240 NumPy 数值计...
241 创建 NumPy ...
242 NumPy 数组之...
243 NumPy 之 ara...
244 NumPy 之 lins...
245 NumPy 之 logs...
246 NumPy 之 创...
247 NumPy 之创建...
248 NumPy之创建...
249 NumPy 之创建...
250 NumPy 之索引...

Python 全栈 400 之NumPy数值计算练习

240 NumPy 数值计算更高效的案列

Python 已经提供了很多丰富的内置包，我们为什么还要学习 NumPy 呢？

先看一个例子，找寻学习 NumPy 的必要性和重要性。

打开 `IPython`，创建 Python 的列表 `a` 对象。然后，使用列表生成式，创建一个元素都为原来两倍的新列表 `a2`，并统计这一行的用时为 `95.7 ms`。

复制
```
In [76]: a = list(range(1000000))

In [77]: %time a2 = [i*2 for i in a]
Wall time: 95.7 ms
```

使用 NumPy，创建同样大小和取值的数组 `na`。然后，对每个元素乘以 2，返回一个新数组 `na2`，用时为 `2 ms`。

复制
```
In [78]: import numpy as np

In [79]: na = np.array(range(1000000))

In [80]: %time na2 = na * 2
Wall time: 2 ms
```

完成同样的都对元素乘以 2 的操作，NumPy 比 Python 快了 45 倍之多。

这就是我们要学好 NumPy 的一个重要理由，它在处理更大数据量时，处理效率明显快于 Python；并且内置的向量化运算和广播机制，使得使用 NumPy 更加简洁，会少写很多嵌套的 for 循环，因此代码的可读性大大增强。

252 NumPy 计算为什么如此快？

有多个原因：

- Python的 list 是一个通用结构。它能包括任意类型的对象，并且是动态类型。
- NumPy 的 ndarray 是静态、同质的类型，当 ndarray 对象被创建时，元素的类型就确定。由于是静态类型，所以 ndarray 间的加、减、乘、除用 C 和 Fortran 实现才成为可能，所以运行起来就会更快。根据官当介绍，底层代码用 `C` 语言和 `Fortran` 语言实现，实现性能无限接近 C 的处理效率。

由此可见，NumPy 就非常适合做大规模的数值计算和数据分析。

今天，我们一起学习 NumPy 的基本使用，借助实际的 `iris` 数据集，使用例子帮助大家最快掌握 NumPy 那些最高频使用的函数。

241 创建 NumPy 数组五种常用方法

创建一个 `ndarray` 数组对象，有很多种方法。`array` 函数能创建新的数组；`arange`，`linspace` 等方法；从文件中读入数据返回一个 `ndarray` 对象；多个 `ndarray` 对象又能构造生成一个新的 `ndarray` 对象。

1) 通过构造函数 `array` 创建一维 `array`：

复制
```
import numpy as np

In [2]: v = np.array([1,2,3,4])

In [3]: v
Out[3]: array([1, 2, 3, 4])
```

2) 创建二维 `array`：

复制
```
In [4]: m = array([[1,2],[3,4]])

In [5]: m
Out[5]:
array([[1, 2],
       [3, 4]])
```

v 和 m 的类型都是 `ndarray`，NumPy 中最主要的数据结构。

复制
```
In [6]: type(v),type(m)
Out[6]: (numpy.ndarray, numpy.ndarray)
```

3) `arange` 数组

复制
```
In [94]: ara = np.arange(1,10)

In [95]: ara
Out[95]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

4) `linspace` 数组，15 个元素：

复制
```
In [97]: np.linspace(1,10,15)
Out[97]:
array([ 1.        ,  1.64285714,  2.28571429,  2.92857143,  3.57142857,
        4.21428571,  4.85714286,  5.5       ,  6.14285714,  6.78571429,
        7.42857143,  8.07142857,  8.71428571,  9.35714286, 10.        ])
```

5) 组合 `ndarray` 对象：

如下创建一个 `ndarray` 对象 `a`：

复制
```
In [98]: a = np.arange(10).reshape(2,-1)
```

```
In [99]: a
Out[99]:
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

找出 `a` 中大于 3 的元素索引，使用 `where` 方法，返回一个元组，带有 2 个 `ndarray` 对象，分别表示大于 3 的元素第一维、第二维度中的位置：

复制

```
In [100]: np.where(a>3)
Out[100]:
(array([0, 1, 1, 1, 1, 1], dtype=int64),
 array([4, 0, 1, 2, 3, 4], dtype=int64))
```

`where` 方法返回值可读性不强，我们把它拼接为一个 `ndarray` 对象：

复制

```
In [101]: np.array(np.where(a>3))
Out[101]:
array([[0, 1, 1, 1, 1, 1],
       [4, 0, 1, 2, 3, 4]], dtype=int64)
```

然后，再转置，使用 `np.transpose` 方法：

复制

```
tuple_to_array = np.array(np.where(a>3))
np.transpose(tuple_to_array)
```

结果，这回一看就明白了，`[0,4]` 表示在原 `ndarray` 对象 `a` 上的索引：

复制

```
Out[102]:
array([[0, 4],
       [1, 0],
       [1, 1],
       [1, 2],
       [1, 3],
       [1, 4]], dtype=int64)
```

242 NumPy 数组之shape,size,dtype

1) shape 属性

数组的形状信息，非常重要。在深度学习中，构建网络模型，调试多维数组运算代码时，`shape` 的作用更加凸显。

`shape` 属性返回数组的形状信息，是一个元组对象。

如下，分别创建一维数组 `v` ，二维数组 `m` ：

复制

```
In [108]: v = np.zeros(10)

In [109]: m = np.ones((3,4))
```

打印它们的 `shape` 信息，`(10,)` 表示为一维数组，且第一维的长度为 10. 当元组只有一个元素时，为什么写成这样，我们在前面的 Python 基础部分、进阶部分都提到过。

