**[python数据分析入门](https://www.cnblogs.com/baijinshuo/p/10485902.html)**

什么是数据分析?

数据分析:就是把隐藏在一些看似杂乱无章的数据背后的信息提炼出来,总结出所研究对象的内在规律.

数据分析三剑客:Numpy,Pandas,Matplotlib

**NumPy简介**

NumPy(Numerical python)是python语言的一个扩展程序库,支持大量的维度数组与矩阵运算,此外也针对数组运算提供大量的数学函数库.

NumPy的前身Numeric 最早是由 Jim Hugunin 与其它协作者共同开发，2005 年，Travis Oliphant 在 Numeric 中结合了另一个同性质的程序库 Numarray 的特色，并加入了其它扩展而开发了 NumPy。NumPy 为开放源代码并且由许多协作者共同维护开发。

NumPy是一个运行速度非常快的数学库,主要用于数组计算,包含:

- 一个强大的N维数组对象ndarray

- 广播功能函数

- 整合 c/c++/Fortran 代码的工具

- 线性代数, 傅里叶变换、随机数生成等功能

**NumPy应用**

NumPy 通常与 SciPy（Scientific Python）和 Matplotlib（绘图库）一起使用， 这种组合广泛用于替代 MatLab，是一个强大的科学计算环境，有助于我们通过 Python 学习数据科学或者机器学习。

SciPy 是一个开源的 Python 算法库和数学工具包。

SciPy 包含的模块有最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解和其他科学与工程中常用的计算。

Matplotlib 是 Python 编程语言及其数值数学扩展包 NumPy 的可视化操作界面。它为利用通用的图形用户界面工具包，如 Tkinter, wxPython, Qt 或 GTK+ 向应用程序嵌入式绘图提供了应用程序接口（API）。

**入门**

NumPy的主要对象是同构多维数组.他是一个元素表(通常是数字),都是相同的类型,有正整数元组索引,在NumPy维度中成为轴.

例如,3D空间中的点的坐标[1,2,1]具有一个轴,该轴有三个元素,所以说它的长度为3,在下面所示的例子中,数组有两个轴,第一个轴长度为2,第二个轴的长度为3.

[[ 1., 0., 0.],

[ 0., 1., 2.]]

NumPy的数组类被调用ndarray。它也被别名所知 array。请注意，numpy.array这与标准Python库类不同array.array，后者只处理一维数组并提供较少的功能。ndarray对象更重要的属性是：

ndarray.ndim --- 数组的轴数(尺寸)

ndarray.shape --- 数组的大小,这是一个整数元组,表示每个维度中数组的大小,对于具有n行和m列的矩阵,shape将是(n,m).shape因此元组的长度是轴的数量ndim

ndarray.size --- 数组的元素总数。这等于元素的乘积shape

ndarray.itemsize --- 数组中每个元素的大小（以字节为单位）。例如，类型的元素数组float64有itemsize8（= 64/8），而其中一个类型complex32有itemsize4（= 32/8）。它相当于ndarray.dtype.itemsize。

ndarray.data --- 包含数组实际元素的缓冲区。通常，我们不需要使用此属性，因为我们将使用索引工具访问数组中的元素。

例子:

>>> import numpy as np

>>> a = np.arange(15).reshape(3, 5) # 创建一个二维数组

>>> a

array([[ 0, 1, 2, 3, 4],

[ 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14]])

>>> a.shape # 查看每个维度中数组的大小

(3, 5) # 3行5列

>>> a.ndim # 数组的轴数,二维数组

2

>>> a.dtype.name # 描述数组中元素类型的对象

'int64'

>>> a.itemsize # 数组中每个元素的大小为8字节

8

>>> a.size # 数组的元素总数

15

>>> type(a)

<type 'numpy.ndarray'>

>>> b = np.array([6, 7, 8])

>>> b

array([6, 7, 8]) # 创建一个数组

>>> type(b)

<type 'numpy.ndarray'>  
>>> np.array([[1,2,3],['a','b',1.1]]) # 创建一个二维数组

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

['a', 'b', '1.1']], dtype='<U11')

**数组的创建**

有以下集中方法可以创建数组.

