Numpy

Prof.Dr. Bahadır AKTUĞ

803400815021 Machine Learning with Python

**Compiled from sources given in the references.*

Numpy

- Numpy has arrays of a single numeric data type in contrast to the arrays in Python (list, tuple).
- Numpy arrays are particularly useful for mathematical operations.
- Numpy package is not a part of Python programming language (not a built-in modüle). It needs to be installed separately and should be imported as a module.
- The elements of Numpy arrays can be accessed by indexing similar to the Python arrays.
- Numpy arrays can be one or more dimensional. The dimension of the arrays are specified by it axes and rank.
- Many popular Python modules (e.g. Scipy, Pandas etc.) also depend on Numpy.

Numpy

- The data type of Numpy arrays is "ndarray. It is also named as «numpy.array»

- Fundamental ndarray commands
  - ndarray.ndim
    - The number of axes. For the dimension of the arrays «rank» is also used.
  - ndarray.shape
    - The size of the array along axes. For a 2x3 matrix, its "shape" is (2,3).
  - ndarray.size
    - The total number of elements in an array. For a 2x3 matrix, its «size» is 6.

Numpy

- Fundamental ndarray commands (cont.)

  - ndarray.dtype
    - The type of the elements of the array. In addition to the basic Python data types, Numpy data types such as numpy.int32, numpy.int16, and numpy.float64.
  - ndarray.itemsize
    - The size of each element of the array in bytes. For instance, an element of "float64" has an itemsize of 8, an element of "complex32" has an itemsize of 4.
  - ndarray.data
    - All the elements of an array. Since the elements can be accessed by slicing and indexing, not frequently needed.

Numpy

```
>>> import numpy as np
>>> dizi = np.arange(12).reshape(3,4)
>>> dizi
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> dizi.shape
(3, 4)
>>> dizi.dtype
dtype('int32')
>>> dizi.itemsize
4
>>> dizi.ndim
2
>>> dizi.size
12
```

Numpy

- Examples of “shape” command, one of the most frequently used array commands:

```
import numpy as np

a = np.array([1, 2, 3])  # One dimensional array
b = np.array([[1,2,3],[4,5,6]])   # Two dimensional array

print(a.shape) → (3,)
print(b.shape) → (2,3)
```

Numpy

- Numpy array can be constructed through the following methods:
  - Using np.array command with standard Python lists and tuples
  - Using pre-defined numpy commands (numpy.arange, numpy.ones, numpy.eye, numpy.zeros, numpy.empty, numpy.full,numpy.random, numpy.linspace)
- The data type can be given while constructing the array by using the numpy.array command.
- If data tye is not given, numpy automatically determines the data type. The default values for integers are "int32/int64“ and “float64” for decimal/floating numbers.

Numpy Data Types

- When a new array is formed, the best data type is determined by Numpy. But this can be overridden and the data type can be given explicitly:

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

x.dtype → "int64" veya "int32"

```
x = np.array([1.0, 2.0])
```

x.dtype → "float64"

```
x = np.array([1, 2], dtype=np.int64)
```

x.dtype → "int64"

Numpy

```
import numpy as np
>>> a = np.array([1, 2, 3])
>>> b = np.array([[1,2,3],[4,5,6]],dtype='float64')
```

or

```
>>> b = np.array([[1,2,3],[4,5,6]],dtype=np.float64)
>>> a.dtype
dtype('int32')
>>> a = np.array([.1, 2, 3])
>>> a.dtype
dtype('float64')
>>> b.dtype
dtype('float64')
```

Numpy

- Some special functions which produce Numpy arrays:

```
a = np.zeros((2,2))  # 2x2 zero matrix

b = np.ones((1,2))   # # 1x2 ones matrix

c = np.full((2,2), 7) # 2x2 matrix of a constant number

d = np.eye(2)        # 2x2 identity matrix

e = np.random.random((2,2)) # 2x2 matrix of random numbers
```

Numpy

- Some special functions which produce Numpy arrays:

```
>>> import numpy as np
>>> np.arange(2,10,2)
array([2, 4, 6, 8])
>>> np.arange(20,0,-5)
array([20, 15, 10,  5])
>>> np.linspace(0,50,9)
array([  0.  ,   6.25,  12.5 ,  18.75,  25.  ,  31.25,  37.5 ,  43.75,  50.  ])
>>> np.linspace(0,50,11)
array([  0.,   5.,  10.,  15.,  20.,  25.,  30.,  35.,  40.,  45.,  50.])
```

Numpy

- Some aggregate functions of Numpy arrays:

a.sum()	Array-wise sum
a.min()	Array-wise minimum value
a.max(axis=0)	Maximum value of an array row
a.cumsum(axis=1)	Cumulative sum of the elements
a.mean()	Mean
a.median()	Median
a.corrcoef()	Correlation coefficient
np.std(b)	Standard deviation

