4.2 - NumPy Basics: Part 1

Ha Khanh Nguyen (hknguyen)

---

1. What is NumPy?

- NumPy is short for Numerical Python.
- Some of the main features of NumPy that we will look at in our course:
  - `ndarray` : a multi-dimensional array providing fast array-oriented arithmetic access and operations.
  - Mathematical functions for fast operation on entire arrays of data without having to write loops! (just like R)
  - Tools for reading/writing array data to disk and working with memory-mapped files.
  - Linear algebra, random number generation, etc.
- Let's compare the performance of NumPy array and the equivalent Python list!

```
import numpy as np

my_arr = np.arange(1000000)
my_list = list(range(1000000))
```

```
%time for _ in range(10): my_arr2 = my_arr*2
%time for _ in range(10): my_list2 = [x * 2 for x in my_list]
```

- NumPy-based algorithms are generally 10-100 times faster than their pure Python counterparts and use significantly less memory! (again, sound just like R!)

2. NumPy ndarray

- NumPy arrays enable you to perform mathematical operations on whole blocks of data using similar syntax to the equivalent operations between scalar elements.

```
# set the seed for the random generator
np.random.seed(430)

# generate some random data
data = np.random.randn(2, 3)
data
```

```
## array([[ 0.4742312 , -0.85940424, -0.75509958],
##        [-0.54634703,  1.01367802, -1.48055291]])
```

- An `ndarray` is a generic multidimensional container for **homogeneous** data: all of the elements must be the same type.
  - Again, like vector and array in R!

- Every array has a `ndim` and a `shape` attributes which describe the dimension of the array.

```
data.ndim
```

```
## 2
```

```
data.shape
```

```
## (2, 3)
```

- An object describing the data type of the array:

```
data.dtype
```

```
## dtype('float64')
```

2.1 Creating `ndarray`

- You can use the `array()` function to creata a NumPy `ndarray`.
- This function accepts any sequence-like object like list, tuple, etc. (including other arrays) and produces a new NumPy array.

```
data_list = [5, 5.2, 9.1, 3, 6.4]
data_array = np.array(data_list)
data_array
```

```
## array([5. , 5.2, 9.1, 3. , 6.4])
```

- Nested sequence (like a list of equal-length lists) will be converted into a multi-dimensional array:

```
data_nested_list = [[4, 2, 1], [3, 0, 7]]
data_2darray = np.array(data_nested_list)
data_2darray
```

```
## array([[4, 2, 1],
##        [3, 0, 7]])
```

- NumPy also provides functions to create special arrays such as arrays of all 0s or all 1s.

```
# array with 0s
np.zeros(10)
```

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

```
# array with 1s
np.ones((3, 6))
```

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

```
# array without initializing its value to any particular value
np.empty((2, 3, 2))
```

```
## array([[[2.68156159e+154, 1.49457826e-154],
##         [2.96439388e-323, 0.00000000e+000],
##         [0.00000000e+000, 0.00000000e+000]],
##
##        [[0.00000000e+000, 0.00000000e+000],
##         [0.00000000e+000, 0.00000000e+000],
##         [0.00000000e+000, 8.34402697e-309]]])
```

- `np.arange()` is an array-valued version of the built-in Python `range()` function:

```
np.arange(15)
```

```
## array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

- Here is the full list of functions for creating array in NumPy:

Function	Description
`array`	Convert input data (list, tuple, array, or other sequence type) to an `ndarra` y either by inferring a dtype or explicitly specifying a dtype; copies the input data by default
`asarray`	Convert input to `ndarray` , but do not copy if the input is already an `ndarray`
`arange`	Like the built-in `range` but returns an `ndarray` instead of a list
`ones`	Produce an array of all 1s with the given shape and dtype
`ones_like`	Take another array and produces a 1s array of the same shape and dtype
`zeros`	Like `ones` producing arrays of 0s instead
`zeros_like`	Like `ones_like` producing arrays of 0s instead
`empty`	Create new arrays by allocating new memory, but do not populate with any values like `ones` and `zeros`
`empty_like`	Take another array and produces a new array of the same shape and dtype, but do not populate it with values
`full`	Produce an array of the given shape and dtype with all values set to the indicated “fill value”
`full_like`	Take another array and produces a filled array of the same shape and dtype
`eye` , `identity`	Create a square NxN identify matrix (1s on the diagonal and 0s elsewhere)

2.2 Arithmetic with NumPy Arrays

- Arrays are important because they enable you to express batch operations on data WITHOUT writing any loops.

- NumPy users call this **vectorization** (also known as elementwise-operation).
  - So does R, as some of you have seen this in STAT 385 last semester!
- Any arithmetic operations between equal-size arrays applies the operation element-wise:

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
arr
```

```
## array([[1, 2, 3],
##        [4, 5, 6]])
```

```
arr + arr
```

```
## array([[ 2,  4,  6],
##        [ 8, 10, 12]])
```

```
arr - arr
```

```
## array([[0, 0, 0],
##        [0, 0, 0]])
```

```
arr ** 2
```

```
## array([[ 1,  4,  9],
##        [16, 25, 36]])
```

```
1 / arr
```

```
## array([[1.        , 0.5       , 0.33333333],
##        [0.25      , 0.2       , 0.16666667]])
```

- We can also use boolean operator with arrays!

```
arr2 = np.array([[0, 4, 1], [7, 2, 12]])
arr2
```

```
## array([[ 0,  4,  1],
##        [ 7,  2, 12]])
```

```
arr > arr2
```

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

- What happens if the two arrays have different sizes? In that case, the operation is called broadcasting and it is discussed in detailed in the Appendix A of the textbook.

This lecture note is modified from Chapter 4 of Wes McKinney's Python for Data Analysis 2nd Ed (https://www.oreilly.com/library/view/python-for-data/9781491957653/).