《数据科学与大数据导论》实验课程

第2章 使用NumPy进行数据计算

目录

2.1 安装NumPy

NumPy是什么？

NumPy（Numerical Python）是Python用于科学计算的基础软件包，是Python的开源数值计算扩展，NumPy主要用来存储数组对象和处理大型矩阵（matrix），比Python自身的嵌套列表（nested list structure，也可用来表示矩阵）结构要高效的多。此外NumPy是一个运行速度非常快的数学库，主要用于数组计算，包含强大的n维数组对象ndarray、广播功能函数、线性代数等功能。

2.1 安装NumPy

在cmd下使用命令： pip install numpy

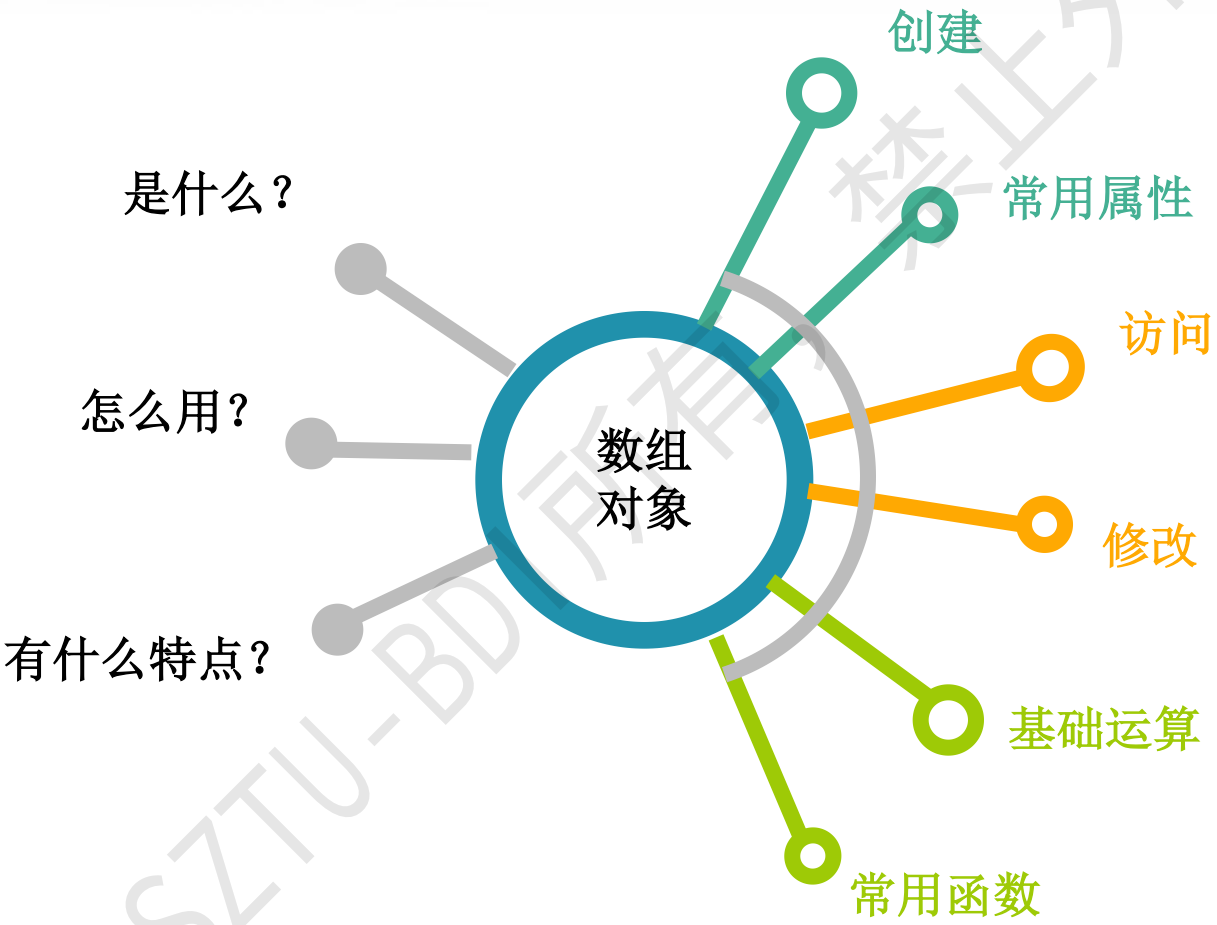
```
C:\Users\Administrator>pip install numpy
```

注意：若没有配置环境变量，需要先cd到python的安装目录

```
C:\Users\Administrator>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from numpy import *
>>> random.rand(3,4)
array([[0.42102008, 0.55586969, 0.21712186, 0.67042703],
       [0.81249156, 0.45128924, 0.52166022, 0.90980144],
       [0.9240672 , 0.39724709, 0.8794015 , 0.81773515]])
>>>
```