复制

```
In [110]: v.shape
Out[110]: (10,)

In [111]: m.shape
Out[111]: (3, 4)
```

`size` 属性获取数组内元素个数：

复制

```
In [112]: m.size
Out[112]: 12
```

`dtype` 属性获取数组内元素的类型：

复制

```
In [113]: m.dtype
Out[113]: dtype('float64')
```

如果我们尝试用 `str` 类型赋值给 `m` ，就会报错：

复制

```
In [10]: m[0,0]='hello'

ValueError: could not convert string to float: 'hello'
```

创建数组时，还可以通过为 `dtype` 赋值，指定元素类型：

复制

```
In [117]: m = np.array([1,2,3],dtype='float')

In [118]: m
Out[118]: array([1., 2., 3.])
```

`dtype` 更多取值：`int`, `complex`, `bool`, `object` ，还可以显示的定义数据位数的类型，如：`int64`, `int16`, `float128`, `complex128`。

243 NumPy 之 arange 函数

起始点，终点，步长；不包括终点。

复制

```
In [94]: ara = np.arange(1,10)

In [95]: ara
Out[95]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

244 NumPy 之 linspace 函数

起始点，终点，分割份数；包括终点。

复制

```
In [5]: np.linspace(0,10,5)
Out[5]: array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

245 NumPy 之 logspace 函数

创建以 `e` 为底，指数为 1,2，...,10 的数组：

复制
```
In [17]: np.logspace(1, 10, 10, base=np.e)
Out[17]:
array([2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01,
       1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03,
       8.10308393e+03, 2.20264658e+04])
```

246 NumPy 之 创建对角数组

创建对角数组：

复制
```
In [22]: np.diag([1,2,3])
Out[22]:
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

主对角线偏移 1 的数组：

复制
```
In [23]: np.diag([1,2,3],k=1)
Out[23]:
array([[0, 1, 0, 0],
       [0, 0, 2, 0],
       [0, 0, 0, 3],
       [0, 0, 0, 0]])
```

247 NumPy 之创建全零数组

创建元素全都为 0 的数组：

复制
```
In [24]: np.zeros((3,3))
Out[24]:
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

248 NumPy之创建全一数组

创建元素全都为 1 的数组：

复制
```
In [25]: np.ones((3,3))
Out[25]:
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

249 NumPy 之创建随机数组

`np.random` 模块生成随机数组，更加方便。

生成 0~1 ，shape 为 (3,5) 的随机数数组：

复制
```
In [2]: np.random.rand(3,5)
Out[2]:
array([[0.25366147, 0.18996607, 0.01599463, 0.08113353, 0.99794258],
       [0.38813147, 0.33669704, 0.97040282, 0.0836301 , 0.55533133],
       [0.17767781, 0.94982834, 0.52045864, 0.58504198, 0.40904079]])
```

250 NumPy 之索引案例

NumPy 索引，功能强大，不仅支持切片操作，还支持布尔型按条件筛选操作。

复制
```
In [2]: m = np.arange(18).reshape(2,3,3)

In [3]: m
Out[3]:
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]]])
```

`:` 表示此维度的所有元素全部获取：

复制
```
In [4]: m[:,1:3,:]
Out[4]:
array([[[ 3,  4,  5],
        [ 6,  7,  8]],

       [[12, 13, 14],
        [15, 16, 17]]])
```

复制
```
In [5]: m[:,1:3,:1]
Out[5]:
array([[[ 3],
        [ 6]],

       [[12],
        [15]]])
```

按照维度赋值：

复制
```
In [7]: m[:,0,:] = -1

In [8]: m
Out[8]:
array([[[-1, -1, -1],
        [ 3,  4,  5],
        [ 6,  7,  8]],
```

```
       [[-1, -1, -1],
        [12, 13, 14],
        [15, 16, 17]]])
```

复制

```
In [11]: m[:,:,0]
Out[11]:
array([[-1,  3,  6],
       [-1, 12, 15]])
```

NumPy 还支持掩码索引，用于元素筛选，非常方便。

判断上面切片 `m[:,:,0]` 中大于 5 的元素，写法简洁，无需写 for 循环。

复制

```
In [12]: mt = m[:,:,0]

In [13]: mt[mt>5]
Out[13]: array([ 6, 12, 15])
```

251 NumPy之数据归一化案例

1) 下载数据

使用 NumPy，下载 `iris` 数据集。

仅提取 `iris` 数据集的第二列 `usecols = [1]`

复制

```
import numpy as np

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iri
s.data'
wid = np.genfromtxt(url, delimiter=',', dtype='float', usecols=[1])
```

展示数据

复制

```
array([3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3. ,
       3. , 4. , 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3. ,
       3.4, 3.5, 3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.1, 3. ,
       3.4, 3.5, 2.3, 3.2, 3.5, 3.8, 3. , 3.8, 3.2, 3.7, 3.3, 3.2, 3.2,
       3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2. , 3. , 2.2, 2.9, 2.9,
       3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3. , 2.8, 3. ,
       2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. , 3.4, 3.1, 2.3, 3. , 2.5, 2.6,
       3. , 2.6, 2.3, 2.7, 3. , 2.9, 2.9, 2.5, 2.8, 3.3, 2.7, 3. , 2.9,
       3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2, 2.7, 3. , 2.5, 2.8, 3.2, 3. ,
       3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2, 2.8, 3. , 2.8, 3. ,
       2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4, 3.1, 3. , 3.1, 3.1, 3.1, 2.7,
       3.2, 3.3, 3. , 2.5, 3. , 3.4, 3. ])
```

单变量(univariate)，长度为 150 的一维 NumPy 数组。

2) 归一化

求出最大值、最小值

复制

