クレジット:
UTokyo Online Educationデータマイニング入門 2018 森　純一郎

東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

データマイニング入門

Pythonの基礎

NumPyライブラリ

NumPyライブラリを用いることにより、Python標準のリストよりも効率的に多次元の配列を扱うことができます。これにより高速な行列演算が可能になるため、行列演算を行う科学技術計算などでよく活用されています。以下では、NumPyライブラリの配列の基本的な操作や機能を説明します。

配列の作成

NumPyライブラリを使用するには、まず `numpy` モジュールをインポートします。慣例として、同モジュールを `np` と別名をつけてコードの中で使用します。

In [ ]:
```
import numpy as np
```

NumPyの**配列**は `numpy` モジュールの **`array`** `()` 関数で作ります。配列の要素はPython標準のリストやタプルで指定します。

In [ ]:
```
# リストから配列作成
list_to_array = np.array([1,2,3,4,5])
list_to_array
```

NumPyの配列は `ndarray` オブジェクトとなります。

In [ ]:
```
# タプルからの配列作成
tuple_to_array = np.array((1,2,3,4,5))

# 配列の型
type(tuple_to_array)
```

`print()` 関数を使って配列を出力すると、要素が空白で区切られて出力されます。

In [ ]:
```
# 配列のprint
print(list_to_array)
print(tuple_to_array)
```

`array()` 関数では、第2引数 `dtype` で配列の要素の型を指定することができます。NumPyの配列はリストと異なり、要素の型を混在させることはできません。配列の要素の型は `dtype` 属性で調べることができます。

In [ ]:
```
# 配列要素の型の指定
list_to_array = np.array([1,2,3,4,5], dtype=float)

# 配列要素の型の確認
list_to_array.dtype
```

配列を `array()` 関数に渡し、第2引数dtypeで型を指定すると、既存の配列を別の型に変換した配列を作成できます。

In [ ]:
```
int_array = np.array([1,2,3,4,5])

# 配列要素の型の変換
float_array = np.array(int_array, dtype=float)
float_array
```

多次元配列の作成

多次元配列は、配列の中に配列がある入れ子の配列です。NumPyは多次元配列を効率的に扱うことができます。NumPyでは、 `array()` 関数の引数にリストが入れ子になった多重リストを与えると多次元配列が作成できます。

`shape` 属性で、配列が何行何列かを調べることができます。また、 **`ndim`** 属性で、何次元の配列かを調べることができます。 **`size`** 属性では、配列の要素の個数を調べることができます。

In [ ]:
```
# 多次元配列の作成
mul_array = np.array([[1,2,3],[4,5,6]])
print(mul_array)

# 多次元配列の行数と列数
print(mul_array.shape)

# 多次元配列の次元数
print(mul_array.ndim)

# 多次元配列の要素数
print(mul_array.size)
```

reshape () メソッドを使うと、 reshape (行数、列数）と指定して、1次元配列を多次元配列に変換することができます。 reshape() で変換した多次元配列の操作の結果は元の配列にも反映されることに注意してください。

ravel () メソッドまたは **flatten** () メソッドを使うと、多次元配列を1次元配列に戻すことができます。

In [ ]:
```
mydata = [1,2,3,4,5,6]
a1 = np.array(mydata)

# 2行3列の多次元配列に変換
a2 = a1.reshape(2,3)
print(a1)
print(a2)

# 1行1列の要素に代入（後述）
a2[0,0]=0
print(a1)
print(a2)

# 多次元配列を1次元配列に戻す
print(a2.ravel())
```

練習

以下のリストから1次元配列を作成し、その配列から3行3列の多次元配列を作成してください。作成した配列のshape属性を確認してください。

In [ ]:
```
mylist=[1,2,3,4,5,6,7,8,9]
```

様々な配列の作り方

zeros()関数

zeros () 関数を用いると、すべての要素が0の配列を作成することができます。 zero() 関数の第1引数には0の個数を（多次元配列の場合は行数と列数をタプルで）指定し、第2引数の dtype に数値の型を指定します。

In [ ]:
```
# 5つの0要素からなる配列
zero_array1=np.zeros(5, dtype=int)
print(zero_array1)

# 3行3列の0要素からなる多次元配列
zero_array2=np.zeros((3,3), dtype=int)
print(zero_array2)
```

ones()関数

ones () 関数を用いると、すべての要素が1の配列を作成することができます。 ones() 関数の第1引数には1の個数を（多次元配列の場合は行数と列数をタプルで）指定し、第2引数の dtype に数値の型を指定します。

