清晰易懂的Numpy入门教程

原创 石头 机器学习算法那些事 2019-08-22

翻译 | 石头

来源 | Machine Learning Plus

Numpy是python语言中最基础和最强大的科学计算和数据处理的工具包，如数据分析工具pandas也是基于numpy构建的，机器学习包scikit-learn也大量使用了numpy方法。本文介绍了Numpy的n维数组在数据处理和分析的所有核心应用。

目录

1. 如何构建numpy数组

构建numpy数组的方法很多，比较常用的方法是用np.array函数对列表进行转化。

```
# 通过列表创建一维数组
import numpy as np
list1 = [0,1,2,3,4]
arr1d = np.array(list1)

#打印数组和类型
print(type(arr1d))
arr1d
```

```
<type 'numpy.ndarray'>
[0 1 2 3 4]
```

数组和列表最关键的区别是：数组是基于向量化操作的，列表不是，我们在实际项目中处理的数据一般是矩阵结构，对该数据以行向量或列向量的形式进行计算，向量计算是基于数组实现的，因此数组比列表的应用更广。

函数可以应用到数组的每一项，列表不行。

比如，不可以对列表的每一项数据都加2，这是错误的。

```
list1 + 2  # 错误
```

可以对数组的某一项数据都加2

```
# Add 2 to each element of arr1d
arr1d + 2

#> array([2, 3, 4, 5, 6])
```

另一个区别是已经定义的numpy数组不可以增加数组大小，只能通过定义另一个数组来实现，但是列表可以增加大小。

然而，numpy有更多的优势，让我们一起来发现。

numpy可以通过列表中的列表来构建二维数组。

```
# Create a 2d array from a list of lists
list2 = [[0,1,2], [3,4,5], [6,7,8]]
arr2d = np.array(list2)
arr2d

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

你也可以通过dtype参数指定数组的类型，一些最常用的numpy类型是：'float'，'int'，'bool'，'str'和'object'。

```
# Create a float 2d array
arr2d_f = np.array(list2, dtype='float')
arr2d_f
```

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

输出结果的小数点表示float类型，你也可以通过 astype方法转换成不同的类型。

```
# 转换成‘int’类型
arr2d_f.astype('int')

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

```
# 先转换‘int’类型，再转换‘str’类型
arr2d_f.astype('int').astype('str')

#> array([['0', '1', '2'],
#>        ['3', '4', '5'],
#>        ['6', '7', '8']],
#>       dtype='U21')
```

另一个区别是数组要求所有项是同一个类型，list没有这个限制。如果你想要一个数组包含不同类型，设置‘dtype’为'object'。

```
# 构建布尔类型数组
arr2d_b = np.array([1, 0, 10], dtype='bool')
arr2d_b

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

```
# 构建包含数值和字符串的数组
arr1d_obj = np.array([1, 'a'], dtype='object')
arr1d_obj

#> array([1, 'a'], dtype=object)
```

最终使用 tolist()函数使数组转化为列表。

```
# Convert an array back to a list
arr1d_obj.tolist()

#> [1, 'a']
```

总结数组和列表主要的区别：

1. 数组支持向量化操作，列表不支持；
2. 数组不能改变长度，列表可以；
3. 数组的每一项都是同一类型，list可以有多种类型；
4. 同样长度的数组所占的空间小于列表；

2. 如何观察数组属性的大小和形状（shape）

一维数组由列表构建，二维数组arr2d由列表的列表构建，二维数组有行和列，比如矩阵，三维数组由嵌入了两个列表的列表构建。

假设给定一个数组，我们怎么去了解该数组的属性。

数组的属性包括：

数组的维度（ndim）

数组的形状（shape）

数组的类型（dtype）

数组的大小（size）

数组元素的表示（通过索引）

```
# 定义3行4列的二维数组
list2 = [[1, 2, 3, 4],[3, 4, 5, 6], [5, 6, 7, 8]]
arr2 = np.array(list2, dtype='float')
arr2

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

```
# 形状（shape）
print('Shape: ', arr2.shape)

# 数组类型（dtype）
print('Datatype: ', arr2.dtype)

# 数组大小（size）
print('Size: ', arr2.size)

# 数组维度（ndim）
print('Num Dimensions: ', arr2.ndim)

# 取数组第3行3列元素
print('items of 3 line 3 column: ', c[2,2])

#> Shape:  (3, 4)
#> Datatype:  float64
#> Size:  12
#> Num Dimensions:  2
#> items of 3 line 3 column:  7
```

3. 如何从数组提取特定的项

数组的索引是从0开始计数的，与list类似。numpy数组通过方括号的参数以选择特定的元素。

```
# 选择矩阵的前两行两列
arr2[:2, :2]
list2[:2, :2]  # 错误
```

