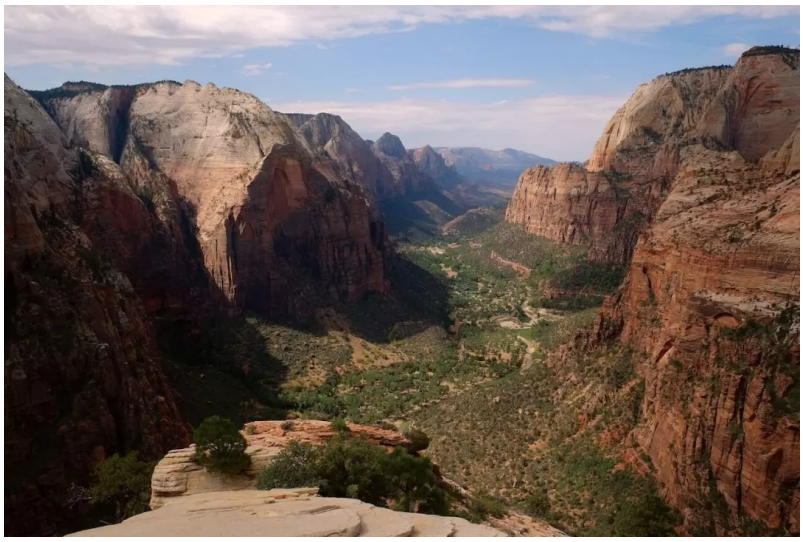
清晰易懂的Numpy进阶教程

原创 石头 机器学习算法那些事 2019-04-12

> 推荐阅读：
> 清晰易懂的Numpy入门教程

Numpy是数据分析和科学计算的核心包，上文详细介绍了Numpy的入门教程，本文将详细介绍Numpy的高级特性，这些特性对于数据分析和处理非常重要。



> 来源：Machine Learning Plus
> 翻译：石头

目录

1. 如何获取满足条设定件的索引

上一文介绍根据数组是否满足条件，输出为True或False.

```
# 定义数组
import numpy as np
```

```
arr_rand = np.array([8, 8, 3, 7, 7, 0, 4, 2, 5, 2])

#根据数组是否大于4，满足为True，不满足为False
b = arr_rand > 4
b

#> array([ True,  True, False,  True,  True, False, False, False,  True,
       False])
```

若我们想得到满足条件的索引，用np.where函数实现.

```
# 定位数组大于5的索引
index_gt5 = np.where(arr_rand > 5)
print("Positions where value > 5: ", index_gt5)

#> Positions where value > 5:  (array([0, 1, 3, 4]),) # 索引
```

由索引得到满足条件的值.

```
# 根据索引得到满足条件的数组值
arr_rand.take(index_gt5)

#> array([[8, 8, 7, 7]])
```

np.where也可以接受另两个可选择的参数x和y。当条件满足时，输出x，反之输出y.

```
# 如果值大于5，输出字符串'gt5'，反之输出'le5'
np.where(arr_rand > 5, 'gt5', 'le5')

#> array(['gt5', 'gt5', 'le5', 'gt5', 'gt5', 'le5', 'le5', 'le5', 'le5', 'le5'],
      dtype='<U3')
```

np.argmax和np.argmin分别获取数组最大值和最小值的索引.

```
# 最大值索引
print('Position of max value: ', np.argmax(arr_rand))

# 最小值索引
print('Position of min value: ', np.argmin(arr_rand))

#> Position of max value:  0
#> Position of min value:  5
```

np.max和np.min分别获取数组的最大值和最小值.

```
# 最大值
print('max value: ', np.max(arr_rand))

# 最小值
print('min value: ', np.min(arr_rand))

#> max value:  8
#> min value:  0
```

2. 如何将数据导入和导出csv文件

导入数据的标准方法是使用np.genfromtxt函数，它可以从web URLs导入数据，处理缺失值，多种分隔符，处理不规则的列数等功能。一个不太通用的版本是用np.loadtxt函数导入数据，它假设数据集无缺失值.

