学习Numpy

Numpy介绍与创建

1. numpy内置了并行运算功能，当系统有多个核心时，做某种计算时，numpy会自动做并行计算。
2. Numpy底层使用C语言编写，内部解释器（全局解释器锁），其对数组的操作速度不受Python解释器的限制，效率远高于纯Python代码。
3. 有一个强大的N维数组对象Array（一种类似于列表的东西）。

安装

```
通过pip install numpy即可安装,
通过pip install jupyter即可安装,
```

Numpy数组和Python列表性能对比：

```
import time
import numpy as np
start_time = time.time()
a = []
for x in range(100000):
    a.append(x**2)
end_time = time.time()
print("%.6f"%float(end_time-start_time))# 0.011967897415161133

start_time = time.time()
a = np.arange(100000)**2
end_time = time.time()
print("%.6f"%float(end_time-start_time))

```

numpy中的数组：

1. 创建数组(np.array对象)

Python

```
#%%
# 1. 创建数组(np.array对象)
import  numpy as np
a1 = [1,2,3,'4']
print(a1)
a2 = np.array([1,2,3,'4'])
print(a2)
type(a2)
#%%
```

2. 使用np.arange生成，np.arange的用法类似于Python中的range:

```
a3 = np.arange(2,21,2)
a3
print(a3)
```

3. 使用np.random生成随机数的数组：

```
a4 = np.random.random((2,2)) # 生产一个2行2列的数组
a4
a5 = np.random.randint(0,18,size=(3,3)) # 生产一个3行3列的随机数字的数组
a5
```

4. 使用函数生成特殊的数组：

```
# 4. 使用函数生成特殊的数组：
a6 = np.zeros((2,2)) # 生产一个所有元素都是0的2行2列的数组
a6
#%%
a7 = np.ones((3,2))  # 生产一个所有元素都是1的3行2列的数组
a7
#%%
a8 = np.full((2,2),6) # 生产一个所有元素都是x:6的2行2列的数组
a8
#%%
a9 = np.eye(3)#生产一个在对角线元素都为1,其他元素都为0 的3x3的矩阵
a9
#%%

```

2.ndarray常用属性：

ndarray.dtype:

因为数组中只能存储同一种数据类型，因此可以dtype获取数组中的元素的数据类型。以下是ndarray.dtype的常用的数据类型：

数据类型	描述
bool_	布尔型数据类型（True 或者 False）
int_	默认的整数类型（类似于 C 语言中的 long，int32 或 int64）
intc	与 C 的 int 类型一样，一般是 int32 或 int 64
intp	用于索引的整数类型（类似于 C 的 ssize_t，一般情况下仍然是 int32 或 int64）
int8	字节（-128 to 127）

int16	整数（-32768 to 32767）
int32	整数（-2147483648 to 2147483647）
int64	整数（-9223372036854775808 to 9223372036854775807）
uint8	无符号整数（0 to 255）
uint16	无符号整数（0 to 65535）
uint32	无符号整数（0 to 4294967295）
uint64	无符号整数（0 to 18446744073709551615）
float_	float64 类型的简写
float16	半精度浮点数，包括：1 个符号位，5 个指数位，10 个尾数位
float32	单精度浮点数，包括：1 个符号位，8 个指数位，23 个尾数位
float64	双精度浮点数，包括：1 个符号位，11 个指数位，52 个尾数位
complex_	complex128 类型的简写，即 128 位复数
complex64	复数，表示双 32 位浮点数（实数部分和虚数部分）
complex128	复数，表示双 64 位浮点数（实数部分和虚数部分）
object_	Python对象
string_	字符串
unicode_	unicode类型

```
#%%
import numpy as np
#%%
a = np.arange(10)
a
#%%
print(a.dtype)
# windows系统 默认是32
# mac 和linux 根据系统
#%%
b = np.array([1,2,3,4],dtype=np.int8)
print(b)
print(b.dtype)
#%%
c = np.array([1,2,3,4],dtype=np.float16)
print(c)
print(c.dtype)
#%%
class Student:
    def __init__(self,name,age):
        self.name = name
        self.age = age
d = np.array(Student("小红",18))
print(d)
print(d.dtype)
#%%
e = np.array(['ab','c'],dtype=np.string_)
print(e)
print(e.dtype)
#%%
d = np.array(['abcdefg','b'],dtype=np.unicode_)
print(d)
print(d.dtype)
#%%
# 修改dtype
a1 = np.array([1,2,3])
print(a1)
print(a1.dtype)
#%%
a2 = a1.astype(np.int8) # astype不会影响数据的本身  而是修改后返回一个新的对象
print(a2.dtype)
#%%
'''
总结
1. 为什么numpy的数值中有这么多的数据类型?
    Numpy本身是基于C语言的 C语言中本身就有很多的数据类型 所以直接引用过来了