In [ ]:
```
# 2行2列の1要素からなる多次元配列
one_array=np.ones((2,2), dtype=int)
print(one_array)
```

arange()関数

arange () 関数を用いると、開始値から一定の刻み幅で生成した値の要素からなる配列を作成できます。 arange() 関数の第1引数には開始値、第2引数には終了値、第3引数には刻み幅を指定します。終了値は生成される値に含まれないことに注意してください。開始値を省略すると0、刻み幅を省略すると1がそれぞれ初期値となります。 arange() 関数では dtype で数値の型も指定できますが、省略すると開始値、終了値、刻み幅に合わせて型が選ばれます。

numpy.arange(開始値、終了値、刻み幅、dtype=型)

In [ ]:
```
# 0から1刻みで5つの要素を持つ配列
a1=np.arange(5)
print(a1)

# 0から0.1刻みで1未満の値の要素を持つ配列
a2=np.arange(0,1,0.1)
print(a2)

# 0から1刻みで4つの要素を持つ配列を2行2列の多次元配列に変換
a3=np.arange(2*2).reshape(2,2)
print(a3)
```

linspace()関数

linspace () 関数を用いると、分割数を指定することで値の範囲を等間隔で分割した値の要素からなる配列を作成できます。 linspace() 関数の第1引数には開始値、第2引数には終了値、第3引数には分割数を指定します。

In [ ]:
```
# 0から100の値を11分割した値を要素に持つ配列
a=np.linspace(0,100,11)
print(a)
```

random.rand()関数

`random.rand` `()` 関数を用いると、乱数の配列を作成することができます。 `random.rand()` 関数では、引数で与えた個数の乱数が0から1の間の値で生成されます。この他にも、 `random.randn()` 関数、 `random.binomial()` 関数、 `random.poisson()` 関数を用いると、それぞれ正規分布、二項分布、ポアソン分布から乱数の配列を作成することができます。

- random.*関数 (https://docs.scipy.org/doc/numpy/reference/routines.random.html)

In [ ]:
```
# 5つのランダムな値の要素からなる多次元配列
rand_array=np.random.rand(5)
print(rand_array)

# 平均mu,分散sigmaの正規分布に従う値の要素からなる多次元配列
mu, sigma = 0, 1
rand_normal_array=sigma*np.random.rand(10)+mu
print(rand_normal_array)
```

練習

9つのランダムな値の要素から3行3列の多次元配列を作成してください。

In [ ]:
```
```

csvファイルからの配列の作成

`loadtxt()` 関数を用いて、以下のように**csvファイル**を読み込んで、配列を作成することができます。 `loadtxt()` 関数の `delimiter` 引数には区切り文字を指定します。また、 `skiprows` 引数を `=1` とすることで、csvファイルの先頭行（ヘッダ）を飛ばすように指定します（配列は異なる型の要素を混在できないため）。

```
## simple_score.csv
user, kokugo, shakai, sugaku, rika
1, 30, 43, 51, 63
2, 39, 21, 49, 56
...
```

In [ ]:
```
score = np.loadtxt("simple_score.csv", delimiter=",", skiprows=1)
score
```

配列要素の操作

インデックス

NumPyの配列の要素には、リストと同様に0から始まる**インデックス**を使ってアクセスします。リストと同じく、配列の先頭要素のインデックスは0、最後の要素のインデックスは-1となります。

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

# 配列aのインデックス0の要素
print(a[0])

# 配列aのインデックス-1(終端）の要素
print(a[-1])

# 配列aのインデックス-1の要素に代入
a[-1]=0
print(a)
```

多次元配列では、 `array[行,列]` のように行と列で要素にアクセスできます。この時、行と列はインデックスと同じくそれぞれ0から始まります。また、多次元リストと同様に、 `array[インデックス][インデックス]` のようにリストごとのインデックスを使っても要素にアクセスできます。

In [ ]:
```
a = np.array([[1,2,3],[4,5,6]])
print(a)

# 0行1列の要素
print(a[1,2])

# 0行1列の要素に代入
a[1,2]=0

# 0行1列の要素
print(a[1][2])
```

練習

csvファイル、simple_score.csv、から配列を作成し、3行4列の要素を抽出してください。

In [ ]:
```

```

スライス

リストと同様に、NumPyの配列でも、 `array[開始位置:終了位置:ステップ]` のように**スライス**を用いて配列の要素を抜き出すことができます。リストと同じく、スライスの開始位置や終了位置は省略が可能です。

In [ ]:
```
a = np.array([1,10,100,1000,10000])