```
smax = np.max(wid)
smin = np.min(wid)

In [51]: smax,smin
Out[51]: (4.4, 2.0)
```

`

归一化公式：

复制

```
s = (wid - smin) / (smax - smin)
```

还有一个更简便的方法，使用 `ptp` 方法，它直接求出最大值与最小值的差

复制

```
s = (wid - smin) / wid.ptp()
```

3) NumPy 的打印设置

只打印小数点后三位的设置方法：

复制

```
np.set_printoptions(precision=3)
```

归一化结果：

复制

```
array([0.625, 0.417, 0.5  , 0.458, 0.667, 0.792, 0.583, 0.583, 0.375,
       0.458, 0.708, 0.583, 0.417, 0.417, 0.833, 1.   , 0.792, 0.625,
       0.75 , 0.75 , 0.583, 0.708, 0.667, 0.542, 0.583, 0.417, 0.583,
       0.625, 0.583, 0.5  , 0.458, 0.583, 0.875, 0.917, 0.458, 0.5  ,
       0.625, 0.458, 0.417, 0.583, 0.625, 0.125, 0.5  , 0.625, 0.75 ,
       0.417, 0.75 , 0.5  , 0.708, 0.542, 0.5  , 0.5  , 0.458, 0.125,
       0.333, 0.333, 0.542, 0.167, 0.375, 0.292, 0.   , 0.417, 0.083,
       0.375, 0.375, 0.458, 0.417, 0.292, 0.083, 0.208, 0.5  , 0.333,
       0.208, 0.333, 0.375, 0.417, 0.333, 0.417, 0.375, 0.25 , 0.167,
       0.167, 0.292, 0.292, 0.417, 0.583, 0.458, 0.125, 0.417, 0.208,
       0.25 , 0.417, 0.25 , 0.125, 0.292, 0.417, 0.375, 0.375, 0.208,
       0.333, 0.542, 0.292, 0.417, 0.375, 0.417, 0.417, 0.208, 0.375,
       0.208, 0.667, 0.5  , 0.292, 0.417, 0.208, 0.333, 0.5  , 0.417,
       0.75 , 0.25 , 0.083, 0.5  , 0.333, 0.333, 0.292, 0.542, 0.5  ,
       0.333, 0.417, 0.333, 0.417, 0.333, 0.75 , 0.333, 0.333, 0.25 ,
       0.417, 0.583, 0.458, 0.417, 0.458, 0.458, 0.458, 0.292, 0.5  ,
       0.542, 0.417, 0.208, 0.417, 0.583, 0.417])
```

分布可视化

复制

```
import seaborn as sns
sns.distplot(s,kde=False,rug=True)
```

频率分布直方图：

复制

```
sns.distplot(s,hist=True,kde=True,rug=True)
```

带高斯密度核函数的直方图：

252 创建一个 [3,5] 所有元素为 True 的数组

复制
```
In [15]: np.ones((3,5),dtype=bool)
Out[15]:
array([[ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

253 一维数组转二维

复制
```
In [18]: a =  np.linspace(1,5,10)

In [19]: a.reshape(5,2)
Out[19]:
array([[1.        , 1.44444444],
       [1.88888889, 2.33333333],
       [2.77777778, 3.22222222],
       [3.66666667, 4.11111111],
       [4.55555556, 5.        ]])
```

254 数组所有奇数替换为 -1

复制
```
In [14]: m = np.arange(10).reshape(2,5)

In [16]: m[m%2==1] = -1
In [17]: m
Out[17]:
array([[ 0, -1,  2, -1,  4],
       [-1,  6, -1,  8, -1]])
```

255 提取出数组中所有奇数

复制
```
In [18]: m = np.arange(10).reshape(2,5)

In [19]: m[m%2==1]
Out[19]: array([1, 3, 5, 7, 9])
```

256 求 2 个 NumPy 数组的交集

复制
```
In [21]: m ,n = np.arange(10), np.arange(1,15,3)

In [22]: np.intersect1d(m,n)
Out[22]: array([1, 4, 7])
```

257 求 2 个 NumPy 数组的差集

复制
```
In [21]: m ,n = np.arange(10), np.arange(1,15,3)

In [23]: np.setdiff1d(m,n)
Out[23]: array([0, 2, 3, 5, 6, 8, 9])
```

258 筛选出指定区间内的所有元素

注意：（m >2），必须要添加一对括号

复制
```
In [21]: m = np.arange(10).reshape(2,5)
In [34]: m[(m > 2) & (m < 7)]
Out[34]: array([3, 4, 5, 6])
```

259 二维数组交换 2 列

复制
```
In [21]: m = np.arange(10).reshape(2,5)
In [37]: m
Out[37]:
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])

In [36]: m[:,[1,0,2,3,4]]
Out[36]:
array([[1, 0, 2, 3, 4],
       [6, 5, 7, 8, 9]])
```

可以一次交换多列：

复制
```
In [38]: m[:,[1,0,2,4,3]]
Out[38]:
array([[1, 0, 2, 4, 3],
       [6, 5, 7, 9, 8]])
```

260 二维数组反转行

复制
```
In [39]: m = np.arange(10).reshape(2,5)

In [40]: m
Out[40]:
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])

In [41]: m[::-1]
Out[41]:
array([[5, 6, 7, 8, 9],
       [0, 1, 2, 3, 4]])
```

261 生成数值 5~10 ，shape 为 (3,5) 的随机浮点数：

复制
```
In [9]: np.random.seed(100)

In [42]: np.random.randint(5,10,(3,5)) + np.random.rand(3,5)
Out[42]:
```

```
array([[9.31623868, 5.68431289, 9.5974916 , 5.85600452, 9.3478736 ],
       [5.66356114, 7.78257215, 7.81974462, 6.60320117, 7.17326763],
       [7.77318114, 6.81505713, 9.21447171, 5.08486345, 8.47547692]])
```