```

```
    Numpy为了考虑到处理海量的数据的性能，针对不同的数据给出不同的数据类型 来节省内存空间 所以有不同的数据类型
2. numpy的数值的元素数据类型有哪些-->看笔记
3.使用ndarray.astype可以修改数组元素的数据类型
'''
```

3.多维数组

3.1 ndarray.ndim:

```
#%%
import numpy as np
#%%
a1 = np.array([1,2,3])
a1
#%%
a1.ndim  #维度1
#%%
a2 = np.array([[1,2,3],[4,5,6]])
a2
a2.ndim # 维度2
#%%
a3 = np.array([[[1,2,3,4],[4,5,6,4],[7,8,9,4]],[[1,2,3,4],[4,5,6,4],[7,8,9,4]]])
a3
a3.ndem #维度3
#%%
a3.shape # 查询数组的形状
#%%
a3.size # 总元素
#%%
a3.dtype
#%%
a3.itemsize # 返回的是4  4表示4个字节  每一个字节占8位   4*8= 32
#%%
a3.itemsize*a3.size # 1. 获取总字节
#%%
a3.nbytes # 2. 获取总字节
#%%
```

3.2 ndarray.shape:

```
# shape
#%%
a1 = np.array([1,2,3])
a1.shape # 返回的3 表示 是一个一维的数组 有3个数据
#%%
a2 = np.array([[1,2,3],[4,5,6]])
a2.shape
#%%
a3 = np.array([[[1,2,3,4],[4,5,6,4],[7,8,9,4]],[[1,2,3,4],[4,5,6,4],[7,8,9,4]]])
a3.shape # 2,3,4  返回的意思是      包含2个3*4的数组:  有两个大组 每个大组包含3个小组 每个小组里面有4个元素
#%%
# reshape
#%%
a1 = np.arange(12)
a1
c = a1.reshape((3,4)) # 修改为一个3行4列的数字  2维数组
c.ndim
#%%
b = a1.reshape((2,3,2))#修改一个3维数组  2大组 3小组 每组有2个元素
b.ndim
#%%
m = c.reshape(12,)# 将一个二维的数组 变成一维
# 无论reshape怎么去修改 都不会影响原来的数组 而是将修改后的结果返回给一个新的对象 如果想要直接修改数组本身 那么可以使用resize
m.ndim
#%%
h = b.flatten()
h.ndim
#%%
a1
#%%
```

3.3 ndarray.itemsize:

```
# itemsize
a1 = np.array([1,2,3],dtype=np.int16)
a1.itemsize # 每个字节占8位 返回的是以字节为单位  2字节  2*8=16
#%%
'''
总结
1. 数值一般达到3维就已经很复杂了,所以我们一般都会把3维以上的数组转换为2维数值进行计算
2. 可以通过 ndarray.ndim来查看数组的维度
3. 可以通过 ndarray.shape来查看数值的形状(几组几行几列)
4, 可以通过 reshape来修改数组的一个形状  不会影响原来的数组 而是将修改后的结果返回给一个新的对象
    注意:  修改后的元素个数必须和原来的个数一致  比如原来是(2,6) 那么修改后可以是(3,4) 但是不能这样写(1,4)
5. 可以通过 ndarray.size 来查看数组总共有多少元素
6. 可以通过itemsize来看到每个元素所占的内存的大小,单位是字节 1字节=8位
'''
```

