11.2 Numpy 与矩阵运算

学些目标

- Numpy 安装与基本使用
- 核心数据结构 numpy.ndarray

Numpy 安装与基本使用

安装

```
$ sudo pip3 install numpy
```

```
import numpy as np
np.__version__
```

重要数据类型 ndarray

numpy.ndarray 可以将数组当成矩阵使用

Python 的 list 的优缺点

Python 的 array 模块（限制数据类型）

```
import array
arr = array.array('i', list(range(10)))  # i表示整数，f表示浮点数
```

为什么要使用 Numpy 中的 numpy.ndarray

```
import numpy as np

nparr1 = np.array([1, 2, 3])    # int型
nparr2 = np.array([1, 2, 3.0])  # float型
nparr3 = np.array([1, 2, 3], dtype=float)  # 指定为float型
```

性能对比

```
def python_test(n):
    a = [i**2 for i in range(n)]
    b = [i**3 for i in range(n)]
    c = []
    for i in range(n):
        c.append(a[i] + b[i])
    return c
```

Numpy 版本

```
def numpy_test(n):
    a = np.arange(n) ** 2
    b = np.arange(n) ** 3
    c = a + b
    return c
```

测试

```
%time res = python_test(10000000)
CPU times: user 5.9 s, sys: 333 ms, total: 6.23 s
Wall time: 6.24 s

%time res = numpy_test(10000000)
CPU times: user 141 ms, sys: 129 ms, total: 270 ms
Wall time: 271 ms
```

list & array & ndarray 异同

Python 中的 list 很灵活，但是性能较低

Python 中的 array 相较于 list 降低了灵活性，使用时不能像 list 那么随心所欲， 但是换来了性能的提升。

array 只是把存放其中的数据当成数组，对于一维或二维数组，它并不会看成是一个向量或矩阵。这时候，我们就需要使用 Numpy 中的 ndarray

```
import numpy as np
import array

a = np.array([0, 2, 4])
a*2 # 结果是 array([0, 4, 8])

a = [0, 2, 4]
a*2 # 结果是 [0, 2, 4, 0, 2, 4]

a = array.array('i', [0, 2, 4])
a*2 # 结果是 array('i', [0, 2, 4, 0, 2, 4])
```

```
[i*2 for i in a] # 结果是 [0, 4, 8]

res = []
for i in a:
    res.append(i*2)

res # 结果是 [0, 4, 8]
```

Numpy 中矩阵和随机数生成

```
import numpy as np
```

函数	功能	示例
np.array	创建ndarray数组/向量	`np.array(list(range(10)))` `np.array(range(10))`

np.arange	指定范围创建ndarray数组	`np.arange(10)`
		`np.arange(2, 20, 3)`
		`np.arange(2, 20, 0.3)`

np.zeros	创建ndarray全零数组
	默认是浮点数
		`np.zeros(10)`
		`np.zeros(10, dtype=int)`
		`np.zeros(shape=(3, 5), dtype=int)`

np.ones	创建ndarray全一数组	`np.ones(10)` `np.ones((2,3))`

np.full	指定填充值创建ndarray数组	`np.full(10, 99)`
		`np.full((3, 5), 99)`
		`np.full((3, 5), fill_value=99.0)`

np.linspace	创建ndarray线性等分数组
	第3个参数表示截取点数，
	默认包含起始点和终止点
		`np.linspace(0, 20, 10)`
		`np.linspace(0, 20, 11)`
		`np.linspace(0, 20, 10, endpoint=False)`

np.random	创建ndarray随机数数组	`np.random.random(5)`
		`np.random.randint(0, 10, 5)`
		`np.random.randint(0, 10, size=(3, 5))`
		正态分布 `np.random.normal(size=(3, 5))`

```
# random是伪随机数，指定seed，每次执行都一样
np.random.seed(100)
np.random.randint(0, 10, 5)
```

Numpy 中 ndarray 基础操作

数组维度

ndim & **shape** & **size**

```
import numpy as np

A = np.ones(shape=(3, 5))

A.ndim    # 查看维数，这里为 2
A.shape   # 查看形状，这里为 (3, 5)
A.size    # 查看元素个数，这里为 15
```

reshape

```
a = np.arange(10)
a.reshape(2, 5)  # 不会覆盖原来的a

a.reshape(2, -1) # 两行，列根据情况而定
a.reshape(-1, 2) # 两列，行根据情况而定
a.reshape(-1, 3) # ValueError: cannot reshape array of size 10 into shape (3)
```

