Topic 10

Data Processing with Ndarrays

Murdoch
UNIVERSITY

Last Topic

Murdoch UNIVERSITY

- Class and objects
- Class definition
- Create objects from a class
- Operator methods
- Solving problems with classes and objects

This Topic

Murdoch UNIVERSITY

- Numpy package
- Ndarray
- Create ndarrays
- Numpy functions for creating ndarray: zeros, ones, empty, arange, linspace
- Ndarray type
- Ndarray indexing
- Copy, view and concatenate
- Element-wide operations
- Matrix multiplication
- Universal functions

Numpy– Numerical Python

Murdoch UNIVERSITY

- Numpy is a Python package. It stands for "Numerical Python".
- Numpy allows you to create special multi-dimensional array objects, known as *ndarrays*. An ndarray is an object of `ndarray` class.
- Numpy has a collection of functions and classes for processing these ndarrays.
- Travis Olipant created Numpy by incorporating the features from two earlier modules, *Numeric* and *Numarray*, in 2005.
- Numpy is widely used for numerical processing, including mathematical and logical operations on arrays, Fourier transforms and shape manipulation, linear algebra, and random number generation.

Numpy– Numerical Python

Murdoch UNIVERSITY

- Numpy is often used along with packages like *SciPy* (Scientific Python) and *Matplotlib* (a plotting library) by scientists and engineers.
- This combination is widely used as a replacement for MatLab, a popular platform for technical computing.

Install Numpy Package

Murdoch UNIVERSITY

- A Python package is a set of modules/packages stored under a directory.
- The directory contains the initialisation code in file `__init__.py` which is executed when the package is imported.
- Numpy is not a core package of Python, therefore you need to install it on your computer:

```
pip install numpy
```

Install Numpy Package

Murdoch UNIVERSITY

- PIP stands for **P**ackage **I**nstaller for **P**ython, it is used to perform various package management operations, eg:

`pip install` *`package_name`*	install a package
`pip uninstall` *`package_name`*	uninstall a package
`pip show` *`package_name`*	display information of a package
`pip list`	list installed modules/packages

- `pip` is to Python what `npm` is to Node.js.

Why Ndarrays?

Murdoch UNIVERSITY

- Although Python has a builtin type `list` which can be used for array operations, it is very inefficient when the size of the array is large.
- In numerical processing, most data sets are in the form of an array (eg, an array of 100000 floats). The data sizes are usually very large. It would be very slow to process a large array if the list is used to store the array.
- For an ndarray, the elements in the array are stored in one continuous area of the memory (no matter how many dimensions), which affords efficient processing.
- For a large array, the processing speed can increase by 50+ times if an ndarray is used (compared to list).

Why Ndarrays?

Murdoch UNIVERSITY

- Furthermore, Numpy provides many useful and convenient functions and classes for common array operations, including operations for multi-dimensional arrays.
- Given all these advantages, ndarrays do have one limitation: unlike a list which can be heterogenous (elements in a list may have mixture of types), an ndarray is homogenous – the elements in an ndarray is of the same type!
- This is not a big issue for numerical processing, as in most cases, the data elements in an array is of the same type.

Ndarray: Dimensions and Axes

```
>>> a = np.array(3)
```

Dimensions: 0
Axes: 0
Shape: ()
Size: 1

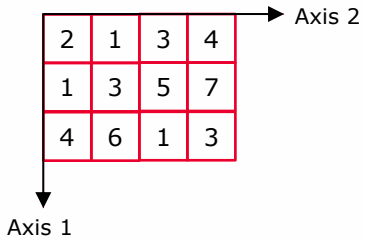
```
>>> print(a)
3
```

```
>>> b = np.array([2,1,3,4])
```

