
機械学習

Numpyについて

管理工学科

篠沢佳久

Numpy

- 科学技術計算用のパッケージ
 - <https://www.numpy.org/>
- ベクトル, 行列演算に用いられる
- 多次元配列 (ndarray)
 - リストと異なり, 配列の大きさは変えられない, 同じ型の要素のみ
 - ただし, リストと比較して高速
 - 行列演算が可能

Numpyのインストール

- Anacondaの場合, インストール済み
- Pythonの場合のインストール
 - > pip install numpy
- マニュアル
 - <https://www.numpy.org/doc/1.17/reference/index.html>

配列の作成①

対話型シェル上での実行

```
>>> import numpy as np
```

numpyのインポート

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

一次元配列の作成
np.array(list型)

```
>>> a
```

```
array([1, 2, 3, 4, 5])
```

import numpy としてインポートした場合
a = numpy.array([1,2,3,4,5])

```
>>>
```

```
>>> a.size
```

size
配列の要素数

```
5
```

```
>>> a.shape
```

shape
配列の大きさ

```
(5,)
```

```
>>> a.ndim
```

ndim
配列の次元数

```
1
```

配列の作成②

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

二次元配列の作成
np.array(list型)
listは二次元配列で指定

```
>>> b
```

```
array([[1, 2, 3],  
       [4, 5, 6]])
```

```
>>> b.size
```

配列の要素数

```
6
```

```
>>> b.shape
```

配列の大きさ

```
(2, 3)
```

```
>>> b.ndim
```

配列の次元数

```
2
```

配列の作成③

```
>>> c = np.array( [[[1,2,3],[4,5,6]],  
                  [[7,8,9],[10,11,12]],  
                  [[13,14,15],[16,17,18]]] )
```

三次元配列の作成
np.array(list型)
listは二次元配列で指定

```
>>> c  
array([[[ 1,  2,  3],  
        [ 4,  5,  6]],  
       [[ 7,  8,  9],  
        [10, 11, 12]],  
       [[13, 14, 15],  
        [16, 17, 18]]])
```

```
>>> c.size  
18
```

配列の要素数

```
>>> c.shape  
(3, 2, 3)
```

配列の大きさ

```
>>> c.ndim  
3
```

配列の次元数

配列の要素の参照と代入

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

一次元配列の作成

```
>>> a[0]
```

a[0]: 0番目の要素

```
1
```

```
>>> a[0:3]
```

0番目から2番目の要素

a[0],a[1],a[2](a[3]は含まないことに注意)

```
array([1, 2, 3])
```

```
>>> a[2:]
```

a[2]以降

```
array([3, 4, 5])
```

```
>>> a[:3]
```

a[2]まで(a[3]まででないことに注意)

```
array([1, 2, 3])
```

```
>>> a[0]=10
```

a[0]に代入

```
>>> a
```

```
array([10, 2, 3, 4, 5])
```

配列の参照

配列[start:end:step]
(注意)end-1まで

型の指定①

整数型

```
>>> a = np.zeros( (3,3) , dtype=np.int32 )
```

```
>>> a
```

```
array([[0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0]])
```

zeros(配列の大きさ, dtype=値の型)
要素が0の値, 大きさ(3,3)の配列を作成

```
>>> a.dtype
```

型の表示→整数型

```
dtype('int32')
```

numpyの配列(ndarray)
では異なる型の値は格納
できません

```
>>> a = np.ones( (3,3) , dtype=np.int32)
```

```
>>> a
```

```
array([[1, 1, 1],  
       [1, 1, 1],  
       [1, 1, 1]])
```

ones(配列の大きさ, dtype=値の型)
要素が1の値, 大きさ(3,3)の配列を作成

```
>>> a.dtype
```

型の表示→整数型

```
dtype('int32')
```


型の指定②

小数型(単精度)

```
>>> b = np.zeros( (3,3) , dtype=np.float32 )
```

```
>>> b
```

```
array([[0., 0., 0.],  
       [0., 0., 0.],  
       [0., 0., 0.]], dtype=float32)
```

要素が0の値, 大きさ(3,3)の配列を作成

```
>>> b.dtype
```

型の表示→小数型(単精度)

```
dtype('float32')
```

```
>>> c = np.zeros( (3,3) , dtype=np.float64)
```

```
>>> c.dtype
```

