Impoty numpy as np

[ndarray object class structure]

class ndarray:

* attribute:

\_\_internal\_\_ {dict} (the variable save array element)

min {<one of np.dtype>} (the min value)

max {<one of np.dtype>} (the max value)

dtype {dtype} (the data dype of the element in this array)

shape {tuple}

size {int}

* method:

astype(dtype= <newType>) (copy the array and return its copy with <newType>)

[Create a 1-d array]

>> np.array(<1-d list>)

ex: np.array([1,2,3])

[Create a 2D array]

>> np.array([[1,2,3],[4,5,6],[7,8,9]])

[Create a sequence of consecutive number]

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

>> np.arange(a,b,c) # sequence: [a, a+c, a+2c, .... “the last number < b”]

Ex: np.arange(1,10,3) >>> [1,3,5,7,9]

[Index array]

B = A[<row>][<col>] or A[<row>,<col>]

* negative index

Each numpy entry has two index: 1. normal index, 2. negative index

ex: assume there are a 1-d ndarray with size 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| -5 | -4 | -3 | -2 | -1 |

[data slicing]

* want to slice data from <a> to <b> (not including <b>)

>> array[<a>:<b>]

* want to slice data from <a> to <b> with step <c>(not including <b>)

>> array[<a>:<b>:<c>]

* want to slice data including end/start

>> array[<a>:] # from <a> to end

>> array[: <a>] # from start to <a>

>> array[<a>::-1] # from start to <a> reversely

>> array[:] # slice all

* slicing data using array
* The size of index array is the same as the array, and element of index array is true or false

>> array[<index\_array>]

# return the array element of the index of which <index\_array>[index] is True (return the array[index], where the index makes <index\_array>[index] == True)

ex:

a = array([[1,2,3],[4,5,6],[7,8,9]])

b = array([[True,False,False], [False,True,False],[False, False, False]])

a[b] >> [1,5]

* index array structure is tuple of list ([…],[…],[…],..)

>> array[<index\_array>]

# return the selected array with indices where first tuple is axis 0 and second tuple is axis 1 and so on.

ex:

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

b = ([0,0,1],[0,1,2])

a[b] >> [1,2,6] # 1 is a[0,0], 2 is a[0,1], 6 is a[1,2]

[Data changing]

A[0:2] = 2 # the A[0:2] elements will all be 2

(the left size can be non-equal to right size)

A[0:2] = np.array([1,2]) # the A[0:2] will be vector [1,2]

(the left size is euqal to right size)

[Output the shape]

B = A.shape

B is the tuple contain size of each dimension

Ex: A = np.array([[1,2,3],[4,5,6]]) B = (2,3)

[Output the shape or size] (row# \* column#)

A = np.zeros((3,2))

Print(A.size) >> 6

Print(np.size(A)) >> 6

Print(np.size(A,0)) >> 3

Print(np.size(A,1)) >> 2

Print(np.shape) >> (3,2)

[Reshape the matrix]

B = A.reshape(<shape>) or B = np.reshape(A,<shape>)

<shape> can be a list or a np.array

[Create a zero/ones matrix]

Np.zeros(size)

Np.ones(size)

Size is tuple. Ex: (5,5,3,6)

[Copy an array]

b = np.copy(a)

[Construct an array by repeating A the number of times given by reps]

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

C = np.tiles(B,(1,2))

>> C = [1,2,3,1,2,3]

C = np.tiles(B,(2,2))

>> C = [[1,2,3,1,2,3],  
 [1,2,3,1,2,3]]

[Stack the array: concatenate]

>> np.concatenate((<a>, <b>), <axis>)

this function will concatenate <a>, <b> ndarray along <axis>and return. (the <a>, <b> array should have same dimension number and axis lengths except <axis>)

[Data selecting]

A = [[1,2,3],[4,5,6],[7,8,9]]

A == 7 >> False

A = np.array([[1,2,3],[4,5,6],[7,8,9]])

A == 7 >> [[False, False, False],[False,False,False] [True, False, False]]

[Count the true number]

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

B = np.array([1,2,0])

C = np.count\_nonzero(A==B)

C >> 2

[return the nonzero indice]

* np.nonzero()

>> np.nonezero(<array>)

# this will return a tuple of list, where first list store value of axis 0 and second list store value of axis 1 and so on

Ex:

a = np.array([[1,2,0],[0,0,1],[0,0,0]]) # only element of indice (0,0), (0,1), (1,2) are nonzero

np.nonzero(a) >> ([0,0,1],[0,1,2])

* <array>.nonzero()

>> <array>.nonzero() # the same as np.nonzero(<array>)

[Vectorize a function]

When a function can only deal with input value rather than input vector, you can vectorize it as function that can take vector as input

1. The input array can either be list, tuple or numpy array
2. The input array can be multidimensional
3. The output array is numpy array

Ex:

myfunction = lambda x: x\*2

vectorized = np.vectorize(myfunction)

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

print(vectorized(a))

>> [4,5,6]