作为示例，我们尝试从下面的URL读取.csv文件，由于numpy数组的所有元素都应该是同一种类型，因此文本的最后一列默认为'nan'。通过设置参数'filling_values'，你可以用其他值代替缺失值.

```
# 关闭数字的科学表示方法
np.set_printoptions(suppress=True)

# 从url的csv文件导入数据
path = 'https://raw.githubusercontent.com/selva86/datasets/master/Auto.csv'
# delimiter：分隔符，skip_header：从多少行开始读数据，以0开始，filling_values：缺失值表示，dtype:
data = np.genfromtxt(path, delimiter=',', skip_header=1, filling_values=-999, dtype='float')

data[:3]  # 显示前3行数据

#> array([[  18. ,    8. ,  307. ,  130. , 3504. ,   12. ,   70. ,
#>           1. , -999. ],
#>        [  15. ,    8. ,  350. ,  165. , 3693. ,   11.5,   70. ,
#>           1. , -999. ],
#>        [  18. ,    8. ,  318. ,  150. , 3436. ,   11. ,   70. ,
#>           1. , -999. ]])
```

若设置参数dtype为'object'或'None'，np.genfromtxt在未设置占位符的前提下能同时处理具有数字和文本列的数据集.

```
# data2 = np.genfromtxt(path, delimiter=',', skip_header=1, dtype='object')
data2 = np.genfromtxt(path, delimiter=',', skip_header=1, dtype=None)
data2[:3]  # 显示前三行

#> array([( 18., 8,  307., 130, 3504,  12. , 70, 1, b'"chevrolet chevelle malibu"'),
#>        ( 15., 8,  350., 165, 3693,  11.5, 70, 1, b'"buick skylark 320"'),
#>        ( 18., 8,  318., 150, 3436,  11. , 70, 1, b'"plymouth satellite"')],
#>       dtype=[('f0', '<f8'), ('f1', '<i8'), ('f2', '<f8'), ('f3', '<i8'), ('f4', '<i8'), ('f5
```

最后，'np.savetxt'将数据保存为csv文件.

```
# 保存数据为csv文件
np.savetxt("out.csv", data, delimiter=",")
```

3. 如何保存和加载numpy数据

Numpy提供了.npy和.npz文件类型来实现。如果保存一个ndarray数据，使用np.save保存为.npy文件；若保存多个ndarray数据，使用np.savez保存为.npz文件。加载numpy数据，则统一用np.load函数.

```
# 保存单一的numpy数据，使用.npy文件
np.save('myarray.npy', arr2d)

# 保存多个numpy数据，使用.npz文件
np.savez('array.npz', arr2d_f, arr2d_b)

# 加载.npy文件
a = np.load('myarray.npy')
print(a)

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

加载.npz文件，获取特定的数组值.

```
# 加载.npz文件
b = np.load('array.npz')
print(b.files)
b['arr_0']

#> ['arr_0', 'arr_1']

#> array([[ 0.,  1.,  2.],
#>        [ 3.,  4.,  5.],
#>        [ 6.,  7.,  8.]])
```

虽然通过'arr_0'和'arr_1'获取了数组值，但是我们对这两个索引比较陌生，下面介绍手动设置索引保存和加载数组.

```
# 增加索引保存数据
b=np.savez('array.npz',arr2d_f=arr2d_f,arr2d_b=arr2d_b)
c = np.load('array.npz')
print(c.files)

c['arr2d_f']

#> ['arr2d_f', 'arr2d_b']

#> array([[1., 2., 3., 4.],
       [3., 4., 5., 6.],
       [5., 6., 7., 8.]])
```

4. 如何按列或行拼接numpy数组

本节介绍三种拼接numpy数组的方法：

- 方法1：设置np.concatenate参数axis的值为1或0，实现数组的列拼接或行拼接。
- 方法2：np.vstack和np.hstack
- 方法3：np.r_和np.c_

需要注意的是，np.r_和np.c_使用方括号来拼接数组，其他两种方法使用括号。

首先，定义两个需要拼接的数组.

```
# 定义两个拼接的数组
a = np.zeros([4, 4])
b = np.ones([4, 4])
print(a)
print(b)

#> [[ 0.  0.  0.  0.]
#>  [ 0.  0.  0.  0.]
#>  [ 0.  0.  0.  0.]
#>  [ 0.  0.  0.  0.]]