安装完成后实验NumPy是否安装成功

2.2 NumPy中的数组对象



2.2.1 数组的创建

1.使用array()函数创建数组：

使用NumPy的array()函数可以创建一维、二维、n维数组，array()函数要求传入Python列表数据，传入Python列表数据的嵌套层次决定了创建数组的维数。

```
>>> import numpy as np  #引入Numpy包
>>> array1=np.array([0.1,0.4,0.9,0.16,0.25,0.36])        #创建一个一维数组
>>> array1                                  #查看创建的一维数组
array([0.1 , 0.4 , 0.9 , 0.16, 0.25, 0.36])
>>> array2=np.array([[88,90,94,90,85],[93.5,93.5,92.5,90.5,90.5],[0,0,0,0,0]])#
创建一个二维数组
>>> array2
array([[88. , 90. , 94. , 90. , 85. ],
       [93.5, 93.5, 92.5, 90.5, 90.5],
       [ 0. ,  0. ,  0. ,  0. ,  0. ]])
```

```
>>> array3=np.array([[[88,90],[78,87]],[84,88]])        #创建一个多维数组
>>> array3
array([[list([88, 90]), list([78, 87])],
       [84, 88]], dtype=object)
>>>
```

2.2.1 数组的创建

2.使用zeros()、ones()、empty()函数创建数组：

zeros()函数创建数组元素全部为0的数组，默认情况下数组元素的类型为float 64；ones()函数创建数组元素全部为1的数组，默认情况下数组元素的类型为float 64；empty()函数创建创建一个没有任何具体值的ndarray数组。

一维数组：

```
>>> import numpy as np
>>> array1=np.zeros(5)                    #使用zeros()函数创建一维数组
>>> array1                              #查看创建的一维数组，元素均为0
array([0., 0., 0., 0., 0.])
>>> array2=np.ones(7)                     #使用ones()函数创建一维数组
>>> array2                              #查看创建的一维数组，元素均为1
array([1., 1., 1., 1., 1., 1., 1.])
>>> array3=np.empty(3)                    #使用empty()函数创建一维数组
>>> array3                              #查看创建的一维数组，元素为随机数
array([4.01702319e-305, 4.02580180e-305, 6.09601514e-317])
>>>
```

2.2.1 数组的创建

2.使用zeros()、ones()、empty()函数创建数组：

二维或多维数组：

```
>>> import numpy as np
>>> array1=np.zeros((3,3))                    #使用zeros()函数创建二维数组
>>> array1                                    #查看创建的二维数组，元素均为0
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
>>> array2=np.ones((2,2))                     #使用ones()函数创建二维数组
>>> array2                                    #查看创建的二维数组，元素均为1
array([[1., 1.],
       [1., 1.]])
>>> array3=np.empty((2,4))                    #使用empty()函数创建二维数组
>>> array3                                    #查看创建的二维数组，元素为随机数
array([[1.e-323, 0.e+000, 0.e+000, 0.e+000],
       [0.e+000, 0.e+000, 0.e+000, 0.e+000]])
>>> array4=np.ones((3,5,4))                   #使用ones()函数创建三维数组
>>> array4                                    #查看创建的三维数组，元素均为1
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.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
>>>
```

2.2.1 数组的创建

2.使用zeros()、ones()、empty()函数创建数组：

创建指定类型的数组
例如：complex的二维数组

```
>>> import numpy as np
>>> array1=np.zeros((3,4),dtype=complex)
>>> #使用zeros()函数创建数据类型为complex的二维数组
>>> array1                          #查看创建的二维数组
array([[0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j]])
>>>
```

2.2.1 数组的创建

2.使用random函数创建随机数组：

1.np.random.random()函数参数

np.random.random((1000, 20))
代表生成1000行 20列的浮点数，浮点数都是从0-1中随机。

2.numpy.random.rand()函数用法

numpy.random.rand(d0, d1, ..., dn)：
生成一个[0,1)之间的随机浮点数或N维浮点数组。

3.numpy.random.randn()函数用法：

numpy.random.randn(d0, d1, ..., dn)：
生成一个浮点数或N维浮点数组，取数范围：正态分布的随机样本数。

2.2.1 数组的创建

3.使用arange()函数创建等间隔的数字数组：

使用arange()函数可以创建等间隔的数字数组，其函数参数有三个，第一个为起始值，第二个为终止值，第三个参数为间隔距离，第三个参数默认值为None，即一个单位。

```
>>> import numpy as np
>>> array1=np.arange(6,10,dtype=None)
>>> #用arange()函数创建从6开始，10结束，间隔为1的数组，不包含末尾的10
>>> array1                          #查看创建的一维数组
array([6, 7, 8, 9])
>>> array2=np.arange(6.6,12.7)
>>> #使用arange()函数创建从6.6开始，12.7结束，间隔为1的数组
>>> array2                          #查看创建的一维数组
array([ 6.6,  7.6,  8.6,  9.6, 10.6, 11.6, 12.6])
>>> array3=np.arange(3.8,8.2,0.6)
>>> #使用arange()函数创建从3.8开始，8.2结束，间隔为0.6的数组
>>> array3                          #查看创建的一维数组
array([3.8, 4.4, 5. , 5.6, 6.2, 6.8, 7.4, 8. ])
>>>
```

2.2.2 数组对象的常用属性

常用的ndarray对象属性

属性属性	说明
ndarray.ndim	秩，即轴的数量或维度的数量
ndarray.shape	数组的维度，对于矩阵，n行m列
ndarray.size	数组元素的总个数，相当于.shape中n*m的值
ndarray.dtype	ndarray对象的元素类型
ndarray.itemsize	ndarray对象中每个元素的大小，以字节为单位
ndarray.flags	ndarray对象的内存信息
ndarray.real	ndarray元素的实部
ndarray.imag	ndarray元素的虚部
ndarray.data	包含实际数组元素的缓冲区，由于一般通过数组的索引获取元素，所以通常不需要使用这个属性

