Python 科学计算包入门

Numpy

Matplotlib

Pandas

Scipy

WxPython

Introduction:

Orgnaize and Translate the official document of Python by TOM

bluedream770@sina.com

目录

第一部分

Numpy 用户指南

第一章　设置 Numpy

1.1　Numpy 是什么？

Numpy 是 Python 科学计算基础包。它以 Python 库的形式提供了多维数组对象，以及多种多样的衍生对象，比如掩码式数组（掩码式数组是指可能含有缺失或者无效项的数组）和矩阵，Numpy 包含丰富多样的用于快速操作数组的程序，比如数学运算，逻辑运算，阵形运算，分类和选取，I/O 操作，离散傅里叶变换，基本的线性代数，基本统计运算以及随机模拟等等。

Numpy 的核心是 ndarray 对象，它封装了 n 维包含同类数据的数组，并且附带许多已经编译好的高性能的运算方法。Numpy 数组和 Python 原生态序列有几点非常重要的区别：

- Numpy 数组在创建的时候就必须指定大小，而 Python 序列可以动态增长或者缩减。改变某个 ndarray 的大小将重新创建一个新的数组，并且删除原有的数组
- Numpy 数组的元素要求是同一类型，因此他们在内存中的大小相同。一个例外是：当 Numpy 的元素是 Python 对象的时候（包括 ndarray）有可能占有的内存不一样大
- Numpy 数组使得高等数学运算以及其他类型的运算变得容易，尤其是数据量巨大的时候。通常情况下，与 Python 原生态的序列类型相比，Numpy 的运算更高效，代码量更少
- 越来越多的基于 Python 的科学计算包用到了 Numpy 数组。虽然这些库支持 Python 序列类型的输入，但是内部实现还是需要将其转换为 Numpy 数组类型，并且输出通常就是 Numpy 数组。换句话说，为了能更高效使用当今大多数 Python 计算包，仅仅知道 Python 原生态的序列类型是不够的，还需要知道如何使用 Numpy 数组

在科学计算中，序列能容纳的大小和运算速度尤其重要。作为一个简单的例子，将一个一维数组序列 a 的每个元素和另外一个一维数组序列 b 相对应的元素相乘，并将计算结果保存在序列 c。

```
c = []
for i in range(len(a)):
c.append(a[i]*b[i])
```

以上将获得正确的结果。但是，如果 a 和 b 都包含上百万的元素，我们需要付出在 Python 中循环上百万次的代价。当然，我们可以直接用 C 语言来完成同样的运算，可以从中获得 C 语言运算速度的优势（为了说明问题，我们在以下 C 语言中忽略变量的申明、初始化、内存分配等）。

```
for (i = 0; i < rows; i++){
c[i] = a[i]*b[i];
}
```

利用 C 语言来实现省去了解释 Python 代码和操作 Python 对象的开销，但 Python 语言的开发速度、维护性等优点却无法从 C 语言中获取。若不用向量的思想，无论 C 语言或者 Python，其代码量都会随着数据维度的增加而成倍增加。想象一下，如果这个例子是两个矩阵的运算，那么采用 C 语言将可能写成这样：

```
for (i = 0; i < rows; i++) {
  for (j = 0; j < columns; j++) {
  c[i][j] = a[i][j]*b[i][j];
  }
}
```

Numpy 将 C 语言的速度和 Python 语言带来的好处都给予了我们。ndarray 在运算的时候默认会采用将内部元素按顺序一个一个进行运算的特性，即“element-by-element”。并且其内部实现运算的方式其实是已经预先编译好的 C 代码，非常高效。在 Numpy 中，

```
c = a * b
```

以上用一行代码就可以做到刚才我们的例子所做的事情，而不用去关心 a 和 b 的维度。Numpy 高效又简介除了得益于 Python 语言的优势和预编译 C 代码的方式外，矢量运算和广播是关键的原因。

