7 - Numpy

Data Science Practicum 2021/22, Lesson 7

Marko Tkalčič

Univerza na Primorskem

Table of Contents

Numpy

Numpy Exercises

Numpy

- NumPy is a python library used for working with arrays.
- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra and random number generation.

ndarray

- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster that traditional Python lists.
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.

Scipy

- Numpy: a high-performance multidimensional array and basic tools to manipulate these arrays.
- SciPy: a large number of functions that operate on numpy arrays
    - Vector quantization / Kmeans
    - Physical and mathematical constants
    - Fourier transform
    - Integration routines
    - Interpolation
    - Data input and output
    - Linear algebra routines
    - n-dimensional image package
    - Orthogonal distance regression
    - Optimization
    - Signal processing
    - Sparse matrices
    - Spatial data structures and algorithms
    - Any special mathematical functions
    - Statistics

SciPy vs Numpy

- unclear distinction
- the scipy `__init__` method executes a

```
from numpy import *
```

- hence, all numpy functions are available to scipy
- historical perspective:

In an ideal world, NumPy would contain nothing but the array data type and the most basic operations: indexing, sorting, reshaping, basic element-wise functions, etc. All numerical code would reside in SciPy. However, one of NumPy's important goals is compatibility, so NumPy tries to retain all features supported by either of its predecessors. Thus NumPy contains some linear algebra functions, even though these more properly belong in SciPy. In any case, SciPy contains more fully-featured versions of the linear algebra modules, as well as many other numerical algorithms. If you are doing scientific computing with python, you should probably install both NumPy and SciPy. Most new features belong in SciPy rather than NumPy.

Numpy ndarray

- We can create a NumPy **ndarray** object by using the `array()` function.

```
import numpy as np

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

print(arr)

print(type(arr))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

Numpy ndarray

- We can create a NumPy **ndarray** object by using the `array()` function.

```
import numpy as np

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

print(arr)

print(type(arr))
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
```

- To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

```
import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)
```

Dimensions

- 0-D arrays (scalars)

```
import numpy as np

arr = np.array(42)

print(arr)
```

Dimensions

- 0-D arrays (scalars)

```
import numpy as np

arr = np.array(42)

print(arr)
```

- 1-D array (vector): has 0-D arrays as its elements

```
import numpy as np

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

print(arr)
```

Dimensions

- 0-D arrays (scalars)

```
import numpy as np

arr = np.array(42)

print(arr)
```

- 1-D array (vector): has 0-D arrays as its elements

```
import numpy as np

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

print(arr)
```

- 2-D array (matrix): has 1-D arrays as its elements

```
import numpy as np

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

print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

Dimensions

- 3-D array (3rd order tensor): has 2-D arrays (matrices) as its elements

```
import numpy as np

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

print(arr)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

Check Dimensions

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

Accessing Elements

```
import numpy as np

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

print(arr[2] + arr[3])
```

Accessing Elements

```
import numpy as np

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

print(arr[2] + arr[3])
```

- for matrices: `arr[row,col]`

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st dim: ', arr[0, 1])
```

```
2nd element on 1st dim:  2
```

Accessing Elements

```
import numpy as np

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

print(arr[2] + arr[3])
```

- for matrices: `arr[row,col]`

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st dim: ', arr[0, 1])
```

```
2nd element on 1st dim:  2
```

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('5th element on 2nd dim: ', arr[1, 4])
```

```
5th element on 2nd dim:  10
```

Accessing Multidimensional Arrays

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
```

```
6
```

- 0 -> [[1, 2, 3], [4, 5, 6]]
- 1 -> [4, 5, 6]
- 2 -> 6

Negative Indexing

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

```
Last element from 2nd dim:  10
```

Slicing

- `[start:end:step]`
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1

```
import numpy as np

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

print(arr[1:5])
```

```
[2 3 4 5]
```

Numpy Data Types

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