2.2.2 数组对象的常用属性

ndarray.flags中的内存信息介绍

属性	描述
C_CONTIGUOUS(C)	数据是在一个单一的C风格的连续段中
F_CONTIGUOUS(F)	数据是在一个单一的Fortran风格的连续段中
OWNDATA(O)	数组拥有它所使用的内存或从另一个对象中借用它
WRITEABLE(W)	数据区域可以被写入，将该值设置为False，则数据为只读
ALIGNED(A)	数据和所有元素都适当地对齐到硬件上
UPDATEIFCOPY(U)	这个数组是其它数组的一个副本，当这个数组被释放时，原数组的内容将被更新

2.2.3 数组元素的访问与修改

1.一维数组元素的访问与修改：

访问的时候是用的变量名加上数组的下标（数组下标从0开始，数组下标是对应的元素个数-1，因此第三个元素的数组下标为2），并用[]框起来，修改同理。

```
>>> import numpy as np
>>> array1=np.array([1,2,3])          #创建一个一维数组
>>> array1[2]                         #查看数组下标为2的元素
3
>>> array1[2]=4                       #修改数组中的元素值
>>> array1[2]                         #查看修改后数组下标为2的元素
4
>>> array1                            #查看修改后数组所有元素
array([1, 2, 4])
>>>
```

2.2.3 数组元素的访问与修改

2.二维数组元素的访问与修改：

如果只给出一个[]，就会输出一行的元素，因此这个二维数组可以看做是由多个一维数组组成的一个新的数组。
如果要访问到某个位置的元素，就需要同时确定它的行标和列标，所以在访问时，需要用2个[]，其中第一个框代表行标，第二个框代表列标，其数组下标意义与一维数组类似。

```
>>> import numpy as np
>>> array2=np.array([[1,2,3],[4,5,6],[7,8,9]])#创建一个二维数组
>>> array2[1]                                  #查看数组下标为1的元素
array([4, 5, 6])
>>> array2[0,2]                                #查看数组下标为(0,2)的元素
3
>>> array2[0,2]=11                             #修改数组中的元素值
>>> array2[0,2]                                #查看修改后数组下标为(0,2)的元素
11
>>> array2[0]                                  #查看修改后数组下标为0的元素
array([ 1,  2, 11])
>>> array2                                     #查看修改后数组所有元素
array([[ 1,  2, 11],
       [ 4,  5,  6],
       [ 7,  8,  9]])
>>>
```

n维数组的访问可以参考二维数组依次类推。

2.2.4 数组对象的基础运算

1.不改变原数组的自增自减运算

```
>>> import numpy as np
>>> array1=np.array([1,3,5,7,9])          #创建一个一维数组
>>> array2=array1+5                       #让array1数组的元素+5并赋给array2数组
>>> array2                                #查看做加法后array2数组的元素值
array([ 6,  8, 10, 12, 14])
>>> array3=array2*2                       #让array2数组的元素*2并赋给array3数组
>>> array3                                #查看做乘法后array3数组的元素值
array([12, 16, 20, 24, 28])
>>> array1                                #查看array1数组的元素值
array([1, 3, 5, 7, 9])
>>>
```

2.改变原数组的自增自减运算

```
>>> import numpy as np
>>> array1=np.array([1,3,5,7,9])          #创建一个一维数组
>>> array1+=5                             #让array1数组进行自加操作
>>> array1                                #查看做自加后array1数组的元素值
array([ 6,  8, 10, 12, 14])
>>> array1*=2                             #让array1数组进行自乘操作
>>> array1                                #查看做自乘后array1数组的元素值
array([12, 16, 20, 24, 28])
>>>
```

2.2.5 数组对象的基本操作

1.reshape()函数：

reshape()函数的功能是改变数组形状，可以把x维数组改成y维数组。
函数原型是reshape(n)。
参数：n代表数组形状。

```
>>> import numpy as np
>>> array1=np.array([1,2,3,4,5,6,7,8])          #创建一个一维数组
>>> array1                  #查看创建的一维数组
array([1, 2, 3, 4, 5, 6, 7, 8])
>>> array2=array1.reshape((2,4))#使用reshape()函数将原数组改变成二维数组
>>> array2                  #查看使用reshape()函数后的数组
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> array3=array2.reshape((2,2,2))
>>> #使用reshape()函数将原数组改变成三维数组
>>> array3                  #查看使用reshape()函数后的数组
array([[[1, 2],
        [3, 4]],

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

2.2.5 数组对象的基本操作

2.ravel()函数：

ravel()函数的功能是将多维数组展开为一维数组。

```
>>> import numpy as np
>>> array1=np.array([[1,2,3],[4,5,6]])          #创建一个二维数组
>>> array1                  #查看创建的二维数组
array([[1, 2, 3],
       [4, 5, 6]])
>>> array2=array1.ravel()          #使用ravel()函数将原数组展开成一维数组
>>> array2                  #查看使用ravel()函数后的数组
array([1, 2, 3, 4, 5, 6])
>>> array3=np.array([[[1,2],[3,4]],[[5,6],[7,8]]])#创建一个三维数组
>>> array3                  #查看创建的三维数组
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
>>> array4=array3.ravel()          #使用ravel()函数将原数组展开成一维数组
>>> array4                  #查看使用ravel()函数后的数组
array([1, 2, 3, 4, 5, 6, 7, 8])
>>>
```

2.2.5 数组对象的基本操作

3.concatenate()函数：

concatenate()函数的功能是将多个数组连接。
函数原型是concatenate(arr,axis)。
参数：arr是要拼接的数组，要求数组维数要一致；axis默认值是0，表示在第0个维度上拼接，也可以给其赋值，拼接在指定维度上。

