第8课　import numpy as np

numpy

- 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。

- 学习numpy可以为未来的学习打下基础：
  - Matplotlib （Python 的绘图库）
  - SciPy （Python 算法库和数学工具包）
  - Pandas （数据分析）
  - 机器学习

- 根据Python社区的习惯，都是用下面的方式来导入numpy

```
import numpy as np
```

- 官网 https://numpy.org/

- 开源代码 https://github.com/numpy/numpy

numpy

有了numpy，不需要再…

1. **矩阵**：在 Python 中，我们可以使用列表的列表（a list of lists）来存储矩阵，每个内部列表代表矩阵一行。例如，我们可以用

M = [[5, 6, 7],
[0, -3, 5]]

来存储矩阵

$$\begin{pmatrix} 5 & 6 & 7 \\ 0 & -3 & 5 \end{pmatrix}$$

我们可以使用 M[1]来访问矩阵的第二行（即[0, -3, 4]），也可以使用 M[1][2]来访问矩阵的第二行的第三项（即 5）。

(a) 编写函数 *matrix_dim(M)*，该函数输入上面格式的矩阵 M，返回矩阵 M 的维度。例如, matrix_dim([[1,2],[3,4],[5,6]]) 返回 [3, 2]。

(b) 编写函数 *mult_M_v(M, v)*，返回 $n \times m$ 矩阵 M 和 $m \times 1$ 向量 v 的乘积。

(c) 编写函数 *transpose(M)*，返回矩阵的转置。

(d) 编写函数 *largest_col_sum(M)*，寻找矩阵 M 中元素总和最大的列，如上面矩阵中第三列元素的总和最大，则返回 12（7+5=12）。

(e) 编写函数 *switch_columns(M, i, j)*，交换矩阵 M 的第 i 列和第 j 列，返回新的矩阵 M。

(f) 把以上函数（a-e）写进模块 matrix.py。编写 test_matrix.py 调用模块 matrix 并测试函数（a-e）。

注意：numpy毕竟是第三方库，有它的bias。

Numpy： 注意

- numpy毕竟是第三方库，有它设计的bias：

```
array([[ 0.],
       [ 0.],
       [ 0.]])
```

```
array([[ 0.,  0.,  0.]])
```

```
>> a=[0;0;0]

a =

     0
     0
     0
```

```
>> a=[0, 0, 0]

a =

     0     0     0
```

- 喜欢numpy：遵守numpy的规则，仔细查询每个方法的用法；
- 不喜欢numpy/满足不了你的功能：编写自己的numpy!
- 合理运用第三方库/编写属于自己的库：都是能力的体现

numpy: 生成数组

```
>>> np.array([1, 2, 3, 4, 5])
array([1, 2, 3, 4, 5])

>>> np.array(range(5))
array([0, 1, 2, 3, 4])

>>> np.array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3],
       [4, 5, 6]])

>>> np.linspace(0, 100, 11) #从0到100，生成等差的11个数
array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,
   80.,  90., 100.])
>>> np.arange(0, 100, 11) #类似range(0,100,11)
array([ 0, 11, 22, 33, 44, 55, 66, 77, 88, 99])
```

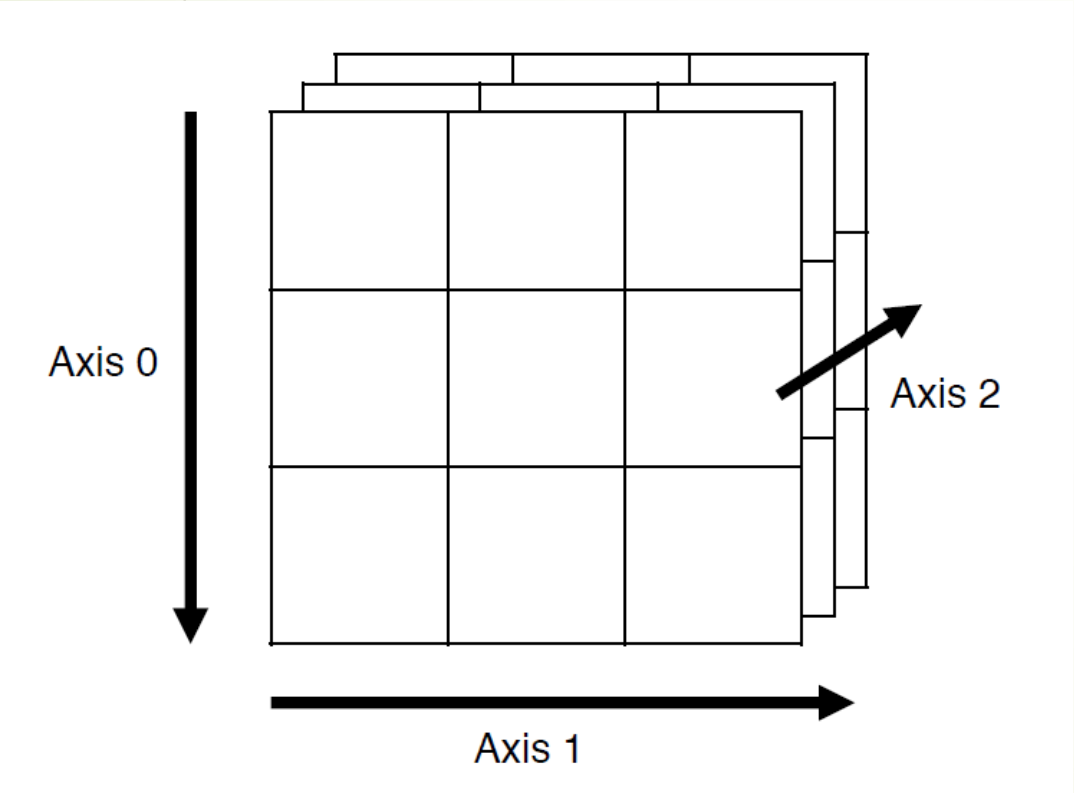
numpy:生成数组

```
>>> np.zeros((2,2))                    #全0二维数组
array([[0., 0.],
       [0., 0.]])
>>> np.zeros((3,1))                    #全0一维数组,列向量
array([[ 0.],
     [ 0.],
     [ 0.]])
>>> np.zeros((1,3))                   #全0一维数组,行向量
array([[ 0.,  0.,  0.]])
>>> np.ones((2,2))                     #全1二维数组
array([[1., 1.],
       [1., 1.]])
>>> np.identity(2)                      #单位矩阵
array([[ 1.,  0.],
     [ 0.,  1.]])
```

numpy:数组与数值的运算

```
>>> x = np.array((1, 2, 3, 4, 5))
>>> x
array([1, 2, 3, 4, 5])

>>> x + 2
array([3, 4, 5, 6, 7])
>>> x * 2
array([ 2, 4, 6, 8, 10])
>>> x / 2
array([ 0.5, 1. , 1.5, 2. , 2.5])
>>> x // 2
array([0, 1, 1, 2, 2], dtype=int32)

>>> x ** 3
array([1, 8, 27, 64, 125], dtype=int32)
>>> x % 3
array([1, 2, 0, 1, 2], dtype=int32)
```