262 揭秘 Shape

一个一维数组，长度为 12，为什么能变化为二维 `(12,1)` 或 `(2,6)` 等，三维 `(12,1,1)` 或 `(2,3,2)` 等，四维 `(12,1,1,1)` 或 `(2,3,1,2)` 等。总之，能变化为任意多维度。

reshape 是如何做到的？使用了什么魔法数据结构和算法吗？

这篇文章对于 reshape 方法的原理解释，会很独到，尽可能让朋友们弄明白数组 reshape 的魔法。

如同往常一样，导入 NumPy 包：

复制
```
import numpy as np
```

创建一个一维数组 `a`，从 0 开始，间隔为 2 ，含有 12 个元素的数组：

复制
```
a = np.arange(0,24,2)
```

打印数组 a

复制
```
In [48]: a
Out[48]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22])
```

如上数组 a, NumPy 会将其解读成两个结构，一个 `buffer`，还有一个 `view`。

`buffer` 的示意图如下所示：

`view` 是解释 `buffer` 的一个结构，比如数据类型，`flags`信息等：

复制
```
In [50]: a.dtype
Out[50]: dtype('int32')

In [51]: a.flags
Out[51]:
  C_CONTIGUOUS : True
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False
```

使用 `a[6]` 访问数组 a 中 index 为 6 的元素。从背后实现看， NumPy 会辅助一个轴，轴的取值为 0 到 11 。

从概念上看，它的示意图如下所示：

所以，借助这个轴 `i` ，`a[6]` 就会被索引到元素 12，如下所示：

至此，大家要建立一个轴的概念。

接下来，做一次 reshape 变化，变化数组 a 的 shape 为 (2,6):

复制
```
b = a.reshape(2,6)
```

打印 b:

复制
```
In [53]: b
Out[53]:
array([[ 0,  2,  4,  6,  8, 10],
       [12, 14, 16, 18, 20, 22]])
```

此时，NumPy 会建立两个轴，假设为 `i`,`j` ，`i` 的取值为 0 到 1,`j` 的取值为 0 到 5，示意图如下：

使用 `b[1][2]` 获取元素到 16

复制
```
In [54]: b[1][2]
Out[54]: 16
```

两个轴的取值分为 1,2，如下图所示，定位到元素 16

平时，有些读者朋友可能会混淆两个 shape,(12,) 和 (12,1) ，其实前者一个轴，后者两个轴，示意图分别如下：

一个轴，取值从 0 到 11

两个轴，i 轴取值从 0 到 11， j 轴取值从 0 到 0

至此，大家要建立两个轴的概念。

并且，通过上面几幅图看到，无论 shape 如何变化，变化的是视图，底下的 buffer 始终未变。

接下来，上升到三个轴，变化数组 `a` 的 shape 为 `(2,3,2)`：

复制

```
c = a.reshape(2,3,2)
```

打印 c:

复制

```
In [55]: c = a.reshape(2,3,2)

In [56]: c
Out[56]:
array([[[ 0,  2],
        [ 4,  6],
        [ 8, 10]],

       [[12, 14],
        [16, 18],
        [20, 22]]])
```

数组 c 有三个轴，取值分别为 0 到 1， 0 到 2， 0 到 1，示意图如下所示：

读者们注意体会，`i` ，`j`，`k` 三个轴，其值的分布规律。如果去掉 i 轴取值为 1 的单元格后，

实际就对应到数组 c 的前半部分元素：

复制

```
array([[[ 0,  2],
        [ 4,  6],
        [ 8, 10]],
```

也就是如下的索引组合 ：

至此，三个轴的 reshape 已经讲完，再说一个有意思的问题。

还记得，原始的一维数组 a 吗？它一共有 12 个元素，后来，我们变化它为数组 c ，shape 为 (2,3,2)，那么如何升级为 4 维或 任意维呢？

4 维可以为：(1,2,3,2)，示意图如下：

看到，轴 i 索引取值只有 0，它被称为自由维度，可以任意插入到原数组的任意轴间。

比如，5 维可以为：(1,2,1,3,2)：

至此，你应该完全理解 reshape 操作后的魔法：

- buffer 是个一维数组，永远不变；
- 变化的 shape 通过 view 传达；
- 取值仅有 0 的轴为自由轴，它能变化出任意维度。

关于 reshape 操作，最后再说一点，reshape 后的数组，仅仅是原来数组的视图 view，并没有发生复制元素的行为，这样才能保证 reshape 操作更为高效。

复制

```
In [58]: v1 = np.arange(10)

In [59]: v2 = v1.reshape(2,5)

In [60]: v2
Out[60]:
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

改变 `v2` 的第一个元素：

复制

```
In [61]: v2[0,0] = 10

In [62]: v2
Out[62]:
array([[10,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9]])
```

如果 v2 是 v1 的视图，那么 v1 也会改变，如下，v1 的第一个元素也发生相应改变，所以得证。

复制

```
In [63]: v1
Out[63]: array([10,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

在了解完 reshape 操作的奥秘后，相信大家都建立轴和多轴的概念，这对灵活使用高维数组很有帮助。

263 元素级操作

NumPy 中两个数组加减乘除等，默认都是对应元素的操作：

复制

```
In [59]: v1 = np.arange(5)

In [60]: v1
Out[60]: array([0, 1, 2, 3, 4])
```