```
>>> import numpy as np
>>> array1=np.array([[1,2],[3,4]])                    #创建一个二维数组
>>> array1                    #查看创建的二维数组
array([[1, 2],
       [3, 4]])
>>> array2=np.array([[5,6]])                          #创建一个二维数组
>>> array2                    #查看创建的二维数组
array([[5, 6]])
>>> array3=np.concatenate((array1,array2))
>>> #使用concatenate()函数将两个二维数组拼接成一个二维数组，且维度为0
>>> array3                    #查看使用concatenate()函数后的数组
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> array4=np.concatenate((array1,array2.T),axis=1)
>>> #使用concatenate()函数将两个二维数组拼接成一个二维数组，且维度为1
>>> array4                    #查看使用concatenate()函数后的数组
array([[1, 2, 5],
       [3, 4, 6]])
>>>
```

2.2.5 数组对象的基本操作

4.delete()函数：

delete()函数的功能是从数组中删除指定值。
函数原型是delete(arr,obj,axis)。
参数：arr是需要处理的矩阵；obj在什么位置处理；axis是一个可选参数，axis=None，1，0。当axis=None时，arr会先按行展开，然后按照obj，删除第obj-1（从0开始）位置的数，返回一个行矩阵；当axis=0时，arr按行删除；当axis=1时，arr按列删除。

```
>>> import numpy as np
>>> array1=np.array([[1,2,3,4],[5,6,7,8]])      #创建一个二维数组
>>> array1                  #查看创建的二维数组
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> array2=np.delete(array1,3)          #使用delete()函数删除第三个元素
>>> array2                  #查看使用delete()函数后的数组
array([1, 2, 3, 5, 6, 7, 8])
>>> array3=np.delete(array1,1,0)        #使用delete()函数删除第二行元素
>>> array3                  #查看使用delete()函数后的数组
array([[1, 2, 3, 4]])
>>> array4=np.delete(array1,0,1)        #使用delete()函数删除第二列元素
>>> array4                  #查看使用delete()函数后的数组
array([[2, 3, 4],
       [6, 7, 8]])
>>>
```

2.2.5 数组对象的基本操作

5.sort()函数：

sort()函数返回输入数组的排序副本。
函数原型是sort(arr, axis, kind, order)。
参数：arr是要排序的数组；axis是沿着它排序数组的轴，如果没有数组会被展开，沿着最后的轴排序，axis=0按列排序，axis=1按行排序；kind是排序方法，默认为'quicksort'（快速排序），排序方法及其一些情况如表2-3所示；order是排序的字段，可以不包含。

种类	速度	最坏情况	工作空间	稳定性
'quicksort'（快速排序）	1	O(n^2)	0	否
'mergesort'（归并排序）	2	O(n*log(n))	~n/2	是
'heapsort'（堆排序）	3	O(n*log(n))	0	否

2.2.5 数组对象的基本操作

5.sort()函数：

```
>>> import numpy as np
>>> array1=np.array([[3,9],[1,7]])                    #创建一个二维数组
>>> array1                          #查看创建的二维数组
array([[3, 9],
       [1, 7]])
>>> array2=np.sort(array1)                            #使用sort()函数按行进行排序
>>> array2                          #查看排序后数组
array([[3, 9],
       [1, 7]])
>>> array3=np.sort(array1,axis=0)                     #使用sort()函数按列进行排序
>>> array3                          #查看排序后数组
array([[1, 7],
       [3, 9]])
>>>
```

2.2.5 数组对象的基本操作

6.where()函数：

where()函数筛选出满足条件元素的下标。
函数原型是where(condition, x, y)。
参数：满足条件condition，输出x，不满足输出y。
若没有x和y，则输出满足条件（即非0）元素的坐标。