#> [[ 1.  1.  1.  1.]
#>  [ 1.  1.  1.  1.]
#>  [ 1.  1.  1.  1.]
#>  [ 1.  1.  1.  1.]]
```

行拼接数组

```
# 行拼接数组
np.concatenate([a, b], axis=0)
np.vstack([a,b])
np.r_[a,b]

#> array([[ 0.,  0.,  0.,  0.],
#>        [ 0.,  0.,  0.,  0.],
#>        [ 0.,  0.,  0.,  0.],
#>        [ 0.,  0.,  0.,  0.],
#>        [ 1.,  1.,  1.,  1.],
#>        [ 1.,  1.,  1.,  1.],
#>        [ 1.,  1.,  1.,  1.],
#>        [ 1.,  1.,  1.,  1.]])
```

列拼接数组

```
# 列拼接
np.concatenate([a, b], axis=1)
np.hstack([a,b])
np.c_[a,b]

#> array([[ 0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.],
#>        [ 0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.],
```

```
#>        [ 0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.],
#>        [ 0.,  0.,  0.,  0.,  1.,  1.,  1.,  1.]])
```

5. 如何按列对数据进行排序

本节介绍三种按列排序方法：np.sort，np.argsort和np.lexsort。在介绍三种排序方法前，先定义一个2维数组.

```
arr = np.random.randint(1,6, size=[8, 4])
arr

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

np.sort基于列对arr数组进行排序.

```
# axis=0表示列排序，1表示行排序
np.sort(arr, axis=0)

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

由上面结果分析，np.sort排序函数认为所有列是相互独立的，对所有列进行排序，破坏了每行的结构，使用np.argsort函数可以保留行的完整性 .

```
# 对arr的第一列进行排序，返回索引
sorted_index_1stcol = arr[:, 0].argsort()

# 根据第一列的索引对数组排序，保留了行的完整性
arr[sorted_index_1stcol]

#> array([[1, 5, 4, 5],
#>        [2, 4, 5, 5],
#>        [2, 4, 1, 3],
#>        [2, 2, 4, 3],
#>        [3, 3, 2, 1],
#>        [3, 1, 4, 2],
```

```
#>        [3, 4, 5, 5],
#>        [4, 4, 4, 2]])
```

倒转argsort索引实现递减排序

```
# 递减排序
arr[sorted_index_1stcol[::-1]]

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

若要基于多个列对数组进行排序，使用np.lexsort函数，它的参数是元组类型，元组的每个元素表示数组的某一列，排序规则是：越靠近右边的列，优先级越高.

```
# 先比较第一列，第一列相同的情况下再比较第二列
lexsorted_index = np.lexsort((arr[:, 1], arr[:, 0]))
arr[lexsorted_index]

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

6. 如何用Numpy处理日期

np.datatime64创建日期对象，精确度达到纳秒，你可以使用标准的YYYY-MM-DD格格式的字符串作为参数创建日期.

```
# 创建datetime64对象
date64 = np.datetime64('2018-02-04 23:10:10')
date64

#> numpy.datetime64('2018-02-04T23:10:10')
```

从datatime64对象分离时间

```
# 从datatime64对象分离时间
dt64 = np.datetime64(date64, 'D')
dt64
```

```
#> numpy.datetime64('2018-02-04')
```

如果你想增加天数或其他任何时间单元，比如月份，小时，秒等，使用np.timedelta函数非常方便.

```
# np.delta建立多个时间单元
tenminutes = np.timedelta64(10, 'm')                # 10 分钟
tenseconds = np.timedelta64(10, 's')                # 10 秒钟
tennanoseconds = np.timedelta64(10, 'ns')    # 10 纳秒

print('Add 10 days: ', dt64 + 10)            # 增加10天
print('Add 10 minutes: ', dt64 + tenminutes)        # 增加10分钟
print('Add 10 seconds: ', dt64 + tenseconds)        # 增加10秒
print('Add 10 nanoseconds: ', dt64 + tennanoseconds)  # 增加10纳秒

#> Add 10 days:  2018-02-14
#> Add 10 minutes:  2018-02-04T00:10
#> Add 10 seconds:  2018-02-04T00:00:10
#> Add 10 nanoseconds:  2018-02-04T00:00:00.000000010
```

dt64对象转化为字符串

```
# dt64转化为字符串
np.datetime_as_string(dt64)

#> '2018-02-04'
```

np.is_busday函数判断日期是否为工作日，工作日默认为周一至周五.

```
print('Date: ', dt64)
print("Is it a business day?: ", np.is_busday(dt64))

#> Date:  2018-02-04
#> Is it a business day?:  False    # False表示不是
```

手动设置工作日的时间，如设置工作日为周六周日，其他时间为休息日.

```
date64 = np.datetime64('2019-04-14')
# 设置周六周日为工作日
np.is_busday(date64,weekmask='Sat Sun')

#> True
```

np.busday_offset查看后几个工作日的日期.

```
# 周四
date64 = np.datetime64('2019-04-11')
# 查看两个工作日后的日期
t = np.busday_offset(date64, 2)
t

#> numpy.datetime64('2019-04-15')
```

若当前工作日为非工作日，则默认是报错的.

