeros)	创建全为0的ndarray
ones (https://docs.scipy.org/doc/numpy/reference/generated/numpy.ones.html#numpy.ones)	创建全为1的ndarray
full (https://docs.scipy.org/doc/numpy/reference/generated/numpy.full.html#numpy.full)	创建全为指定元素的ndarray
identity (https://docs.scipy.org/doc/numpy/reference/generated/numpy.identity.html#numpy.identity)	创建单位矩阵
eye (https://docs.scipy.org/doc/numpy/reference/generated/numpy.eye.html#numpy.eye)	创建对角线（可指定哪个对角线）为1，其它元素为0的二维数组
arange (https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html#numpy.arange)	基于开始值、终值和步长来创建一维数组
linspace (https://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html#numpy.linspace)	基于开始值、终值和元素个数来创建一维数组
logspace (https://docs.scipy.org/doc/numpy/reference/generated/numpy.logspace.html#numpy.logspace)	和linspace类似，对数刻度上均匀分布的一维数据
frombuffer (https://docs.scipy.org/doc/numpy/reference/generated/numpy.frombuffer.html#numpy.frombuffer)	从buffer创建数组
fromstring (https://docs.scipy.org/doc/numpy/reference/generated/numpy.fromstring.html#numpy.fromstring)	从string序列中创建数组
fromfunction (https://docs.scipy.org/doc/numpy/reference/generated/numpy.fromfunction.html#numpy.fromfunction)	从函数创建数组
fromfile (https://docs.scipy.org/doc/numpy/reference/generated/numpy.fromfile.html#numpy.fromfile)	基于tofile命令保存的文件创建ndarray
load (https://docs.scipy.org/doc/numpy/reference/generated/numpy.load.html#numpy.load)	基于save命令保存的文件创建ndarray
loadtxt (https://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html#numpy.loadtxt)	基于csv文本文件内容创建ndarray

参考：https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html (https://docs.scipy.org/doc/numpy/reference/routines.array-creation.html)

2.2.1 基于CSV文件创建ndarray（loadtxt实例）

假设“1.txt”内容如下。

```
$ cat 1.txt
0, 1, 2
3, 4, 5
```

使用loadtxt可以把文本文件中的数据加载为ndarray。如：

```
>>> import numpy as np
>>> x = np.loadtxt("1.txt", delimiter=",")       # 指定分隔符为逗号，默认使用空格为分隔
>>> x
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]])
```

2.3 存取元素(Indexing & Slicing)

数组元素的存取方法和Python的标准方法相同，如：

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[4]                  # 用整数作为下标可以获取数组中的某个元素
4
>>> a[1:3]                # 用范围作为下标获取数组的一个切片，含头(a[1])不含尾(a[3])
array([1, 2])
>>> a[:3]                 # 省略开始下标，表示从a[0]开始
array([0, 1, 2])
>>> a[:-2]                # 下标可以使用负数，表示从数组后往前数
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> a[0:5:2]              # 开始下标0，结束下标5，步长2
array([0, 2, 4])
```

和Python的列表序列不同， **NumPy**中通过“下标范围”获取的新的数组是原始数组的一个视图。也就是说它与原始数组共享同一块数据空间：

```
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b=a[1:3]                                          # 通过“下标范围”获取的新的数组是原始
>>> b[0]=99                                           # 修改了b[0]
>>> a
array([ 0, 99,  2,  3,  4,  5,  6,  7,  8,  9])   # a也被修改了
```

2.3.1 多维数组

多维数组有多个轴，所以它的下标需要用多个值来表示，NumPy中采用组元(tuple)表示数组的下标，也

就是用逗号分开多个轴的索引。请看下面实例：