4.索引和切片

1. 获取某行的数据

Python

```
#%%
import numpy as np
#%%
a1 = np.arange(0,29)
a1
#%%
# 1. 获取某行的数据  数组名[索引]
a1[1]
#%%
a1 = np.arange(0,24).reshape((4,6))
a1
#%%
a1[2]
#%%
```

2. 连续获取某几行的数据

```python
#2.1 获取不连续行的数据
a1[[0,2,4]]# 获取索引为0  和2 和4的数据
#%%
#2.2 通过负数来进行索引
a1[[-1,-2]]
#%%
```

3. 获取某行某列的数据

```python
# 3. 获取某行某列的数据
a1 = np.arange(0,30).reshape((5,6))
a1
#%%
a1[1,1]# 1行1列的数据
#%%
a1[0:2,0:2]
#%%
a1[[1,2],[2,3]] # 花式索引
#%%
```

4. 获取某列的数据:

```python
# 4. 获取某列的数据:
a1 = np.arange(0,30).reshape((5,6))
a1
#%%
a1[:,1] # 获取第一列的数据
#%%
```

5. 多维数组:

```python
# 5. 多维数组:
a2 = np.random.randint(0,10,size=(4,6))
a2
#%%
# 获取第一行所有的元素
a2[0]
#%%
# 获取第二行到第三行的所有元素 不包括第四行
a2[1:3]
#%%
# 获取不连续的行
a2[[0,2,3]]
#%%
# 获取第二行第二列的元素
a2[1,1]
#%%
# 使用花式索引
a2[[1,2],[4,5]]
#%%
# 获取第二行和第三行的最后两列
a2[1:3,4:]
#%%
```

6. 总结

```python
'''
总结
1. 如果数组是一维的 那么索引和切片就是和python中的列表是一样的
2. 如果是多维的 那么在中扩中 给两个值 两个值是通过逗号分隔的 逗号前面是行 后面是列 如果中括号中只有一个值 那么就表示行
3. 如果是多维数组 那么行的部分和列的部分 都是遵循一维数组的方式 可以使用整形 切片 还可以使用中括号的形式 来代表不连续的
'''
```

5.布尔索引

```python
#%%
import numpy as np
#%%
a1 = np.arange(0,24).reshape((4,6))
a1
# a1<10
#%%
a2 = a1[a1<10] # 这样就会把a1中小于10 的数据提取出来
a2

#%%
# > < != == >= <=   并且(&)  或者(|)
print(">",a1[a1>20])
print("<",a1[a1<5])
print("!=",a1[a1!=19])
print("==",a1[a1==19])
print(">=",a1[a1>=20])
print("<=",a1[a1<=5])
print("&",a1[(a1>5) & (a1<10)])
print("|",a1[(a1==5) | (a1==10)])
#%%
'''
总结
布尔索引是通过相同数组上的Truc还是False来进行提取的  提取的条件可以有多个 那么如果有多个
可以使用 并且(&) 或者(|)来表示  如果有多个条件 那么每个条件需要使用圆括号括起来
'''
```

6.值的替换

```python
#%%
import numpy as np
#%%
a1 = np.arange(0,24).reshape(4,6)
a1
#%%
a1[2] = 0 # 将索引为2的一行 所有的元素 替换成0
a1
#%%
a2 = np.arange(0,24).reshape(4,6)
a2
#%%
a2[a2>5] = 0  # 通过条件来实现
a2
#%%
a3 = np.arange(0,24).reshape(4,6)
a3
#%%
a4 = np.where(a3<10,1,0)  # 使用函数
a4
#%%
a5 = np.arange(0,24).reshape(4,6)
a5
#%%
a5[2,4] = 36
a5
#%%
'''
总结:
1. 可以使用所有或者切片来替换
2. 使用条件替换
3. 使用where来进行替换
'''
```

7.Numpy索引和切片练习

> 1.将np,arange(10)数组中的奇数全部都替换成-1.
>
> 2.有一个4行4列的数组（比如：np.random.randint(0,10,size=(4,4)),请将其中对角线的数取出来形成一个一维数组，提示(使用np.eye)·
>
> 3.有一个4行4列的数组，请取出其中(0,0),(1,2),(3,2)的点。