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

numpy数组支持布尔类型的索引，布尔型索引数组与过滤前（array-to-be-filtered）的数组大小相等，布尔型数组只包含Ture和False变量，Ture变量对应的数组索引位置保留了过滤前的值 。

```
arr2

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

```
# 对数组每一个元素是否满足某一条件，然后获得布尔类型的输出
b = arr2 > 4
b

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

```
# 取布尔型数组保留的原始数组的值
arr2[b]

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

3.1 如何反转数组

```
# 反转数组的行
arr2[::-1, ]

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

```
# Reverse the row and column positions
# 反转数组的行和列
arr2[::-1, ::-1]

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

3.2 如何处理数组的缺失值（missing）和无穷大（infinite）值

缺失值可以用np.nan对象表示，np.inf表示无穷大值，下面用二维数组举例：

```
# 插入nan变量和inf变量
arr2[1,1] = np.nan  # not a number
```

```
arr2[1,2] = np.inf  # infinite
arr2

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

```
# 用-1代替nan值和inf值
missing_bool = np.isnan(arr2) | np.isinf(arr2)
arr2[missing_bool] = -1
arr2

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

3.3 如何计算n维数组的平均值，最小值和最大值

```
# 平均值，最大值，最小值
print("Mean value is: ", arr2.mean())
print("Max value is: ", arr2.max())
print("Min value is: ", arr2.min())

#> Mean value is:  3.58333333333
#> Max value is:  8.0
#> Min value is:  -1.0
```

如果要求数组的行或列的最小值，使用np.amin函数

```
# Row wise and column wise min
# 求数组行和列的最小值
# axis=0表示列，1表示行
print("Column wise minimum: ", np.amin(arr2, axis=0))
print("Row wise minimum: ", np.amin(arr2, axis=1))

#> Column wise minimum:  [ 1. -1. -1.  4.]
#> Row wise minimum:  [ 1. -1.  5.]
```

对数组的每个元素进行累加，得到一维数组，一维数组的大小与二维数组相同。

```
# 累加
np.cumsum(arr2)

#> array([  1.,   3.,   6.,  10.,  13.,  12.,  11.,  17.,  22.,  28.,  35.,  43.])
```

4. 如何从现有的数组定义新数组

如果使用赋值运算符从父数组定义新数组，新数组与父数组共占同一个内存空间，如果改变新数组的值，那么父数组也相应的改变。

为了让新数组与父数组相互独立，你需要使用copy（）函数。所有父数组都使用copy（）方法构建新数组。

```
# Assign portion of arr2 to arr2a. Doesn't really create a new array.
# 分配arr2数组给新数组arr2a，下面方法并没有定新数组
arr2a = arr2[:2,:2]
arr2a[:1, :1] = 100  # arr2相应位置也改变了
arr2

#> array([[ 100.,    2.,    3.,    4.],
#>        [   3.,   -1.,   -1.,    6.],
#>        [   5.,    6.,    7.,    8.]])
```

```
# 赋值arr2数组的一部分给新数组arr2b
arr2b = arr2[:2, :2].copy()
arr2b[:1, :1] = 101  # arr2没有改变
arr2

#> array([[ 100.,    2.,    3.,    4.],
#>        [   3.,   -1.,   -1.,    6.],
#>        [   5.,    6.,    7.,    8.]])
```

5. 多维数组的重构（reshaping）和扁平（flattening）

重构（reshaping）是改变了数组项的排列，即改变了数组的形状，未改变数组的维数。

扁平（flattening）是对多维数组转化为一维数组。

```
# 3x4数组重构为4x3数组
arr2.reshape(4, 3)

#> array([[ 100.,    2.,    3.],
#>        [   4.,    3.,   -1.],
#>        [  -1.,    6.,    5.],
#>        [   6.,    7.,    8.]])
```

5.1 flatten()和ravel()的区别

数组的扁平化有两种常用的方法，flatten()和ravel() 。flatten处理后的数组是父数组的引用，因此新数组的任何变化也会改变父数组，因其未用复制的方式构建数组，内存使用效率高，ravel通过复制的方式构建新数组。

```
# flatten方法
arr2.flatten()

#> array([ 100.,    2.,    3.,    4.,    3.,   -1.,   -1.,    6.,    5.,  6.,    7.,    8.])
```

```
# flatten方法
b1 = arr2.flatten()
b1[0] = 100  # 改变b1的值并未影响arr2
arr2
```

```
#> array([[ 100.,    2.,    3.,    4.],
#>        [   3.,   -1.,   -1.,    6.],
#>        [   5.,    6.,    7.,    8.]])
```