```
>>> import numpy as np
>>> x = np.array([[ 0,  1,  2,  3,  4,  5],
...               [10, 11, 12, 13, 14, 15],
...               [20, 21, 22, 23, 24, 25],
...               [30, 31, 32, 33, 34, 35],
...               [40, 41, 42, 43, 44, 45],
...               [50, 51, 52, 53, 54, 55]])
>>>
>>> x[0]
array([0, 1, 2, 3, 4, 5])
>>> x[1]
array([10, 11, 12, 13, 14, 15])
>>> x[1, 4]                      # 逗号分开。逗号前是0轴索引，逗号后是1轴索引。
14
>>> x[0, 2:5]                    # 逗号分开。逗号前是0轴索引，逗号后是1轴索引。
array([2, 3, 4])
>>> x[1, 2:5]                    # 逗号分开。逗号前是0轴索引，逗号后是1轴索引。
array([12, 13, 14])
>>> x[:, 2]
array([ 2, 12, 22, 32, 42, 52])
>>> x[1:4, 2:6]                  # 逗号分开。逗号前是0轴索引，逗号后是1轴索引。
array([[12, 13, 14, 15],
       [22, 23, 24, 25],
       [32, 33, 34, 35]])
>>> x[1:, 2:6]                   # 逗号分开。逗号前是0轴索引，逗号后是1轴索引。
array([[12, 13, 14, 15],
       [22, 23, 24, 25],
       [32, 33, 34, 35],
       [42, 43, 44, 45],
       [52, 53, 54, 55]])
```

2.3.2 高级索引：整数序列索引

可以使用“整数序列”对数组元素进行存取，这时将使用整数序列中的每个元素作为下标，整数序列可以是Python中的列表或者NumPy中的数组。 使用整数序列作为下标获得的数组不和原始数组共享数据空间。 例如：

```
>>> import numpy as np
>>> x = np.array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
>>> y = x[[0, 2, 3]]                    # 使用整数序列 [0, 2, 3] 对数组元素进行存取，
>>> y
array([9, 7, 6])
>>> y[0] = 1111                         # 使用整数序列作为下标获得的数组y和原数组x不共享
>>> y
array([1111,    7,    6])
>>> x
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

下面是多维数组中，使用整数序列索引的例子：

```
>>> import numpy as np
>>> x = np.array([[ 0,  1,  2,  3,  4,  5],
...               [10, 11, 12, 13, 14, 15],
...               [20, 21, 22, 23, 24, 25],
...               [30, 31, 32, 33, 34, 35],
...               [40, 41, 42, 43, 44, 45],
...               [50, 51, 52, 53, 54, 55]])
>>>
>>> x[2, [1,2,3,5]]           # 第0轴是一个数，第1轴是整数序列 [1,2,3,5]
array([21, 22, 23, 25])
>>> x[2:, [1,2,3,5]]          # 第0轴是一个范围，第1轴是整数序列 [1,2,3,5]
array([[21, 22, 23, 25],
       [31, 32, 33, 35],
       [41, 42, 43, 45],
       [51, 52, 53, 55]])
```

2.3.3 高级索引：布尔数组索引

当使用布尔数组b作为下标存取数组x中的元素时，将收集数组x中所有在数组b中对应下标为True的元素。 使用布尔数组作为下标获得的数组不和原始数组共享数据空间。 注意这种方式只对应于NumPy中的布尔数组，不能使用Python中的布尔列表。

```
>>> import numpy as np
>>> x = np.array([4, 3, 2, 1, 0])
>>> x[np.array([True, False, False, True])]    # 使用布尔数组 np.array([True, Fals
array([4, 1])
>>> x[[True, False, True, False, False]]       # 如果使用布尔列表 [True, False, Tru
array([3, 4, 3, 4, 4])
```

NumPy布尔数组一般不是手工产生，而是使用布尔运算的Universal Function函数产生。如：

```
>>> x = np.random.rand(10)            # 产生长度为10，元素值在(0,1)区间内的随机数的数组
>>> x
array([ 0.31405475,  0.25280231,  0.08440324,  0.27874072,  0.81103738,
        0.19872166,  0.62412707,  0.62278573,  0.94702961,  0.18089782])
>>> x > 0.5                           # > 是Universal Function函数，返回布尔数组
array([False, False, False, False,  True, False,  True,  True,  True, False], dt
>>> x[x > 0.5]                        # 使用x>0.5返回的布尔数组收集x中的元素，因此得到的
array([ 0.81103738,  0.62412707,  0.62278573,  0.94702961])
```