小数型(倍精度)

```
dtype('float64')
```

型の表示→小数型(倍精度)

要素がランダムな配列の作成

`random.rand(配列の大きさ)`

大きさは(3,3), 要素が0から1の一樣乱数

```
>>> np.random.rand(3,3)
array([[0.3960435 , 0.19573042, 0.30261743],
       [0.05841145, 0.82443788, 0.6092646 ],
       [0.59970793, 0.18129925, 0.60186834]])
```

`random.randn(配列の大きさ)`

大きさは(3,3), 平均0, 分散1の標準正規分布

```
>>> np.random.randn(3,3)
array([[ -0.41918265, -0.98458284, -0.64019514],
       [-0.74227906,  0.4265353 ,  0.59531986],
       [-0.76386238, -0.50602818, -0.75626888]])
```

配列のコピー①

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

一次元配列の作成

```
>>> b = a
```

bにaを代入

```
>>> b
```

```
array([1, 2, 3, 4, 5])
```

```
>>> a[0]=10
```

注意！

a[0]の値を変えるとb[0]も変わる

```
>>> b
```

```
array([10, 2, 3, 4, 5])
```

配列のコピー②

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

一次元配列の作成

```
>>> b = a.copy()
```

```
>>> b
```

```
array([1, 2, 3, 4, 5])
```

配列名.copy()
bにaの値をコピー

```
>>> a[0]=10
```

a[0]の値を変えてもb[0]は変わらない

```
>>> b
```

```
array([1, 2, 3, 4, 5])
```

配列の変形①

```
>>> a = np.arange(6)
```

arange(N)

要素が0からN-1までの一次元配列を作成

```
>>> a
```

```
array([0, 1, 2, 3, 4, 5])
```

要素が0,1,2,3,4,5の一次元配列を作成

```
>>> b = np.reshape( a , (2,3) )
```

reshape(配列名, 配列の大きさ)
指定した配列の大きさに変形

```
>>> b
```

大きさ(2,3)の二次元配列に変形

```
array([[0, 1, 2],  
       [3, 4, 5]])
```

```
>>> c = b.flatten()
```

配列名.flatten()
一次元配列に変形

```
>>> c
```

```
array([0, 1, 2, 3, 4, 5])
```

配列の変形②

```
>>> a = np.arange(27)
```

要素が0,1,2,...,26の一次元配列を作成

```
>>> a
```

```
array([ 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, 25, 26])
```

```
>>> b = np.reshape( a, (3,3,3) )
```

大きさ(3,3,3)の三次元配列に変形

```
>>> b
```

```
array([[[ 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, 25, 26]]])
```

配列の要素の参照①

```
>>> a = np.arange(12).reshape(4,3)
```

```
>>> a
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

```
>>> a[0][0]
```

```
0
```

```
>>> a[0,0]
```

```
0
```

```
>>> a[0]
```

```
array([0, 1, 2])
```

```
>>> a[3]
```

```
array([ 9, 10, 11])
```

0~11の要素を持つ一次元配列を作成
→ 大きさ(4,3)の二次元配列に変形

```
a = array(
[
    [ 0,  1,  2],
    [ 3,  4,  5],
    [ 6,  7,  8],
    [ 9, 10, 11]
])
```

a[0]

a[1]

a[2]

a[3]

(0,0)の要素

「,」で区切る

0行目(一次元配列)

3行目(一次元配列)

配列aの(i,j)要素
a[i][j]
a[i, j]

「,」で区切る

配列の要素の参照②

```
>>> a[0:2]
array([[0, 1, 2],
       [3, 4, 5]])
```

0から1行目
(2行目は含まれないことに注意)

配列の参照
配列[start:end:step]
(注意)end-1まで

```
>>> a[:]
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

全ての行

```
a = array([[ 0,  1,  2],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
```

```
>>> a[:,1]
array([ 1,  4,  7, 10])
```

全ての行→1列目→一次元配列

区切り

配列の要素の参照③

```
>>> a[:,2]
```

← 全ての行→2列目→一次元配列

```
array([ 2,  5,  8, 11])
```

```
>>> a[:,[2]]
```

← 全ての行→2列目→二次元配列

```
array([[ 2],  
       [ 5],  
       [ 8],  
       [11]])
```

```
a = array([[ 0,  1,  2],  
          [ 3,  4,  5],  
          [ 6,  7,  8],  
          [ 9, 10, 11]])
```

```
>>> a[1:3,2]
```

