
NumPy

Numpy

→ Numpy arrays are of fixed size

Changing the size of ndarray will create a new array and delete the original array.

→ Numpy array elements should be of same datatype.

Why is Numpy Fast?

→ Vectorized code is more concise and easier to read.

→ Vectorization results in more "Pythonic" code. Without vectorization, our code would be littered with inefficient and difficult to read for loops.

Basics

→ Numpy object is homogenous multidimensional array

Ex:- [[1., 0., 0.], [0., 1., 2.]] ndarray

ndarray attributes:

ndarray.ndim
→ No. of dimensions in array

ndarray.shape
→ Dimensions of array

ndarray.size
→ Total no. of elements in array.

ndarray.dtype

→ Datatype of elements

ndarray.itemsize

→ Size in bytes of each element

ndarray.data

→ The buffer containing the actual elements of the array.

Creating a Numpy array

```
np.array([2,3,4], dtype=int16)

np.zeros((3,4))   # Array full of 0
np.ones((2,3,4))  # Array full of 1
np.empty((2,3), dtype=int16)  # Random
```

(2,3) → dimension

→ Default dtype : float64

```
np.arrange(10, 35, 5)   # Sequence
```

start = 10, end = 35, increment = 5

```
np.arrange(5, 6, 0.1)

np.linspace(0, 1, 5)    #linear spacing
#  (0., 0.25, 0.5, 0.75, 1.)
```

Printing Numpy array

```
print(arr)
```

Basic Operators

a , b ⇒ numpy array

```
c = a+b
c = a-b
c = a*b
c = a/b
```

Matrix product

```
c = a @ b
c = a.dot(b)
```

} Matxic Cross product

Can type cast from

int32 → float64 → complex

but can't type cast from

complex ↛ float64 ↛ int32

```
np.sum(axis=0) #Sum of each column
np.sum(axis=1) #Sum of each row
```

axis = 0

axis = 1

Indexing, Slicing and Iterating

```
a[2]
a[2:5]        start : end
a[2:5:2]      start : end : increment
a[::-1]       Reversed array
```

```
np.fromfunction(fun, (5, 4), dtype=int)

a[2,3]      # a[2][3]
a[0:5, 1]   # a[i][1]
a[1, 0:5]   # a[1][i]

a[1:3, 0:5]
```

b[i, ...]

→ ... are used to fill remaining ":"

b[1, ...] == b[1, :, :] == b[1]

b.flat

→ Iterator over all the elements of the array.

Change the shape of Numpy

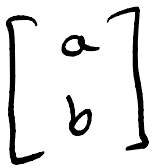
```
a.ravel()  # flattens array
a.reshape(6,2)  # changes shape
a.T  # Returns transpose of a
a.resize(2,6)
a.resize(3, -1)
```

→ -1 automatically calculates other dimension

Stacking different arrays

np.vstack(a,b)

np.concatenate((a,b),axis=0)

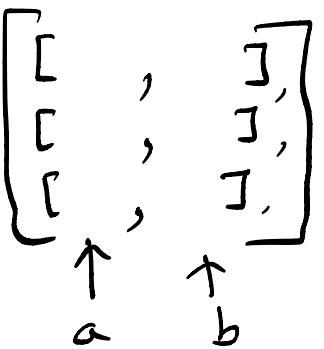


np.hstack (a,b)

[a b]

np.column_stack (a,b)

np.concatenate((a,b))



Splitting Array

np.hsplit (a, 3)

3 equal parts

np.hsplit (a, (3, 4))

Split at column 3 and column 4

View or Shallow Copy

→ Creates a new object that references the main object.

a.view()

Deep Copy

→ Creates a new object with new data

a.copy()

Indexing with array of indices

```
>>> a = np.arange(12)**2  # the first 12 square numbers
>>> i = np.array([1, 1, 3, 8, 5])  # an array of indices
>>> a[i]  # the elements of `a` at the positions `i`
array([ 1,  1,  9, 64, 25])
>>>
>>> j = np.array([[3, 4], [9, 7]])  # a bidimensional array of indices
>>> a[j]  # the same shape as `j`
array([[ 9, 16],
       [81, 49]])
```

→ Multi dimensional indexing

```
>>> palette = np.array([[0, 0, 0],         # black
...                     [255, 0, 0],       # red
...                     [0, 255, 0],       # green
...                     [0, 0, 255],       # blue
...                     [255, 255, 255]])  # white
>>> image = np.array([[0, 1, 2, 0],  # each value corresponds to a color in the pa
...                   [0, 3, 4, 0]])
>>> palette[image]  # the (2, 4, 3) color image
array([[[  0,   0,   0],
        [255,   0,   0],
        [  0, 255,   0],
        [  0,   0,   0]],

       [[  0,   0,   0],
        [  0,   0, 255],
        [255, 255, 255],
        [  0,   0,   0]]])
```

Ex:-

```
>>> a = np.arange(12).reshape(3, 4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> i = np.array([[0, 1],  # indices for the first dim of `a`
...               [1, 2]])
>>> j = np.array([[2, 1],  # indices for the second dim
...               [3, 3]])
>>>
>>> a[i, j]  # i and j must have equal shape
array([[ 2,  5],
       [ 7, 11]])
>>>
>>> a[i, 2]
array([[ 2,  6],
       [ 6, 10]])
>>>
>>> a[:, j]
array([[[ 2,  1],
        [ 3,  3]],

       [[ 6,  5],
        [ 7,  7]],

       [[10,  9],
        [11, 11]]])
```

np.argmax(axis=0)

→ Returns the index of maximum number in axis 0 (↓)

Indexing with boolean Array

```
>>> a = np.arange(12).reshape(3, 4)
>>> b = a > 4
>>> b  # `b` is a boolean with `a`'s shape
array([[False, False, False, False],
       [False,  True,  True,  True],
       [ True,  True,  True,  True]])
>>> a[b]  # 1d array with the selected elements
array([ 5,  6,  7,  8,  9, 10, 11])
```

```
>>> a[b] = 0  # All elements of `a` higher than 4 become 0
>>> a
array([[0, 1, 2, 3],
       [4, 0, 0, 0],
       [0, 0, 0, 0]])
```

```
data.min(axis=0) #Min of columns
data.max() #Max value in matrix
data.sum() #Sum of all values

np.unique(data, #Unique values
          return_index=True, #Returns index
          return_counts=True, #Occurrences
          axis=0) #Optional

np.flip(data, #Reverse an array
        axis=0) #Optional
```

Ex: $MSE = \frac{1}{n} \sum_{i=1}^{n} (Y\_pred_i - Y_i)^2$

```
error = (1/n) * np.sum(np.square(
                pred - labels))
```

Save & Load Numpy File

```
np.save ('filename', data)

data = np.load ('filename.npy')

np.savetxt ('newfile.csv', data)

data = np.loadtxt('newfile.csv')
```

np.nan ⇒ Null value in numpy