2.4 Shape manipulation

表 3 总结了ndarray形状操作的相关函数。

Table 3: Shape manipulation

Shape manipulation function	Description
ndarray.reshape (https://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.reshape.html#numpy.reshape)(shape[, order])	Returns an array containing the same data with a new shape.
ndarray.resize(new_shape[, refcheck])	Change shape and size of array in-place.
ndarray.transpose(*axes)	Returns a view of the array with axes transposed.
ndarray.swapaxes(axis1, axis2)	Return a view of the array with axis1 and axis2 interchanged.
ndarray.flatten (https://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.ndarray.flatten.html#numpy.ndarray.flatten)([order])	Return a copy of the array collapsed into one dimension.
ndarray.ravel (https://docs.scipy.org/doc/numpy-dev/reference/generated/numpy.ravel.html#numpy.ravel)([order])	Return a flattened array.
ndarray.squeeze([axis])	Remove single-dimensional entries from the shape of a.

参考：
https://docs.scipy.org/doc/numpy-dev/user/quickstart.html#shape-manipulation (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html#shape-manipulation)
https://docs.scipy.org/doc/numpy-dev/reference/arrays.ndarray.html#shape-manipulation (https://docs.scipy.org/doc/numpy-dev/reference/arrays.ndarray.html#shape-manipulation)

2.4.1 reshape实例

下面是 `reshape` 的使用实例：

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> x.reshape(2, 4)
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> x.reshape(4, 2)
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
>>> x.reshape(2, 2, 2)
array([[[1, 2],
        [3, 4]],
       [[5, 6],
        [7, 8]]])
```

3 Array Calculation

表 4 中是数组计算的相关函数。它们中的大部分函数接受参数axis：
(1) 如果省略参数axis，则表示对整个数组进行操作；
(2) 如果提供了参数axis，则表示在对应坐标轴上进行操作。

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> x.sum()              # 省略了axis，对所有元素求和
21
>>> x.sum(0)
array([5, 7, 9])         # 5=1+4, 7=2+5, 9=3+6
>>> x.sum(1)
array([ 6, 15])          # 6=1+2+3, 15=4+5+6
>>> x.min()              # 省略了axis，在所有元素中找最小元素
1
>>> x.min(0)
array([1, 2, 3])
>>> x.min(1)
array([1, 4])
```

Table 4: 数组计算相关函数

Calculation函数	说明
ndarray.argmax([axis, out])	Return indices of the maximum values along the given axis.
ndarray.min([axis, out, keepdims])	Return the minimum along a given axis.
ndarray.argmin([axis, out])	Return indices of the minimum values along the given axis of a.

Calculation函数	说明
ndarray.ptp([axis, out])	Peak to peak (maximum - minimum) value along a given axis.
ndarray.clip([min, max, out])	Return an array whose values are limited to [min, max].
ndarray.conj()	Complex-conjugate all elements.
ndarray.round([decimals, out])	Return a with each element rounded to the given number of decimals.
ndarray.trace([offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.
ndarray.sum([axis, dtype, out, keepdims])	Return the sum of the array elements over the given axis.
ndarray.cumsum([axis, dtype, out])	Return the cumulative sum of the elements along the given axis.
ndarray.mean([axis, dtype, out, keepdims])	Returns the average of the array elements along given axis.
ndarray.var([axis, dtype, out, ddof, keepdims])	Returns the variance of the array elements, along given axis.
ndarray.std([axis, dtype, out, ddof, keepdims])	Returns the standard deviation of the array elements along given axis.
ndarray.prod([axis, dtype, out, keepdims])	Return the product of the array elements over the given axis
ndarray.cumprod([axis, dtype, out])	Return the cumulative product of the elements along the given axis.
ndarray.all([axis, out, keepdims])	Returns True if all elements evaluate to True.
ndarray.any([axis, out, keepdims])	Returns True if any of the elements of a evaluate to True.