矢量运算将循环、索引这些隐藏起来，后台以优化过的 C 代码的速度运行。矢量运算有许多优点，比如：

- 矢量运算的代码更简洁易读
- 更少的代码意味着更少的 Bug
- 代码更接近与标准的数学符号公式，使得更容易将数学公式转化为代码和修改、维护代码
- 矢量运算可以产生更加“Pythonic”[1]的代码。没有矢量运算我们的代码可能一团糟，不容易阅读和维护。

广播描述了以“element-by-element”方式运算过程中遵守的隐含规则。总的来说，Numpy 中所有的运算，包括数学运算，逻辑运算，位运算，功能运算等都遵守这个隐含规则。在上一个例子中，a 和 b 可以是有着相同 shape[2]的多维数组，或者是一个标量和一个数组，甚至是两个不同 shape 的多维数组，结果是小一点的数组会“扩展”到和大一点的数组一样大，然后再进行相关运算。广播的规则描述的就是什么时候应该“扩展”和如何“扩展”。只要遵守广播规则，最终计算结果不仅是唯一的，而且是有物理意义的。具体详细信息可以参考Numpy 广播。

1.2　安装 Numpy

大多数情况下，安装 Numpy 最好的方式都是用与你操作系统相对应的已经编译好的软件包。可以参考http://scipy.org/install.html查询可选项。若已经安装好 pip[3]，推荐直接通过命令

```
pip install numpy
```

来单独安装 Numpy。不过更推荐的方式是安装 Python 发行版如anaconda，它集成了 Python 环境、Numpy、Pandas、Scipy、Matplotlib 等基于 Python 的科学计算包，并且截止 2017 年 10 月 27 日其 Python 版本已经更新到了 3.6。

用户也可以从源码安装，请参考Numpy 源码安装。这类信息主要是为高级用户准备的。

[1] Python 下输入“import this”，会告知“Pythonic”的具体含义

[2] 一个元祖，比如 shape 为（2,3）的数组表示两行三列的矩阵

[3] windows 下安装好 Python 后在 Script 文件夹可以找到 pip

第二章　Numpy 快速入门

2.1　先决条件

在阅读本章之前，你需要对 Python 有一定的了解。如果你想回顾一下 Python，可以参看Python 教程。中文教程可以参考廖雪峰老师的教程：廖雪峰 Python3 教程。该教程通俗易懂，非常适合入门；教程最后的实战部分有一定难度，可以视情况而做。

如果你想运行本教程的示例，你必须预先安装一些必要的软件。详情请见`http://scipy.org/install.html`。也可以直接安装 Python 发行版anaconda。

本书的运行环境是 **Python 3.6.1 |Anaconda 4.4.0 (64-bit)**。

2.2　基本知识

Numpy 中最主要的对象是 ndarray，即 n 维数组。它封装了 n 维同类数据以及对这些数据的操作函数。他将相同的元素（一般来说都是一些数字）组合成一张表，用一个元祖来表示元素的索引。在 Numpy 中，维数被称为 axes，即“轴”；轴的数量叫做 rank，即“秩”。在线性代数中，一个矩阵 A 的列秩是 A 的线性独立的纵列的极大数。若 A 的秩为 n，则 A 的列向量可以表示 n 维空间。

举个例子，3D 坐标 [1,2,1] 是一个 rank 为 1，axes 长度是 3 的数组，他只有一维。接下来的例子中，rank 为 2（它是二维的）。第一维的长度是 2，第二维的长度是 3。

```
[[ 1., 0., 0.],
[ 0., 1., 2.]]
```

Numpy 的数组类叫做 ndarray，通常也可以直接称作 array。注意，numpy.array 和 Python 标准的 class 类 array.array 是有区别的。array.array 仅仅只能处理一维数组，且提供的功能函数少很多。ndarray 中重要的属性有：