4.有一个4行4列的数组，请取出其中2:3行(包括第3行)的所有数据。

5.有一个8行9列的数组，请将其中1-6行(包含第5行)的第8列大于3的数全部都取出来。

6.替换数组中所有小于平均值的元素

7.将一个数组的边框元素设置为0

8.反转二维数组的行

9. 将3D数组的最后一个维度进行求和

10.选择数组中的非对角线元素

Python

```python
#%%
import numpy as np
#%%
# 1.将np.arange(10)数组中的奇数全部都替换成-1.
arr = np.arange(10)
# arr[arr%2==1] = -1
arr[arr%2!=0] = -1
arr
#%%
# 2.有一个4行4列的数组（比如：np.random.randint(0,10,size=(4,4)),请将其中对角线的数取出来形成一个一维数组，提示(使用np.eye)·
arr = np.random.randint(0,10,size=(4,4))
print(arr)
eye_mask = np.eye(4,dtype=bool)
eye_mask
arr[eye_mask]
#%%
# 3.有一个4行4列的数组，请取出其中(0,0),(1,2),(3,2)的点。
arr = np.random.randint(0,10,size=(4,4))
print(arr)
arr[[0,1,3],[0,2,2]]
#%%
# 4.有一个4行4列的数组，请取出其中索引为2:3行(包括第3行)的所有数据。
arr = np.random.randint(0,10,size=(4,4))
print(arr)
arr[2:4]
#%%
# 5.有一个8行9列的数组，请将其中1-6行(包含第5行)的第8列大于3的数全部都取出来。
arr = np.random.randint(0,10,size=(8,9))
print(arr)
# arr[1:6,8][arr[1:6,8]>3]
# [1:6,8] 这是得到一个一维数组
# [arr[1:6,8]>3] 返回一个bool类型的数组

a2 = arr[1:6,8]
print(a2)
# a2[a2[1:6,8]>3]
a2[a2>3]
#%%
# 6.替换数组中所有小于平均值的元素
arr = np.random.randint(0,20,size=(5,5))
print(arr)
mean_value =  arr.mean()
print(mean_value)
arr[arr<mean_value] = mean_value
```

```
arr
#%%
# 7.将一个数组的边框元素设置为0
arr = np.random.randint(0,10,size=(6,6))
arr[0,:] = 0 # 将第一行设置为0
arr[-1,:] = 0 # 将最后一行设置为0
arr[:,0] = 0# 将第一列设置为0
arr[:,-1] = 0 # 将最后一列设置为0
arr
#%%
# 8.反转二维数组的行
arr = np.arange(16).reshape(4,4)
print(arr)
arr[::-1]
#%%
# 9. 将3维数组的最后一个维度进行求和
arr = np.random.randint(0,10,size=(3,4,5))
print(arr)
arr.sum(axis=-1)[2]
#%%
# 10.选择数组中的非对角线元素
arr = np.random.randint(0,10,size=(5,5))
print(arr)
eye_mask = np.eye(5,dtype=bool)
print(eye_mask)
arr[~eye_mask]
#%%
```

8.Numpy数组操作

数组广播机制

数组与数的计算:

数组与数组的计算：

1. 结构相同的数组之间的运算:
2. 与行数组相同并且只有1列的数祖之间的运算: ；’、
3. 与列数组相同并且只有1行的数组之间的运算:

广播原则：

1. shape为(3,8,2)的数组能和(8,3)的数组进行运算吗?

```python
# 如果两个数组不相同你玩么小维度的数组的形状将会在最左边补1
# 如果两个数字的形状在某个维度上不匹配 并且其中一个数字的该维度上大小为1 则该数组在该维度上的大小
# 将会拉伸致匹配另外一个数组的大小
```

2. shape为(3,8,2)：的数组能和(8,1)的数组进行运算吗?

```python
# 2. shape为(3,8,2)：的数组能和(8,1)的数组进行运算吗?
a1 = np.random.randint(0,5,size=(3,8,2))
print(a1)
# a2 = np.random.randint(0,5,size=(8,3))
a2 = np.random.randint(0,5,size=(8,1))
print(a2)
a3 = a1+a2
a3
# 如果在任何维度上大小不一致两个数组子啊该维度上的大小都不为1
```

分析：能，因为按照广播原则，从后面往前而数，(3,8,2)和(8,1)中的2和1虽然不相等，但是因为有一方的长度为1，所以能参与运算。

3. shape为(3,1,8)的数组能和(8,1)的数组进行运算吗?