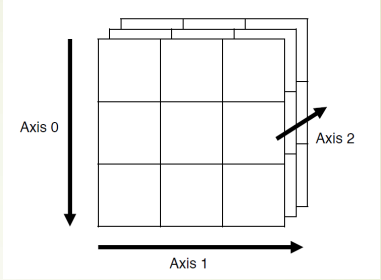
numpy:维度



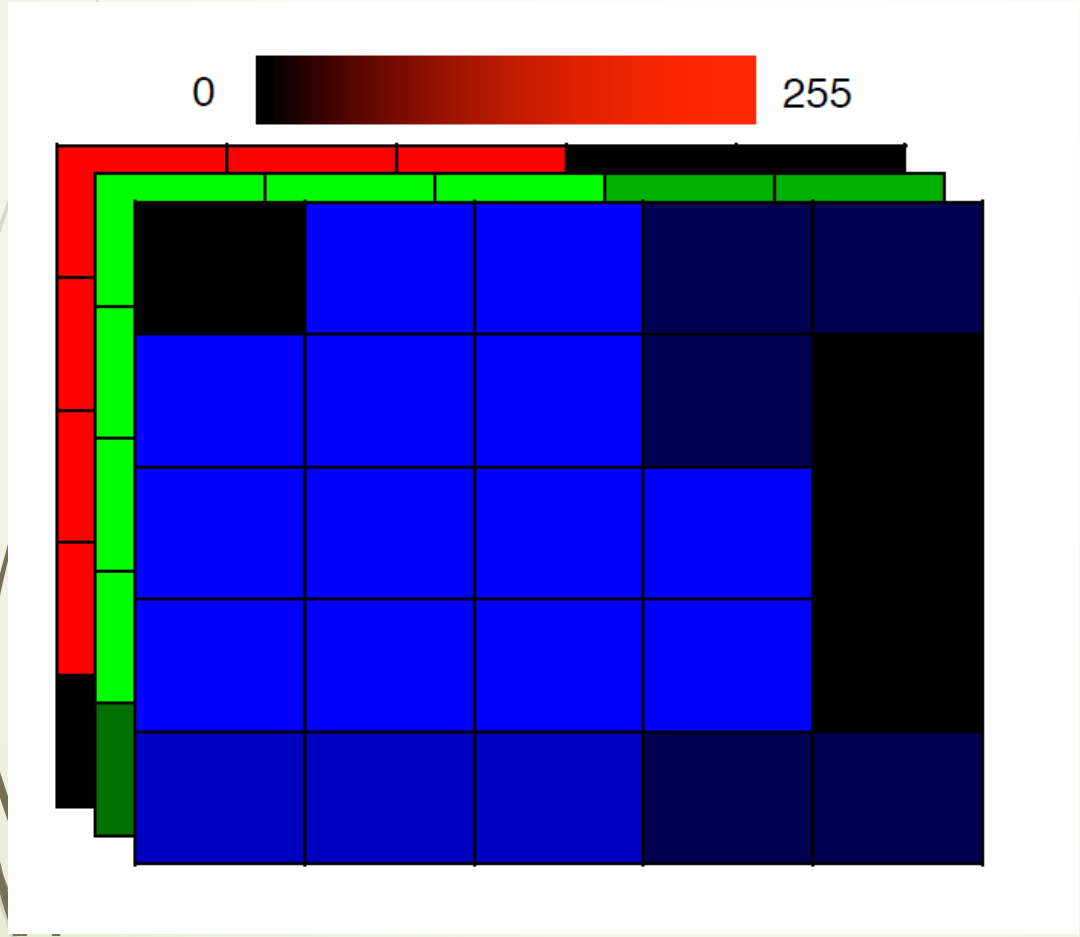

```
arr.shape
(3, 3, 3)
Axis (0, 1, 2)
```

Shape返回一个元组，由每个轴上的元素个数组成。

numpy:维度



应用：图片，每个像素（RGB）




```
arr.shape
（5，5，3）
```

红色（255，0，0）
绿色（0，255，0）
蓝色（0，0，255）

青色（0，255，255）
黄色（255，255，0）
洋红色（255，0，255）

白色（255，255，255）
黑色（0，0，0）

numpy:数组与数组的运算

```
#按元素一个一个加法  array1 + array2
np.add(array1, array2)
#按元素一个一个减法  array1 - array2
np.subtract(array1, array2)
#按元素一个一个乘法  array1 * array 2
np.multiply(array1, array2)
#按元素一个一个除法  array1 / array2
np.divide(array1, array2)
```

- array1和array2必须是相同的形状shape （或者可以广播成相同的形状shape ），才能正常运行。
- 什么是可广播的 （*broadcastable*） ？？

numpy:数组与数组的运算

$$\begin{bmatrix} 5 & 1 \\ 2 & 0 \\ 4 & 3 \end{bmatrix} + \begin{bmatrix} 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 2 & 0 \\ 4 & 3 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 6 & 0 \\ 3 & -1 \\ 5 & 2 \end{bmatrix}$$

这里，我们说第二个矩阵已经沿着轴0广播，以使维度对齐

numpy:数组与数组的运算

```
>>> a = np.array([1, 2, 3])
>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> c = a * b
>>> c
array([[ 1,  4,  9],
       [ 4, 10, 18],
       [ 7, 16, 27]])
>>> c / b
array([[ 1.,  2.,  3.],
       [ 1.,  2.,  3.],
       [ 1.,  2.,  3.]])
>>> c / a
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 7.,  8.,  9.]])
```

numpy:数组与数组的运算

- 广播例子

```
>>> a = np.arange(0,60,10).reshape(-1,1)          #列向量
>>> b = np.arange(0,6)                            #行向量
>>> a
array([[ 0],
       [10],
       [20],
       [30],
       [40],
       [50]])
>>> b
array([0, 1, 2, 3, 4, 5])

>>>a+b???
>>>a*b???
```

numpy:数组与数组的运算

```
>>> a + b
array([[ 0,  1,  2,  3,  4,  5],
       [10, 11, 12, 13, 14, 15],
       [20, 21, 22, 23, 24, 25],
       [30, 31, 32, 33, 34, 35],
       [40, 41, 42, 43, 44, 45],
       [50, 51, 52, 53, 54, 55]])
>>> a * b
array([[  0,   0,   0,   0,   0,   0],
       [  0,  10,  20,  30,  40,  50],
       [  0,  20,  40,  60,  80, 100],
       [  0,  30,  60,  90,  120, 150],
       [  0,  40,  80,  120, 160, 200],
       [  0,  50,  100, 150,  200, 250]])
```

```
a
array([[0,0,0,0,0,0],
       [10,10,10,10,10,10],
       [20,20,20,20,20,20],
       [30,30,30,30,30,30],
       [40,40,40,40,40,40],
       [50,50,50,50,50,50]])
b
array([[0, 1, 2, 3, 4, 5],
       [0, 1, 2, 3, 4, 5],
       [0, 1, 2, 3, 4, 5],
       [0, 1, 2, 3, 4, 5],
       [0, 1, 2, 3, 4, 5],
       [0, 1, 2, 3, 4, 5]])
```

numpy:数组与数组的运算

什么时候可以广播？

思考以下**A**和**B**能否使用+ − * /

- $\mathbf{A} \in \mathbb{R}^{3x2}, \mathbf{B} \in \mathbb{R}^{1x2}$
- $\mathbf{A} \in \mathbb{R}^{2x5}, \mathbf{B} \in \mathbb{R}^{3x5}$
- $\mathbf{A} \in \mathbb{R}^{1x3x4x2}, \mathbf{B} \in \mathbb{R}^{5x1x4x1}$
- $\mathbf{A} \in \mathbb{R}^{4x3}, \mathbf{B} \in \mathbb{R}^{5x4x3}$

numpy:数组与数组的运算

A 和 B 在 i 轴上:（1） 必须相等, 或者 （2）至少一个是1维

思考以下**A**和**B**能否使用+ － * /

- $\mathbf{A} \in \mathbb{R}^{3x2}, \mathbf{B} \in \mathbb{R}^{1x2}$

  结果是 $\mathbb{R}^{3x2}$

- $\mathbf{A} \in \mathbb{R}^{2x5}, \mathbf{B} \in \mathbb{R}^{3x5}$

  不行！ operands could not be broadcast together with shapes (2,5) (3,5)

