



NumPy数值计算

NumPy数值计算

Python

5

线性代数

NumPy库提供用于矩阵乘法的dot函数。另外NumPy库的linalg模块来完成具有线性代数运算方法。

函数	描述
dot	两个数组的点积
vdot	两个向量的点积
det	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

1、dot()

- 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之和；
- 对于二维数组，计算的是两个数组的矩阵乘积；
- **二维数组和一维数组的点积**，则将一维数组转置后计算矩阵乘积，**最后结果还**
原成向量。

`numpy.dot(a, b, out=None)` 或 `a.dot(b,out=None)`

参数说明：

- `a : ndarray` 数组
- `b : ndarray` 数组
- `out : ndarray`, 可选，用来保存dot()的计算结果，**必须与乘积结果的维度、长度、类型一致。**

[示例] 一维数组点积

```
import numpy as np
```

```
x = np.arange(1,5)      # [1 2 3 4]
```

```
y = np.arange(2,10,2)   # [2,4,6,8]
```

```
print('dot(x)=', np.dot(x,y))  # 1*2+2*4+3*6+4*8=60
```

dot(x)= 60

[示例] 二维数组矩阵乘积

```
import numpy as np

a = np.array([[1.,2.],[3.,4.]])
b = np.array([[5.,6.],[7.,8.]])
c = np.dot(a, b)
print('dot(a,b)=\n', c)
```

dot(a,b)=
[[19. 22.]
[43. 50.]]

$$\begin{bmatrix} 1. & 2. \\ 3. & 4. \end{bmatrix} \times \begin{bmatrix} 5. & 6. \\ 7. & 8. \end{bmatrix}$$

[示例] 二维数组和一维数组乘积 (线性回归模型的预测)

线性回归模型通常表示为: $\hat{y} = w_1x_1 + w_2x_2 + \cdots + w_mx_m + b$

```
import numpy as np
```

```
X = np.array([[1, 2], [3, 4], [5, 6]]) # 输入特征矩阵
```

```
w = np.array([0.5, 0.6]) # 权重向量
```

```
b = 0.1 # 偏置项
```

```
dot_product = np.dot(X, w) # 计算点积
```

```
predictions = dot_product + b # 加上偏置项
```

```
print("预测值: ", predictions)
```

x=

[[1 2]

[3 4]

[5 6]]

y= [0.5 0.6]

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 0.5 \\ 0.6 \end{bmatrix} = \begin{bmatrix} 1.7 \\ 3.9 \\ 6.1 \end{bmatrix}$$

还原成向量(一维数组)

[1.7 3.9 6.1]

2、vdot()

计算 两个数组的展平 (Flatten) 后的点积。

如果**第一个参数**是复数，则将第一个参数的**复共轭**用于点积的计算。

```
import numpy as np  
  
a = np.array([[1.,2.],[3.,4.]])  
b = np.array([[5.,6.],[7.,8.]])  
print (np.vdot(a,b))    # 1.*5. + 2.*6. + 3.*7. + 4.*8. = 70
```

70.0

[示例] 复数点积

```
import numpy as np

va = 2 + 3j
vb = np.array([2]) # 或 vb = 2

p2 = np.vdot(va, vb)
print("复数点积: ", p2)
```

复数点积: (4-6j)

```
import numpy as np

va = 2 + 3j
vb = 4 + 5j

p1 = np.vdot(va, vb)
print("复数点积: ", p1)
```

复数点积: (23-2j)

2+3j的共轭复数为: 2-3j
 $(2-3j) \cdot (4+5j)$
 $= 2 \cdot 4 + 2 \cdot 5j - 3j \cdot 4 - 3j \cdot 5j$
 $= 8 + 10j - 12j + 15$
 $= 23 - 2j$

3、 linalg.det()

返回矩阵的行列式。

行列式最直接的用途是判断一个方阵是否可逆（即是否存在逆矩阵）：

行列式 $\neq 0 \rightarrow$ 矩阵可逆 (非奇异矩阵)

行列式 = 0 \rightarrow 矩阵不可逆 (奇异矩阵)

```
import numpy as np
a = np.array([[1,2],[3,4]])
b = np.linalg.det(a)
print(b)    # 1*4 - 2*3 = -2
```

-2.00000000000000004

$$\begin{vmatrix} a1 & b1 \\ a2 & b2 \end{vmatrix} = a1*b2-a2*b1$$

4、linalg.solve()