```
# ravel方法
b2 = arr2.ravel()
b2[0] = 101  # 改变b2值，相应的改变了arr2值
arr2

#> array([[ 101.,    2.,    3.,    4.],
#>        [   3.,   -1.,   -1.,    6.],
#>        [   5.,    6.,    7.,    8.]])
```

6. 如何通过numpy生成序列数（sequences），重复数（repetitions）和随机数（random）

np.arrange函数手动生成指定数目的序列数，与ndarray作用一样。

```
# 默认下限为0
print(np.arange(5))

# 0 to 9，默认步数为1
print(np.arange(0, 10))

# 递增步数2
print(np.arange(0, 10, 2))

# 降序
print(np.arange(10, 0, -1))

#> [0 1 2 3 4]
#> [0 1 2 3 4 5 6 7 8 9]
#> [0 2 4 6 8]
#> [10  9  8  7  6  5  4  3  2  1]
```

上例是通过np.arrange设置初始位置和结束位置来生成序列数，如果我们设置数组的元素个数，那么可以自动计算数组的递增值。

如构建1到50的数组，数组有10个元素，使用np.linspace总动计算数组的递增值。

```
# 起始位置和结束位置分别为1和50
np.linspace(start=1, stop=50, num=10, dtype=int)

#> array([ 1,  6, 11, 17, 22, 28, 33, 39, 44, 50])
```

我们注意到上面例子的递增值并不相等，有5和6两个值，原因是计算递增值采用了四舍五入的算法（rounding）。与np.linspace类似，np.logspace以对数尺度的方式增长。

```
# 设置数组的精度为小数点后两位
np.set_printoptions(precision=2)

# 起点为 10^1 and 终点为 10^50，数组元素个数10，以10为底数
np.logspace(start=1, stop=50, num=10, base=10)
```

```
#> array([  1.00e+01,   2.78e+06,   7.74e+11,   2.15e+17,   5.99e+22,
#>          1.67e+28,   4.64e+33,   1.29e+39,   3.59e+44,   1.00e+50])
```

初始化数组的元素全为1或全为0。

```
np.zeros([2,2])
#> array([[ 0.,  0.],
#>        [ 0.,  0.]])
```

```
np.ones([2,2])
#> array([[ 1.,  1.],
#>        [ 1.,  1.]])
```

7.1 如何构建重复的序列数

np.tile重复整个的数组或列表n次，np.repeat重复数组每一项n次。

```
a = [1,2,3]

# 重复数组a两次
print('Tile:   ', np.tile(a, 2))

# 重复数组a每项两次
print('Repeat: ', np.repeat(a, 2))

#> Tile:    [1 2 3 1 2 3]
#> Repeat:  [1 1 2 2 3 3]
```

7.2 如何生存随机数

random模块包含的函数可以生成任一数组形状的随机数和统计分布。

```
# 生成2行2列的[0,1)的随机数
print(np.random.rand(2,2))

# 生成均值为0方差为1的2行2列的正态分布值
print(np.random.randn(2,2))

# 生成[0,10)的2行2列的随机整数
print(np.random.randint(0, 10, size=[2,2]))

# 生成一个[0,1)的随机数
print(np.random.random())

# 生成[0,1)的2行2列的随机数
print(np.random.random(size=[2,2]))

# 从给定的列表等概率抽样10次
print(np.random.choice(['a', 'e', 'i', 'o', 'u'], size=10))

# 从给定的列表和对应的概率分布抽样10次
print(np.random.choice(['a', 'e', 'i', 'o', 'u'], size=10, p=[0.3, .1, 0.1, 0.4, 0.1]))  # pick

#> [[ 0.84   0.7 ]
```

```
#>  [ 0.52  0.8 ]]

#> [[-0.06 -1.55]
#>  [ 0.47 -0.04]]

#> [[4 0]
#>  [8 7]]

#> 0.08737272424956832

#> [[ 0.45  0.78]
#>  [ 0.03  0.74]]

#> ['i' 'a' 'e' 'e' 'a' 'u' 'o' 'e' 'i' 'u']
#> ['o' 'a' 'e' 'a' 'a' 'o' 'o' 'o' 'a' 'o']
```

7.3 如何得到数组独特（unique）的项和个数（counts）

np.unique函数去除数组中重复的元素，设置return_counts参数为True，得到数组每一项的个数。

```
# 定义范围为[0,10)，个数为10的随机整数数组
np.random.seed(100)
arr_rand = np.random.randint(0, 10, size=10)
print(arr_rand)

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

```
# 得到数组独特的项和相应的个数
uniqs, counts = np.unique(arr_rand, return_counts=True)
print("Unique items : ", uniqs)
print("Counts       : ", counts)

#> Unique items :  [0 2 3 4 5 7 8]
#> Counts       :  [1 2 1 1 1 2 2]
```

8 小结

本文比较全面的介绍了numpy的基本用法，希望对numpy还不熟悉的同学有所帮助。

推荐阅读

Python代码写得丑怎么办？推荐几个神器拯救你

110道python面试