- $\mathbf{A} \in \mathbb{R}^{1x3x4x2}, \mathbf{B} \in \mathbb{R}^{5x1x4x1}$

  结果是 $\mathbb{R}^{5x3x4x2}$

- $\mathbf{A} \in \mathbb{R}^{4x3}, \mathbf{B} \in \mathbb{R}^{5x4x3}$

  结果是 $\mathbb{R}^{5x4x3}$，因为numpy将A看作$\mathbb{R}^{1x4x3}$

numpy:转置

```
>>> b = np.array(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
>>> b
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> b.T                          #转置
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

注意：一维数组无效，应使用a.reshape(a.shape[0],1)

```
>>> a = np.array((1, 2, 3, 4))
>>> a
array([1, 2, 3, 4])
>>> a.T
array([1, 2, 3, 4])
```

numpy:点积

```
>>> a = np.array((1, 2, 3))
>>> b = np.array((2, 3, 4))
>>> a.dot(b)
20
>>> np.dot(a,b)
20
>>> c = np.array(([1,2,3],[4,5,6],[7,8,9]))
>>> c
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> c.dot(a)
array([14, 32, 50])
>>> a.dot(c)
array([30, 36, 42])
>>> c[0].dot(a)
14
```

```
[1,2,3]   [1]
[4,5,6] x [2]
[7,8,9]   [3]
```

```
                [1,2,3]
[1,2,3] x       [4,5,6]
                [7,8,9]
```

numpy:访问元素

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


>>> x=np.linspace(0,100,11)
>>> x
array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,
70.,  80.,  90., 100.])
>>> x[[2,3,7]]
array([20., 30., 70.])
```

numpy:数学计算

```
>>> x=np.linspace(0,100,11)
>>> np.sin(x)
array([ 0.        , -0.54402111,  0.91294525, -
   0.98803162,  0.74511316,
       -0.26237485, -0.30481062,  0.77389068, -
   0.99388865,  0.89399666, -0.50636564])

>>> b = np.array(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
>>> c=np.cos(b)
array([[ 0.54030231, -0.41614684, -0.9899925 ],
       [-0.65364362,  0.28366219,  0.96017029],
       [ 0.75390225, -0.14550003, -0.91113026]])

>>> np.round(c)
array([[ 1., -0., -1.],
       [-1.,  0.,  1.],
       [ 1., -0., -1.]])
```

numpy:不同维度

```
>>> x = np.arange(0,10).reshape(2,5)   #一维数组变二维数组
>>> x
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> np.sum(x)                          #数组中所有元素求和
45
>>> np.sum(x, axis=0)                  #数组纵向求和
array([ 5,  7,  9, 11, 13])
>>> np.sum(x, axis=1)                  #数组横向求和
array([10, 35])
>>> np.mean(x, axis=0)                 #数组纵向计算平均值
array([ 2.5,  3.5,  4.5,  5.5,  6.5])

>>> weight = [0.3, 0.7]                          #权重
>>> np.average(x, axis=0, weights=weight) #数组纵向计算加权平均值
array([ 3.5,  4.5,  5.5,  6.5,  7.5])
```

numpy:不同维度

```
>>> x
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> np.max(x)                          #所有元素中最大值
9
>>> np.max(x, axis=0)                  #每列元素的最大值
array([5, 6, 7, 8, 9])
>>> np.std(x)                          #所有元素标准差
2.8722813232690143
>>> np.std(x, axis=1)                  #每行元素的标准差
array([1.41421356, 1.41421356])
```

- numpy设计的bias：既然是求每行的标准差，那为什么返回行向量，而不是列向量？
- 有什么办法解决？（课后思考；查看numpy文档）

numpy:矩阵的不同维度

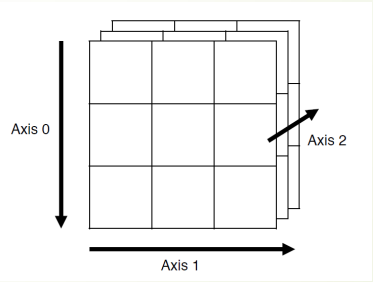
```
>>> x
array([[0, 2, 4, 1, 3],
       [6, 5, 9, 8, 7]])
>>> np.var(x, axis=0)                    #每列元素的方差
array([ 9.  ,  2.25,  6.25, 12.25,  4.  ])

>>> np.sort(x, axis=0)                   #纵向排序
array([[0, 2, 4, 1, 3],
       [6, 5, 9, 8, 7]])

>>> np.sort(x, axis=1)                   #横向排序
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

numpy:改变数组大小

```
>>> a = np.arange(1, 11, 1)
>>> a
array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> a.shape = 2, 5                          #改为2行5列
>>> a
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
>>> a.shape = 5, -1                         #-1表示自动计算
>>> a
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10]])

>>> b = a.reshape(2,5)
>>> b
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

numpy:切片

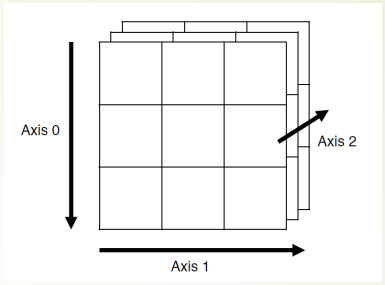
```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[::-1]                                    #倒序
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
>>> a[::2]                                     #隔一个取一个元素
array([0, 2, 4, 6, 8])

>>> c=a.reshape(2,5)
>>> c
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> c[:,2:5:2]
array([[2, 4],
       [7, 9]])
```



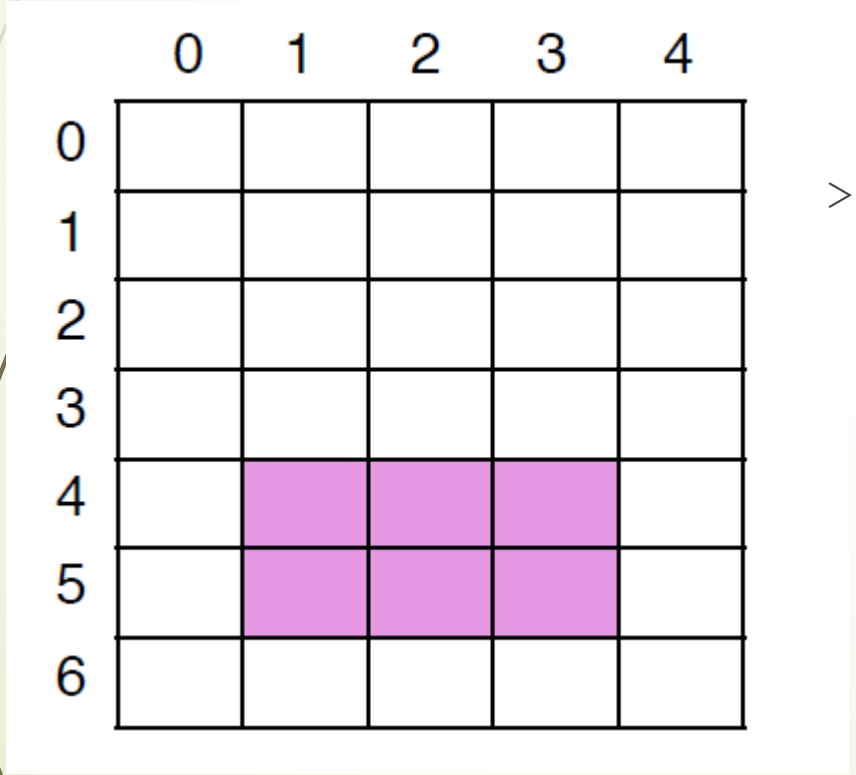
冒号表示我们从轴0获取所有元素

```
arr[index_axis_0, index_axis_1, ... index_axis_N]
```

numpy:切片




```
arr[index_axis_0, index_axis_1, ... index_axis_N]
```




```
>>> segment = arr [4:6, 1:4]
```