- **ndarray.ndim** 数组的维数，即秩
- **ndarray.shape** 数组的组成方式。以一个元祖的形式来表示每一个维度的大小。如果一个矩阵有 m 行 n 列，则 shape 可以表示成 (m, n)。shape 的元祖长度实际上就是 ndim，本例中 ndim=2
- **ndarray.size** 数组包含元素个数的总量。size 值等于 shape 元祖里各个维度的乘积。size 值并不表示数据占有的内存的大小，还需要知晓每个元素占有的内存大小
- **ndarray.dtype** dtype 用于描述数组元素的类型。dtype 既支持创建新的 type，也支持 Python 标准的 type。当然，Numpy 也提供了自己的 type，比如 numpy.int32, numpy.int16, and numpy.float64
- **ndarray.itemsize** 数组中每个元素的大小。比如，若数组中的元素的 dtype 是 float64，则它的 itemsize 是 8（=64/8），若数组中元素的 dtype 是 comlex32，则它的 itemsize 是 4（=32/8）。ndarray.itemsize 与 ndarray.dtype.itemsize 值相等

- **ndarray.data** 数组中包含元素数据的 buffer。一般来说我们没有必要直接用这个属性，因为我们一般是通过 index 来访问数据元素

总结一下，一个 ndarray 类包含数据和处理数据的方法。数据中几个重要的属性是 ndim、shape、size、dtype、itemsize、data。其中，shape 可以推导出 ndim。shape+dtype 可以推导出除了 data 以外的其他全部属性。而 data 更接近底层访问数据的接口，可以用于和 C 语言等低级语言对接。

2.2.1 一个例子

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int32'
>>> a.itemsize
4
>>> a.size
15
>>> type(a)
<class 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<class 'numpy.ndarray'>
>>> b.data
<memory at 0x0000000003494948>
```

2.2.2 Numpy 数组的创建

有好几种创建 Numpy 数组的方法。首先，你可以用 Numpy 的 array 函数来创建一个 Numpy 数组，只需要给这个函数传入一个 Python 列表或者元祖作为参数。数组元素的 dtype 将从 Python 序列元祖的 type 中推导出来[1]。

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int32')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

array 函数正确的用法应该是传入一个 Python 序列。一个经常犯的错误是直接给 array 函数出入多个数字作为参数，比如：

[1]若 Python 序列中有字符串和数字，经过推导后 Numpy 会统一成字符串类型

```
>>> b = np.array([6, 7, 8])    #RIGHT
>>> b = np.array(6, 7, 8)      #WRONG
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: only 2 non-keyword arguments accepted
```

若传入的参数是序列的序列，则 array 会将其转换为二维数组，以此类推，序列的序列的序列将被转换为三维数组...

```
>>> b = np.array([(1.5,2,3), (4,5,6)])
>>> c = np.array([((1,2),(3,4)), ((5,6),(7,8))])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
>>> c
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

数组元素的 dtype 也可以在创建数组的时候显式地指出：

```
>>> d = np.array( [ [1,2], [3,4] ], dtype=complex )
>>> d
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

通常情况下，数组的元素的值刚开始是不确定的，但数组的大小却能定下来。因此，Numpy 提供了一些可以创建固定大小数组的函数。数组元素的值可以用特定值填充占位或者直接使用随机值。这样的做法可以减少一些对于增长数组的需求，毕竟维护一个可增长数组是一个不小的开销。

numpy.zero 和 numpy.ones 分别创建一个元素值全为 0 和 1 的数组，numpy.empty 创建一个元素值随机的数组，元素的值根据当时的内存状态而视，不一定为 0。采用这些方法创建数组的时候都可以指定 dtype。若不指定 dtype，则 dtype 默认是 float64。

```
>>> a=np.zeros( (3,4) )
>>> a
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> a.dtype
dtype('float64')
>>> b=np.ones( (2,3,4), dtype=np.int16 )
>>> b
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int16)
>>> np.empty( (2,3) )
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