求解线性方程组。

$$\begin{cases} x + y + z = 6 \\ 2y + 5z = -4 \\ 2x + 5y - z = 27 \end{cases}$$

```
import numpy as np
```

```
A = np.array([[1,1,1],[0,2,5],[2,5,-1]])
```

```
b = np.array([6,-4,27])
```

```
x, y, z = np.linalg.solve(A,b)
```

```
print('方程解: x={},y={},z={}'.format(x,y,z))
```

方程解: 方程解: x=5.0,y=3.0,z=-2.0

5、linalg.inv()

计算方阵的乘法逆矩阵 ($A*B=I$) 。

注意：如果矩阵是奇异的或者非方阵，使用inv函数求逆矩阵，会出现错误。

```
import numpy as np
a = np.array([[1,2],[3,4]])
b = np.linalg.inv(a)
print('a=', a)
print('b=', b)
np.set_printoptions(precision=0)
np.set_printoptions(suppress=True)
print('dot(a,b)=', np.dot(a,b))
```

```
a= [[1 2]
     [3 4]]
```

```
b= [[-2.   1.]
     [ 1.5 -0.5]]
```

```
dot(a,b)= [[1.  0.]
            [0.  1.]]
```

逆矩阵点积求解线性方程组：

$$\begin{cases} x + y + z = 6 \\ 2y + 5z = -4 \\ 2x + 5y - z = 27 \end{cases}$$

```
import numpy as np
```

```
A = np.array([[1,1,1],[0,2,5],[2,5,-1]])
```

```
b = np.array([6,-4,27])
```

```
x, y, z = np.linalg.inv(A).dot(b) #  $x = A^{-1} \cdot b$ 
```

```
print('方程解： x={:.1f},y={:.1f},z={:.1f}'.format(x,y,z))
```

方程解： 方程解： $x=5.0, y=3.0, z=-2.0$

补充: `numpy.linalg.pinv()`

如果A不是方阵或者不可逆, 求其**伪逆矩阵**。

```
import numpy as np

a = np.array([[1,2,3],[3,4,5]])
b = np.linalg.pinv(a)
print('a=', a)
print('b=', b)
np.set_printoptions(precision=0)
np.set_printoptions(suppress=True)
print('dot(a,b)=', np.dot(a,b))
```

```
a= [[1 2 3]
     [3 4 5]]
```

```
b= [[-1.16666667  0.66666667]
     [-0.16666667  0.16666667]
     [ 0.83333333 -0.33333333]]
```

```
dot(a,b)= [[ 1.  0.]
            [-0.  1.]]
```

伪逆矩阵点积求解线性方程组：

$$\begin{cases} x_1 + x_2 + x_3 + x_4 + x_5 = 7 \\ 3x_1 + x_2 + 2x_3 + x_4 - 3x_5 = -4 \\ 2x_2 + x_3 + 2x_4 + 6x_5 = 23 \end{cases}$$

```
import numpy as np
```

```
a = np.array([[1,1,1,1,1],[3,1,2,1,-3],[0,2,1,2,6]])
```

```
b = np.array([7,-2,23])
```

```
c = np.linalg.pinv(a)
```

```
x = np.dot(c,b)
```

```
print('x=', x)
```

```
x= [0.59615385  1.32692308  0.96153846  1.32692308  2.78846154]
```

6、linalg.eigvals()和linalg.eig()

计算特征值时，我们可以用numpy.linalg程序包提供的eigvals()函数eig()函数，其中函数eigvals返回矩阵的特征向量， eig函数返回一个元组，其元素为特征值和特征向量（列向量）。

设 A 是 n 阶方阵，若存在数 λ 和非零向量 x ，
使得 $Ax = \lambda x$ ($x \neq 0$)