Slicing of Numpy Arrays

- Numpy arrays can be accessed through slicing:

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

- However, when a subarray is modified, the main array is also modified!!!

```
b[0, 0] = 77    → a[0, 1] → 77
```

Slicing of Numpy Arrays

- A multiple assignment is also possible with slicing:

```
>>> a = np.arange(10)*2
>>> a
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
>>> a[:9:3] = 50
>>> a
array([50,  2,  4, 50,  8, 10, 50, 14, 16, 18])
>>>
```

- The use of ":" operator is the same as standard Python array slicing.

Slicing of Numpy Arrays

- If the number of indices is less than the dimension of the array, the indices are assigned to the axes respectively. For the missing axes, a “:” operator is assumed.

```
>>> a = np.array([[2,10,2],[3,7,9],[4,8,1]])
>>> a[-1]
array([4, 8, 1])
>>> a[2,:]
array([4, 8, 1])
>>> a[:,-1]
array([2, 9, 1])
```

Loop over Numpy Arrays

- When a loop is constructed over Numpy arrays, the loop is iterated only on the first axis (dimension).

```
>>> a
array([[ 2, 10,  2],
       [ 3,  7,  9],
       [ 4,  8,  1]])
>>> for x in a:
...    print(x)
...
[ 2 10  2]
[3 7 9]
[4 8 1]
```

Slicing of Numpy Arrays

- Beside slicing, the subarrays of a Numpy array can also be accessed by the row and column numbers.
- However, the subarrays formed this way will have a different dimension.

```
import numpy as np

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

row_r1 = a[1, :]     # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
row_r1.shape → (4,)
row_r2.shape → (1, 4)
```

Slicing of Numpy Arrays

- It also holds true for the column dimension.

```
import numpy as np

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

col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
col_r1.shape → (3,)
col_r2.shape → (3, 1)
```

Slicing of Numpy Arrays

- Another way of accessing Numpy arrays is the use of integer arrays:

```
import numpy as np
a = np.array([[1,2], [3, 4], [5, 6]])
a[[0, 1, 2], [0, 1, 0]] → [1 4 5]
```

- The indexing above is the same as below:

```
b = np.array([a[0, 0], a[1, 1], a[2, 0]]) → [1 4 5]
```

- The first approach, however, allows using the same element one than once:

```
a[[0, 0], [1, 1]]  → [2 2]
np.array([a[0, 1], a[0, 1]]) → [2 2]
```

Slicing of Numpy Arrays

- A numpy array could be indices list of another numpy array:

```
import numpy as np
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
b = np.array([0, 2, 0, 1])
a[np.arange(4), b]  → [ 1  6  7 11]
```

- The elements of the array could be modified in this way if one wishes:

```
a[np.arange(4), b] += 10
a →  array([[11,  2,  3],
            [ 4,  5, 16],
            [17,  8,  9],
            [10, 21, 12]])
```

Slicing of Numpy Arrays

- «Boolean» filtering and indexing is also possible:

```
import numpy as np
a = np.array([[1,2], [3, 4], [5, 6]])
bool_idx = (a > 2) → a'nın 2'den büyük elemanlarını bulur
bool_idx → [[False False]
             [ True  True]
             [ True  True]]
a[bool_idx] →[3 4 5 6]
```

- Or in a shorter notation:

```
print a[a > 2]    → [3 4 5 6]
```

Stacking of numpy arrays of different sizes

- Horizontal of vertical stacking of numpy arrays is possible:

```
>>> a = np.array([[2,4,6],[7,1,9]])
>>> b = np.array([[8,3,2],[1,6,0]])
>>> np.vstack((a,b))
array([[2, 4, 6],
       [7, 1, 9],
       [8, 3, 2],
       [1, 6, 0]])
>>> np.hstack((a,b))
array([[2, 4, 6, 8, 3, 2],
       [7, 1, 9, 1, 6, 0]])
```

Vector / Matrix Operations

- Numpy provides the fundamental functions of vector/matrix operations. Operator overloading is also possible:

```
import numpy as np
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

x + y or np.add(x, y) → Elementwise adding
x - y or np.subtract(x, y) → Elementwise subtraction
x * y or np.multiply(x, y) → Elementwise multiplication
x / y or np.divide(x, y) → Elementwise division
np.sqrt(x) → Elementwise squareroot
```

