Numpy

April 1, 2024

- Written partially in python, most of the parts requiring fast computation utilise C or C++
- It's the universal standard for working with numerical data in Python, and it's at the core of the scientific python and libraries like numpy, scikit-learn, and pandas
- It provides ndarray, a homogeneous n-dimensional array object, with methods to efficiently operate on it
- It can be used to perform a wide variety of mathematical operations on arrays
- It adds powerful data structures to Pthon that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices
- NumPy is a general-purpose array-processing package
- It provides a high-performance multidimensional array object, and tools for working with these arrays
- It contains various features including these important ones:
  - A powerful N-dimensional array object
  - Sophisticated (broadcasting) functions
  - Tools for integrating C/C++ and Fortran code

1 Why use numpy?

- Numpy aims to provide an array object that is up to 50x faster than traditional Python lists
- Numpy array consumes less memory than lists

2 Element wise operations (2 arrays)

(Cant be arsed with showing all of them via code)

These basically take in two arrays as arguments. Both should have the same shape. They are:

- `np.add()`
- `np.subtract()`
- `np.multiply()`
- `np.divide()`
- `np.remainder()`
- `np.power()` (First array elements raised to powers from second array)
- `np.maximum()`
- `np.minimum()`
- `np.logical_or()`
- `np.logical_and()`

3 Element wise operations (1 array)

These basically take in an array and return an array of the same shape. They are:

- `np.reciprocal()` (It makes a zero array for arrays with with the dtype being int)
- `np.sign()` (1 for positive, -1 for negative)
- `np.abs()`
- `np.floor()`
- `np.ceil()`
- `np.round()` (Can take another argument, ie the argument you would pass in built in function `round`)
- `np.sqrt()`

4 Parameters/arguments

4.1 `np.array`

- `dtype`: Data type that all the elements will be converted to. If a value in the list cannot be converted to the desired datatype, a `ValueError` is raised. It can be a python class (like `complex`, `float`, or `bool`), or can be a string representing the desired datatype like:
  - `int32` or `float32` (Here `32` represents the number of bytes the item takes)
  - `i4` or `f4` (Here `4` represents the number of bits the item takes)
  - `uint` (Unsigned integer)
  - `U` (Unicode string)
  - `S3` (String) (`3` represents the max number of characters in a element)

4.2 `np.zeros` and `np.ones`

- `size`: Must be a tuple, with first elem being no. of rows and second elem being no. of columns
- `dtype`

5 Differences

5.1 `.shape` vs `.size` vs `.itemsize`

- `.shape` returns a tuple containing the number of rows and number of columns of the array
- `.size` returns an integer ie the total number of elements in the array
- `.itemsize` returns the number of bytes occupied by a single elem

5.2 `min/max` vs `argmin/argmax`

- `min/max` returns the element
- `argmin/argmax` returns the index(s)

5.3 `.shape` vs `.reshape` vs `.resize`

(Note: I am talking about attributes/methods here. `np.resize` returns a copy and also starts repeating elements instead of using `0`. `np.reshape` works in the same way as the method)

Aspect	`.shape`	`.reshape`	`.resize`
Error	Throws an error if mxn != no. of elems	Throws an error if mxn != no. of elems	If mxn != no. of elems, sets the new elements to `0`
Inplace/returns	Inplace	Returns just a "view". So if changes to the original array is made, they are reflected	Inplace

5.4 `.ravel` vs `.flatten`

(Note: `.flat` is an attribute which represents the array in a 1D view) (Also note: there is also a function called `np.ravel` which works in the same way as the `ravel` method)

- They both return shit, neither is inplace
- `.ravel` returns a "view", while `.flatten` returns a copy
- `.ravel` is faster as it does not occupy any memory

6 Other shit

- `numpy.ndarray` and `numpy.array` are essentially the same thing. The `numpy.array` function is a constructor for creating a new array, and the result is an instance of the `numpy.ndarray` class. So, when you use `numpy.array`, you are creating a NumPy array object, which is an instance of the `numpy.ndarray` class
- axis 0 means column, axis 1 means row
- `any`, `all`, and `median` work in the same syntax as `min` or `max`
- Like pandas dataframes, doing something like `2*arr` will double all the elems in `arr`
- When doing comparison operators between two arrays, it must be ensured that both arrays have the same shape. The resultant would be an array of the same shape with each element being `True` or `False` depending on the corresponding elements in the original arrays