记住，和之前学过的切片一样：

（1）一维变二维

（2）包含下界

（3）不包含上界

numpy:定义简单函数

- 分段函数

```
>>> x = np.random.randint(0, 10, size=(1,10))
>>> x
array([[3, 6, 8, 7, 0, 0, 8, 8, 2, 7]])
>>> np.where(x<5, 0, 1)    #小于5的元素值对应0，其他对应1
array([[0, 1, 1, 1, 0, 0, 1, 1, 0, 1]])

#小于4的元素乘以2，大于7的元素乘以3，其他元素变为0
>>> np.piecewise(x,[x<4, x>7],[lambda x:x*2,lambda x:x*3])
array([[ 6,  0, 24,  0,  0,  0, 24, 24,  4,  0]])
```

numpy: 矩阵对象

```
>>> a_list = [3, 5, 7]
>>> a_mat = np.matrix(a_list)
>>> a_mat
matrix([[3, 5, 7]])
>>> a_mat.T                                    #矩阵转置
matrix([[3],
        [5],
        [7]])
>>> a_mat.shape                                #矩阵形状
(1, 3)
>>> a_mat.size
3
```

numpy: 矩阵对象

```
>>> c_mat = np.matrix([[1, 5, 3], [2, 9, 6]]) #创建二维矩阵
>>> c_mat
matrix([[1, 5, 3],
        [2, 9, 6]])
>>> c_mat.argsort(axis=0)                      #纵向排序后的元素序号
matrix([[0, 0, 0],
        [1, 1, 1]], dtype=int64)
>>> c_mat.argsort(axis=1)                      #横向排序后的元素序号
matrix([[0, 2, 1],
        [0, 2, 1]], dtype=int64)
>>> d_mat = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

>>> d_mat.diagonal()                           #矩阵对角线元素
matrix([[1, 5, 9]])
```

numpy:线性代数

```
np.linalg.norm(A) # 求范数
np.linalg.eig(A) # Tuple: 求特征值，特征向量
np.linalg.det(A) #求行列式
np.linalg.matrix_rank(A) # 求秩
np.linalg.qr(A) # Tuple: QR分解
np.linalg.svd(A) # Tuple: SVD分解
np.linalg.matrix_power(A, k) # A^k
```

numpy:应用

数据处理：处理实验1到实验4的成绩

```
import numpy as np
labs = np.loadtxt(“grade.csv”, dtype=np.int,
delimiter=',',usecols=(1,2,3,4))

np.mean(labs,axis=0) #每次实验平均
np.mean(labs,axis=1) #每个学生平均

np.median(labs,axis=0) #每次实验中位数
np.median(labs,axis=1) #每个学生中位数

np.average(labs,axis=1,weights=[0.1,0.2,0.
3,0.4])                #加权算学生总分
```

	A	B	C	D	E
1	292477	80	90	95	100
2	290761	90	90	95	100
3	332092	90	95	100	100
4	295295	95	95	100	100
5	295815	80	90	95	95
6	246113	70	80	90	95
7	291248	100	95	100	100
8	296477	100	85	95	100
9	291311	90	95	100	100
10	290647	100	80	95	100
11	105737	90	65	70	100
12	291299	90	85	100	100
13	291310	100	100	100	95
14	291290	95	95	100	100
15	291282	100	65	90	80
16	291246	90	75	95	70
17	291256	100	70	95	90
18	291320	90	100	100	100
19	291303	90	85	100	95
20	291263	90	80	95	100
21	291241	100	90	100	100
22	291332	100	70	80	100
23	292897	100	85	100	100
24	291272	95	90	90	100
25	295259	100	100	100	100
26	291244	90	95	100	100
27	291308	90	90	100	100
28	336178	80	85	80	90
29	126474	0	0	0	0
30	291321	100	75	90	90
31	291240	100	100	80	95
32	291294	85	80	100	100
33	291264	80	80	95	95
34	291285	95	90	95	90