← 1行目から2行目→2列目→一次元配列

```
array([5, 8])
```

```
>>> a[1:,2]
```

← 1行目以降→2列目→一次元配列

```
array([ 5,  8, 11])
```

配列の要素の参照④

```
>>> a[:,1:3]  
array([[ 1,  2],  
       [ 4,  5],  
       [ 7,  8],  
       [10, 11]])
```

全ての行→ 1列目から2列目
→二次元配列

```
a = array([[ 0,  1,  2],  
          [ 3,  4,  5],  
          [ 6,  7,  8],  
          [ 9, 10, 11]])
```

```
>>> a[:,[0,1]]  
array([[ 0,  1],  
       [ 3,  4],  
       [ 6,  7],  
       [ 9, 10]])
```

全ての行→
0列目と1列目
→二次元配列

指定できない

```
>>> a[:,0,1]  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
IndexError: too many indices for array
```

配列の要素の参照⑤

```
>>> a = np.arange(12).reshape(4,3)
```

```
>>> a
```

```
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 9, 10, 11]])
```

```
>>> a[[3],:]
```

```
array([[ 9, 10, 11]])
```

```
>>> a[[1,2],:]
```

```
array([[3, 4, 5],  
       [6, 7, 8]])
```

0~11の要素を持つ一次元配列を作成
→ 大きさ(4,3)の二次元配列に変形

全列→3行目→二次元配列

全列→1行目と2行目→二次元配列

全列→3行目→一次元配列

```
>>> a[3,:]
```

```
array([ 9, 10, 11])
```

```
>>> a[1,2,:]
```

指定できない

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: too many indices for array

配列の演算①

```
>>> a = np.arange(6).reshape(2,3)
```

```
>>> a
```

```
array([[0, 1, 2],  
       [3, 4, 5]])
```

0～5の要素を持つ一次元配列を作成
→ 大きさ(2,3)の二次元配列に変形

```
>>> b = a+1
```

各要素に1を加算

```
>>> b
```

```
array([[1, 2, 3],  
       [4, 5, 6]])
```

```
>>> b = a * 2
```

各要素を2倍

```
>>> b
```

```
array([[ 0,  2,  4],  
       [ 6,  8, 10]])
```

配列の演算②

```
>>> a = np.arange(6).reshape(2,3)
```

```
>>> a
```

```
array([[0, 1, 2],  
       [3, 4, 5]])
```

0～5の要素を持つ一次元配列を作成
→ 大きさ(2,3)の二次元配列に変形

```
>>> b = a * a
```

各要素ごとに掛け算

```
>>> b
```

```
array([[ 0,  1,  4],  
       [ 9, 16, 25]])
```

```
c = np.sqrt( a )
```

各要素ごとに平方根

```
>>> c
```

```
array([[0.          , 1.          , 1.41421356],  
       [1.73205081, 2.          , 2.23606798]])
```

配列の演算③

```
>>> a = np.arange(6).reshape(2,3)
```

```
>>> a
```

```
array([[0, 1, 2],  
       [3, 4, 5]])
```

```
>>> np.sum(a)
```

sum(配列名)
要素の和

```
15
```

```
>>> np.mean(a)
```

mean(配列名)
要素の平均

```
2.5
```

```
>>> np.mean(a, axis=0)
```

mean(配列名, axis=0)
列方向(axis=0)の平均

```
array([1.5, 2.5, 3.5])
```

```
>>> np.mean(a, axis=1)
```

mean(配列名, axis=1)
行方向(axis=1)の平均

```
array([1., 4.])
```

0~5の要素を持つ一次元配列を作成
→ 大きさ(2,3)の二次元配列に変形

```
>>> a = np.arange(18).reshape(3,3,2)
```

```
>>> a
```

```
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],
       [[ 6,  7],
        [ 8,  9],
        [10, 11]],
       [[12, 13],
        [14, 15],
        [16, 17]]])
```