[1]:
```python
import numpy as np

arr = np.array([[1,2,3,4,5,6]], dtype='f4')
print('1 ', type(arr))
print('2 ', arr[0,1])
print('3 ', arr)
print('4 ', arr.astype(int))
print('5 ', arr.dtype)
print('6 ', arr.size)
print('7 ', arr.shape)
print('8 ', arr.ndim)
print('9 ', arr.itemsize)
print('10', arr.T)
print('11', np.empty((2,3), float))
print('12', np.zeros((2,3), bool))
print('13', np.ones((1,4), int))
print('14', np.eye(2, dtype=int)) # Create a (square) identity matrix of n rows and n cols
print('15', np.random.random( (1,3) )) # Create a 1x3 array with random values
print('16', np.arange(0,10,2)) # Basically np.array( range(0,10,2)
print('17', np.linspace(4,100,25)) # Create an array with 25 elems, with the first elem being 4 and the last elem being 25. Gaps b/w the elems is equal
arr2 = np.array([[1,2,3,4,5,6]], dtype='i')
print('20', arr2)
print('19', np.array_equal(arr, arr2))

print('\n'+'-'*100+'\n')

print('SHAPE, RESHAPE, & RESIZE\n')
arr = np.array([[1,2,3,4,5,6]], dtype='f4')
arr2 = arr.copy()
print('20', arr2)
arr2.shape=(2,3)
print('21', arr2)
print('22', arr.reshape(2,3))
arr2 = arr.copy()
arr2.resize(3,3)
print('23', arr2)

print('\n'+'-'*100+'\n')

print('SUM & CUMSUM\n')
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('24', arr)
print('25', arr.sum()) # Returns an int ie sum of all vals
print('26', arr.sum(axis=0)) # Returns a 1D arr that contains sums of all cols
print('27', np.sum(arr,axis=1)) # Returns a 1D arr that contains sums of all rows
print('28', arr.cumsum()) # Returns a 1D arr
print('29', arr.cumsum(axis=0)) # Returns a 2D arr
print('30', np.cumsum(arr,axis=1)) # Returns a 2D arr

print('\n'+'-'*100+'\n')

print('PROD & CUMPROD\n')
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('31', arr)
print('32', arr.prod()) # Returns an int ie product of all vals
print('33', arr.prod(axis=0)) # Returns a 1D arr that contains product of all cols
print('34', np.prod(arr,axis=1)) # Returns a 1D arr that contains product of all rows
print('35', arr.cumprod()) # Returns a 1D arr
```

```python
print('36', arr.cumprod(axis=0)) # Returns a 2D arr
print('37', np.cumprod(arr,axis=1)) # Returns a 2D arr

print('\n'+'-'*100+'\n')

print('MEAN\n')
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('38', arr)
print('39', arr.mean()) # Returns an int ie mean of all vals
print('40', arr.mean(axis=0)) # Returns a 1D arr that contains means for every col
print('41', np.mean(arr, axis=1)) # Returns a 1D arr that contains means for every row

print('\n'+'-'*100+'\n')

print('MIN & MAX\n')
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('42', arr)
print('43', arr.min())
print('44', arr.min(axis=0))
print('45', np.min(arr, axis=1))
print('46', arr.max())
print('47', arr.max(axis=0))
print('48', np.max(arr, axis=1))
```

```
1 <class 'numpy.ndarray'>
2 2.0
3 [[1. 2. 3. 4. 5. 6.]]
4 [[1 2 3 4 5 6]]
5 float32
6 6
7 (1, 6)
8 2
9 4
10 [[1.]
 [2.]
 [3.]
 [4.]
 [5.]
 [6.]]
11 [[4.9e-324 9.9e-324 1.5e-323]
 [2.0e-323 2.5e-323 3.0e-323]]
12 [[False False False]
 [False False False]]
13 [[1 1 1 1]]
14 [[1 0]
```

```
 [0 1]]
15 [[0.90229271 0.08104771 0.94961901]]
16 [0 2 4 6 8]
17 [  4.   8.  12.  16.  20.  24.  28.  32.  36.  40.  44.  48.  52.  56.
  60.  64.  68.  72.  76.  80.  84.  88.  92.  96. 100.]
20 [[1 2 3 4 5 6]]
19 True

--------------------------------------------------------------------------------
--------------------

SHAPE, RESHAPE, & RESIZE

20 [[1. 2. 3. 4. 5. 6.]]
21 [[1. 2. 3.]
 [4. 5. 6.]]
22 [[1. 2. 3.]
 [4. 5. 6.]]
23 [[1. 2. 3.]
 [4. 5. 6.]
 [0. 0. 0.]]

--------------------------------------------------------------------------------
--------------------

SUM & CUMSUM

24 [[1 2 3]
 [4 5 6]
 [7 8 9]]
25 45
26 [12 15 18]
27 [ 6 15 24]
28 [ 1  3  6 10 15 21 28 36 45]
29 [[ 1  2  3]
 [ 5  7  9]
 [12 15 18]]
30 [[ 1  3  6]
 [ 4  9 15]
 [ 7 15 24]]

--------------------------------------------------------------------------------
--------------------

PROD & CUMPROD

31 [[1 2 3]
 [4 5 6]
 [7 8 9]]
32 362880
33 [ 28  80 162]
34 [  6 120 504]
35 [     1      2      6     24    120    720   5040  40320 362880]
36 [[  1   2   3]
 [  4  10  18]
 [ 28  80 162]]
37 [[  1   2   6]
 [  4  20 120]
 [  7  56 504]]

--------------------------------------------------------------------------------
--------------------

MEAN

38 [[1 2 3]
 [4 5 6]
 [7 8 9]]
39 5.0
40 [4. 5. 6.]
41 [2. 5. 8.]

--------------------------------------------------------------------------------
--------------------

MIN & MAX

42 [[1 2 3]
 [4 5 6]
 [7 8 9]]
43 1
44 [1 2 3]
45 [1 4 7]
46 9
47 [7 8 9]
48 [3 6 9]
```