则称 λ 是 A 的一个特征值，

x 为 A 的对应于特征值 λ 的特征向量。

稳态分布 v 满足：

$$Pv = v$$

$$Pv = 1 \cdot v$$

因此：

稳态分布 v 就是 P 的特征值 $\lambda=1$ 对应的特征向量。

[示例] 特征值和特征向量

```
A = np.array([[1,2],[3,4]])  
print('A=', A)
```

```
lambda_A = np.linalg.eigvals(A)  
print('lambda_A=', lambda_A)
```

```
lambda_A, x = np.linalg.eig(A)  
print("lambda_A=", lambda_A)  
print("x=", x)
```

```
a = np.dot(A,x)  
print(a)
```

```
b = lambda_A*x  
print(b)
```

```
A= [[1 2]  
     [3 4]]
```

```
lambda_A= [-0.37228132  5.37228132]
```

```
lambda_A= [-0.37228132  5.37228132]
```

```
x= [[-0.82456484 -0.41597356]  
     [ 0.56576746 -0.90937671]]
```

```
[[ 0.30697009 -2.23472698]  
 [-0.21062466 -4.88542751]]
```

```
[[ 0.30697009 -2.23472698]  
 [-0.21062466 -4.88542751]]
```

[示例]分析一个简单的人口迁移模型，预测长期人口分布。

```
import numpy as np
```

```
# 定义迁移矩阵:
```

```
P = np.array([[0.90, 0.05],      # 每年有90%的人留在城市，10%迁往农村  
              [0.10, 0.95]])    # 每年有5%的农村人迁往城市，95%留在农村
```

```
evs, evt = np.linalg.eig(P)
```

```
print("特征值:", evs)
```

```
print("特征向量:\n", evt)
```

```
# 找到特征值为1对应的特征向量 (稳态分布)
```

```
s = evt[:, np.isclose(evs, 1)]
```

```
s = s[:, 0] # 取第一列
```

```
# 归一化得到概率分布
```

```
g = s / np.sum(s)
```

```
print("\n长期人口分布: 城市 {:.2f}, 农村 {:.2f}".format(g[0], g[1]))
```

特征值: [0.85 1.]

特征向量:

$\begin{bmatrix} -0.70710678 & -0.4472136 \end{bmatrix}$

$\begin{bmatrix} 0.70710678 & -0.89442719 \end{bmatrix}$

长期人口分布: 城市 0.33, 农村 0.67

NumPy数值计算

Python

6

数组的存取

- Numpy中通过`np.savetxt`方法对数组进行存储

numpy.savetxt(

fname, **# 文件名或文件句柄**

X, **# 要保存的数组**

fmt='%.18e', **# 数据格式**

delimiter=' ', **# 列分隔符**

newline='\n', **# 行分隔符**

header='', **# 文件头注释**

footer='', **# 文件尾注释**

comments='# ', **# 注释符号**

encoding=None **# 文件编码**

)

[示例] 数组存储

```
import numpy as np
data=np.arange(50).reshape(5,10)
np.savetxt('e:/data_txt.txt', data, fmt='%d')
np.savetxt('e:/floatdata_txt.txt', data, fmt='%.2f')
np.savetxt('e:/data_csv.csv', data, fmt='%d',delimiter=',')
```

data_csv.csv - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
0,1,2,3,4,5,6,7,8,9
10,11,12,13,14,15,16,17,18,19
20,21,22,23,24,25,26,27,28,29
30,31,32,33,34,35,36,37,38,39
40,41,42,43,44,45,46,47,48,49
```

intdata_txt.txt - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
```


floatdata_txt.txt - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
0.00 1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00 9.00
10.00 11.00 12.00 13.00 14.00 15.00 16.00 17.00 18.00 19.00
20.00 21.00 22.00 23.00 24.00 25.00 26.00 27.00 28.00 29.00
30.00 31.00 32.00 33.00 34.00 35.00 36.00 37.00 38.00 39.00
40.00 41.00 42.00 43.00 44.00 45.00 46.00 47.00 48.00 49.00
```

[示例] 数组存储

```
import numpy as np
dt = [('id', int), ('name', 'U10'),('score', float)]
data = np.array([
    (1001, "张三", 85.5),
    (1002, "李四", 92.0),
    (1003, "王五", 78.5),
    (1004, "吴六", 88.0)
],dtype=dt)
np.savetxt("e:/formatted.txt",
    data,
    fmt="%d\t| %s\t| %.2f\t",
    delimiter="|",
    header="学号\t姓名\t分数", # 文件头
    comments="", # 禁用注释符号
    encoding="utf-8") # UTF-8编码
```

 formatted.txt - 记事本

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)
学号	姓名	分数		
1001	张三	85.50		
1002	李四	92.00		
1003	王五	78.50		
1004	吴六	88.00		

- Numpy中通过`np.loadtxt`方法对存储数组文件进行读取，并将其**加载到一个数组中**。