# 配列aのインデックス1からインデックス3までの要素をスライス
print(a[1:4])

# 配列aのインデックス1から終端までの要素をスライス
print(a[1:])

# 配列aの先頭から終端から3番目までの要素をスライス
print(a[:-2])

# 配列aの先頭から1つ飛ばしで要素をスライス
print(a[::2])

# 配列aの終端から先頭までの要素をスライス
print(a[::-1])
```

NumPyの配列では、配列からスライスで抜き出した要素に値をまとめて代入することができます。配列においてスライスに対する変更は元の配列にも反映されることに注意してください。

In [ ]:
```
a = np.array([1,10,100,1000,10000])

# 配列aのインデックス1からインデックス3までの要素に0を代入
a[1:4]=0
print(a)
```

多次元配列のスライスでは、 `array[行のスライス, 列のスライス]` のように行と列のスライスのそれぞれの指定をカンマで区切って指定します。

In [ ]:
```
a = np.array( [1,2,3,4,5,6,7,8,9]).reshape(3,3)
print(a)

# 多次元配列aの先頭行から2行目、先頭列から2列目までの要素をスライス
print(a[:2,:2])

# 多次元配列aの2行目から終端行、2列目から終端列までの要素をスライス
print(a[1:,1:])
```

練習

csvファイル、simple_score.csv、から配列を作成し、1列目を除いた全ての列を抽出してください。

In [ ]:
```
```

要素の順序取り出し

リストと同様に、 `for...in` 文を用いて、配列の要素を順番に取り出すことができます。 **`enumerate ()`** 関数を使うと、リストと同じく、取り出しの繰り返し回数も併せて数えることができますが、多次元配列の要素の取り出しでは `enumerate` 関数の代わりに `ndenumerate()` 関数を用います。 `ndenumerate()` 関数は取り出した要素とともに、その要素の位置を行と列のタプルで返します。

In [ ]:
```
a1 = np.array([1,2,3,4,5,6])

# 配列a1から要素の取り出し
for num in a1:
    print(num)

a2 = np.array([1,2,3,4,5,6]).reshape(2,3)

# 多次元配列a2から行の要素と繰り返し回数の取り出し
for i, num in enumerate(a2):
    print(i, num)

# 多次元配列a2から要素の取り出し、要素の位置を行と列のタプルで取得
for i, num in np.ndenumerate(a2):
    print(i, num)
```

練習

csvファイル、simple_score.csv、から配列を作成し、得点の要素の値を取り出してください。

In [ ]:
```
score = np.loadtxt("simple_score.csv", delimiter=",", skiprows=1)
score=score[:,1:]
```

要素の並び替え

配列の要素の並び替えには、 ndarray オブジェクトの **sort ()** メソッドまたはNumPyライブラリの sort() 関数を使います。 sort() メソッドは、メソッドを呼び出した自身の配列の要素を並び替えますが、 sort() 関数は引数で与えた配列の要素を並び替えた新しい配列を返します。 sort() 関数の引数にリストやタプルを指定し、それらの並び替えを行った結果を配列として取得することもできます。

In [ ]:
```
a1 = np.array([5,3,1,4,2])
# 配列a1の要素を並び替え
a1.sort()
print(a1)

a2 = np.array([5,3,1,4,2])
# 配列a2の要素を並び替えた結果から新たな配列a3を作成
a3 = np.sort(a2)

print(a2)
print(a3)
```

要素の条件取り出し

条件式を用いて、配列の要素の中から条件に合う要素のみを抽出し、要素の値を変更したり、新たな配列を作成することができます。配列と比較演算を組み合わせることで、比較演算が配列の個々の要素に適用されます。

条件式のブール演算では、 and 、 or 、 not の代わりに & 、 | 、 ~ を用います。

In [ ]:
```
a1 = np.array([1,2,-3,-4,5,-6,-7])

# 0未満で2で割り切れる値を持つ要素に0を代入
a1[(a1<0) & (a1%2==0)]=0
print(a1)
```

以下の例において、 print(a1>0) とすると [True True False False True False False] というブール値の配列が返ってきていることがわかります。 True は条件（この場合は要素が正）に対して真な要素（この場合は1,2,5）に対応しています。配列要素の条件取り出しでは、このブール値の配列を元の配列に渡して、条件に対して真な要素のインデックスを参照していることになります。これを**ブールインデックス参照**と呼びます。

In [ ]:
```
a1 = np.array([1,2,-3,-4,5,-6,-7])