```
import numpy as np

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

print(arr.dtype)
```

```
int32
```

Creating Arrays With a Defined Data Type

```
import numpy as np

arr = np.array([1, 2, 3, 4], dtype='S')

print(arr)
print(arr.dtype)
```

```
[b'1' b'2' b'3' b'4']
|S1
```

Converting Data Type on Existing Arrays

```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype('i')

print(newarr)
print(newarr.dtype)
```

```
[1 2 3]
int32
```

Copy/View

- Copy

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

- View

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
x[1] = 31

print(arr)
print(x)
```

```
[42 31  3  4  5]
[42 31  3  4  5]
```

Shape

```
import numpy as np

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

print(arr.shape)
```

```
(2, 4)
```

Reshape

- Reshape From 1-D to 2-D

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Reshape

- Reshape From 1-D to 2-D

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

- Reshape From 1-D to 3-D

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(2, 3, 2)
print(newarr)
```

```
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

- We can reshape in any shape, as long as the elements required for reshaping are equal in both shapes

Iterating

```python
import numpy as np

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

for x in arr:
  for y in x:
    print(y)
```

```
1
2
3
4
5
6
```

Ranges

- arange: Return evenly spaced values within a given interval.

```
numpy.arange([start, ]stop, [step, ]dtype=None)
```

```
np.arange(3,7,2)
```

- *When using a non-integer step, such as 0.1, the results will often not be consistent. It is better to use numpy.linspace for these cases.*

Ranges

- arange: Return evenly spaced values within a given interval.

```
numpy.arange([start, ]stop, [step, ]dtype=None)
```

```
np.arange(3,7,2)
```

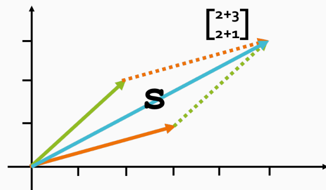
- *When using a non-integer step, such as 0.1, the results will often not be consistent. It is better to use numpy.linspace for these cases.*

- Linspace: Evenly spaced numbers over a specified interval.

```
a = np.linspace(1, 10, num=10)
print(a)
```

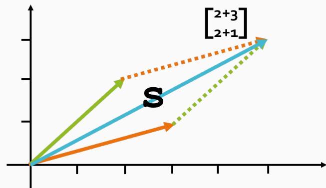
```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

Vector operations




```
v = np.array([2,2])
u = np.array([3,1])
s = v + u
```

Vector operations



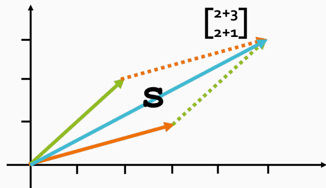

```
v = np.array([2,2])
u = np.array([3,1])
s = v + u
```

- Hadamard product

```
v = np.array([2,2])
u = np.array([3,1])
s = v * u
```

```
[6 2]
```

Vector operations




```
v = np.array([2,2])
u = np.array([3,1])
s = v + u
```

- Hadamard product

```
v = np.array([2,2])
u = np.array([3,1])
s = v * u
```

```
[6 2]
```

- Dot product

```
v = np.array([2,2])
u = np.array([3,1])
s = np.dot(v,u)
print(s)
```

```
8
```

Joining Arrays

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))

print(arr)
```

- we join arrays by axes
  - if not specified, is 0 (i.e. first dimension)

```
[1 2 3 4 5 6]
```

Joining Arrays

```
import numpy as np

arr1 = np.array([[1, 2], [3, 4]])

arr2 = np.array([[5, 6], [7, 8]])

arr = np.concatenate((arr1, arr2), axis=1)

print(arr)
```

```
[[1 2 5 6]
 [3 4 7 8]]
```

Stacking

- Joining along a new axis
  - 2 1-D arrays become a 2-D array

```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.stack((arr1, arr2), axis=1)

print(arr)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

Search

- Return an index matching the query

```
import numpy as np

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

x = np.where(arr == 4)

print(x)
```

```
(array([3, 5, 6], dtype=int64),)
```

Sorting

```
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

Filtering

```
import numpy as np

arr = np.array([41, 42, 43, 44])

filter_arr = arr > 42

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