Dimensions: 1
Axes: 1
Shape: (4,)
Size: 4

```
>>> print(b)
[2 1 3 4]
```

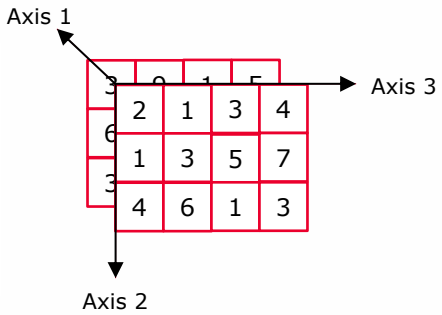
```
>>> c = np.array([[2,1,3,4],
...               [1,3,5,7],
...               [4,6,1,3]])
```



Dimensions: 2
Axes: 1, 2
Shape: (3, 4)
Size: 12

```
>>> print(c)
[[2 1 3 4]
 [1 3 5 7]
 [4 6 1 3]]
```

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



Dimensions: 3
Axes: 1, 2, 3
Shape: (2, 3, 4)
Size: 24

```
>>> print(d)
[[[2 1 3 4]
  [1 3 5 7]
  [4 6 1 3]]

 [[3 9 1 5]
  [6 9 3 9]
  [3 3 5 7]]]
```

Create Ndarrays

Murdoch UNIVERSITY

- To create an ndarray, we need to import Numpy package:

```
>>>
>>> import numpy
>>> a = numpy.array([1,2,3,4])
>>> print(a)
    [1 2 3 4]
>>> print(type(a))
    <class 'numpy.ndarray'>
>>>
```

- The `array` function from `numpy` returns an `ndarray` object. The function takes a sequence (usually either a list or a tuple).
  - If a single value is given, the function returns an ndarray of dimension 0.
- Most Numpy users use the alias `np` for Numpy, as in:

```
import numpy as np
a = np.array([1,2,3])
```

Create Ndarray

Murdoch UNIVERSITY

- The Numpy function `array` allows us to create n-dimensional arrays.
- Example 1: an 2d array - we provide a list of lists to the array function. The last axis is specified in inner lists and the first axis is specified by the outer list:

```
import numpy as np
table = np.array([[1,2,3,4],
                  [5,6,7,8]])
print(table)
```

- Example 2: an 3d array:

```
import numpy as np
c = np.array([[[1,2,3,4],
               [5,6,7,8]],
              [[8,7,6,5],
               [4,3,2,1]]])
print(c)
```

Properties of an Ndarray

Murdoch UNIVERSITY

- The `numpy.ndarray` class consists of many properties, including the following about the array object:
  - `ndarray.ndim`: the number of dimension
  - `ndarray.shape`: the size of each dimension
  - `ndarray.size`: the total number of elements
  - `ndarray.dtype`: the type of the elements
  - `ndarray.itemsize`: the number of bytes taken up by each element
  - `ndarray.data`: the buffer containing the actual elements of the array

Properties of an Ndarray

Murdoch UNIVERSITY

- Example:

```
import numpy as np

a = np.array([[[1,2,3,4],
               [5,6,7,8],
               [9,0,1,2]],
              [[9,8,7,6],
               [5,4,3,2],
               [1,0,9,8]]])

print(a.ndim)             # 3
print(a.shape)            # (2,3,4)
print(a.size)             # 24
print(a.dtype)            # int64
print(a.itemsize)         # 8
```

Create Ndarrays with 0 and 1

- Apart from function `array`, Numpy package has several other functions to create ndarrays filled with 0, 1 or random values

```
import numpy as np

a = np.zeros((2,4))   # a 2x4 array filled with all 0
print(a)

b = np.ones((3,2))    # a 3x2 array filled with all 1
print(b)

c = np.empty((3,5))   # a 3x5 array whose values are the values
print(c)              # in the allocated memory
```