```
>>> import numpy as np
>>> array1=np.arange(10)                    #创建等距的一维数组
>>> array1                                  #查看创建的一维数组
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.where(array1,1,-1)                   #使用where()函数进行判断
array([-1,  1,  1,  1,  1,  1,  1,  1,  1,  1])
>>> np.where(array1>4,2,-2)                 #使用where()函数进行判断
array([-2, -2, -2, -2, -2,  2,  2,  2,  2,  2])
>>> np.where(array1>5)                      #返回索引
(array([6, 7, 8, 9], dtype=int32),)
>>> array1[np.where(array1>5)]              #等价于 a[a>5]
array([6, 7, 8, 9])
>>> np.where([[0,1], [1,0]])                          #查看结
(array([0, 1], dtype=int32), array([1, 0], dtype=int32))
>>>
```

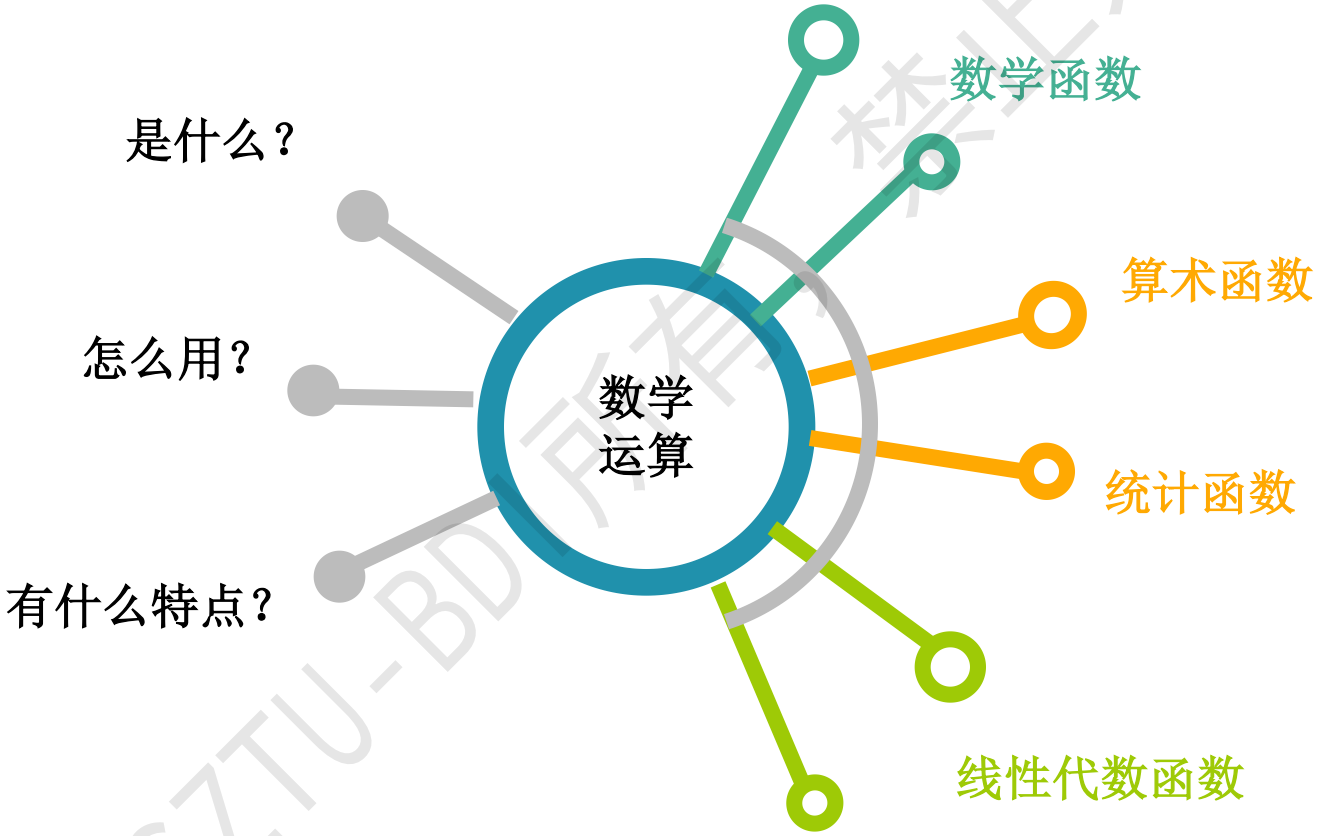
2.2.5 数组对象的基本操作

7.extract()：

extract()和where()函数有一点类似，不过extract()函数是筛选出满足条件元素的值并返回，而不是元素索引。

```
>>> import numpy as np
>>> array1=np.arange(10)                    #创建等距的一维数组
>>> array1                                  #查看创建的一维数组
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> #设置条件，如：能整除3
>>> x=np.mod(array1,3)==0
>>> x                                       #查看判断结果
array([ True, False, False,  True, False, False,  True, False, False,
        True])
>>> np.extract(x,array1)
array([0, 3, 6, 9])
>>>
```

2.3 使用NumPy进行数学运算



2.3.1 数学函数

1.三角函数

标准的三角函数：sin()、cos()、tan()
反三角函数arcsin()、arccos()和arctan()
degrees()函数将弧度转换为角度

```
>>> import numpy as np
>>> np.sin(1)                          #使用sin()函数
0.8414709848078965
>>> np.cos(1)                          #使用cos()函数
0.5403023058681398
>>> np.tan(1)                          #使用tan()函数
1.5574077246549023
>>> np.arcsin(0.5)                     #使用arcsin()函数
0.5235987755982989
>>> np.arccos(0.5)                     #使用arccos()函数
1.0471975511965979
>>> np.arctan(0.5)                     #使用arctan()函数
0.4636476090008061
>>> np.degrees(np.arcsin(0.5))         #使用degrees()函数将弧度转化为角度
30.000000000000004
>>> np.degrees(np.arccos(0.5))         #使用degrees()函数将弧度转化为角度
60.00000000000001
>>> np.degrees(np.arctan(0.5))         #使用degrees()函数将弧度转化为角度
26.56505117707799
>>>
```

2.3.1 数学函数

2.舍入函数

（1）around()函数返回指定数字的四舍五入值。
函数原型是around(array,decimals)。
参数：array表示数组；decimals表示舍入的小数位数，默认值为0，如果为负，整数将四舍五入到小数点左侧的位置。

