Numpy

1.基本操作

1.1数组转换

创建数组的最简单的方法就是使用array函数，将Python下的list转换为ndarray

```
#通过数组创建一个ndarray
data1 = [6,7.5,8,0,1]
arr1 = np.array(data1)
arr1


#输出为:
array([6,7.5,8,0,1])
```

创建二维数组

```
#通过数组创建一个二维的ndarray
data2 = [[1,2,3,4],[5,6,7,8]]
arr2 = np.array(data2)
arr2


#输出为:
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

重新定义矩阵的形状
array.reshape((n,m))

1.2数组生成

> 除了通过数组转换而来之外，我们可以利用np中的一些内置函数来创建数组，比如我们创建全0的数组，也可以创建全1数组，或者等差数列数组

创建全0数组

```
np.zeros(10)


#输出为:
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

创建全1数组

```
np.ones(10)


#输出为:
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

创建元素接近空的数组

```
np.empty((3,4))


#输出为:
array([[4.9e-324, 4.9e-324, 9.9e-324, 9.9e-324],
       [9.9e-324, 9.9e-324, 9.9e-324, 1.5e-323],
       [2.0e-323, 2.0e-323, 2.5e-323, 2.5e-323]])
注：创建初始是随机数，而不是空，需要重新赋值
```

创建等差数组

```
np.arange(1,15,2)


#输出为:
array([ 1,  3,  5,  7,  9, 11, 13])
```

创建正态分布随机数组

```
samples = np.random.normal(2,3,size=(4,4))
samples


#输出为:
array([[ 6.15721917,  3.42393052,  0.65418836, -0.6927128 ],
       [ 2.79557529,  1.1205289 ,  1.63283966,  6.18085827],
       [-1.02241948, -2.77444011, -0.7293639 ,  1.9849088 ],
       [ 2.52981105,  1.80517346, -1.98852316,  1.71483645]])
```

创建正态分布数组

```
samples = np.random.normal(4,4)
samples


#输出为:
array([[-0.96184087,  0.86026662,  1.13674982,  2.74464916],
       [ 0.14419425, -0.57185231,  0.61601683, -0.18976333],
       [-0.25594082, -1.84514383,  0.54483433,  1.77408903],
       [-0.16996494,  0.18802037,  1.54856742,  0.18296107]])
```

randint生成随机整数数组

```
samples = np.random.randint(0,10,size=(3,4))
samples
```

```
#输出为:
array([[6, 7, 2, 2],
       [1, 3, 5, 6],
       [5, 8, 4, 9]])
```

1.3文件读取

save方法保存ndarray到一个npy文件，也可以使用savez将多个array保存到一个.npz文件中：

```
x = np.array([1,2,4,5])
y = np.array([3,4,5])
#save方法可以存取一个ndarray
np.save("x_arr",x)
#如果要存取多个数组，要是用savez方法
np.savez("some_array.npz",xarr = x,yarr=y)
```

load方法来读取存储的数组，如果是.npz文件的话，读取之后相当于形成了一个k-v类型的变量，通过保存时定义的key来获取相应的array。

```
np.load('x_arr.npy')
#array([1, 2, 4, 5])
arch = np.load("some_array.npz")
arch['yarr']
#array([3, 4, 5])
```

np.loadtxt 和 np.savetxt可以用来存取txt或csv文件

```
arr=[[6, 7, 2, 2],
    [1, 3, 5, 6],
    [5, 8, 4, 9]]
#储存数组到txt文件
np.savetxt("array_ex.txt",arr)
#读取txt文件，delimiter为分隔符，dtype为数据类型
np.loadtxt("array_ex.txt",delimiter=" ",dtype=np.int32)
```

1.4查看操作

查看维度

```
array.ndim
```

查看形状

```
array.shape
```

查看元素个数

```
array.size
```

2.数据类型

> ndarray的数据类型：
>
> - int: int8、int16、int32、int、64
> - float: float16、float32、float64
> - string

2.1指定数据类型：

```
#指定array的数据类型
arr1 = np.array([1,2,3],dtype=np.int32)
arr2 = np.array([1,2,3],dtype=np.float32)
```

输出为：

```
#arr1
array([1, 2, 3], dtype=int32)
#arr2
array([1., 2., 3.], dtype=float32)
```

2.2查看数据类型

```
#查看array的数据类型
arr2.dtype
### dtype('float32')
```

2.3数据类型转换

使用astype将一个数组的数据类型进行转换，这样会返回一个新的数组，对原数组不会产生影响

```
#数据类型进行转换，会产生一个新的array，原array不产生影响
arr1.astype(np.float32)
arr1.dtype
# dtype('int32')
```

如果一个数组中的字符串只含有数字，可以将string转换为数值形式：