你可以使用array函数从常规python列表或元组创建数组,结果数组的类型是从序列中元素的类型推导出来的

[复制代码](javascript:void(0);)

>>> import numpy as np

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

>>> a

array([2, 3, 4])

>>> a.dtype

dtype('int64')

>>> b = np.array([1.2, 3.5, 5.1])

>>> b.dtype

dtype('float64')

[复制代码](javascript:void(0);)

 常见的错误在于array使用多个数字参数调用,而不是提供单个数字列表作为参数.

>>> a = np.array(1,2,3,4) # 错误的

>>> a = np.array([1,2,3,4]) # 正确的

 array将两个序列转换成二维阵列,将序列转换成三维阵列.等.

>>> b = np.array([(1.5,2,3), (4,5,6)])

>>> b

array([[ 1.5, 2. , 3. ],

[ 4. , 5. , 6. ]])

**注意:**

　　- numpy默认ndarray的所有元素的类型是相同的

       - 如果传进来的列表中包含不同的类型,则统一为统一类型,优先级 str > float > int

也可以在创建时显式指定数组类型:

>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )

>>> c

array([[ 1.+0.j, 2.+0.j],

[ 3.+0.j, 4.+0.j]])

 通常数组的元素最初是未知的,但其大小是已知的.因此,NumPy提供了几个函数来创建具有初始占位符内容的数组.这些最小化了增加举证的必要性,这是一项昂贵的操作,

函数**zeros**创建一个充满零的数组,

函数**ones**创建一个完整的数组,

函数**empty**创建一个数组,其初始内容是随机的,取决于内存的状态.默认情况下,创建的数组的dtype是float64.

[复制代码](javascript:void(0);)

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

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

[ 0., 0., 0., 0.],

[ 0., 0., 0., 0.]])

>>> np.ones( (2,3,4), dtype=np.int16 ) # dtype can also be specified

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) ) # uninitialized, output may vary

array([[ 3.73603959e-262, 6.02658058e-154, 6.55490914e-260],

[ 5.30498948e-313, 3.14673309e-307, 1.00000000e+000]])

[复制代码](javascript:void(0);)

为了创建数字序列,NumPy提供了一个类似于range返回数组而不是列表的函数

[复制代码](javascript:void(0);)

>>> 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])  
  
第一个参数开始数字,第二个是结束数字,第三个类似步长

[复制代码](javascript:void(0);)

 当arange与浮点参数一起使用时，由于有限的浮点精度，通常不可能预测所获得的元素的数量。出于这个原因，通常最好使用linspace作为参数接收我们想要的元素数量的函数，而不是步骤：

[复制代码](javascript:void(0);)

>>> from numpy import pi

>>> np.linspace( 0, 2, 9 ) # 9 numbers from 0 to 2

array([ 0. , 0.25, 0.5 , 0.75, 1. , 1.25, 1.5 , 1.75, 2. ])

>>> x = np.linspace( 0, 2\*pi, 100 ) # useful to evaluate function at lots of points

>>> f = np.sin(x)

[复制代码](javascript:void(0);)

**数组的打印**

当您打印数组时，NumPy以与嵌套列表类似的方式显示它，但具有以下布局：

* 最后一个轴从左到右打印，
* 倒数第二个是从上到下打印的，
* 其余部分也从上到下打印，每个切片用空行分隔。

然后将一维数组打印为行，将二维数据打印为矩阵，将三维数据打印为矩阵列表。

[复制代码](javascript:void(0);)

>>> a = np.arange(6) # 然后将一维数组打印为行

>>> print(a)

[0 1 2 3 4 5]

>>>

>>> b = np.arange(12).reshape(4,3) # 将二维数据打印为矩阵

>>> print(b)

[[ 0 1 2]

[ 3 4 5]

[ 6 7 8]

[ 9 10 11]]

>>>

>>> c = np.arange(24).reshape(2,3,4) # 将三维数据打印为矩阵列表

>>> print(c)

[[[ 0 1 2 3]

[ 4 5 6 7]

[ 8 9 10 11]]

[[12 13 14 15]

[16 17 18 19]

[20 21 22 23]]]

[复制代码](javascript:void(0);)

 如果数组太大而无法打印，NumPy会自动跳过数组的中心部分并仅打印角落：

[复制代码](javascript:void(0);)

>>> print(np.arange(10000))

[ 0 1 2 ..., 9997 9998 9999]

>>>

>>> print(np.arange(10000).reshape(100,100))

[[ 0 1 2 ..., 97 98 99]

[ 100 101 102 ..., 197 198 199]