numpy:应用

问题：什么问题/应用需要矩阵？

希望能给你们的期末大作业带来启发：

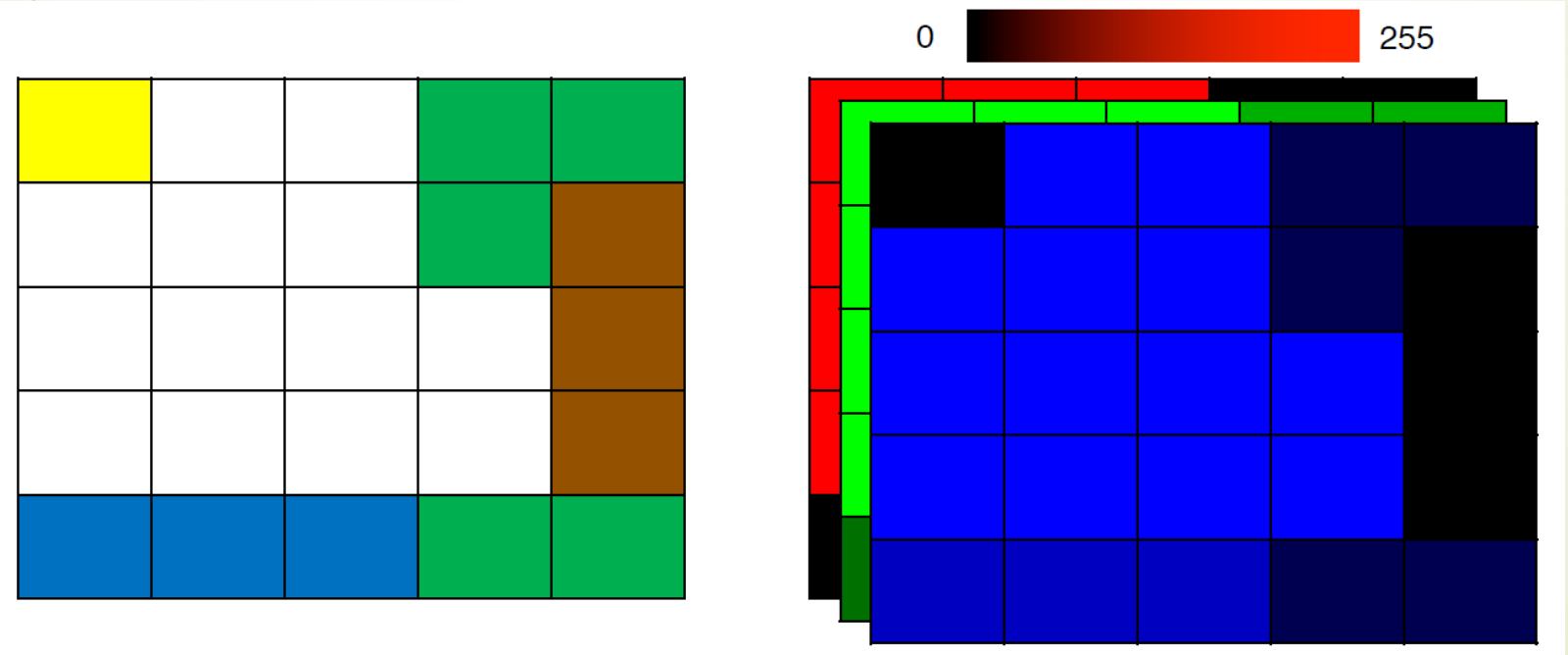
1.图像处理与可视化

2.图论相关

3.机器学习

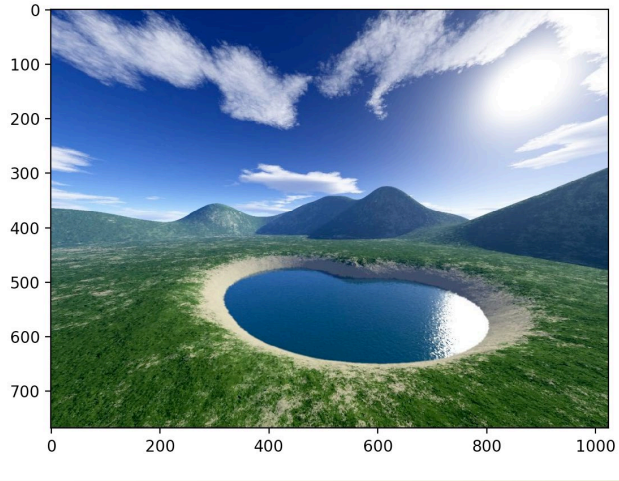
numpy:应用

应用：图片，每个像素（RGB）

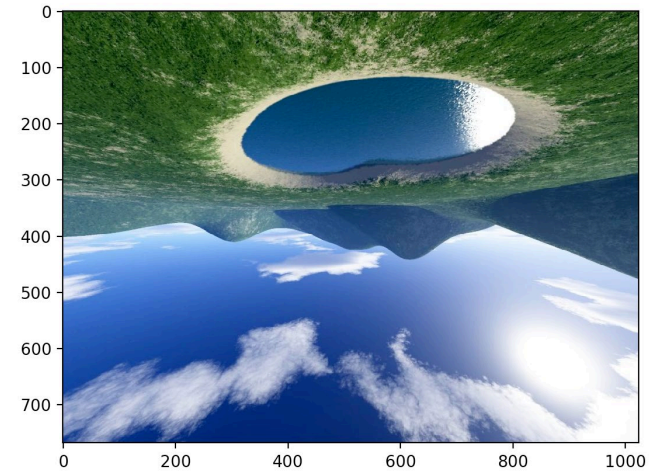


numpy:应用

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

# 读入图片
image = Image.open('view.jpg')
Image = np.array(image)
print(image.shape) # (768, 1024, 3)
plt.imshow(image)
plt.show()
```

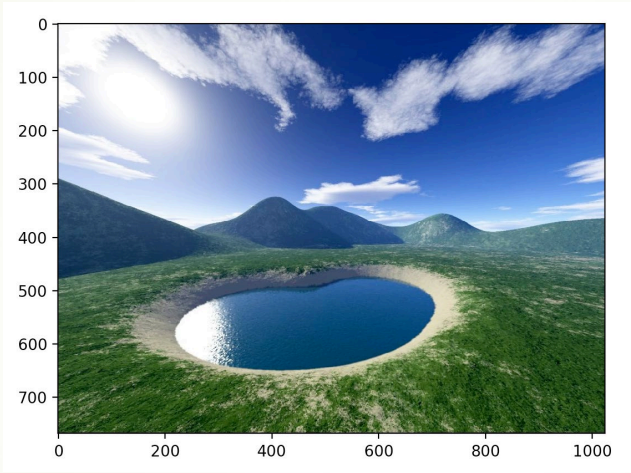



```
image2 = image[::-1, :, :] ???
plt.imshow(image2)
plt.show()
```

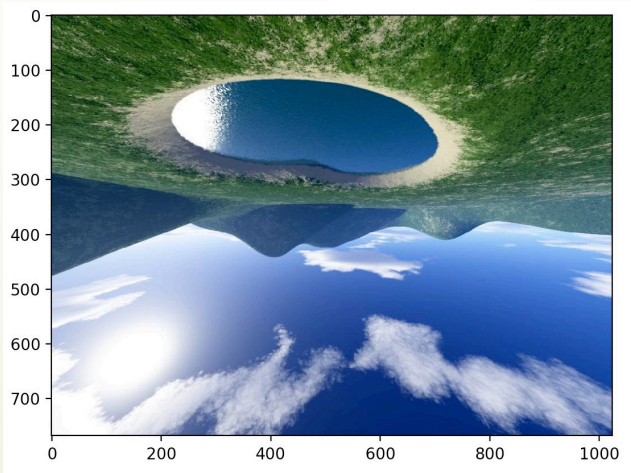


numpy:应用

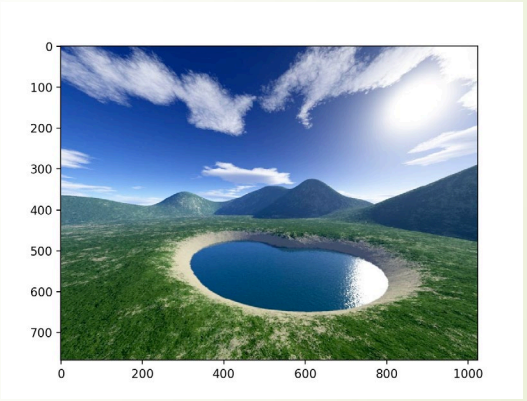
```
image3 = image[:, ::-1, :]
```