执行，`v1+2` 操作，按照元素顺序逐个加 2：

复制

```
In [57]: v1+2
Out[57]: array([2, 3, 4, 5, 6])
```

执行 `v1 * v1`，注意是按照元素逐个相乘：

复制

```
In [58]: v1 * v1
Out[58]: array([ 0,  1,  4,  9, 16])
```

264 矩阵运算

线性代数中，矩阵的乘法操作在 `NumPy` 中怎么实现？

常见两种方法：使用dot 函数，另一种是转化为 matrix 对象。

`dot` 操作：

复制
```
# 数值[1,10)内, 生成shape为(5,2)的随机整数数组
In [1]: import numpy as np

In [2]: v1 = np.arange(5)

In [3]: v2 = np.random.randint(1,10,(5,2))

In [4]: np.dot(v1,v2)
Out[4]: array([49, 51])
```

另一种方法，将 `v1` 和 `v2` 分别转化为 `matrix` 对象：

复制
```
In [6]: np.matrix(v1)*np.matrix(v2)
Out[6]: matrix([[49, 51]])
```

需要注意，数组 `v1` 经过 matrix 转化后，shape 由原来 (5,) 变化为 (1,5) ，的确变得更像线性代数中的矩阵：

复制
```
In [83]: matrix(v1).shape
Out[83]: (1, 5)
```

首先，导入与求行列式相关的模块 `linalg` ，求矩阵的行列式，要求数组的最后两个维度相等。

复制
```
In [10]: from numpy import linalg

In [11]: v1 = np.arange(12)
In [12]: v2 = v1.reshape(3,2,2)

In [13]: linalg.det(v2)
Out[13]: array([-2., -2., -2.])

In [14]: v3 = np.arange(9).reshape(3,3)
In [15]: linalg.det(v3)
Out[15]: 0.0
```

265 NumPy 求 9 大统计变量

NumPy 能方便的求出统计学常见的描述性统计量。

1) 求平均值

复制
```
In [21]: m1 = np.random.randint(1,10,(3,4))

In [22]: m1
Out[22]:
array([[8, 4, 8, 2],
       [4, 9, 2, 1],
       [7, 7, 7, 5]])

In [23]: m1.mean() # 默认求出数组所有元素的平均值
Out[23]: 5.333333333333333

In [24]: m1.sum() / 12
Out[24]: 5.333333333333333
```

若想求某一维度的平均值，设置 `axis` 参数，求 axis 等于 1 的平均值：

复制
```
In [26]: m1.mean(axis = 1)
Out[26]: array([5.5, 4. , 6.5])
```

2) 求标准差

如下，分别求所有元素的标准差、某一维度上的标准差：

复制
```
In [28]: m1.std()
Out[28]: 2.592724864350674

In [29]: m1.std(axis=1)
Out[29]: array([2.59807621, 3.082207  , 0.8660254 ])
```

3) 求方差

如下，分别求所有元素的方差、某一维度上的方差：

复制
```
In [30]: m1.var()
Out[30]: 6.722222222222221

In [31]: m1.var(axis=1)
Out[31]: array([6.75, 9.5 , 0.75])
```

4) 求最大值

如下，分别求所有元素的最大值、某一维度上的最大值：

复制
```
In [34]: m1.max()
Out[34]: 9

In [35]: m1.max(axis=1)
Out[35]: array([8, 9, 7])
```

5) 求最小值

如下，分别求所有元素的最小值、某一维度上的最小值：

复制
```
In [36]: m1.min()
Out[36]: 1

In [37]: m1.min(axis=1)
```

```
Out[37]: array([2, 1, 5])
```

6) 求和

如下，分别求所有维度上元素的和、某一维度上的元素和：

复制

```
In [38]: m1.sum()
Out[38]: 64

In [39]: m1.sum(axis=1)
Out[39]: array([22, 16, 26])
```

7) 求累乘

如下，分别求所有维度上元素的累乘、某一维度上的累乘：

复制

```
In [22]: m1
Out[22]:
array([[8, 4, 8, 2],
       [4, 9, 2, 1],
       [7, 7, 7, 5]])

In [40]: m1.cumprod()
Out[40]:
array([      8,       32,      256,      512,     2048,    18432,
         36864,    36864,   258048,  1806336, 12644352, 63221760],
      dtype=int32)

In [42]: m1.cumprod(axis=1)
Out[42]:
array([[   8,   32,  256,  512],
       [   4,   36,   72,   72],
       [   7,   49,  343, 1715]], dtype=int32)
```

8) 求累和

如下，分别求所有维度上元素的累加和、某一维度上的累加和：

复制

```
In [43]: m1.cumsum()
Out[43]: array([ 8, 12, 20, 22, 26, 35, 37, 38, 45, 52, 59, 64], dtype=in
t32)

In [44]: m1.cumsum(axis=1)
Out[44]:
array([[ 8, 12, 20, 22],
       [ 4, 13, 15, 16],
       [ 7, 14, 21, 26]], dtype=int32)
```

9) 求迹

对角线上元素的和：

复制

```
In [22]: m1
Out[22]:
array([[8, 4, 8, 2],
       [4, 9, 2, 1],
       [7, 7, 7, 5]])

In [45]: m1.trace()
Out[45]: 24
```

266 NumPy 之 flatten 函数

NumPy 的 `flatten` 函数也有改变 shape 的能力，它将高维数组变为向量。但是，它会发生数组复制行为。

复制

```
In [68]: v1 = np.random.randint(1,10,(2,3))

In [69]: v1
Out[69]:
array([[3, 8, 5],
       [2, 3, 4]])

In [70]: v2 = v1.flatten()

In [71]: v2
Out[71]: array([3, 8, 5, 2, 3, 4])
```

`v2[0]` 被修改为 30 后，原数组 `v1` 没有任何改变。

复制

```
In [73]: v2[0] = 30

In [74]: v1
Out[74]:
array([[3, 8, 5],
       [2, 3, 4]])
```

