Numpy 使用教程

一、实验介绍

1.1 实验内容

如果你使用 Python 语言进行科学计算，那么一定会接触到 Numpy。Numpy 是支持 Python 语言的数值计算扩充库，其拥有强大的高维度数组处理与矩阵运算能力。除此之外，Numpy 还内建了大量的函数，方便你快速构建数学模型。

1.2 实验知识点

- Numpy 数组索引
- Numpy 其他用法

1.3 实验环境

- python2.7
- Xfce 终端
- ipython 终端

1.4 适合人群

本课程难度为一般，属于初级级别课程，适合具有 Python 基础，并对使用 Numpy 进行科学计算感兴趣的用户。

二、Numpy 数组索引和切片

我们已经明确了，Ndarray 是 Numpy 的组成核心，那么对于 Numpy 的多维数组，其实它完整集成了 python 对于数组的索引语法 array[obj]。随着 obj 的不同，我们可以实现字段访问、数组切片、以及其他高级索引功能。

2.1 数组索引

我们可以通过索引值（从 0 开始）来访问 Ndarray 中的特定位置元素。Numpy 中的索引和 python 对 list 索引的方式非常相似，但又有所不同。我们一起来看一下：

首先是，一维数据索引：

```
>>> import numpy as np

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

# 获取索引值为 1 的数据
>>> a[1]
1
# 分别获取索引值为 1, 2, 3 的数据
>>> a[[1, 2, 3]]
array([1, 2, 3])
```

对于二维数据而言：

```
>>> import numpy as np

>>> a = np.arange(20).reshape(4,5)

>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])

# 获取第 2 行，第 3 列的数据
>>> a[1,2]
7
```

如果，我们使用 python 中的 list 索引同样的值，看看有什么区别：

```
# 创建一个数据相同的 list
>>> a = [[ 0,  1,  2,  3,  4],[ 5,  6,  7,  8,  9],[10, 11, 12, 13, 14],[15, 16, 17, 18, 19]]

# 按照上面的方法获取第 2 行，第 3 列的数据，报错。
>>> a[1,2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers or slices, not tuple

# python 中 list 索引 2 维数据的方法
>>> a[1][2]
7
```

如何索引二维 Ndarray 中的多个元素值，这里使用逗号，分割：

```
>>> import numpy as np

>>> a = np.arange(20).reshape(4,5)

>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])

# 索引
>>> a[[1,2],[3,4]]
array([ 8, 14])
```

这里需要注意索引的对应关系。我们实际获取的是 `[1,3]` ，也就是第 2 行和第 4 列对于的值 8 。以及 `[2, 4]` ，也就是第 3 行和第 5 列对于的值 14 。

那么，三维数据呢？

```
>>> import numpy as np

>>> a = np.arange(30).reshape(2,5,3)
>>> a
array([[[ 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]]])

# 索引
>>> a[[0,1],[1,2],[1,2]]
array([ 4, 23])
```

这里，`[0,1]` 分布代表 `axis = 0` 和 `axis = 1` 。而，后面的 `[1,2],[1,2]` 分别选择了第 `2` 行第 `2` 列和第 `3` 行第 `3` 列的两个数。

2.2 数组切片

Numpy 里面针对 Ndarray 的数组切片和 python 里的 list 切片操作是一样的。其语法为：

```
Ndarray[start:stop:step]
```

start:stop:step 分布代表 起始索引：截至索引：步长 。对于一维数组：

```
>>> import numpy as np

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

>>> a[:5]
array([0, 1, 2, 3, 4])

>>> a[5:10]
array([5, 6, 7, 8, 9])

>>> a[0:10:2]
array([0, 2, 4, 6, 8])
```

对于多维数组，我们只需要用逗号，分割不同维度即可：

```
>>> import numpy as np

>>> a = np.arange(20).reshape(4,5)

>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])

# 先取第 3，4 列（第一个维度），再取第 1，2，3 行（第二个维度）。
>>> a[0:3,2:4]
array([[ 2,  3],
       [ 7,  8],
       [12, 13]])

# 按步长为 2 取所有列和所有行的数据。
>>> a[:,::2]
array([[ 0,  2,  4],
       [ 5,  7,  9],
       [10, 12, 14],
       [15, 17, 19]])
```

当超过 3 维或更多维时，用 2 维数据的切片方式类推即可。

2.3 索引与切片区别

你可能有点疑问，上面的索引和切片怎么看起来这么相似呢？

它们的语法的确很相似，但实际上有区别：

1. 修改切片中的内容会影响原始数组。

```
>>> import numpy as np

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

>>> a[1] = 100

>>> a
array([0, 100, 2, 3, 4, 5, 6, 7, 8, 9])
```