```
image4 = image[::-1, ::-1, :]
```

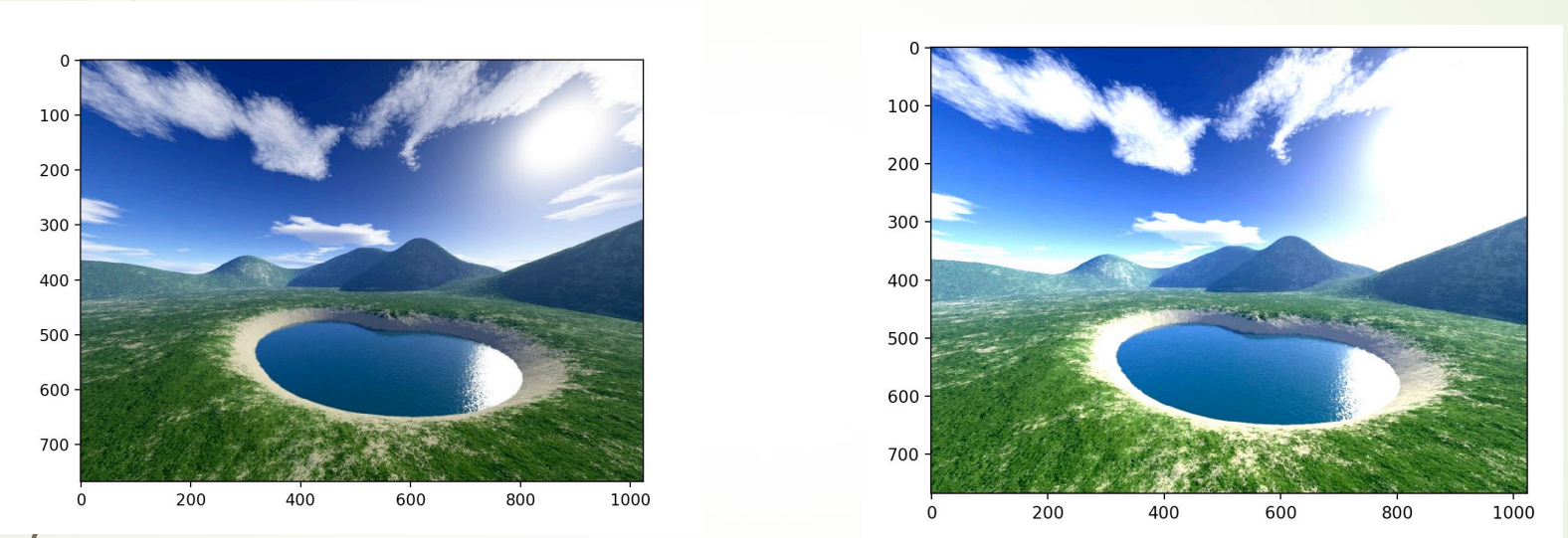


原图



numpy:应用

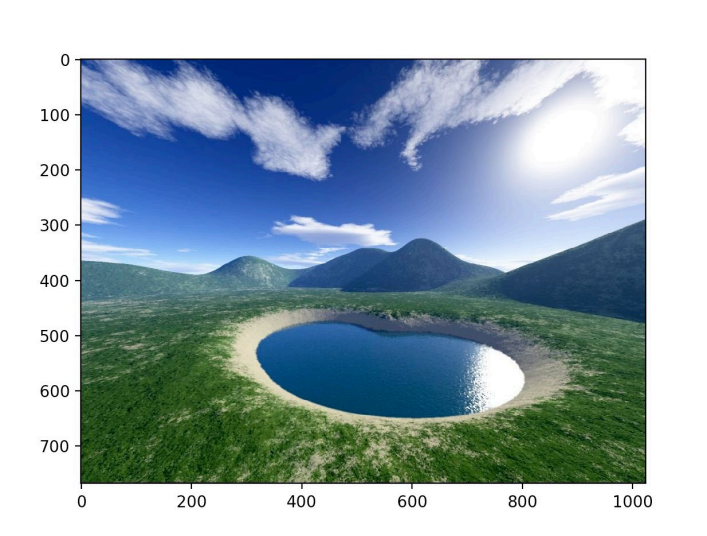
调整亮度

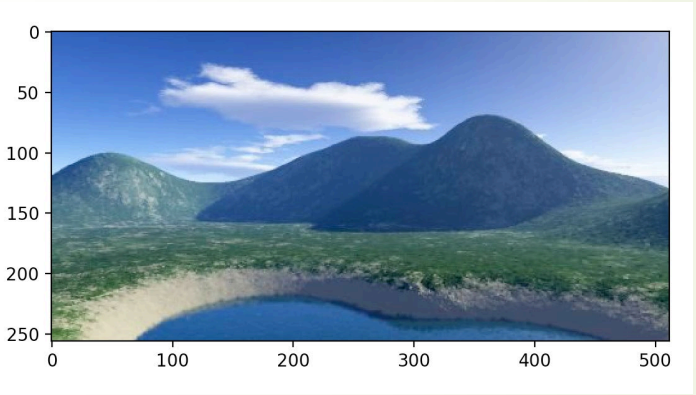



```
image3 = np.clip(image*1.5, a_min=0., a_max=255.)
plt.imshow(image3.astype('uint8'))
plt.show()
```

numpy:应用

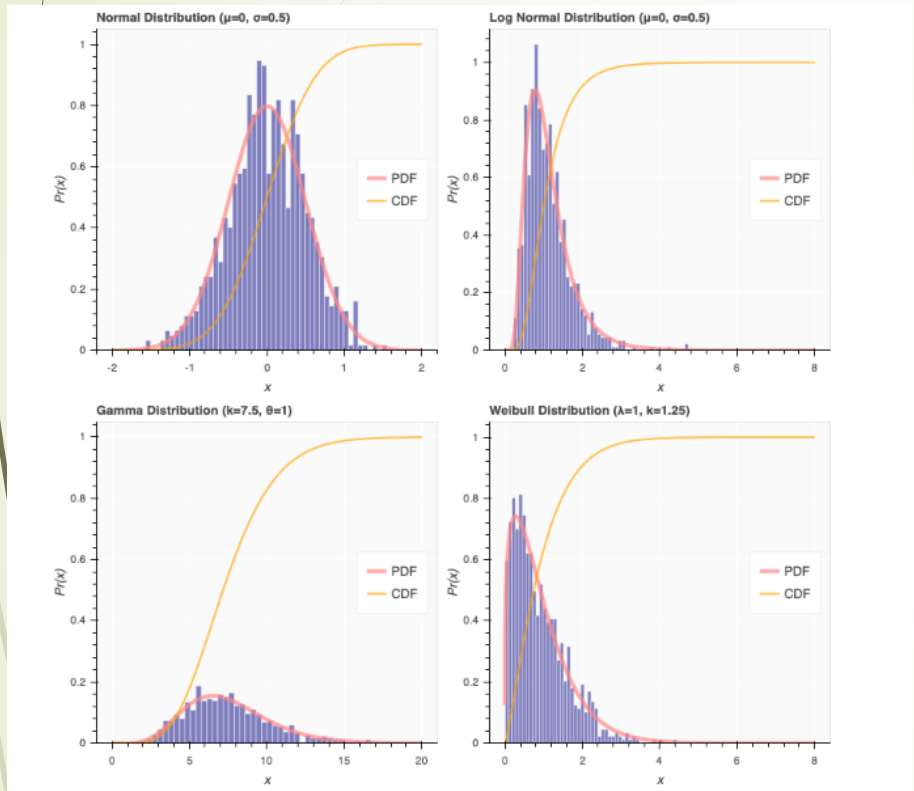
剪切图片



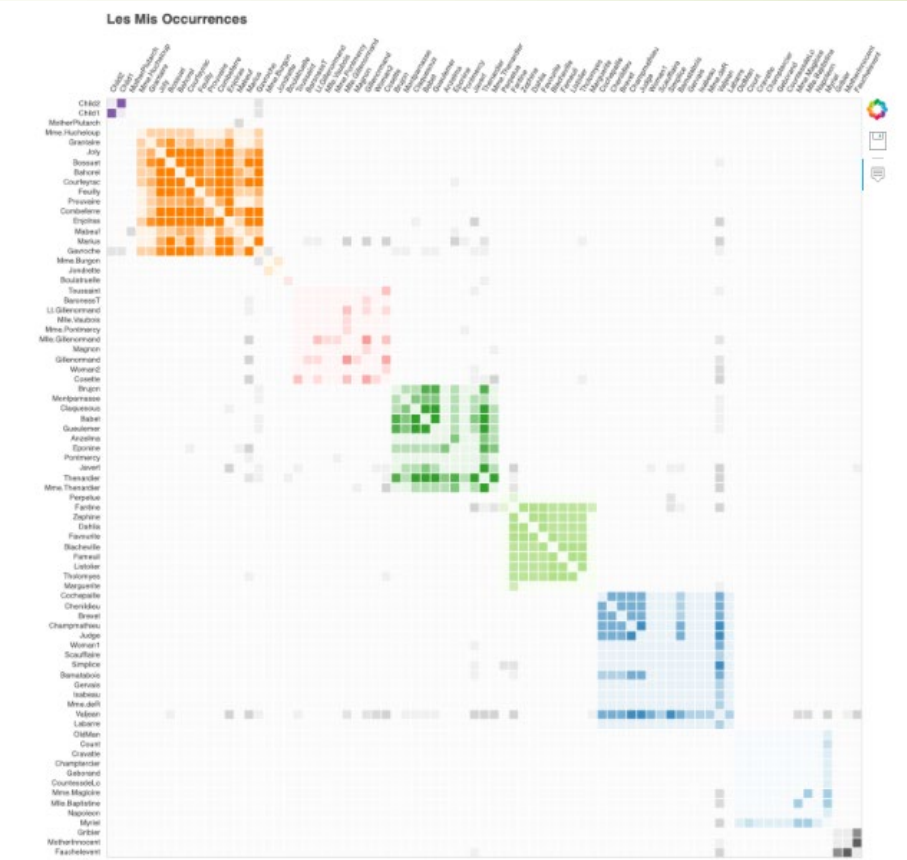