```
numpy.loadtxt(fname, dtype=np.float, delimiter=None)
```

函数中参数说明如下：

fname: 文件名

dtype: 文件中读入的数据以什么类型返回，默认为 `np.float` 类型

delimiter: 分隔符，缺省时为空

[示例] 数组存取

```
loaded_data = np.loadtxt("e:/formatted.txt",  
                          dtype=dt,  
                          delimiter="|",  
                          skiprows=1,    # 跳过标题行  
                          encoding="utf-8")
```

提取所有分数

```
scores = loaded_data['score']  
print("所有分数:", scores)
```

所有分数: [85.5 92. 78.5 88.]

 formatted.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

学号	姓名	分数
1001	张三	85.50
1002	李四	92.00
1003	王五	78.50
1004	吴六	88.00

Python

7

练习题

如何从数组np.arange(15)中提取5到10之间的所有数字？

```
import numpy as np

a = np.arange(15)
b = a[(a>5) & (a<11)] # &两边的条件表达式一定要加括号
print(b)
```

[6 7 8 9 10]

如何交换数组np.arange(9).reshape(3,3)中的第0列和第1列?

```
import numpy as np

a = np.arange(9).reshape(3,3)
print(a)
print('*****')
a = a[:, [1,0,2] ]
print(a)
```

[[0 1 2]

[3 4 5]

[6 7 8]]

[[1 0 2]

[4 3 5]

[7 6 8]]

如何交换数组np.arange(9).reshape(3,3)中的第0行和第1行

```
import numpy as np

a = np.arange(9).reshape(3,3)
print(a)
print('*****')
a = a[[1,0,2], :]
print(a)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
*****
[[3 4 5]
 [0 1 2]
 [6 7 8]]
```

将用户的缺失评分数据用其平均评分填充。

```
import numpy as np
```

```
# 创建含NaN的数组
```

```
data = np.array([[3, 2, 3,np.nan], [4, 5,np.nan,  
5],[np.nan,3,3,5]])
```

```
print(data)
```

```
# 用均值填充NaN
```

```
mean_val = np.nanmean(data,axis=0) # 忽略NaN计算均值
```

```
cleaned_data = np.where(np.isnan(data), mean_val, data)
```

```
print(cleaned_data)
```

```
[[ 3.  2.  3. nan]
```

```
 [ 4.  5. nan  5.]
```

```
[nan  3.  3.  5.]]
```

```
[[3.  2.  3.  5.]
```

```
 [4.  5.  3.  5.]
```

```
[3.5  3.  3.  5.]]
```

如何查找1,2,6,2,3,4,6,4,5,1中第二大的值?

```
import numpy as np

a = np.array([1,2,6,2,3,4,6,4,5,1])
b = np.sort(a)
print(b)
print(b[-2])
c = np.unique(a)
print(c)
print(c[-2])
```

[1 1 2 2 3 4 4 5 6 6]

6

[1 2 3 4 5 6]

5

我们收集了用户a、b、c对10件商品的评分，计算a、b和a、c之间的欧式距离，以判断b和c中谁与a的评分更接近？

```
import numpy as np

a = np.array([5,5,3,4,2,4,3,4,5,5])
b = np.array([4,5,3,3,3,3,5,5,4,5])
c = np.array([2,3,1,3,2,5,5,3,2,1])

dab = np.sqrt(np.sum((a-b)**2))
dac = np.sqrt(np.sum((a-c)**2))

print(f"dab={dab:.2f},dac={dac:.2f}")
```

$$dist(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

<http://blog.csdn.net/linvo>

dab=3.16, dac=7.00

如何知道数组np.array([7,2,10,2,7,4,9,4,9,8])中的第二大值是多少？

```
import numpy as np
```

```
a = np.array([7,2,10,2,7,4,9,4,10,8])
```

```
a = np.unique(a)
```

```
print(a)
```

```
print(a[-2])
```

[2 4 7 8 9 10]

9

投票箱中有11张票，票上标记了被选人序号，分别为：
7,2,10,2,7,2,4,9,4,9,8，得票数最高的人是谁？

```
import numpy as np

a = np.array([7,2,10,7,7,2,4,2,4,7,8])
b, n = np.unique(a,return_counts=True)
print(b)
print(n)
max = np.argmax(n)
print(b[max])
```

```
[ 2  4  7  8 10]
[3 2 4 1 1]
7
```

END