267 NumPy 之 newaxis

使用 `newaxis` 增加一个维度，维度的索引只有 0，本篇的开头已经详细解释过，不再赘述。

复制

```
In [81]: v1 = np.arange(10) # shape 为一维，(10, )
In [82]: v2 = v1[:,np.newaxis] # shape 为二维，(10,1)
```

268 NumPy 之 repeat

`repeat` 操作，实现某一维上的元素复制操作。

在维度 0 上复制元素 2 次：

复制

```
In [132]: a = np.array([[1,2],[3,4]])

In [138]: np.repeat(a,2,axis=0)
Out[138]:
array([[1, 2],
       [1, 2],
       [3, 4],
       [3, 4]])
```

在维度 1 上复制元素 2 次：

```
In [137]: np.repeat(a,2,axis=1)
Out[137]:
array([[1, 1, 2, 2],
       [3, 3, 4, 4]])
```

269 NumPy 之 tile 按块复制元素

tile 实现按块复制元素：

```
In [4]: a = np.array([[1,2],[3,4]])

In [89]: np.tile(a,3)
Out[89]:
array([[1, 2, 1, 2, 1, 2],
       [3, 4, 3, 4, 3, 4]])

In [6]: np.tile(a,(2,3))
Out[6]:
array([[1, 2, 1, 2, 1, 2],
       [3, 4, 3, 4, 3, 4],
       [1, 2, 1, 2, 1, 2],
       [3, 4, 3, 4, 3, 4]])
```

270 NumPy 之 vstack

`vstack` ：vertical stack，沿竖直方向合并多个数组：

```
In [4]: a = np.array([[1,2],[3,4]])
In [5]: b = np.array([[-1,-2]])
In [6]: c = np.vstack((a,b)) # 注意参数类型:元组
Out[5]:
array([[ 1,  2],
       [ 3,  4],
       [-1, -2]])
```

271 NumPy 之 hstack

`hstack` 沿水平方向合并多个数组。

值得注意，不管是 `vstack` ，还是 `hstack` ，沿着合并方向的维度，其元素的长度要一致。

```
In [4]: a = np.array([[1,2],[3,4]])
In [5]: b = np.array([[5,6,7],[8,9,10]])
In [4]: c = np.hstack((a,b))
In [5]: c
Out[5]:
array([[ 1,  2,  5,  6,  7],
       [ 3,  4,  8,  9, 10]])
```

272 NumPy 之 concatenate

`concatenate` 指定在哪个维度上合作数组。

```
In [4]: a = np.array([[1,2],[3,4]])
In [5]: b = np.array([[-1,-2]])
In [6]: np.concatenate((a,b),axis=0) # 效果等于vstack
Out[6]:
array([[ 1,  2],
       [ 3,  4],
       [-1, -2]])

In [7]:  c = np.array([[5,6,7],[8,9,10]])
In [108]: np.concatenate((a,c),axis=1) # 效果等于hstack
Out[108]:
array([[ 1,  2,  5,  6,  7],
       [ 3,  4,  8,  9, 10]])
```

NumPy 还有一些小 track，比如 `r_` 类，`c_` 类，也能实现合并操作。

```
In [4]: a = np.array([[1,2],[3,4]])
In [5]: b = np.array([[-1,-2]])
In [6]: np.r_[a,b] # r_类
Out[6]:
array([[ 1,  2],
       [ 3,  4],
       [-1, -2]])
```

```
In [7]: a = np.array([[1,2],[3,4]])
In [8]: c = np.array([[5,6,7],[8,9,10]])
In [9]: np.c_[a,c] # c_ 类
Out[9]:
array([[ 1,  2,  5,  6,  7],
       [ 3,  4,  8,  9, 10]])
```

```
In [10]: np.r_[a,c[:,:2]]
Out[10]:
array([[1, 2],
       [3, 4],
       [5, 6],
       [8, 9]])
```

273 NumPy 之 argmax，argmin

`argmax` 返回数组中某个维度的最大值索引，当未指明维度时，返回 buffer 中最大值索引。如下所示：

```
In [131]: a = np.random.randint(1,10,(2,3))

In [132]: a
Out[132]:
array([[8, 1, 4],
       [5, 4, 3]])

In [133]: a.argmax()
Out[133]: 0

In [134]: a.argmax(axis = 0)
```

```
Out[134]: array([0, 1, 0], dtype=int64)

In [135]: a.argmax(axis = 1)
Out[135]: array([0, 0], dtype=int64)
```

274 初步认识 NumPy 的广播

广播，英文 broadcasting，有些读者是在使用 NumPy 时，从报错信息中第一次见到 broadcasting。

那么，什么是广播？广播的规则又是怎样的？在 NumPy 中，下列操作是有效的：

复制

```
In [1]: import numpy as np

In [2]: v1 = np.arange(10).reshape(2,5) # v1 shape: (2,5)

In [3]: v2 = np.array([2]) # v2 shape: (1,)

In [4]: v1 + v2
Out[4]:
array([[ 2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11]])
```

v1 的 shape 为 `(2,5)`，v2 的 shape 为 (1,)，一个为二维，一个为一维。如果没有广播机制，一定会抛(出，维数不等无法相加的异常。

但是，因为广播机制的存在，v2 数组会适配 v1 数组，按照第 0 ,1 维度，分别发生一次广播，广播后的 v2 变为：

复制

```
v2 = np.tile(2,(2,5))
Out[10]:
array([[2, 2, 2, 2, 2],
       [2, 2, 2, 2, 2]])
```

然后，执行再执行加法操作时，因为 v1, v2 的 shape 变得完全一致，所以就能实现相加操作了。

但是，如果 v2 的 shape 为 `(2,)`，如下， 是否 v2 广播后，能实现 v1 + v2 操作？

复制

```
v2 = np.array([1,2])
```

执行 v1 + v2 后，抛出 shapes (2,5) (2,) 无法广播到一起的异常。

复制

```
In [11]: v2 = np.array([1,2])

In [12]: v1 + v2

ValueError: operands could not be broadcast together with shapes (2,5) (2
,)
```