为了创建一个数字序列，numpy 提供了一个和 Python 内建函数 range 相类似的函数，即 numpy.arange，该方法返回一个一维数组。

```
>>> np.arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> np.arange( 0, 2, 0.3 ) # it accepts float arguments
array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
```

当给 arange 函数传入一个浮点数参数的时候，由于浮点数的精度是有限的，因此无法准确得知创建数组的元素个数。出于以上原因，numpy 提供了一个更好用的函数 numpy.linspace。linspace 的用法和 Matlab 中的 linspace 用法基本类似，其两个参数的意义和 arange 一样，第三个参数表示元素的个数。它会根据元素的个数自动推导元素间的 step。另外需要注意的是，arange 函数的第二个参数不是闭区间，而 linspace 是闭区间。比如 linspace(-5,5,1) 最终产生的数组就包含了“5”这个点，而 arange 不会，需要注意其区别。

```
>>> from numpy import pi
>>> np.arange( 0, 2, 0.3 ) # it accepts float arguments
array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
>>> np.arange(10,30,5)
array([10, 15, 20, 25])
>>> from numpy import pi
>>> np.linspace(0,2,5)
array([ 0. ,  0.5,  1. ,  1.5,  2. ])
>>> np.arange(0,2,0.5)
array([ 0. ,  0.5,  1. ,  1.5])
>>> x=np.linspace(0, 2*pi, 10)
>>> x
array([ 0.        ,  0.6981317 ,  1.3962634 ,  2.0943951 ,  2.7925268 ,
        3.4906585 ,  4.1887902 ,  4.88692191,  5.58505361,  6.28318531])
```

另外还可以根据已知的数组的 shape 依样画葫芦创建一个新的数组。涉及到的函数有 zeros_like、empty_like、ones_like 等。下面的示例中，a 是 shape 为 (3,2) 的数组，通过 ones_like 创建一个 shape 为（3,2），元素值全 1 数组 b。

```
>>> a=np.linspace(1. ,6. ,6).reshape(3,2)
>>> a
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
>>> b=np.ones_like(a)
>>> b
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]])
```

总结一下，如表2.1所示，numpy 提供了大量的数组创建函数。创建数组的关键在于告知数组的形状，元素的类型，元素的值。元素的 dtype 可以不用显式给出，numpy 可以自主推导合适的 dtype。元素的值也可以不用暂时不用指定，用“占位”的方式来减少对动态增长数组的需求。

2.2.3 print 数组

当你 print 一个数组的时候，numpy 会采用和打印 Python 嵌入式列表相类似的方式打印，为了方便查看，Numpy 打印时添加了以下特征：

- 排在最后的 axis 打印方向是从左到右

Function	Instruction
array	将输入数据（列表、元组、数组或者其他序列）转换为 ndarray。推断出 dtype 或显式指定 dtype。默认直接复制输入数据
asarray	将输入转换为 ndarray，若输入本身就是一个 ndarray 就不复制
arange	类似内置的 range，但返回的是一个 ndarray 而不是列表
ones、ones_like	根据指定的形状和 dtype 创建一个全 1 的数组。ones_like 以另一个数组为参数，并根据其形状和 dtype 创建一个全 1 数组
zeros、zeros_like	类似于 ones 和 ones_like，只不过产生的是全 0 数组而已
empty、empty_like	创建新数组，只分配内存空间但不填充任何值
eye、identity	创建一个正方的 $n \times n$ 单位矩阵（对角线为 1，其余为 0）

表 2.1: 数组创建函数

- 排在倒数第二的 axis 打印方向是从上到下
- 其他 axis 打印方向是从上到下，但是会单独用一个空行来将不同的片区分开来，方便查看

一维数组打印方式和列表一样，二维数组打印方式如矩阵形式，三维数组打印方式如矩阵的列表。

第二部分

Pandas 用户指南

第三部分

Scipy 用户指南

第四部分

Matplotlib 用户指南

第五部分

WxPython 用户指南