Create Ndrrays using Function `arange`

Murdoch UNIVERSITY

- The Numpy package has function `arange` which creates an ndarray with a sequence of values. The way `arange` generates a sequence is similar to function `range`.

```
import numpy as np

a = np.arange(5)                        # [0,1,2,3,4]
print(a)
b = np.arange(2,8)                      # [2, 3, 4, 5, 6, 7]
print(b)
c = np.arange(1,7,2)                    # [1, 3, 5 ]
print(c)
d = np.arange(12).reshape(3,4)          # [ [0, 1, 2, 3 ],
print(d)                                #   [4, 5, 6, 7 ],
                                        #   [8, 9, 10, 11] ]
```

- In the last example above, we call the `reshape` method (from `ndarray` class) to convert the 1d array to a 3x4 array

Create Ndrrays using Function `linspace`

Murdoch UNIVERSITY

- The `linspace` function from Numpy module is similar to `arrange` function, but it creates an ndarray with a specified number of elements.

```
import numpy as np

a = np.linspace(0,3,9)
print(a)

# output:
# [0. 0.375 0.75 1.125 1.5 1.875 2.25 2.625 3. ]

b = np.linspace(0, np.pi, 10)
print(b)

# output:
# [0.          0.34906585  0.6981317   1.04719755  1.3962634
#  1.74532925  2.0943951   2.44346095  2.7925268   3.14159265]
```

Element Type of Ndarrays

Murdoch UNIVERSITY

- An ndarray is homogenous, ie, all elements in an ndarray have the same type.
- If the elements have both integers and floats, all elements will be converted to floats

```
a = np.array([2, 3.1, 4, 5])
print(a)              # [2.  3.1 4.  5. ]
```

- One can specify the element type when creating the ndarray, with `dtype=`*`type`*, eg.,

```
b = np.array([2, 4, 5, 10], dtype=np.int16)
print(b)              # [ 2  4  5 10]

c = np.arange(1,5, dtype=float)
print(c)              # [ 1. 2. 3. 4. ]
```

The Index of an Ndarray

- Like a list, we can access elements of an ndarray using their indexes, both positive and negative.

```
import numpy as np

a = np.array([1,2,3,4])
print(a[2])   # 3
print(a[-2])  # 3

b = np.array([ [ 1,2,3,4 ],
               [ 5,6,7,8 ] ])
print(b[1,2])  # 7
```

Slice an Ndarray

- Numpy implemented the same index arithmetic as strings and lists, including array slicing.

```
import numpy as np

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

print(a[1:5])  # [ 2 3 4 5 ]

print(a[:5:2])  # [ 1 3 5 ]
```

Copy and View

Murdoch UNIVERSITY

- The `ndarray` class provides `copy` and `view` methods:
  - Use the `copy` method to create a separate copy of the ndarray. Change in the copy would not affect the original array.
  - Use the `view` method to create a "view" (or alias) of the ndarray. The view is similar to an alias. Change in the view would affect the original array.

```
import numpy as np
a = np.array([1,2,3,4,5])

b = a.copy()
b[1] = 10
print(a)                    # [ 1  2  3  4  5 ]
print(b)                    # [ 1 10  3  4  5 ]

c = a.view()
c[1] = 20
print(a)                    # [ 1  20  3  4  5 ]
print(c)                    # [ 1  20  3  4  5 ]
```

Join Two Ndarrays

Murdoch UNIVERSITY

- Use the `concatenate` function of Numpy package to combine two ndarrays and return the new ndarray.

```
import numpy as np

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

c = np.concatenate((a,b))
print(c)    # [ 1 2 3 4 5 6 7 8 ]
```

- For 2d array, concatenation is done along axis 1 (rows)

```
x = np.array([ [  1,  2,  3 ],
               [  4,  5,  6 ] ])
y = np.array([ [ 10, 20, 30 ],
               [ 40, 50, 60 ] ])

print(np.concatenate((x,y)))
```

```
[[ 1  2  3]
 [ 4  5  6]
 [10 20 30]
 [40 50 60]]
```

Element-wise Operations

Murdoch UNIVERSITY

- Numerical operations apply to each element of an ndarray.

```
import numpy as np

a = np.array([1,2,3,4,5])

print(a+2)  # [ 3 4 5 6 7 ]
print(5-a)  # [ 4 3 2 1 0 ]
print(3*a)  # [ 3 6 9 12 15 ]
print(10/a) # [ 10. 5. 3.333333 2.5 2. ]
print(a**2) # [ 1 4 9 16 25 ]
print(a>3)  # [ False False False True True ]
```

- In these operations, the original array is unchanged. The operation results in a new ndarray object.