```
>>> import numpy as np
>>> array1=np.array([1.45,2.15,5.71,6.198,10.982])
>>> array1                    #查看创建的一维数组
array([ 1.45 ,  2.15 ,  5.71 ,  6.198, 10.982])
>>> np.around(array1)         #使用around()函数进行取整，以1为单位
array([ 1.,  2.,  6.,  6., 11.])
>>> np.around(array1,-1)      #使用around()函数进行取整，以10为单位
array([ 0.,  0., 10., 10., 10.])
>>> np.around(array1,1)       #使用around()函数进行取整，以0.1为单位
array([ 1.4,  2.2,  5.7,  6.2, 11. ])
>>>
```

2.3.1 数学函数

2.舍入函数

（2）floor()函数用于返回小于或者等于指定表达式的最大整数，即向下取整。ceil()函数用于返回大于或者等于指定表达式的最小整数，即向上取整。
函数原型：floor(n)，ceil(n)
参数：n为要取整的数

```
>>> import numpy as np
>>> np.floor(2.3)          #使用floor()函数进行向下取整
2.0
>>> np.floor(2.9)          #使用floor()函数进行向下取整
2.0
>>> np.floor(-2.3)         #使用floor()函数进行向下取整
-3.0
>>> np.ceil(2.3)                    #使用ceil()函数进行向上取整
3.0
>>> np.ceil(2.9)                    #使用ceil()函数进行向上取整
3.0
>>> np.ceil(-2.3)          #使用ceil()函数进行向上取整
-2.0
>>>
```

2.3.2 算术函数

1.简单的加减乘除

add()，subtract()，multiply()和divide()是简单的加减乘除运算，也是基本的运算，这里要注意的是数组必须具有相同的形状或符合数组广播规则，否则会报错。

```
>>> import numpy as np
>>> array1=np.array([[1,4,7],[2,5,8],[3,6,9]])          #创建一个二维数组
>>> array1                                              #查看创建的二维数组
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
>>> array2=np.array([2,3,4])                            #创建一个一维数组
>>> np.add(array1,array2)                               #数组相加
array([[ 3,  7, 11],
       [ 4,  8, 12],
       [ 5,  9, 13]])
>>> np.subtract(array1,array2)                          #数组相减
array([[-1,  1,  3],
       [ 0,  2,  4],
       [ 1,  3,  5]])
>>> np.multiply(array1,array2)                          #数组相乘
array([[ 2, 12, 28],
       [ 4, 15, 32],
       [ 6, 18, 36]])
>>> np.divide(array1,array2)                            #数组相除
array([[0.5       , 1.33333333, 1.75      ],
       [1.        , 1.66666667, 2.        ],
       [1.5       , 2.        , 2.25      ]])
>>>
```

2.3.2 算术函数

2.reciprocal()函数

reciprocal()函数返回参数元素的倒数。

```
>>> import numpy as np
>>> array1=np.array([1/5,5/2,5/4])          #创建一个一维数组
>>> array1                                  #查看创建的一维数组
array([0.2 , 2.5 , 1.25])
>>> np.reciprocal(array1)                   #使用reciprocal()函数进行取到数
array([5. , 0.4, 0.8])
>>>
```

2.3.2 算术函数

3.power()函数

power()函数是将第一个输入数组中的元素作为底数，计算它与第二个输入数组中相应元素的幂。

```
>>> import numpy as np
>>> array1=np.array([2,3,4])
>>> array1
array([2, 3, 4])
>>> array2=np.array([4,3,2])
>>> array2
array([4, 3, 2])
>>> np.power(array1,array2)
array([16, 27, 16], dtype=int32)
>>>
```

#创建一个一维数组
#查看创建的一维数组

#创建一个一维数组
#查看创建的一维数组

#使用power()函数求幂

2.3.2 算术函数

4.mod()与remainder()函数

mod()函数计算输入数组中相应元素的相除后的余数。remainder()函数也产生相同的结果。

```
>>> import numpy as np
>>> array1=np.array([15,23,34])                    #创建一个一维数组
>>> array1                                         #查看创建的一维数组
array([15, 23, 34])
>>> array2=np.array([4,3,2])                       #创建一个一维数组
>>> array2                                         #查看创建的一维数组
array([4, 3, 2])
>>> np.mod(array1,array2)                          #使用mod()函数求余数
array([3, 2, 0], dtype=int32)
>>> np.remainder(array1,array2)                    #使用remainder()函数求余数
array([3, 2, 0], dtype=int32)
>>>
```

2.3.3 统计函数

1.最大值amax()与最小值amin()函数

这两个函数的功能分别是用于计算数组中的元素沿指定轴的最小大值和最小值。
另外还有ptp()函数计算数组中元素最大值与最小值的差。

```
>>> import numpy as np
>>> array1=np.array([13,65,89,34,32,11,23,25,46])        #创建一维数组
>>> array1                                               #查看创建的一维数组
array([13, 65, 89, 34, 32, 11, 23, 25, 46])
>>> np.amax(array1)                                      #使用amax()函数求最大值
89
>>> np.amin(array1)                                      #使用amin()函数求最大值
11
>>> np.ptp(array1)                                       #使用ptp()函数求最大值与最小值差值
78
>>>
```

2.3.3 统计函数

2.百分位数percentile()函数

percentile()函数的主要功能是用来计算数组中的百分位数，百分位数是统计中使用的度量，表示小于这个值的观察值的百分比。
函数原型是percentile(array, q, axis)。
参数：array是输入数组；q是要计算的百分位数，在0~100之间；axis是沿着它计算百分位数的轴。