0~17の要素を持つ一次元配列を作成
→ 大きさ(3,3,2)の三次元配列に変形

```
>>> np.mean( a[0] , axis=0 )
```

a[0]での列(axis=0)の平均

```
array([2., 3.])
```

```
>>> np.mean( a[0] , axis=1 )
```

a[0]での行(axis=1)の平均

```
array([0.5, 2.5, 4.5])
```

```
>>> a = np.arange(6).reshape(2,3)
```

0~5の要素を持つ一次元配列を作成
→ 大きさ(2,3)の二次元配列に変形

```
>>> a
```

```
array([[0, 1, 2],  
       [3, 4, 5]])
```

```
>>> b = np.arange(6,12).reshape(2,3)
```

6~11の要素を持つ一次元配列を作成
→ 大きさ(2,3)の二次元配列に変形

```
>>> b
```

```
array([[ 6,  7,  8],  
       [ 9, 10, 11]])
```

```
>>> a+b
```

要素ごとの足し算

```
array([[ 6,  8, 10],  
       [12, 14, 16]])
```

```
>>> a-b
```

要素ごとの掛け算

```
array([[ -6, -6, -6],  
       [-6, -6, -6]])
```

```
>>> a*b
```

要素ごとの掛け算(行列の積ではない)

```
array([[ 0,  7, 16],  
       [27, 40, 55]])
```


行列演算①

前のページの続き

配列を行列をみなして計算

```
>>> a.T  
array([[0, 3],  
       [1, 4],  
       [2, 5]])  
>>> c = np.dot( a.T , b )  
>>> c  
array([[27, 30, 33],  
       [42, 47, 52],  
       [57, 64, 71]])
```

行列の転置

行列の積

転置
配列名.T

積
dot(配列1, 配列2) もしくは
配列1.dot(配列2)

```
>>> c = a.T  
>>> c.dot(b)  
array([[27, 30, 33],  
       [42, 47, 52],  
       [57, 64, 71]])
```

行列の積

```
>>> a = np.array([[0,4,2],[-1,5,2],[1,4,9]])
```

大きさ(3,3)の二次元配列

```
>>> a
```

```
array([[ 0,  4,  2],  
       [-1,  5,  2],  
       [ 1,  4,  9]])
```

`linalg.det(配列名)`
行列式の計算

```
>>> np.linalg.det(a)
```

```
26.000000000000004
```

`linalg.inv(配列名)`
逆行列の計算

```
>>> b = np.linalg.inv(a)
```

```
>>> b
```

```
array([[ 1.42307692, -1.07692308, -0.07692308],  
       [ 0.42307692, -0.07692308, -0.07692308],  
       [-0.34615385,  0.15384615,  0.15384615]])
```

単位行列になることの確認

```
>>> np.dot( a , b )
```

```
array([[ 1.00000000e+00, -5.55111512e-17, -5.55111512e-17],  
       [ 1.11022302e-16,  1.00000000e+00,  0.00000000e+00],  
       [ 0.00000000e+00, -2.22044605e-16,  1.00000000e+00]])
```

行列演算②

```
>>> I = np.identity(3)
```

identity(大きさ)

→ 大きさ(3,3)の単位行列の作成

```
>>> I
```

```
array([[1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])
```

linalg.solve(A,B)

AX=Bの解Xを求める

```
>>> c = np.linalg.solve(a,I)
```

aX=Iの解を求める

```
>>> c
```

```
array([[ 1.42307692, -1.07692308, -0.07692308],  
       [ 0.42307692, -0.07692308, -0.07692308],  
       [-0.34615385,  0.15384615,  0.15384615]])
```

行列演算③

```
>>> a = np.array([[0,4,2],[-1,5,2],[1,4,9]])
```

← 大きさ(3,3)の二次元配列

```
>>>
```

```
>>> lamda , v = np.linalg.eig(a)
```

← `linalg.eig`(配列名)

固有値ベクトル, 固有行列を求める

```
>>> lamda
```

```
array([ 1.          , 2.46887113, 10.53112887])
```

← 固有値ベクトル(lamda)

← 固有行列(v)

```
>>> v
```

```
array([[ -0.88225755,  0.62183946,  0.28100631],  
       [ -0.36760731,  0.62183946,  0.28100631],  
       [ 0.29408585, -0.47605817,  0.91764422]])
```