Element-wise Operations

Murdoch UNIVERSITY

- When both operands are ndarrays of the *same shape*, numerical operations apply to each pair of elements at the same index position.

```
import numpy as np

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

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


print(a+b)          # [[7 7 7]
                    #  [7 7 7]]

print(a*b)          # [[ 6 10 12]
                    #  [12 10  6]]
```

Matrix Multiplication

Murdoch UNIVERSITY

- Numpy supports multiplication of two compatible matrices. Eg., A is a 3x3 matrix and B is a 3x2 matrix:

$$A = \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \qquad B = \begin{vmatrix} 2 & 1 \\ 0 & 2 \\ 1 & 0 \end{vmatrix}$$

$$AB = \begin{vmatrix} 1x2+2x0+3x1 & 1x1+2x2+3x0 \\ 4x2+5x0+6x1 & 4x1+5x2+6x0 \\ 7x2+8x0+9x1 & 7x1+8x2+9x0 \end{vmatrix} = \begin{vmatrix} 5 & 5 \\ 14 & 14 \\ 23 & 23 \end{vmatrix}$$

```
import numpy as np
A = np.array([ [ 1, 2, 3 ],
               [ 4, 5, 6 ],
               [ 7, 8, 9 ] ])
B = np.array([ [ 2, 1 ],
               [ 0, 2 ],
               [ 1, 0 ] ])
C = A @ B
print(C)
D = A.dot(B)
print(D)
```

Universal Functions

Murdoch UNIVERSITY

- Numpy provides functions for many common mathematical operations, known as "universal functions", or ufuncs. These functions are used to implement vectorisation which is much faster than iterating over the ndarray with a loop. Eg
  - `np.add(a1,a2)` add two ndarrays element-wise
  - `np.multiply(a1,a2)` multiply two ndarrays element-wise
  - `np.absolute(a)` apply abs on the ndarray element-wise
  - `np.sum([a1,a2],axis=i)` summation of two arrays along axis i
  - `np.lcm(n1,n2)` the lowest common multiple of two numbers
  - `np.gcd(n1,n2)` greatest common denominator of two numbers
  - `np.sin(a)` apply sin on each element of the ndarray

Universal Functions

Murdoch UNIVERSITY

```
import numpy as np

print(np.lcm(3,6))  # 6
print(np.gcd(3,6))  # 3

a = np.array([1.0, 2.0, 3.0, 4.0])
b = np.array([4.0, 3.0, 2.0, 1.0])

print(np.sin(a)) # [ 0.84147098  0.90929743  0.14112001 -0.7568025 ]
print(np.add(a,b)) # [5. 5. 5. 5.]


c = np.array([ [ 1, 2, 3 ],
               [ 4, 5, 6] ])
d = np.array([ [ 6, 5, 4 ],
               [ 3, 2, 1 ]])

print(np.sum([c,d],axis=1)) # [[5 7 9] [9 7 5]]
print(np.sum([c,d],axis=2)) # [[ 6 15] [15  6]]
print(np.sum([c,d]))        # 42
```

References

- W3school:

  https://www.w3schools.com/python/numpy/default.asp

- Tutorialspoint

  https://www.tutorialspoint.com/numpy/numpy_introduction.htm

- NumPy Quickstart

  https://numpy.org/doc/stable/user/quickstart.html