除此之外，切片只能通过步长控制得到连续的值，而索引可以得到任意值。也就是说，索引的自由度更大。

三、排序、搜索、计数

最后，再介绍几个 numpy 针对数组元素的使用方法，分别是排序、搜索和计数。

3.1 排序

我们可以使用 `numpy.sort` 方法对多维数组元素进行排序。其方法为：

```
numpy.sort(a, axis=-1, kind='quicksort', order=None)
```

其中：

- `a` ：数组。
- `axis` ：要排序的轴。如果为 `None` ，则在排序之前将数组铺平。默认值为 `-1` ，沿最后一个轴排序。
- `kind` ：`{'quicksort', 'mergesort', 'heapsort'}` ，排序算法。默认值为 `quicksort` 。

举个例子：

```
>>> import numpy as np

>>> a = np.random.rand(20).reshape(4,5)
>>> a
array([[ 0.32930243,  0.63665893,  0.67589989,  0.05413352,  0.26090526],
       [ 0.6996066 ,  0.66006238,  0.88240934,  0.17563549,  0.03015105],
       [ 0.79075184,  0.40115859,  0.39336513,  0.64691791,  0.96333534],
       [ 0.20052738,  0.46157057,  0.48653336,  0.34537645,  0.54597273]])

>>> np.sort(a)
array([[ 0.05413352,  0.26090526,  0.32930243,  0.63665893,  0.67589989],
       [ 0.03015105,  0.17563549,  0.66006238,  0.6996066 ,  0.88240934],
       [ 0.39336513,  0.40115859,  0.64691791,  0.79075184,  0.96333534],
       [ 0.20052738,  0.34537645,  0.46157057,  0.48653336,  0.54597273]])
```

除了 numpy.sort，还有这样一些对数组进行排序的方法：

1. `numpy.lexsort(keys ,axis)` ：使用多个键进行间接排序。
2. `numpy.argsort(a ,axis,kind,order)` ：沿给定轴执行间接排序。
3. `numpy.msort(a)` ：沿第 1 个轴排序。
4. `numpy.sort_complex(a)` ：针对复数排序。

3.2 搜索和计数

除了排序，我们可以通过下面这些方法对数组中元素进行搜索和计数。列举如下：

1. `argmax(a ,axis,out)` ：返回数组中指定轴的最大值的索引。
2. `nanargmax(a ,axis)` ：返回数组中指定轴的最大值的索引,忽略 NaN。
3. `argmin(a ,axis,out)` ：返回数组中指定轴的最小值的索引。
4. `nanargmin(a ,axis)` ：返回数组中指定轴的最小值的索引,忽略 NaN。

5. argwhere(a)：返回数组中非 0 元素的索引,按元素分组。
6. nonzero(a)：返回数组中非 0 元素的索引。
7. flatnonzero(a)：返回数组中非 0 元素的索引,并铺平。
8. where(条件,x,y)：根据指定条件,从指定行、列返回元素。
9. searchsorted(a,v ,side,sorter)：查找要插入元素以维持顺序的索引。
10. extract(condition,arr)：返回满足某些条件的数组的元素。
11. count_nonzero(a)：计算数组中非 0 元素的数量。

选取其中的一些方法举例：

```
>>> import numpy as np
>>> a = np.random.randint(0,10,20)

>>> a
array([3, 2, 0, 4, 3, 1, 5, 8, 4, 6, 4, 5, 4, 2, 6, 6, 4, 9, 8, 9])

>>> np.argmax(a)
17

>>> np.nanargmax(a)
17

>>> np.argmin(a)
2

>>> np.nanargmin(a)
2

>>> np.argwhere(a)
array([[ 0],[ 1],[ 3],[ 4],[ 5],[ 6],[ 7],[ 8],[ 9],[10],[11],[12],[1
3],[14],[15],[16],[17],[18],[19]], dtype=int64)

>>> np.nonzero(a)
(array([ 0,  1,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 1
6, 17, 18, 19], dtype=int64),)

>>> np.flatnonzero(a)
array([ 0,  1,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16
, 17, 18, 19], dtype=int64)

>>> np.count_nonzero(a)
19
```

四、实验总结

最后一章，我们熟悉了 Ndarray 索引与切片相关的方法，这将对灵活处理多维数组提供帮助。除此之外，提到的排序、搜索与计数方法你可能不常用到，但是留下印象，以便不时之需。

上一节：Numpy 数学函数及代数运算 (/courses/912/labs/3424/document)