```
[False False  True  True]
[43 44]
```

Random

- 0-1

```
from numpy import random

x = random.rand()

print(x)
```

Random

- 0-1

```
from numpy import random

x = random.rand()

print(x)
```

- random integer 0,100

```
from numpy import random

x = random.randint(100)

print(x)
```

Random

- 0-1

```
from numpy import random

x = random.rand()

print(x)
```

- random integer 0,100

```
from numpy import random

x = random.randint(100)

print(x)
```

```
from numpy import random

x = random.randint(100, size=(3, 5))

print(x)
```

Random

```
import numpy as np
mu = 15
sigma = 5
size=5000
my_sample = np.random.normal(mu, sigma, size)
```

Random

```
import numpy as np
mu = 15
sigma = 5
size=5000
my_sample = np.random.normal(mu, sigma, size)
```

```
from numpy import random

x = random.normal(size=(2, 3))

print(x)
```

```
[[ 0.04544007  1.96473935 -0.15858358]
 [-0.20709777 -1.40407508 -1.08589647]]
```

Random

```
import numpy as np
mu = 15
sigma = 5
size=5000
my_sample = np.random.normal(mu, sigma, size)
```

```
from numpy import random

x = random.normal(size=(2, 3))

print(x)
```

```
[[ 0.04544007  1.96473935 -0.15858358]
 [-0.20709777 -1.40407508 -1.08589647]]
```

```
from numpy import random

x = random.uniform(size=(2, 3))

print(x)
```

```
[[0.90171773 0.95619308 0.22216792]
 [0.91454814 0.78921139 0.54851748]]
```

Table of Contents

Numpy

Numpy Exercises

Exercise

- create an array with the elements [2,6,3,4,7]

Exercise

- create an array with the elements [2,6,3,4,7]

```
a = np.array([2,6,3,4,7])
a
```

Exercise - arange()

- create an array that contains even values from 1 to 15, i.e. 1,3,5. . . 15

```
a = np.arange(1,16,2)
a
```

Exercise

- create an array with the elements [1,3,5,7,9]

Exercise

- create an array with the elements [1,3,5,7,9]

```
a = np.array([1,3,5,7,9])
a
```

```
d = np.arange(1,10,2)
d
```

```
e = np.linspace(1, 9, 5)
e
```

Exercise

- create an array with elements `[0 1 2 3 4 5 6 7 8 9]`
- given the notation from:to
  - print all the elements of the array
  - print the first three elements
  - print the second to fourth element
  - print the last three elements

Exercise

- create an array with elements [0 1 2 3 4 5 6 7 8 9]
- given the notation from:to
  - print all the elements of the array
  - print the first three elements
  - print the second to fourth element
  - print the last three elements

```
x = np.arange(10)

print(x)
print(x[:3])
print(x[1:4])
print(x[-3:])
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2]
[1 2 3]
[7 8 9]
```

Exercise

- given the notation from:to:step
- print every third element starting from the second

Exercise

- given the notation from:to:step
- print every third element starting from the second

```
a = np.arange(0,10)
print(a)
b = a[1:10:3]
print(b)
```

```
[0 1 2 3 4 5 6 7 8 9]
[1 4 7]
```

Exercise

- given the notation from:to:step
- if we omit from and to it assumes `from=0` and `to=length`
    - get every second element
    - get every third element
    - reverse the order

Exercise

- given the notation from:to:step
- if we omit from and to it assumes `from=0` and `to=length`
  - get every second element
  - get every third element
  - reverse the order

```
print(x[::2])
print(x[::3])
print(x[::-1])
```

```
[0 2 4 6 8]
[0 3 6 9]
[9 8 7 6 5 4 3 2 1 0]
```

Exercise - reshape

- create a 5x5 matrix with values from 0 to 24

Exercise - reshape

- create a 5x5 matrix with values from 0 to 24

```
x = np.arange(25).reshape(5, 5)
print(x)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

Exercise

- given the array from above, get the first two columns and the first two rows
- hint: slice per every dimension

Exercise

- given the array from above, get the first two columns and the first two rows
- hint: slice per every dimension

```
print(x[:2, :2])
```

```
[[0 1]
 [5 6]]
```