# 0より大きい値を持つ要素はTrue, それ以外はFalseのブール値配列
print(a1>0)

# 配列a1の0より大きい値を持つ要素から新たな配列a2の作成
a2 = a1[a1>0]
print(a2)
```

練習

csvファイル、simple_score.csv、から配列を作成し、50点以上の得点の要素の値を取り出してください。

In [ ]:
```
score = np.loadtxt("simple_score.csv", delimiter=",", skiprows=1)
score=score[:,1:]
```

配列の演算

NumPyの配列では、配列のすべての要素に数値演算を適用する**ブロードキャスト**という機能により、要素が数値である配列の演算を簡単に行うことができます。

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

# 配列aのすべての要素に1を加算
b = a+1
print(b)

# 配列bのすべての要素に2を乗算
c=b*2
print(c)

# 配列cのすべての要素に2を除算
d=c/2
print(d)
```

この他に、NumPyには**ユニバーサル関数**と呼ばれる、配列を入力としてのそのすべての要素を操作した結果を配列として返す関数が複数あります。ユニバーサル関数については以下を参照してください。

- ユニバーサル関数の一覧 (https://docs.scipy.org/doc/numpy-1.14.0/reference/ufuncs.html#available-ufuncs)

練習

csvファイル、simple_score.csv、から配列を作成し、得点の各要素の値を2乗にしてください。

In [ ]:
```
score = np.loadtxt("simple_score.csv", delimiter=",", skiprows=1)
score=score[:,1:]
```

ndarray オブジェクトのメソッドを用いて、要素の合計、平均値、最大値、最小値を、それぞれ sum() 、 mean() 、 max() 、 min() で求めることができます。各メソッドは引数を指定しなければ配列のすべての要素に適用されます。多次元配列の場合、引数に0を指定すると、各列にメソッドを適用した結果の配列、引数に1を指定すると各行にメソッドを適用した結果の配列が返ります。

In [ ]:
```
score = np.loadtxt("simple_score.csv", delimiter=",", skiprows=1)
score=score[:,1:] # 得点の列だけ抽出

# 多次元配列scoreのすべての要素の平均
print(score.mean())

# 多次元配列aの各列の要素の平均
print(score.mean(0))

# 多次元配列aの各行の要素の平均
print(score.mean(1))
```

この他のNumPyの数学・統計関連のメソッド・関数については以下を参照してください。

- 数学関数 (https://docs.scipy.org/doc/numpy/reference/routines.math.html)
- 統計関数 (https://docs.scipy.org/doc/numpy/reference/routines.statistics.html)

練習

csvファイル、simple_score.csv、から配列を作成し、得点の各列の最大値と最小値を求めてください。

In [ ]:
```
score = np.loadtxt("simple_score.csv", delimiter=",", skiprows=1)
score=score[:,1:]
```

配列同士の演算

行数と列数が同じ配列同士の四則演算は、各要素同士の演算となります。

In [ ]:
```
A = np.array([1,2,3,4]).reshape(2,2)
B = np.array([2,4,6,8]).reshape(2,2)

# 配列の要素同士の足し算
C = A+B
print(C)

# 配列の要素同士の引き算
D = B-A
print(D)

# 配列の要素同士の掛け算
E = A*B
print(E)

# 配列の要素同士の割り算
F = B//A
print(F)
```

練習

csvファイル、simple_score.csv、から配列を作成し、得点の要素同士を掛け算して得点の各要素の値を2乗にしてください。

In [ ]:
```
score = np.loadtxt("simple_score.csv", delimiter=",", skiprows=1)
score=score[:,1:]
```

ブロードキャスト

行数と列数が異なる配列や行列同士の四則演算では、足りない行や列の値を補うブロードキャストが行われます。以下の例では、配列Aと演算に対して、配列 `B` の2行目が足りないため、 `B` の1行目と同じ値で2行目を補い演算を行なっています。このようなブロードキャストが機能するのは、 `B` の行数または列数が `A` のそれらと同じ場合、または配列 `B` が1行・1列の場合です。

In [ ]:
```
A = np.array([1,2,3,4]).reshape(2,2)
B = np.array([2,4])

# 行列Bをブロードキャストして行列Aと足し算
C = A+B
print(C)
```

行列の演算

`dot()` 関数を使うとベクトルの**内積**や**行列積**を計算することができます。この時、それぞれの配列の行数と列数、または列数と行数が同じである必要があります。