[ 200 201 202 ..., 297 298 299]

...,

[9700 9701 9702 ..., 9797 9798 9799]

[9800 9801 9802 ..., 9897 9898 9899]

[9900 9901 9902 ..., 9997 9998 9999]]

[复制代码](javascript:void(0);)

 要禁用此行为并强制NumPy打印整个阵列，可以使用更改打印选项set\_printoptions。

>>> np.set\_printoptions(threshold=np.nan)

**基本操作**

数组上的算术运算符应用于*元素*。创建一个新数组并填充结果。

[复制代码](javascript:void(0);)

>>> a = np.array( [20,30,40,50] )

>>> b = np.arange( 4 )

>>> b

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

>>> c = a-b

>>> c

array([20, 29, 38, 47])

>>> b\*\*2

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

>>> 10\*np.sin(a)

array([ 9.12945251, -9.88031624, 7.4511316 , -2.62374854])

>>> a<35

array([ True, True, False, False])

[复制代码](javascript:void(0);)

 与许多矩阵语言不同，产品运算符\*在NumPy数组中以元素方式运行。矩阵乘积可以使用@运算符（在python> = 3.5中）或dot函数或方法执行：

[复制代码](javascript:void(0);)

>>> A = np.array( [[1,1],

... [0,1]] )

>>> B = np.array( [[2,0],

... [3,4]] )

>>> A \* B # elementwise product

array([[2, 0],

[0, 4]])

>>> A @ B # matrix product

array([[5, 4],

[3, 4]])

>>> A.dot(B) # another matrix product

array([[5, 4],

[3, 4]])

[复制代码](javascript:void(0);)

 某些操作（例如+=和）\*=用于修改现有阵列而不是创建新阵列。

[复制代码](javascript:void(0);)

>>> a = np.ones((2,3), dtype=int)

>>> b = np.random.random((2,3))

>>> a \*= 3

>>> a

array([[3, 3, 3],

[3, 3, 3]])

>>> b += a

>>> b

array([[ 3.417022 , 3.72032449, 3.00011437],

[ 3.30233257, 3.14675589, 3.09233859]])

>>> a += b # b is not automatically converted to integer type

Traceback (most recent call last):

...

TypeError: Cannot cast ufunc add output from dtype('float64') to dtype('int64') with casting rule 'same\_kind'

[复制代码](javascript:void(0);)

 当使用不同类型的数组进行操作时，结果数组的类型对应于更一般或更精确的数组（称为向上转换的行为）

[复制代码](javascript:void(0);)

>>> a = np.ones(3, dtype=np.int32)

>>> b = np.linspace(0,pi,3)

>>> b.dtype.name

'float64'

>>> c = a+b

>>> c

array([ 1. , 2.57079633, 4.14159265])

>>> c.dtype.name

'float64'

>>> d = np.exp(c\*1j)

>>> d

array([ 0.54030231+0.84147098j, -0.84147098+0.54030231j,

-0.54030231-0.84147098j])

>>> d.dtype.name

'complex128'

[复制代码](javascript:void(0);)

 许多一元操作，例如计算数组中所有元素的总和，都是作为ndarray类的方法实现的。

[复制代码](javascript:void(0);)

>>> a = np.random.random((2,3))

>>> a

array([[ 0.18626021, 0.34556073, 0.39676747],

[ 0.53881673, 0.41919451, 0.6852195 ]])

>>> a.sum()

2.5718191614547998

>>> a.min()

0.1862602113776709

>>> a.max()

0.6852195003967595

[复制代码](javascript:void(0);)

 默认情况下，这些操作适用于数组，就像它是一个数字列表一样，无论其形状如何。但是，通过指定axis 参数，您可以沿数组的指定轴应用操作：

[复制代码](javascript:void(0);)

>>> b = np.arange(12).reshape(3,4)

>>> b

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

[ 4, 5, 6, 7],

[ 8, 9, 10, 11]])

>>>

>>> b.sum(axis=0) # 列的总和

array([12, 15, 18, 21])

>>>

>>> b.min(axis=1) # 行的总和

array([0, 4, 8])

>>>

>>> b.cumsum(axis=1) # cumulative sum along each row

array([[ 0, 1, 3, 6],

[ 4, 9, 15, 22],

[ 8, 17, 27, 38]])

[复制代码](javascript:void(0);)