Exercise

- Get every third column of every second row

Exercise

- Get every third column of every second row

```
y = x[::2 , ::3]
print(y)
```

```
[[ 0  3]
 [10 13]
 [20 23]]
```

Exercise

- `np.random.randint(low,high)`
- `np.random.normal(mean, std, num)`
- `.shape`
- create a 2D array with random dimensions from 1 to 5
- fill it with random variables drawn from a random distribution with mean=10 standard deviation=3
- use shape to print out the number of columns

Exercise

- `np.random.randint(low,high)`
- `np.random.normal(mean, std, num)`
- `.shape`
- create a 2D array with random dimensions from 1 to 5
- fill it with random variables drawn from a random distribution with mean=10 standard deviation=3
- use shape to print out the number of columns

```
x = np.random.randint(1,5)
y = np.random.randint(1,5)

a = np.random.normal(10,3,x*y)
b = a.reshape(x,y)
print(b)
s = b.shape
print(s[1])
```

```
[[10.33272803 13.22042969]
 [10.93450619 10.3011933 ]
 [ 6.33357001 10.69188525]]
2
```

Exercise - indeces

- create an array with 6 random integers from 0 to 10
- using an index array, print the first, second to last, and last element

Exercise - indeces

- create an array with 6 random integers from 0 to 10
- using an index array, print the first, second to last, and last element

```
x = np.random.randint(0, 10, 6)
print(x)
indices = [0, -2, -1]
print(x[indices])
```

```
[1 7 4 9 2 5]
[1 2 5]
```

Exercise

- create an array with elements from 0 to 4
- create a boolean array of indexes where the value of the first array is bigger than 2
- use the index array to create a new array that contains only elements >2

Exercise

- create an array with elements from 0 to 4
- create a boolean array of indexes where the value of the first array is bigger than 2
- use the index array to create a new array that contains only elements >2

```
x = np.arange(5)
print(x)
i = x >2
print(i)
print(x[i])
```

```
[0 1 2 3 4]
[False False False  True  True]
[3 4]
```

Exercise

- Create an array from 0 to 26 assign it to a variable x

Exercise

- Create an array from 0 to 26 assign it to a variable x

```
x = np.arange(0,27)
x
```

- Reverse the order of the array and print it out

Exercise

- Create an array from 0 to 26 assign it to a variable x

```
x = np.arange(0,27)
x
```

- Reverse the order of the array and print it out

```
y = x[::-1]
print(y)
```

- Convert the 1-dimensional array you created into a 3 dimensional array

Exercise

- Create an array from 0 to 26 assign it to a variable x

```
x = np.arange(0,27)
x
```

- Reverse the order of the array and print it out

```
y = x[::-1]
print(y)
```

- Convert the 1-dimensional array you created into a 3 dimensional array

```
z = y.reshape((3,3,3))
print(z)
```

```
[[[26 25 24]
  [23 22 21]
  [20 19 18]]

 [[17 16 15]
  [14 13 12]
  [11 10  9]]

 [[ 8  7  6]
  [ 5  4  3]
  [ 2  1  0]]]
```

Standardization

- Create a vector of 10 elements with normal distribution
  - $\mu_x = 50$
  - $\sigma_X = 5$
- Perform standardization on the array:

$$x^i_{std} = \frac{x^i - \mu_x}{\sigma_x}$$

Standardization

- Create a vector of 10 elements with normal distribution
  - $\mu_x = 50$
  - $\sigma_X = 5$
- Perform standardization on the array:

$$x^i_{std} = \frac{x^i - \mu_x}{\sigma_x}$$

- Perform min-max normalization

$$x^i_{norm} = \frac{x^i - x_{min}}{x_{max} - x_{min}}$$

References

Part of the material has been taken from the following sources. The usage of the referenced copyrighted work is in line with fair use since it is for nonprofit educational purposes.

- https://www.w3schools.com/python/numpy_intro.asp
- https://www.tutorialspoint.com/numpy/numpy_introduction.htm
- https://cs231n.github.io/python-numpy-tutorial/#scipy
- https://www.scipy.org/scipylib/faq.html#numpy-vs-scipy-vs-other-packages