PYTHON FOR DATA SCIENCE WORKING WITH DATA (II)

Ana-Maria Dobre

based on EDX Course

SEP 2023

5. ONE-DIMENSIONAL NUMPY

Objectives: NUMPY in 1D; ND arrays
NUMPY:
- library for scientific computing
- many useful functions, speed, memory
- the basis for pandas

Python list - example: (1)

A Numpy array or ND array: (2)
- similar to a list
- usually fixed in size
- each element is of the same type // here, integers

Cast a list to an ND array: (3)

Play around with attributes and methods: (4)
Notes:
- ndim = the number of array dimensions or the rank of the array
- a.shape = tuple of integers indicating the size of the array in each dimension
- b = numpy array with real numbers

Continue with INDEXING & SLICING methods: (5)

```
(1) list = [1, "two", "geronimo", "5", 100]
(2) [1, 2, 3, 4, 5]

(3) import numpy as np
a = np.array(list)

(4)
import numpy as np
list = [100, 2, 3]
a = np.array(list)
print(a[0])
print(type(a)) #<class 'numpy.ndarray'>
print(a.dtype) #int32
print(a.size) # 3
print(a.ndim) # 1
print(a.shape) # (3,)
b = np.array([1.0, 2.5, 4.7])
print(type(b)) #<class 'numpy.ndarray'>
print(b.dtype) #float64

(5)
#b[0]="geronimo" #ValueError: could not convert string to float: 'geronimo'
print(b[1:4]) # select items with indexes 1, 2 and 3
b[3:5] = 30, 40 # replaces last 2 items, so: indexes 3 and 4
```

Numpy makes it easier to do many operations that are commonly performed in data science.
The same operations are usually computationally faster and require less memory in numpy compared to regular Python.

Vector addition and subtraction: (6)

Array multiplication with a scalar: (7)

Hadamard product with 1 line of code in numpy: (8)

(9) # DOT PRODUCT: shows how similar 2 vectors are
We multiply the first component from v and u, we then multiply the second component and add the result together
The result is a number that represents how similar the two vectors are
We can also perform dot product using the numpy function dot and assign it with the variable result as follows

(10)
BROADCASTING = Add a constant to a numpy Array

(11) # Universal functions

(12)
A useful function for plotting mathematical functions is linespace.
linespace returns evenly spaced numbers over specified interval.
We specify the starting point of the sequence, the ending point of the sequence.
The parameter num indicates the number of samples to generate, in this case, 5.

(13) SLICING: We can also define the steps in slicing, like this: [start:end:step].
If we don't pass start its considered 0
print(arr[:4]) # [1 2 3 4]
If we don't pass end it considers till the length of array. print(arr[4:]) # [5 6 7]

```
(6) u = np.array([1, 0])
v = np.array([0, 1])
z = u+v
z_sub = u-v

#Compare this with vector addition on 2 lists:
ul = [12, 0]
vl = [0, 12]
zl = []
for n, m in zip(ul, vl):
    zl.append(n+m)

# in case of vector subtraction on 2 lists, only this line changes:
# zl.append(n-m)

(7) z_mult = 2*z

# in case of multiplication on a list, only these 2 lines change: # for n in y
#    zl.append(2*n)

(8)  h1 = np.array([1, 2])
h2 = np.array([3, 2])
h3 = h1*h2
print(h3) # [3, 4]

# the Hadamard product on 2 lists means that only this line changes: # zl.append(n*m)

(9) d1 = np.array([1, 2])
d2 = np.array([3, 1])
similarity = np.dot(d1, d2)
print(similarity) # 1*3+2*1 = 5

(10) c = np.array([1, 2, 3, -5])
c_res = c + 1

(11) print(c.mean())
print(c.max())
print(np.pi)
x = np.array([0, np.pi/2, np.pi])
print(np.sin(x))

(12) print(np.linspace(-2, 2, num=5)) # [-2. -1. 0. 1. 2.]
print(np.linspace(-2, 2, num=9)) # [-2. -1.5 -1. -0.5 0. 0.5 1. 1.5 2. ]

# We can use the function linespace to generate 10
# evenly spaced samples from interval [0, 2 * np.pi]:
ls = np.linspace(0, 2*np.pi, num=10)

# We can use the numpy function sin to map the array ls to a new array ls_2.
ls_2 = np.sin(ls)
print(ls_2)
# %matplotlib inline
plt.plot(ls, ls_2)

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

6. TWO-DIMENSIONAL NUMPY

Consider the list a (14):
// contains three nested lists each of equal size
It is helpful to visualize the **numpy array** as a rectangular array each nested lists corresponds to a different row of the matrix.

#slicing (15)

We can add the matrices = adding the elements in the same position (16)

Multiplying by a scalar: (17)
If we multiply the matrix by this scalar 2, we simply multiply every element in the matrix by 2.

Hadamar product = multiplying each # of the elements in the same position: (18)
Note: # Explained: From matrix multiplication, to obtain the ith row and jth column of the new matrix, we take the dot product of the ith row of D with the jth columns of E.

```
(14)
import numpy as np
a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]
A = np.array(a)
print(A)
print(A.ndim) # number of nested lists
print(A.shape) # returns a tuple: (3,3) where 3 = number of lists, the second 3 = size
print(A.size) # 3x3 = 9
print(A[0][0]) # rectangular brackets to access array elements # 11

(15)
print(A[0,0:2]) # 0 = first row, 0:2 = columns 0 and 1 # [11 12]
print(A[0:2,2]) # 0:2 = rows 0 and 1, 2 = last or third column (index starts at 0) # [13 23]

(16)
X = np.array([[1,0], [0,1]])
Y = np.array([[2,1], [1,2]])
print(X + Y) # [[3,1], [1,3]]

(17)
Z = np.array([[2,1], [1,2]])
Z_res = 2 * Z
print(Z_res)

(18)
print(X*Y) # [[1*2,0*1], [0*1, 1*2]]

# matrix multiplication

# requirement: number of columns (3) in matrix D = number of rows (3) in matrix E
D = np.array([[0,1,1], [1,0,1]])
E = np.array([[1,1], [1,1], [-1,1]])
print(D)
print(E)
print(np.dot(D, E)) # [[0,2], [0,2]]

# res[0][0] = D[0][0]*E[0][0]+D[0][1]*E[1][0]+D[0][2]*E[2][0] = 0x1 + 1x1 + 1x(-1) = 0+1-1 = 0

# res[0][1] = D[0][0]*E[0][1]+D[0][1]*E[1][1]+D[0][2]*E[2][1] = 0x1 + 1x1 + 1x1 = 0+1+1 = 2
```

NOTES:
print(np.__version__) # check numpy version

 # Get the standard deviation of numpy array
standard_deviation=a.std()