```
H, W = image.shape[0], image.shape[1]
W1 = W//4
W2 = W//4 * 3
H1=  H//3
H2=  H//3*2
image5 = image[H1:H2, W1:W2, :]
```

numpy:应用



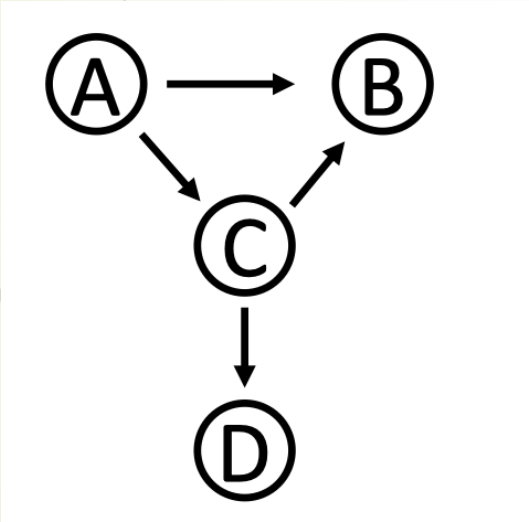

```
import numpy as np
import scipy.special

from bokeh.layouts import gridplot
from bokeh.plotting import figure, output_file, show


def make_plot(title, hist, edges, x, pdf, cdf):
    p = figure(title=title, tools='', background_fill_color="#fafafa")
    p.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:],
           fill_color="navy", line_color="white", alpha=0.5)
    p.line(x, pdf, line_color="#ff8888", line_width=4, alpha=0.7, legend_label="PDF")
    p.line(x, cdf, line_color="orange", line_width=2, alpha=0.7, legend_label="CDF")
```




```
import numpy as np

from bokeh.plotting import figure, output_file, show
from bokeh.sampledata.les_mis import data

nodes = data['nodes']
names = [node['name'] for node in sorted(data['nodes'], key=lambda x: x['group'])]

N = len(nodes)
counts = np.zeros((N, N))
for link in data['links']:
    counts[link['source'], link['target']] = link['value']
    counts[link['target'], link['source']] = link['value']

colormap = ["#444444", "#a6cee3", "#1f78b4", "#b2df8a", "#33a02c", "#fb9a99",
            "#e31a1c", "#fdbf6f", "#ff7f00", "#cab2d6", "#6a3d9a"]
```

numpy:应用

应用：图论



邻接矩阵

	A	B	C	D	
A	0	1	1	0	2
B	0	0	0	0	0
C	0	1	0	1	2
D	0	0	0	0	0
	0	2	1	1	

```
outgoing_edges = np.sum(A, axis=1)