分析：能，因为按照广播原则，从后面往前面数，(3,1,4)和(81)中的4和1虽然不相等且1和8不相等，但是因为这两项中有一方的长度为1，所以能参与运算。

总结

```plain text
'''
1. 数组和数字直接进行运算是没有问题的
2. 两个shape相同的数组是可以进行运算的
3. 如果两个shape不相同的数字想要进行运算那么需要看他们是否满足了广播机制
4. 如果两个数字的后缘维度即从末尾开始算到起维度M轴长度相符合或者其中一方的长度为1 他们是广播兼容的
    广播会在缺失(或者)长度为1的维度上进行
'''
```

9.数组形状的操作

reshape和resize方法:

Python

```python
#%%
import numpy as np
#%%
a1 = np.random.randint(0,10,size=(3,4))
a1
#%%
a1.reshape((2,6))# 返回一个新的对象 并不是对原有对象进行修改
#%%
a1
#%%
a2 = a1.reshape((2,6))
a2
#%%
#resize 和reshape不同的  reshape是不修改数组的本身 而是返回一个新的对象  resize是对本身进行修改 是没有返回值的
a3 = np.random.randint(0,10,size=(3,4))
a3
#%%
a3.resize((2,6))
a3
#%%

```

flatten和ravel方法：

Python

```
#%%
import numpy as np
#%%
# flatten和 ravl  是将多维数组转换成一维数组
#%%
# flatten是将数组转换成一位数数组后 然后将这数组拷贝回去  后续对这个数组进行修改不会影响之前的数组
x = np.array([[1,2],[3,4]])
x
#%%
a3 = x.flatten()
a3[1] = a3[1]-100
a3
#%%
x
#%%
# ravel:把引用地址返回回去,所有对其本数据进行修改会影响之前的数据
x = np.array([[1,2],[3,4]])
x3 = x.ravel()
#%%
x3
#%%
x
#%%
x3[1] = 100
#%%
x3
#%%
x
#%%
"""
总结:
1.reshape和resize都是重新定义形状的, 但是 reshape不会修改数组的本身,而是将修改后的结果返回回去给一个新的对象
    resize 是直接修改数组本身
2. 02.flatten和ravel 都是用来将数组转变问一维数组,并且他们都不会对源数据造成修改,但是flatten返回的是一个拷贝,所有对02.flatten和ravl.ipynb的返回值的修改不会影响其他(原来)的数组,ravel返回的是一个引用地址,那么会对原来的值进行修改
""";
```

不同的数组的组合:

```python
#%%
import numpy as np
#%%
# vstack:将数组按照垂直方向进行叠加 ,主要 数组的列数必须相同 才能叠加
a1 = np.random.randint(0,10,size=(3,5))
print(a1)
a2 = np.random.randint(0,10,size=(1,5))
print(a2)
a3 = np.vstack([a1,a2])
a3
#%%
# hstack:将数组按照水平方法进行叠加 数组的行必须相同才能叠加
a1 = np.random.randint(0,10,size=(3,2))
print(a1)
a2 = np.random.randint(0,10,size=(3,1))
print(a2)
a3 = np.hstack([a1,a2])
a3
#%%
# concatenate([a,b],axis=0/1/None)
a = np.array([[1,2],[3,4]])
b = np.array([[5,6]])
# ab = np.concatenate([a,b],axis=0)
# ab = np.concatenate([a,b.T],axis=1)
ab = np.concatenate([a,b.T],axis=None)
ab
#%%
'''
1.hstack代表的是水平方向叠加,如果叠加成功,那么他们的行必须一致
2.vstack代表的是垂直方向叠加,如果叠加成功,那么他们的列必须一致
3.concatenate可以手动的指定axis参数具体是在那个方向叠加,如果是axis=0表示水平叠加
    axis=1表示垂直叠加 None 那么会线进行叠加然后转换成一维数组
'''
```

10.数组的切割:

1. hsplit:
2. vsplit:
3. split/array_split(array,indicate_or_seciont,axis)