参考：https://docs.scipy.org/doc/numpy-dev/reference/arrays.ndarray.html#calculation (https://docs.scipy.org/doc/numpy-dev/reference/arrays.ndarray.html#calculation)

4 Universal Function (对每个元素进行操作的函数)

Universal Function(简写为**ufunc)**是一种能对数组的每个元素进行操作的函数。 NumPy内置的许多ufunc函数都是在C语言级别实现的，它们的计算速度非常快。

下面是ufunc `numpy.sin` 的例子：

```
>>> import numpy as np
>>> x = np.linspace(0, 2*np.pi, 9)
>>> y = np.sin(x)                    # numpy.sin 是 ufunc ，它对数组每个元素进行操作
>>> y
array([  0.00000000e+00,   7.07106781e-01,   1.00000000e+00,
         7.07106781e-01,   1.22464680e-16,  -7.07106781e-01,
        -1.00000000e+00,  -7.07106781e-01,  -2.44929360e-16])
```

下面是ufunc `numpy.add` 的例子：

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3],
...               [4, 5, 6]])
>>> y = np.array([[11, 12, 13],
...               [14, 15, 16]])
>>> np.add(x, y)            # numpy.add 是 ufunc
array([[12, 14, 16],
       [18, 20, 22]])
>>> x+y                     # 由于Python的操作符重载功能，np.add(x, y)可以简单地写为x+
array([[12, 14, 16],
       [18, 20, 22]])
```

参考：https://docs.scipy.org/doc/numpy-dev/reference/ufuncs.html (https://docs.scipy.org/doc/numpy-dev/reference/ufuncs.html)

4.1 Broadcasting

当我们使用ufunc函数对两个数组进行计算时，ufunc函数会对这两个数组的对应元素进行计算，因此它要求这两个数组有相同的大小(shape相同)。如果两个数组的shape不同的话，会进行广播(broadcasting)处理。

这里不详细介绍广播规则。下面是一个广播的实例：

```
>>> import numpy as np
>>> a = np.array([[  0.0,   0.0,   0.0],
                  [ 10.0,  10.0,  10.0],
                  [ 20.0,  20.0,  20.0],
                  [ 30.0,  30.0,  30.0]])
>>> b = np.array([1.0, 2.0, 3.0])
>>> a+b                          # a和b的形状不一样，会进行广播(broadcasting)处理
array([[  1.,   2.,   3.],
       [ 11.,  12.,  13.],
       [ 21.,  22.,  23.],
       [ 31.,  32.,  33.]])
```

参考：
http://old.sebug.net/paper/books/scipydoc/numpy_intro.html#id6 (http://old.sebug.net/paper/books/scipydoc/numpy_intro.html#id6)
https://www.tutorialspoint.com/numpy/numpy_broadcasting.htm (https://www.tutorialspoint.com/numpy/numpy_broadcasting.htm)
https://docs.scipy.org/doc/numpy-dev/reference/ufuncs.html#broadcasting (https://docs.scipy.org/doc/numpy-dev/reference/ufuncs.html#broadcasting)

5 Matrix

在NumPy中的 matrix (https://docs.scipy.org/doc/numpy/reference/generated/numpy.matrix.html) 是array的子类，它所以继承了array的所有特性并且有自己独特的地方。比如使用matrix时， * 是矩阵乘法；而使用array时， * 是ufunc（每个对应元素相乘）。

```
>>> import numpy as np
>>> a = np.matrix('1 2; 3 4')      # 创建matrix。注：np.matrix 可以简写为 np.mat
>>> a
matrix([[1, 2],
        [3, 4]])
>>> a.T                            # a的转置矩阵
matrix([[1, 3],
        [2, 4]])
>>> a.I                            # a的逆矩阵
matrix([[-2. ,  1. ],
        [ 1.5, -0.5]])
```

参考：
https://docs.scipy.org/doc/numpy/reference/generated/numpy.matrix.html (https://docs.scipy.org/doc/numpy/reference/generated/numpy.matrix.html)

Author: cig01 (http://www.aandds.com)
Created: <2016-04-29 Fri 00:00>
Last updated: <2017-06-11 Sun 20:14>
Creator: Emacs (http://www.gnu.org/software/emacs/) 25.1.1 (Org (http://orgmode.org) mode 9.0.7)