Vector / Matrix Operations

- The usual arithmetic operator operate elementwise
- For matrix/vector multiplication, «dot» function is used:

```
import numpy as np
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([11, 12])

v.dot(w) or np.dot(v, w) → vector product
x.dot(v) or np.dot(x, v) → matrix/vector product
x.dot(y) or np.dot(x, y) → matrix/matrix product
```

Vector / Matrix Operations

- Matrix transpose and row/column summing:

```
import numpy as np
x = np.array([[1,2], [3,4]])
x.T → [[1 3], [2 4]]
```

- While it is possible to use 'T' (transpose) for vectors it has no effect

```
x = np.array([[1,2],[3,4]])
np.sum(x) → 10 sum of all elements
np.sum(x, axis=0)  → [4 6] sum of columns
np.sum(x, axis=1)  → [3 7] sum of rows
```

Vector / Matrix Operations

- Some handy tools while working with large arrays:

```
import numpy as np
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))  # Stack 4 copies of v on top of each other
vv → [[1 0 1]
      [1 0 1]
      [1 0 1]
      [1 0 1]]
y = x + vv  →   [[ 2  2  4
                 [ 5  5  7]
                 [ 8  8 10]
                 [11 11 13]]
```

Numpy Linear Algebra Examples

- Numpy provides a powerfull linear algebra module:

```
import numpy as np
a = np.array([[2,5,14],[7,9,12],[8,2,6]])
b = np.array([[0,0,14],[0,0,12],[0,2,0]])
L = np.linalg.cholesky(a.T.dot(a))
print(a.T.dot(a))
print(L.dot(L.T))

w, v = np.linalg.eig(a.T.dot(a))
v.dot(np.diag(w)).dot(v.T)
print(np.linalg.inv(b))

q,r = np.linalg.qr(a)
print(a)
print(q.dot(r))
```

Numpy File Utilities

- Numpy provides utility functions to load/save ascii/binary files

Value1	Value2	Value3
0.2536	0.1008	0.3857
0.4839	0.4536	0.3561
0.1292	0.6875	0.5929
0.1781	0.3049	0.8928
0.6253	0.3486	0.8791

to return in columns

```
import numpy as np
x, y, z = np.loadtxt('data.txt', skiprows=1, unpack=True)
```

```
>>> x
array([0.2536, 0.4839, 0.1292, 0.1781, 0.6253])
>>> y
(array([0.1008, 0.4536, 0.6875, 0.3049, 0.3486]),)
>>> y
array([0.3857, 0.3561, 0.5929, 0.8928, 0.8791])
```

Numpy File Utilities

- A more general file reader is also provided:

Value1	Value2	Value3
0.2536	0.1008	0.3857
0.4839	0.4536	-
None	0.6875	0.5929
0.1781	Missing	0.8928
0.6253	0.3486	0.8791

to fill the missing values

```
import numpy as np
np.genfromtxt('sil.txt',skip_header=1,filling_values=0)
```

```
array([[0.2536, 0.1008, 0.3857],
      [0.4839, 0.4536, 0.    ],
      [0.    , 0.6875, 0.5929],
      [0.1781, 0.    , 0.8928],
      [0.6253, 0.3486, 0.8791]])
```

Numpy File Utilities

- Saving numpy array with specified delimiters is also very easy:

```
import numpy as np
x = np.arange(0.0,5.0,1.0)
np.savetxt('test.out', x, delimiter=',')
```

test.out

```
0.000000000000000000e+00
1.000000000000000000e+00
2.000000000000000000e+00
3.000000000000000000e+00
4.000000000000000000e+00
```

References

1 Wentworth, P., Elkner, J., Downey, A.B., Meyers, C. (2014). How to Think Like a Computer Scientist: Learning with Python (3nd edition).
2 Pilgrim, M. (2014). Dive into Python 3 by. Free online version: DiveIntoPython3.org ISBN: 978-1430224150.
3 Summerfield, M. (2014) Programming in Python 3 2nd ed (PIP3) : - Addison Wesley ISBN: 0-321-68056-1.
4 Jones E, Oliphant E, Peterson P, et al. SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/.
5 Millman, K.J., Aivazis, M. (2011). Python for Scientists and Engineers, Computing in Science & Engineering, 13, 9-12.
6 John D. Hunter (2007). Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95.
7 Travis E. Oliphant (2007). Python for Scientific Computing, Computing in Science & Engineering, 9, 10-20.
8 Goodrich, M.T., Tamassia, R., Goldwasser, M.H. (2013). Data Structures and Algorithms in Python, Wiley.
9 http://www.diveintopython.net/
10 https://docs.python.org/3/tutorial/
11 http://www.python-course.eu
12 https://developers.google.com/edu/python/
13 https://www.datacamp.com/community/tutorials/python-numpy-tutorial
14 http://learnpythonthehardway.org/book/