数组切片

取值

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

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

X[0]
# array([0, 1, 2, 3, 4])

X[-1]
# array([10, 11, 12, 13, 14])

X[0][1]    # 值为1，Python的写法，但这种写法有时候会取不到我们想要的值
X[(0, 1)]  # 值为1，ndarray的完整写法
X[0, 1]    # 值为1，推荐写法
```

切片

```
a = np.arange(10)

"""
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
"""

a[0:5]    # 取第0到第5（不包含5），array([0, 1, 2, 3, 4])
a[:5]     # 从头开始到第5（不包含5），array([0, 1, 2, 3, 4])
a[5:]     # 取第5到尾，array([5, 6, 7, 8, 9])
a[0:8:2]  # 第0到第8（不包含8），步长为2，array([0, 2, 4, 6])
a[::2]    # 从整个数组中以步长为2进行抽取，array([0, 2, 4, 6, 8])
a[::-1]   # 数组倒序，array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
a[::-2]   # array([9, 7, 5, 3, 1])
```

二维数组切片

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

X[0:2, 0:3]
X[:2, :3]
"""
array([[0, 1, 2],
       [5, 6, 7]])
"""

X[:2][:3] # 先切出前两行，再切出前三行，不是我们想要的
"""
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
"""

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

引用关系

与 list 数据类型不同，ndarray 为了效率考虑，存在引用关系。

也就是说如果修改 ndarray 数组切片之后的变量，其本身也会改变。要解决这个问题就需要使用 copy() 方法。

```
import numpy as np

X = np.arange(15).reshape(3, 5)
X2 = X[:2, :2].copy()
```

ndarray 中的合并和分割

1. np.concatenate
2. np.vstack
3. np.hstack
4. np.split
5. np.vsplit
6. np.hsplit

方法	功能	示例
np.concatenate	合并维数相同的两个数组	`X = np.concatenate([X1, X2])` `X = np.concatenate([X1, X2], axis=1)`
np.vstack	垂直方向合并一维数组和二维数组	
np.hstack	水平方向合并一维数组和二维数组	

np.split	分割	`np.split(x, 2)`
		`np.split(x, [3, 7])`
		`np.split(A, [3], axis=1)`

np.vsplit	垂直分割	
np.hsplit	水平分割	

Numpy 中的矩阵运算

```
A + B
A - B
A * B
A / B
A // B
A % B

# 矩阵乘法
A.dot(B)

# 矩阵的逆
invA = np.linalg.inv(A)
```

矩阵乘以它的逆矩阵，得到单位矩阵；逆矩阵乘以原矩阵，得到的也是单位矩阵。

如果不是方块矩阵（n阶矩阵），将不能求逆矩阵，只能求伪逆矩阵

```
pinvA = np.linalg.pinv(A)
```

矩阵的转置

```
A.T
```

Numpy 中的聚合操作

```
import numpy as np

np.sum(X)  # 元素求和
np.min(X)
np.max(X)
np.sum(X, axis=0)  # 垂直方向求和
np.sum(X, axis=1)  # 水平方向求和
np.prod(X) # 元素相乘
np.mean(X) # 求平均值
np.mean(X, axis=0)
np.median(X, axis=0) # 求中位数
np.percentile(X, q=50)  # 百分位点为50%的值，也就是中位数
np.percentile(X, q=100) # 百分位点为100%的值，也就是最大值
```

```
np.var(X) # 方差：使用每个值减去均值，然后平方之后再整体求和，再除以整个样本的数量
np.sum((X - np.mean(X))**2)/np.size(X) # np.var(X) 相当于这个
np.std(X) # 标准差：方差开根号，相当于 np.var(X)**0.5

x = np.random.normal(0, 1, size=1000000) # 生成均值为0，标准差为1的数据
np.mean(x) # 求均值，应该接近0
np.std(x)  # 求标准差，应该接近1
```

Numpy 中的 arg 运算

在机器学习中经常需要 arg 运算。以下函数不会返回数据本身，而是返回索引

1. argmin
2. argmax
3. argsort
4. argpartition
5. argwhere

Numpy 中的比较运算

Fancy Indexing

```
import numpy as np

x = np.array(list('ABCDEFG'))

[x[1], x[2], x[4]] # ['B', 'C', 'E']

ind = [1, 2, 4]
x[ind] # 同样得到 ['B', 'C', 'E']
```

比较运算

```
x = np.arange(10)
"""
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
"""

x < 5
# array([ True,  True,  True,  True,  True, False, False, False, False, False])
```

```
np.sum(x<8)
```