```
>>> import numpy as np
>>> array1=np.array([13,65,89,34,32,11,23,25,46])        #创建一维数组
>>> array1                    #查看创建的一维数组
array([13, 65, 89, 34, 32, 11, 23, 25, 46])
>>> np.percentile(array1,50)#使用percentile()函数查找位于数组中50%的值
32.0
>>>
```

2.3.3 统计函数

3.中位数median()函数

median()函数的主要功能是用于计算数组**array**中元素的中位数（中值）。

```
>>> import numpy as np
>>> array1=np.array([13,65,89,34,32,11,23,25,46])        #创建一维数组
>>> array1                  #查看创建的一维数组
array([13, 65, 89, 34, 32, 11, 23, 25, 46])
>>> np.median(array1)              #使用median()函数查找中位数
32.0
>>>
```

2.3.3 统计函数

4.算数平均数mean()函数与加权平均数average()函数

mean()函数的主要功能是返回数组中元素的算术平均值。如果提供了轴，则沿其计算。算术平均值是沿轴的元素的总和除以元素的数量。average()函数根据在另一个数组中给出的各自的权重计算数组中元素的加权平均值；该函数可以接受一个轴参数，如果没有指定轴，则数组会被展开。加权平均值即将各数值乘以相应的权数，然后加总求和得到总体值，再除以总的单位数。

```
>>> import numpy as np
>>> array1=np.array([13,65,89,34,32,11,23,25,46])        #创建一维数组
>>> array1                                    #查看创建的一维数组
array([13, 65, 89, 34, 32, 11, 23, 25, 46])
>>> np.mean(array1)                    #使用mean()函数查找算术平均值
37.555555555555556
>>> np.average(array1)  #使用average()函数查找加权平均值
37.555555555555556
>>>
```

2.3.3 统计函数

5.标准差std()函数与方差var()函数

标准差是一组数据平均值分散程度的一种度量，而统计中的方差（样本方差）是每个样本值与全体样本值的平均数之差的平方值的平均数，其中，标准差是方差的平方根。

```
>>> import numpy as np
>>> array1=np.array([13,65,89,34,32,11,23,25,46])        #创建一维数组
>>> array1                                  #查看创建的一维数组
array([13, 65, 89, 34, 32, 11, 23, 25, 46])
>>> np.std(array1)        #使用std()函数查找标准差
24.019025380870758
>>> np.var(array1)        #使用var()函数查找方差
576.9135802469136
>>>
```

2.3.4 线性代数

函数	描述
dot()	两个数组的点积，即元素对应相乘
vdot()	两个向量的点积
inner()	两个数组的内积
matmul()	两个数组的矩阵积
determinant()	数组的行列式
solve()	求解线性矩阵方程
inv()	计算矩阵的乘法逆矩阵

2.3.4 线性代数

1.dot()函数

dot()函数主要对于两个一维的数组，计算的是这两个数组对应下标元素的乘积和（数学上称之为内积）；对于二维数组，计算的是两个数组的矩阵乘积；对于多维数组，它的通用计算公式如下，即结果数组中的每个元素都是：数组a的最后一维上的所有元素与数组b的倒数第二位上的所有元素的乘积和：dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])。

```
>>> import numpy.matlib
>>> import numpy as np
>>> array1=np.array([[2,4],[6,8]])                    #创建一个二维数组
>>> array1                  #查看创建的二维数组
array([[2, 4],
       [6, 8]])
>>> array2=np.array([[22,44],[66,88]])  #创建一个二维数组
>>> array2                  #查看创建的二维数组
array([[22, 44],
       [66, 88]])
>>> np.dot(array1,array2)                    #使用dot()函数求数组的乘积
array([[308, 440],
       [660, 968]])
>>>
```

2.3.4 线性代数

2.vdot()函数

vdot()函数的作用是计算两个向量的点积。如果第一个参数是复数，那么它的共轭复数会用于计算。如果参数是多维数组，它将会被先展开，后计算。

```
>>> import numpy as np
>>> array1=np.array([2,4,6,8])          #创建一个一维数组
>>> array1                              #查看创建的一维数组
array([2, 4, 6, 8])
>>> array2=np.array([1,3,5,7])          #创建一个一维数组
>>> array2                              #查看创建的一维数组
array([1, 3, 5, 7])
>>> np.vdot(array1,array2)              #使用vdot()函数求两数组的点积
100
>>> array3=np.array([[1,3],[5,7]])      #创建一个二维数组
>>> array3                              #查看创建的二维数组
array([[1, 3],
       [5, 7]])
>>> np.vdot(array1,array3)              #使用vdot()函数求两数组的点积
100
>>>
```

2.3.4 线性代数

3.inner()函数

inner()函数的作用是返回一维数组的向量内积。对于更高的维度，它返回最后一个轴上的和的乘积。