```
# 周六
date64 = np.datetime64('2019-04-13')
# 查看两个工作日后的日期
t = np.busday_offset(date64, 2)
t

#> ValueError: Non-business day date in busday_offset  # 非工作日不能作为busday_offset的参数
```

可以增加参数forward或backward来报错，forward的含义是若当前日期非工作日，那么往前寻找最接近当前日期的工作日，backward的含义则是往后寻找最接近当前日期的工作日.

```
#当前时间为周六(2019-04-13)，往前最接近当前时间的工作日是2019-04-15，两个工作日偏移后的日期是2019-
print("Add 2 business days, rolling forward to nearest biz day: ", np.busday_offset(date64, 2, r
#当前时间为周六(2019-04-13)，往后最接近当前时间的工作日是2019-04-12，两个工作日偏移后的日期是2019-
print("Add 2 business days, rolling backward to nearest biz day: ", np.busday_offset(date64, 2,

#> Add 2 business days, rolling forward to nearest biz day:  2019-04-17
#> Add 2 business days, rolling backward to nearest biz day:  2019-04-16
```

6.1 如何创建日期序列

np.arange可简单创建日期序列

```
# 建立日期序列
dates = np.arange(np.datetime64('2018-02-01'), np.datetime64('2018-02-10'))
print(dates)

# 检查是否为工作日
np.is_busday(dates)

#> ['2018-02-01' '2018-02-02' '2018-02-03' '2018-02-04' '2018-02-05'
#>  '2018-02-06' '2018-02-07' '2018-02-08' '2018-02-09']

array([ True,  True, False, False,  True,  True,  True,  True,  True], dtype=bool)
```

6.2 如何把numpy.data64对象转化为datatime.datatime对象

```
# np.datetime64类型转化为datetime.datetime类型
import datetime
dt = dt64.tolist()
dt

#> datetime.date(2018, 2, 4)
```

获取datatime对象的年月日非常简便

```
print('Year: ', dt.year)
print('Day of month: ', dt.day)
print('Month of year: ', dt.month)
print('Day of Week: ', dt.weekday())  # 周五
```

```
#> Year:  2018
#> Day of month:  4
#> Month of year:  2
#> Day of Week:  6
```

7. 高阶numpy函数介绍

7.1 标量函数的向量化

标量函数只能处理标量，不能处理数组

```
# 定义标量函数
def foo(x):
    if x % 2 == 1:
        return x**2
    else:
        return x/2

# On a scalar
print('x = 10 returns ', foo(10))
print('x = 11 returns ', foo(11))

#> x = 10 returns  5.0
#> x = 11 returns  121

# 函数不能处理数组
# print('x = [10, 11, 12] returns ', foo([10, 11, 12]))  # 错误
```

np.vectorize使标量函数也能处理数组，可选参数otypes为输出的类型.

```
# 函数向量化，向量化的输出类型是float
foo_v = np.vectorize(foo, otypes=[float])

print('x = [10, 11, 12] returns ', foo_v([10, 11, 12]))
print('x = [[10, 11, 12], [1, 2, 3]] returns ', foo_v([[10, 11, 12], [1, 2, 3]]))

#> x = [10, 11, 12] returns  [  5.  121.    6.]
#> x = [[10, 11, 12], [1, 2, 3]] returns  [[  5.  121.    6.]
#>  [  1.    1.    9.]]
```

7.2 apply_along_axis函数

首先定义一个二维数组

```
# 定义一个4x10的随机二维数组
np.random.seed(100)
arr_x = np.random.randint(1,10,size=[4,10])
arr_x

#> array([[9, 9, 4, 8, 8, 1, 5, 3, 6, 3],
#>        [3, 3, 2, 1, 9, 5, 1, 7, 3, 5],
```

```
#>        [2, 6, 4, 5, 5, 4, 8, 2, 2, 8],
#>        [8, 1, 3, 4, 3, 6, 9, 2, 1, 8]])
```

如果我们要找数组每行或每列的最大值，numpy的一个最大特点是基于向量化操作的，因此我们使用np.apply_along_axis函数找每行或每列的最大值.

```
# 基于列操作，找每列的最大值
print('max of per column: ', np.apply_along_axis(np.max, 0, arr=arr_x))

# 基于行操作，找每行的最大值
print('max of per row: ', np.apply_along_axis(np.max, 1, arr=arr_x))

#> max of per column:  [9 9 4 8 9 6 9 7 6 8]
#> max of per row:  [9 9 8 9]
```

7.3 searchsorted函数

np.searchsorted函数返回某一变量在有序数组的位置，在该位置插入变量后数组仍然是有序的.