incoming_edges = np.sum(A, axis=0)
```

numpy:应用

地铁线路图

网页重要性：Google's PageRank

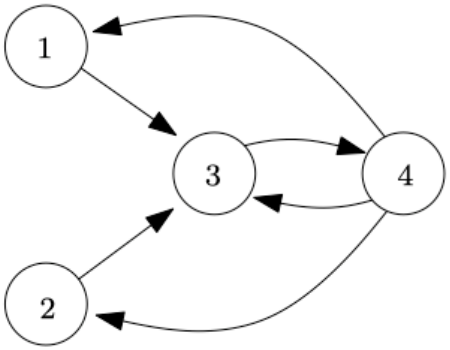
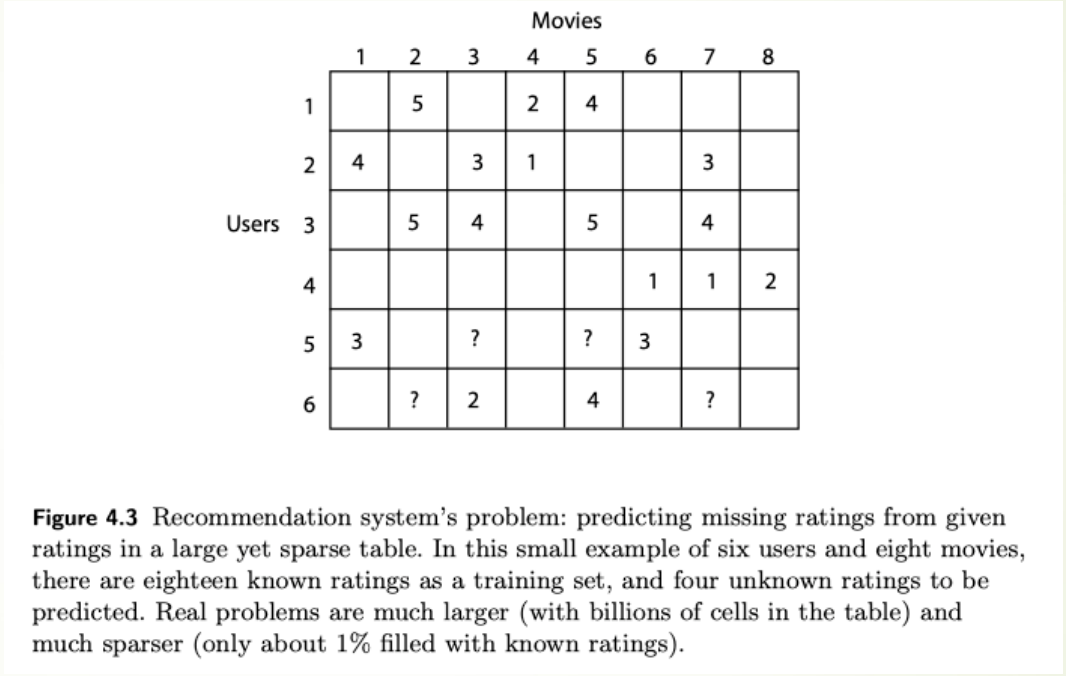


Figure 3.1 A simple example of importance score with four webpages and six hyperlinks. It is a small graph with much symmetry, leading to a simple calculation of the importance scores of the nodes.

numpy:应用

推荐系统



Users \ Movies	1	2	3	4	5	6	7	8
1		5		2	4			
2	4		3	1			3	
3		5	4		5		4	
4						1	1	2
5	3		?		?	3		
6		?	2		4		?	

Figure 4.3 Recommendation system's problem: predicting missing ratings from given ratings in a large yet sparse table. In this small example of six users and eight movies, there are eighteen known ratings as a training set, and four unknown ratings to be predicted. Real problems are much larger (with billions of cells in the table) and much sparser (only about 1% filled with known ratings).

$$\mathbf{R} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 5 & 4 & 4 & - & \mathbf{5} \\ - & 3 & 5 & \mathbf{3} & 4 \\ 5 & 2 & - & \mathbf{2} & 3 \\ - & \mathbf{2} & 3 & 1 & 2 \\ 4 & - & \mathbf{5} & 4 & 5 \\ \mathbf{5} & 3 & - & 3 & 5 \\ 3 & \mathbf{2} & 3 & 2 & - \\ 5 & \mathbf{3} & 4 & - & 5 \\ \mathbf{4} & 2 & 5 & 4 & - \\ \mathbf{5} & - & 5 & 3 & 4 \end{pmatrix} \end{array}$$

$$\hat{\mathbf{R}} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 5.00 & 3.09 & 4.90 & - & \mathbf{4.62} \\ - & 2.89 & 4.69 & \mathbf{3.49} & 4.42 \\ 4.10 & 2.19 & - & \mathbf{2.78} & 3.71 \\ - & \mathbf{1.00} & 2.49 & 1.29 & 2.22 \\ 4.90 & - & \mathbf{4.79} & 3.58 & 4.51 \\ \mathbf{4.88} & 2.96 & - & 3.56 & 4.48 \\ 3.15 & \mathbf{1.23} & 3.03 & 1.82 & - \\ 4.84 & \mathbf{2.92} & 4.72 & - & 4.44 \\ \mathbf{4.84} & 2.92 & 4.72 & 3.51 & - \\ \mathbf{4.61} & - & 4.49 & 3.29 & 4.22 \end{pmatrix} \end{array}$$

$$\tilde{\mathbf{R}} = \mathbf{R} - \hat{\mathbf{R}} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 0.91 & -0.90 & - & ? \\ - & 0.11 & 0.31 & ? & -0.42 \\ 0.90 & -0.19 & - & ? & -0.71 \\ - & ? & 0.51 & -0.29 & -0.22 \\ -0.90 & - & ? & 0.42 & 0.49 \\ ? & 0.040 & - & -0.56 & 0.52 \\ -0.15 & ? & -0.031 & 0.18 & - \\ 0.16 & ? & -0.72 & - & 0.56 \\ ? & -0.87 & 0.33 & 0.54 & - \\ ? & - & 0.51 & -0.29 & -0.22 \end{pmatrix} \end{array}$$

numpy:应用

应用：机器学习

numpy-ml

Machine learning, in NumPy

Watch

Navigation

Hidden Markov models
Gaussian mixture models
Latent Dirichlet allocation
N-gram smoothing models
Multi-armed bandits
Reinforcement learning
Nonparametric models
Tree-based models
Neural networks
Linear models
Preprocessing
Utilities

Quick search

Welcome to numpy-ml

numpy-ml is a growing collection of machine learning models, algorithms, and tools written exclusively in NumPy and the Python standard library.

The purpose of the project is to provide reference implementations of common machine learning components for rapid prototyping and experimentation. With that in mind, don't just read the docs – read the source!

This documentation is under development!

We're working to expand our coverage. During this time there are likely to be typos, bugs, and poorly-worded sections. If you encounter any of the above, please file an issue or submit a pull request!

Disclaimer

This software is provided as-is: there are no guarantees that it fits your purposes or that it is bug-free. Use it at your own risk!

numpy:应用

应用：机器学习

- 监督式学习（supervised learning）
- 建立一个模型，在给定过去学生成绩数据集的情况下，预测学生在课程中的成绩。
- 特征矩阵：每一行表示过去的一个学生，列表示特征。
- 标签向量：列出了过去学生在课堂上的成绩

	Grade in prerequisite class	Hours per week spent on this class	...	Number of times per week they attend OH
Student 0	0.88	8	...	2
Student 1	0.95	15	...	1
⋮	⋮	⋮	⋱	⋮
Student $m-1$	0.78	5	...	0

	Grade in current class
Student 0	0.9
Student 1	0.85
⋮	⋮
Student $m-1$	0.69

numpy:应用

	Grade in prerequisite class	Hours per week spent on this class	...	Number of times per week they attend OH
Student 0	0.88	8	...	2
Student 1	0.95	15	...	1
⋮	⋮	⋮	⋱	⋮
Student $m-1$	0.78	5	...	0

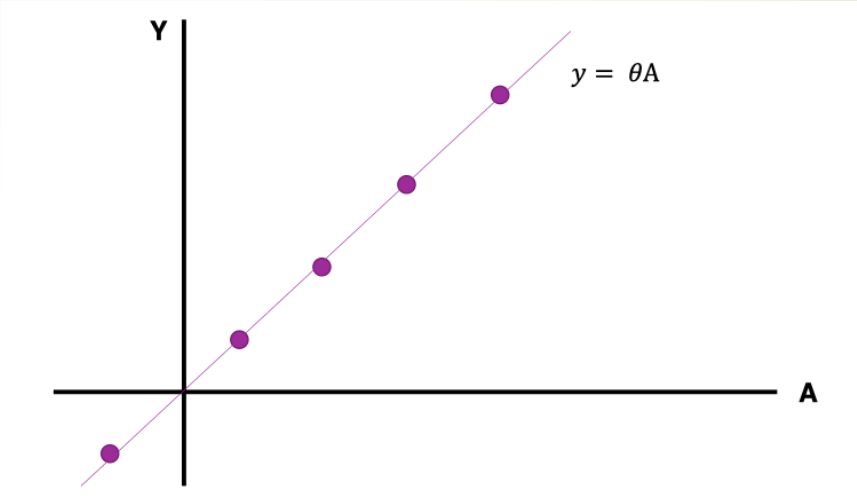
	Grade in current class
Student 0	0.9
Student 1	0.85
⋮	⋮
Student $m-1$	0.69

$$\text{(Row 1)}: \theta_1(0.88) + \theta_2(8) + \cdots + \theta_n(2) \approx 0.9$$
$$\text{(Row 2)}: \theta_1(0.95) + \theta_2(15) + \cdots + \theta_n(1) \approx 0.85$$
$$\text{(Row 3)}: \theta_1(0.78) + \theta_2(5) + \cdots + \theta_n(0) \approx 0.69$$

寻找每一个 θ 系数（θ向量）

numpy:应用

- 特征矩阵 *A*
- 标签向量 *y*



用最小二乘法(Least Square) 拟合数据

```
# 求θ
>>> theta = np.linalg.lstsq(A, y)
array([4.3155, 2.1284, ... -1.9321])
```

使用numpy，一句话解决！

期末大项目安排

选题 Proposal （5%）　截止日期：2021.05.26，23:59pm

内容包括

1. **组队：**每组最多3人，提供组员名单，确定组长。

2. **项目题目和计划：**确定项目（要做什么？）、选题意义（为什么要做），实现计划（要怎么做、简单的计划等）。。。

3. **选题范围：**图像处理、机器学习、数据处理、小游戏等均可。

项目代码和报告 （65%）

项目展示（30%）

数据处理类选题参考