因为 v1 , v2 按照广播的规则，无法达成一致的 shape，所以抛出异常。下面了解广播的具体规则。

275 NumPy 之广播的规则

以上看到，不是任意 shape 的多个数组，操作时都能广播到一起，必须满足一定的约束条件。

NumPy 首先会比较最靠右的维度，如果最靠右的维度相等或其中一个为 1，则认为此维度相等；

那么，再继续向左比较，如果一直满足，则认为两者兼容；

最后，分别在对应维度上发生广播，以此补齐直到维度一致。

如下，两个数组 a, b, shape 分别为 (2,1,3) , (4,3), 它们能否广播兼容？ 我们来分析下。

复制

```
a = np.arange(6).reshape(2,1,3) # shape: (2,1,3)
b = np.arange(12).reshape(4,3) # shape: (4,3)
```

- 1 按照规则，从最右侧维度开始比较，数组 a, b 在此维度上的长度都为 3， 相等；
- 2 继续向左比较，a 在此维度上长度为 1， b 长度为 4，根据规则，也认为此维度是兼容的；
- 3 继续比较，但是数组 b 已到维度终点，停止比较。

结论，数组 a 和 b 兼容，通过广播能实现 shape 一致。

276 NumPy 广播实施的具体步骤拆分案例

下面看看，数组 a 和 b 广播操作实施的具体步骤。

复制

```
In [36]: a # 初始 a
Out[36]:
array([[[0, 1, 2]],

       [[3, 4, 5]]])

In [39]: b # 初始 b
Out[39]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

维度编号从 0 开始，数组 a 在维度 1 上发生广播，复制 4 次：

复制

```
a = np.repeat(a,4,axis=1)
```

打印 a ：

复制

```
Out[35]:
array([[[0, 1, 2],
        [0, 1, 2],
        [0, 1, 2],
```

```
         [0, 1, 2]],

        [[3, 4, 5],
         [3, 4, 5],
         [3, 4, 5],
         [3, 4, 5]]])
```

此时，数组 a 和 b 在后两个维度一致，但是数组 b 维度缺少一维，所以 b 也会广播一次：

复制

```
b = b[np.newaxis,:,:]  # 首先增加一个维度
b = np.repeat(b,2,axis=0) # 在维度 0 上复制 2 次
```

经过以上操作，数组 a 和 b 维度都变为 (2,4,3)，至此广播完成，做个加法操作：

复制

```
In [55]: a + b
Out[55]:
array([[[ 0,  2,  4],
        [ 3,  5,  7],
        [ 6,  8, 10],
        [ 9, 11, 13]],

       [[ 3,  5,  7],
        [ 6,  8, 10],
        [ 9, 11, 13],
        [12, 14, 16]]])
```

验证我们自己实现的广播操作，是否与 NumPy 中的广播操作一致，直接使用原始的 a 和 b 数组相加，看到与上面得到的结果一致。

复制

```
a = np.arange(6).reshape(2,1,3) # shape: (2,1,3)
b = np.arange(12).reshape(4,3) # shape: (4,3)
a + b
Out[56]:
array([[[ 0,  2,  4],
        [ 3,  5,  7],
        [ 6,  8, 10],
        [ 9, 11, 13]],

       [[ 3,  5,  7],
        [ 6,  8, 10],
        [ 9, 11, 13],
        [12, 14, 16]]])
```

至此，广播规则总结完毕。

建议大家都好好理解广播机制，因为接下来使用 NumPy 函数，或者查看文档时，再遇到 broadcast ，就知道它的规则，加快对函数的理解。

并且，即便遇到广播不兼容的 bug 时，相信也能很快解决。

277 返回有规律的数组

已知数组：

复制

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

返回如下数组：

复制

```
array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

分析

数组前半部分 `1, 1, 1, 2, 2, 2, 3, 3, 3` 通过 repeat 函数复制 3 次， 后面部分通过 tile 函数复制 3 次，体会二者区别。然后合并数据。

复制

```
In [59]: np.hstack((np.repeat(a,3), np.tile(a,3)))
Out[59]: array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])
```

278 Python 实现向量化

借助 NumPy 的 `vectorize` 实现操作向量化

原生的 Python 列表不支持向量化操作，两个列表相加默认不是逐个元素相加：

复制

```
a = [1,3,5]
b = [2,4,6]
a + b # 默认实现的不是逐个元素相加操作
```

但是，借助 vectorize 能实现矢量相加：

复制

```
def add(x,y):
    return x+y
addv = np.vectorize(add)
```


复制

```
In [67]: addv(a,b)
Out[67]: array([ 3,  7, 11])
```

279 NumPy 限制打印元素的个数

使用 `set_printoptions` 限制打印元素的个数

复制

```
np.set_printoptions(threshold=5)
```

280 NumPy 求中位数

求如下三维数组 a，沿 axis = 1 的中位数。

使用 median 方法，因为 axis 为 1 的数组元素长度为 4， 所以中位数为中间两个数的平均数。

如切片 a[0,:,0] 为 `[4,8,5,3]` ，排序后的中间两个元素为 `[4,5]` ，平均值为 4.5.