```
numeric_strings = np.array(['1.25','0.96','42'],dtype=np.string_)
numeric_strings.astype(np.float32)
# array([  1.25      ,   0.95999998,  42.        ], dtype=float32)
```

3.数组运算

3.1数组间运算

大小相等的数组之间的任何算数运算都会应用到元素身上

```
arr = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
arr * arr

#array([[  1.,   4.,   9.],
#       [ 16.,  25.,  36.]], dtype=float32)

arr - arr

#array([[ 0.,  0.,  0.],
 #      [ 0.,  0.,  0.]], dtype=float32)
```

不同维度数组间的计算，低维数组每个元素分别与高维数组计算

```
a = np.array([[1,2,3,4],[6,7,8,9]])
b = np.arange(1,5)
print(a+b)
#       array([[ 2,  4,  6,  8],
       [ 7,  9, 11, 13]])

print(a*b)
#       array([[ 2,  4,  6,  8],
        [ 7,  9, 11, 13]])

print(a/b)
#       array([[ 2,  4,  6,  8],
       [ 7,  9, 11, 13]])
```

3.2数组与标量

数组与标量的算术运算也会将标量值传播到各个元素：

```
1 / arr

#array([[ 1.        ,  0.5       ,  0.33333334],
#       [ 0.25      ,  0.2       ,  0.16666667]], dtype=float32)
```

4.索引和切片

4.1基本索引和切片

numpy中数组切片是原始数组的视图，这意味着数据不会被复制，视图上任何数据的修改都会反映到原数组上。

```
arr = np.arange(10)
arr[5]
# 5
arr[5:8]
#array([5, 6, 7])
arr[5:8]=12 #切片赋值会赋值到每个元素上，与列表操作不同
t = arr[5:8]
t[1] = 12345
arr
#array([    0,     1,     2,     3,     4,    12, 12345,    12,     8,     9])
```

使用copy方法,可以看到使用copy之后再修改数据不会影响到原数据:

```
t1 = arr[5:8].copy()
t1[2] = -222
arr

#array([ 0,  1,  2,  3,  4, 64, 64, 64,  8,  9])
```

对于二维数组或者高维数组，我们可以按照之前的知识来索引，当然也可以传入一个以逗号隔开的索引列表来选区单个或多个元素

```
arr2d = np.array([[1,2,3],[4,5,6],[7,8,9]])
arr2d[0,2] #（等于arr2d[0][2]）
#3

arr2d[:2,1:] #等于arr2d[:2][:,1:]
#array([[2, 3],
#       [5, 6]])
```

4.2布尔型索引

```
names = np.array(['Bob','Joe','Will','Bob','Will','Joe','Joe'])
data = np.random.randn(7,4)

names == 'Bob'
#array([ True, False, False,  True, False, False, False], dtype=bool)

data[names=='Bob']
#array([[ 0.75323688,  0.85622553, -2.71974541,  0.37865467],
#       [ 1.35356641,  0.09263267,  1.96207471, -0.05549953]])
```

布尔型索引逻辑运算，“|” 表示或 “&”表示 与

```
data[(names=='Bob')| (names=='Will')]

#array([[ 0.75323688,  0.85622553, -2.71974541,  0.37865467],
       [ 0.93720776, -1.49360063, -0.06471438,  0.62149438],
       [ 1.35356641,  0.09263267,  1.96207471, -0.05549953],
       [ 1.87344915,  1.75085643,  1.9197879 ,  0.47687361]])
```

4.3花式索引

花式索引的方式，它指利用整数数组进行索引,花式索引和切片不一样，它总是将数据复制到新数组中

```
arr = np.empty((8,4))
for i in range(8):
    arr[i] = i
arr[[4,3,0,6]]
```

输出为：

```
array([[ 4.,  4.,  4.,  4.],
       [ 3.,  3.,  3.,  3.],
       [ 0.,  0.,  0.,  0.],
       [ 6.,  6.,  6.,  6.]])
```

选择一块方形区域，同时按照我们指定的顺序排列数据，我们尝试以下的方式：

```
arr = np.arange(32).reshape((8,4))
arr[[1,5,7,2],[0,3,1,2]]
```

输出为：

```
array([ 4, 23, 29, 10])
```

这是因为按照上面的方式进行选取，会将选择出的元素锁定在4个元素上。
正确的方式有下面两种：

```
arr[[1,5,7,2]][:,[0,3,1,2]]
arr[np.ix_([1,5,7,2],[0,3,1,2])]
```

输出为：

```
array([[ 4,  7,  5,  6],
       [20, 23, 21, 22],
       [28, 31, 29, 30],
       [ 8, 11,  9, 10]])
```

5.数组转置和轴对换

转置T属性：