In [ ]:
```
A = np.array([1,2,3,4]).reshape(2,2)
B = np.array([2,4,6,8]).reshape(2,2)

# 行列積
C = np.dot(A,B)
print(C)
```

単位行列は `identity()` 関数または `eye()` 関数で作成することができます。引数に行列のサイズを指定します。

In [ ]:
```
# 3行3列の単位行列
E=np.identity(3, dtype=int)
print(E)
```

`transpose()` 関数または配列の `T` 属性で、配列の行と列の要素を入れ替えた配列を得ることができます。この時、元の配列の形状を変えているだけで元の配列を直接変更していないことに注意してください。

In [ ]:
```
A = np.array([1,2,3,4,5,6]).reshape(2,3)

# 配列の行と列の入れ替え
print(np.transpose(A))
print(A.T)
print(A)
```

NumPyでは、行列の分解、転置、行列式などの計算を含む線形代数の機能は、 `numpy.linalg` モジュールで提供されています。同モジュールについては以下を参照してください。

- 線形代数関連関数 (https://docs.scipy.org/doc/numpy/reference/routines.linalg.html)

練習

csvファイル、simple_score.csv、から配列を作成し、各行を得点の各列を値を要素とするベクトルとみなし、1行目との他の行との内積をそれぞれ計算してください。

In [ ]:
```
score = np.loadtxt("simple_score.csv", delimiter=",", skiprows=1)
score=score[:,1:]
```

配列要素の追加、挿入、削除

append()関数

NumPyの配列の要素の追加には **append** () 関数を使います。 append() 関数の第1引数には配列を指定し、第2引数にはその配列に追加する値を指定します。リストやタプルで複数の値を同時に指定することもできます。NumPyの append() 関数は、要素を追加した新しい配列が返り、元の配列は変化しないことに注意してください。

In [ ]:
```
a1 = np.array([1,2])

# 配列a1に値3を要素として追加
a2 = np.append(a1, 3)

# 配列a2に値4,5を要素として追加
a3 = np.append(a2,[4,5])

print(a1)
print(a2)
print(a3)
```

多次元配列に要素を追加する場合は、 append 関数の axis 引数に対して、行追加であれば0、列追加であれば1を渡します。追加する要素は、追加先の配列の行または列と同じ次元の配列である必要があります。

In [ ]:
```
mul_array1 = np.array([[1,2,3],[4,5,6]])

#  多次元配列mul_array1に行[7,8,9]を追加
mul_array2 = np.append(mul_array1, [[7,8,9]], axis =0)

#  多次元配列mul_array2に列[0,0,0]を追加
mul_array3 = np.append(mul_array2, np.array([[0,0,0]]).T, axis =1)

print(mul_array1)
print(mul_array2)
print(mul_array3)
```

insert()関数

NumPyの配列の要素の挿入には **`insert ()`** 関数を使います。 `insert()` 関数の第1引数には配列、第2引数には要素を挿入する位置、第3引数にはその配列に追加する値を指定します。値は、リストやタプルで複数を同時に指定することもできます。NumPyの `insert()` 関数は、要素を追加した新しい配列が返り、元の配列は変化しないことに注意してください。

In [ ]:
```
a1 = np.array([1,3])

# 配列a1のインデックス1に値2を要素として追加
a2 = np.insert(a1, 1, 2)

# 配列a2のインデックス3に値4,5を要素として追加
a3 = np.insert(a2, 3, [4,5])

print(a1)
print(a2)
print(a3)
```

多次元配列に要素を挿入する場合は、axis引数に対して、行追加であれば0、列追加であれば1を渡します。挿入する要素は、挿入先の配列の行または列と同じ次元の配列である必要があります。

In [ ]:
```
mul_array1 = np.array([[1,2,3],[7,8,9]])

#  多次元配列mul_array1に行[4,5,6]を追加
mul_array2 = np.insert(mul_array1, 1, [[4,5,6]], axis =0)

print(mul_array1)
print(mul_array2)
```

delete()関数

NumPyの配列の要素の削除には **`delete ()`** 関数を使います。 `delete()` 関数の第1引数には配列、第2引数には削除する要素の位置を指定します。 `delete` 関数でも `append()` 関数、 `insert()` 関数と同様に、元の配列は変化しないことに注意してください。

In [ ]:
```
a1 = np.array([0,1,2])

# 配列a1のインデックス2の要素を削除
a2 = np.delete(a1,2)

print(a1)
print(a2)
```