```
# 生成有序数组
x = np.arange(10)
print('Where should 5 be inserted?: ', np.searchsorted(x, 5))
# 若遇到相同大小的数值，输入变量放在右边位置
print('Where should 5 be inserted (right)?: ', np.searchsorted(x, 5, side='right'))

#> Where should 5 be inserted?:  5
#> Where should 5 be inserted (right)?:  6
```

7.4 如何增加数组维度

在不增加任何额外数据的前提下，np.newaxis函数可以增加数组的维数，newaxis在的位置就是要增加的维度.

```
# 定义一维数组
x = np.arange(5)
print('Original array: ', x)
# 数组维度为2
print('ndims of x:', x.ndim)

# 列维度增加
x_col = x[:, np.newaxis]
print(x_col)
# 数组维度为2
print('ndims of x_col: ', x_col.ndim)
print('x_col shape: ', x_col.shape)

# 行维度增加
x_row = x[np.newaxis, :]
print(x_row)
print('x_row shape: ', x_row.shape)
# 数组维度为2
print('ndims of x_row: ', x_row.ndim)

#> Original array:  [0 1 2 3 4]
```

```
#> ndims of x: 1
#> [[0]
    [1]
    [2]
    [3]
    [4]]
#> ndims of x_col:  2
#> x_col shape:  (5, 1)
#> [[0 1 2 3 4]]
#> x_row shape:  (1, 5)
#> ndims of x_row:  2
```

7.5 Digitize函数

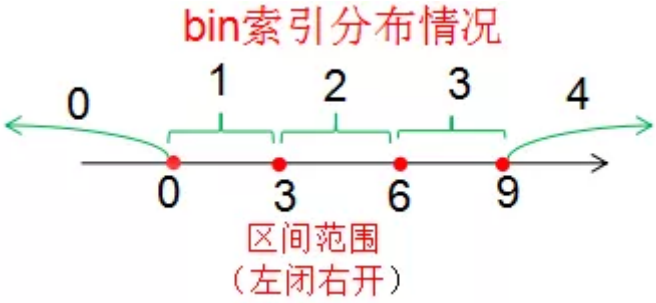
np.digitize函数返回数组每个元素属于bin的索引位置.

```
# 构建数组和bin
x = np.arange(10)
bins = np.array([0, 3, 6, 9])

# 返回bin索引位置
np.digitize(x, bins)

#> array([1, 1, 1, 2, 2, 2, 3, 3, 3, 4])
```

可视化分析digitize算法原理：



如上图，根据变量落在bin的区间范围，返回对应的索引.

7.7 Clip函数

np.clip函数将数字限制在给定截止范围内，所有小于范围下限的数被当成下限值，大于范围上限的数被当成上限值.

```
# 限制x的所有元素位于3和8之间
np.clip(x, 3, 8)

#> array([3, 3, 3, 3, 4, 5, 6, 7, 8, 8])
```

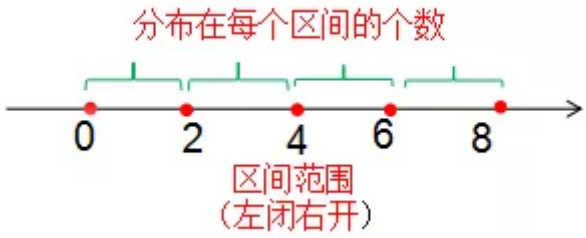
7.8 Histogram函数和Bincount函数

np.bincount函数统计最小值到最大值的个数，最小值为0.

```
# Bincount例子
x = np.array([1,1,2,3,2,4,4,5,6,6,6])
np.bincount(x) # 0出现0次， 1出现2次， 2出现2次， 3出现1次，4出现2次，5出现1次，6出现3次
```

```
#> array([0, 2, 3, 0, 2, 1, 3], dtype=int64)
```

np.histogram函数统计数据落入bins的区间，不考虑bins两侧的区间（如下图所示）.




```
x = np.array([1,1,2,3,2,4,4,5,6,6,6])
counts, bins = np.histogram(x, [0, 2, 4, 6, 8])
print('Counts: ', counts)
print('Bins: ', bins)

#> Counts:  [2 3 3 3]
#> Bins:  [0 2 4 6 8]
```

推荐阅读

清晰易懂的Numpy入门教程

算法目录汇总