```
>>> import numpy as np
>>> array1=np.array([2,4,6,8])   #创建一个一维数组
>>> array1                       #查看创建的一维数组
array([2, 4, 6, 8])
>>> array2=np.array([1,3,5,7])   #创建一个一维数组
>>> array2                       #查看创建的一维数组
array([1, 3, 5, 7])
>>> np.inner(array1,array2)      #使用inner()函数求两数组的内积
100
>>>
```

2.3.4 线性代数

4.matmul()函数

matmul()函数返回两个数组的矩阵乘积。虽然它返回二维数组的正常乘积，但如果任一参数的维数大于2，则将其视为存在于最后两个索引的矩阵的栈，并进行相应广播。另一方面，如果任一参数是一维数组，则通过在其维度上附加1来将其提升为矩阵，并在乘法之后被去除。

```
>>> import numpy.matlib
>>> import numpy as np
>>> array1=np.array([[2,4],[6,8]])                    #创建一个二维数组
>>> array1                                            #查看创建的二维数组
array([[2, 4],
       [6, 8]])
>>> array2=np.array([[22,44],[66,88]])                #创建一个二维数组
>>> array2                                            #查看创建的二维数组
array([[22, 44],
       [66, 88]])
>>> np.matmul(array1,array2)      #使用matmul()函数求两数组的矩阵乘积
array([[308, 440],
       [660, 968]])
>>>
```

2.3.4 线性代数

5.linalg.det()函数

linalg.det()函数用于计算输入矩阵的行列式。

```
>>> import numpy as np
>>> array1=np.array([[1,3],[5,7]])                    #创建一个二维数组
>>> array1                          #查看创建的二维数组
array([[1, 3],
       [5, 7]])
>>> np.linalg.det(array1)#使用linalg.det()函数求数组的行列式
-7.999999999999998
>>>
```

2.3.4 线性代数

6.linalg.solve()函数

linalg.solve()函数用于求矩阵形式的线性方程的解。

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 5 \\ 2 & 5 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 6 \\ -4 \\ 27 \end{bmatrix}$$

```
>>> import numpy as np
>>> array1=np.array([[1,1,1],[0,2,5],[2,5,-1]]) #创建一个二维数组
>>> array1                    #查看创建的二维数组
array([[ 1,  1,  1],
       [ 0,  2,  5],
       [ 2,  5, -1]])
>>> array2=np.array([[6],[-4],[27]])            #创建一个二维数组
>>> array2                    #查看创建的二维数组
array([[ 6],
       [-4],
       [27]])
>>> np.linalg.solve(array1,array2)
array([[ 5.],
       [ 3.],
       [-2.]])
>>>
```

2.3.4 线性代数

7.linalg.inv()函数

linalg.inv()函数用于计算矩阵的乘法逆矩阵。逆矩阵定义：设A是数域上的一个n阶矩阵，若在相同数域上存在另一个n阶矩阵B，使得：AB=BA=E，则称B是A的逆矩阵，而A被称为可逆矩阵。注意，E为单位矩阵。

```
>>> import numpy as np
>>> array1=np.array([[1,2],[3,4]])                    #创建一个二维数组
>>> array1                  #查看创建的二维数组
array([[1, 2],
       [3, 4]])
>>> np.linalg.inv(array1)#使用linalg.inv()函数求数组求逆运算
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
>>>
```

2.4 NumPy使用案例

1.学生成绩数组输入

```
>>> import numpy as np
>>> #创建学生成绩数组，第一行为平时成绩，第二行为期末成绩，第三行为总分
>>> array1=np.array([[88,90,94,90,85],[93.5,93.5,92.5,90.5,90.5],[0,0,0,0,0]])
>>> array1      #查看创建的学生成绩数组
array([[88. , 90. , 94. , 90. , 85. ],
       [93.5, 93.5, 92.5, 90.5, 90.5],
       [ 0. ,  0. ,  0. ,  0. ,  0. ]])
```

2.学生成绩数组属性查看

```
>>> #查看数组array1相关属性
>>> array1.ndim
2
>>> array1.shape
(3, 5)
>>> array1.size
15
>>> array1.dtype
dtype('float64')
>>> array1.itemsize
8
>>> array1.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False
```

3.学生成绩数组统计运算

```
>>> #查看学生期末考试成绩
>>> array1[1]
array([93.5, 93.5, 92.5, 90.5, 90.5])
>>> #修改单一学生成绩
>>> array1[1,1]=90
>>> array1
array([[88. , 90. , 94. , 90. , 85. ],
       [93.5, 90. , 92.5, 90.5, 90.5],
       [ 0. ,  0. ,  0. ,  0. ,  0. ]])
>>> #整体修改学生成绩
>>> array2=array1+2
>>> array2
array([[90. , 92. , 96. , 92. , 87. ],
       [95.5, 92. , 94.5, 92.5, 92.5],
       [ 2. ,  2. ,  2. ,  2. ,  2. ]])
>>> #计算学生成绩总分
>>> array2[2]=array2[0]*0.5+array2[1]*0.5
>>> array2
array([[90.  , 92.  , 96.  , 92.  , 87.  ],
       [95.5 , 92.  , 94.5 , 92.5 , 92.5 ],
       [92.75, 92.  , 95.25, 92.25, 89.75]])
>>> #学生成绩总分中位数
>>> np.median(array2[2])
92.25
>>> #学生成绩总分平均分
>>> np.average(array2[2])
92.4
>>> #学生成绩总分标准差与方差
>>> np.std(array2[2])
1.7578395831246945
>>> np.var(array2[2])
3.09
>>>
```

Thank You!