240 NumPy 数值计...
241 创建 NumPy ...
242 NumPy 数组之...
243 NumPy 之 ara...
244 NumPy 之 lins...
245 NumPy 之 logs...
246 NumPy 之 创...
247 NumPy 之创建...
248 NumPy之创建...
249 NumPy 之创建...
250 NumPy 之索引...

```
In [75]: a
array([[[4, 2, 4],
        [8, 2, 7],
        [5, 3, 6],
        [3, 2, 3]],

       [[2, 6, 1],
        [5, 9, 8],
        [9, 7, 1],
        [2, 1, 1]]])
In [73]: ma = np.median(a,axis = 1)
Out[74]:
array([[4.5, 2. , 5. ],
       [3.5, 6.5, 1. ]])
```

281 NumPy 计算 softmax 得分值

已知数组 a ，求 softmax 得分值。

```
In [81]: a
Out[81]:
array([0.07810512, 0.12083313, 0.23554504, 0.62057901, 0.3437597 ,
       0.10876455, 0.08338525, 0.28873765, 0.54033942, 0.71941148])
```

定义 softmax 函数：

```
def softmax(a):
    e_a = np.exp(a - np.max(a))
    return e_a / e_a.sum(axis=0)
```

调用 softmax ，得到每个元素的得分，因为 softmax 单调递增函数，所以输入值越大，得分值越高。

```
In [85]: sm
Out[85]:
array([0.07694574, 0.08030473, 0.0900658 , 0.13236648, 0.10035914,
       0.07934139, 0.0773531 , 0.09498634, 0.12216039, 0.14611689])
```

sum(sm) 等于 1

282 NumPy 求任意分位数

已知数组 a，求 20 分位数，80 分位数

```
a = np.arange(11)
```

使用 percentile 函数，q 为分位数列表

```
In [95]: np.percentile(a,q=[20,80])
Out[95]: array([2., 8.])
```

283 找到 NumPy 中缺失值

NumPy 使用 `np.nan` 标记缺失值，给定如下数组 a，求出缺失值的索引。

如下使用 where 函数，返回满足条件的位置索引：

```
In [119]: a = np.array([ 0.,  1., np.nan,  3., np.nan, np.nan,  6.,  7.,
8.,  9.])
In [123]: np.where(np.isnan(a))
Out[123]: (array([2, 4, 5], dtype=int64),)
```

284 NumPy 返回无缺失值的行

给定数组，找出没有任何缺失值的行，

```
a = np.array([[ 0., np.nan,  2.,  3.],
       [ 4.,  5., np.nan,  7.],
       [ 8.,  9., 10., 11.],
       [12., 13., np.nan, 15.],
       [16., 17., np.nan, 19.],
       [20., 21., 22., 23.]])
```

求解方法：

```
In [135]: m = np.sum(np.isnan(a), axis = 1) == 0

In [136]: m
Out[136]: array([False, False,  True, False, False,  True])

In [137]: a[m]
Out[137]:
array([[ 8.,  9., 10., 11.],
       [20., 21., 22., 23.]])
```

285 NumPy 求相关系数

求如下二维数组 a 的相关系数

```
a = array([[ 2, 12, 21, 10],
       [ 1, 20,  8, 22],
       [ 7,  1,  5,  1],
       [ 7, 10, 14, 14],
       [12, 13, 13, 14],
       [ 0, 12, 21,  2]])
```

如下使用 corrcoef 方法，求得两列的相关系数，相关系数为 0.242

```
In [149]: np.corrcoef(a[:,1],a[:,2])
Out[149]:
array([[1.        , 0.24230838],
       [0.24230838, 1.        ]])
```

286 缺失值默认用0填充

如下数组，含有缺失值，使用 0 填充：

复制

```
a = np.array([[ 0., np.nan,  2.,  3.],
       [ 4.,  5., np.nan,  7.],
       [ 8.,  9., 10., 11.],
       [12., 13., np.nan, 15.],
       [16., 17., np.nan, 19.],
       [20., 21., 22., 23.]])
```

一行代码，np.isnan(a) 逐元素检查，若为空则为 True，否则为 False，得到一个与原来 shape 相同的值为 True 和 False 的数组。

复制

```
In [157]: a[np.isnan(a)] = 0

In [158]: a
Out[158]:
array([[ 0.,  0.,  2.,  3.],
       [ 4.,  5.,  0.,  7.],
       [ 8.,  9., 10., 11.],
       [12., 13.,  0., 15.],
       [16., 17.,  0., 19.],
       [20., 21., 22., 23.]])
```

287 使用 NumPy 处理 fashion-mnist 数据集

fashion-mnist 是一个与手写字一样经典的数据集，与服饰相关。

导入数据特征数 785，我们先提取 0 到 784 ，然后 reshape 为 28 * 28 的二维数组。

首先导入 NumPy，截取前 784 个元素，reshape 为 28 * 28 的元素。大家自行下载此数据集，下载并导入后，train_data 的 shape 为 (*,785).

复制

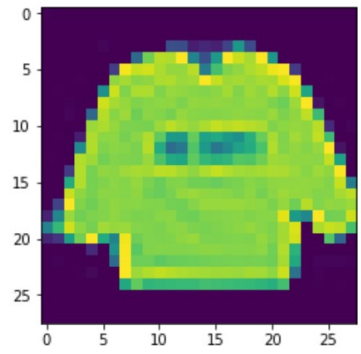
```
import numpy as np
train_data = fashion_mnist_train.to_numpy() # Pandas DataFrame 转 numpy 对象

row0 = train_data[0,:784].reshape(28,-1)
```

导入 matplotlib ，使用 imshow 绘制 784 个像素（取值为 0-255 ）

复制

```
import matplotlib.pyplot as plt
plt.imshow(row0)
plt.show()
```

展示的图像，如下所示：



依次展示前 10 幅图：

复制

```
for i in range(10):
    print('图%d'%(i,))
    plt.imshow(train_data[i,:784].reshape(28,-1))
    plt.colorbar()
    plt.show()
```

下一章

互动评论

说点什么

评论

The Scrapper 1 个月前

不错

鼓掌