```
arr = np.arange(15).reshape((5,3))
arr.T

#array([[ 0,  3,  6,  9, 12],
       [ 1,  4,  7, 10, 13],
       [ 2,  5,  8, 11, 14]])
```

对于高维数组，tranpose需要得到一个由轴编号组成的元组才能对这些轴进行转置，太费脑子:

```
arr.transpose((1,0,2))
#array([[[ 0,  1,  2,  3],
        [ 8,  9, 10, 11]],

       [[ 4,  5,  6,  7],
        [12, 13, 14, 15]]])
```

6.数组函数

6.1通用函数：元素级数字函数

> - 一元函数:
>   abs 绝对值
>   sqrt 开方
>   square 平方根
>   exp e的幂次方
>   log 对数函数
>   sin/cos/tan 三角函数

> - 二元函数：
>   maxinmum 最大值
>   minimum 最小值

```
arr = np.arange(10)
np.sqrt(arr)

x = np.random.randn(8)
y = np.random.randn(8)
np.maximum(x,y)
#array([ 0.68417031,  0.22971426,  1.69724546,  1.19366822, -0.79176777, -0.43557768,  0.66628223,
```

6.2where函数

where 函数，三个参数，条件为真时选择值的数组，条件为假时选择值的数组：

```
xarr = np.array([1.1,1.2,1.3,1.4,1.5])
yarr = np.array([2.1,2.2,2.3,2.4,2.5])
cond = np.array([True,False,True,True,False])
np.where(cond,xarr,yarr)
```

输出为：

```
array([ 1.1,  2.2,  1.3,  1.4,  2.5])
```

也可以使用下面的形式，后两个参数为指定值：

```
np.where(xarr>1.2,2,-2)
#array([-2, -2,  2,  2,  2])
```

6.3数学和统计方法

数学和统计方法既可以当作数组的实例方法调用，也可以当作numpy函数调用,比如下面两种计算数组均值的方法是等效的：

```
arr = np.random.randn(5,4)
arr.mean()
np.mean(arr)
```

mean,sum,max,min这一类函数可以接受一个axis参数，用于计算该轴向上的统计值，最终结果是一个少一维的数组。对于一个二维数组，axis=0相当于按列操作，最终元素的个数和第二维的大小相同，axis=1相当于按行操作，最终元素的个数和第一维的大小相同：

```
arr.mean(axis=1)
#array([ 0.29250253, -0.50119163,  0.11746254,  0.23338843,  0.15912472])

arr.sum(0)
#array([ 1.92728592,  0.67480797, -2.8398905 ,  1.44294295])
```

我们也可以用cumsum(累加值计算）和cumprod（累积值计算）保留中间计算结果：

```
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
arr.cumsum(0)

#array([[ 1,  2,  3],
       [ 5,  7,  9],
       [12, 15, 18]])

arr.cumprod(1)
#array([[  1,   2,   6],
       [  4,  20, 120],
       [  7,  56, 504]])
```

6.4排序方法

np中还提供了排序方法，排序方法是就地排序，即直接改变原数组：

```
arr = np.random.randn(8)
arr
#array([-0.85668922, -2.0049649 , -0.89885165, -0.04185277,  0.73736138,-0.03509021, -1.89745107,

arr.sort()
arr
#array([-2.36576122, -2.0049649 , -1.89745107, -0.89885165, -0.85668922,-0.04185277, -0.03509021,
```

6.5集合运算函数

unique计算x中的唯一元素，并返回有序结果

```
arr = np.array([1,3,2,5,2,4,2,2,1,4,5,2])
np.unique(arr)
#array([1, 2, 3, 4, 5])
```

numpy提供了下面三个常见的集合运算函数：
intersect1d(x,y) 用于计算x和y的公共结果，并返回有序结果
union1d(x,y) 用于计算x和y的并集，并返回有序结果
setdiff1d(x,y),集合的差，即元素在x中不在y中

```
x = np.array([1,2,4,5])
y = np.array([3,4,5])
np.intersect1d(x,y)
#array([4, 5])
np.union1d(x,y)
#array([1, 2, 3, 4, 5])
np.setdiff1d(x,y)
#array([1, 2])
```

线性代数

矩阵的乘积

```
#矩阵的乘积
x = np.array([[1,2,3],[4,5,6]])
y = np.array([[6,23],[-1,7],[8,9]])
np.dot(x,y)
```

下面可以计算矩阵的逆、行列式、特征值和特征向量、qr分解值，svd分解值：

```
#计算矩阵的逆
from numpy.linalg import inv,det,eig,qr,svd
t = np.array([[1,2,3],[2,3,4],[4,5,6]])
inv(t)

#计算矩阵行列式
det(t)

#计算QR分解址
qr(t)

#计算奇异值分解值svd
svd(t)

#计算特征值和特征向量
eig(t)
```