Guia introdutória ao NumPy

Teoria da Informação - LEI

Introdução

Neste documento são apresentadas algumas funções de uso frequente ao trabalhar com o módulo NumPy do Python.

O NumPy trabalha com objetos do tipo *array* (matrizes). O trabalho com *arrays* permite operações com grandes quantidades de elementos de forma mais rápida e eficiente que com o uso de listas.

Uma descrição mais extensa e detalhada das funções do NumPy pode ser consultada no seguinte link: https://numpy.org/doc/stable/user/index.html#user

Os passos para instalação do NumPy podem ser encontrados em: https://numpy.org/install/

1) Importação do NumPy

Comumente ao importar o NumPy atribui-se o acrônimo "np"

```
import numpy as np
```

2) Formas de inicializar um *array*

i. Criar um *array* a partir de uma lista de elementos (função ***array***).

```
A=np.array([1, 2, 4, 8, 16])

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

ii. Criar um array a partir de uma lista previamente definida.

```
a=[1, 2, 4, 8, 16]
A=np.array(a)
```

iii. Criar um *array* indicando o valor inicial, o valor final e o incremento (função ***arange***).

```
A=np.arange(10, 100, 10)

Out:
array([10, 20, 30, 40, 50, 60, 70, 80, 90])
```

O valor final indicado não é incluído no *array*.

iv. Criar um *array* de zeros ou uns (funções ***zeros*** ou ***ones***).

```
A=np.zeros(5)

Out:
array([0., 0., 0., 0., 0.])
```

Podem ser criados *arrays* de uma ou mais dimensões.

```
A=np.ones([4,2])
Out:
array([[1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.]])
```

Também é possível criar *arrays* de zeros ou uns com as mesmas dimensões de *arrays* existentes (funções ***zeros_like*** ou ***ones_like***).

```
A=np.arange(1, 10, 2)
Out:
array([1, 3, 5, 7, 9])

A0=np.zeros_like(A)
Out:
array([0, 0, 0, 0, 0])
```

v. Criar um *array* de valores aleatórios

A função ***np.random.rand*** irá gerar valores aleatórios no intervalo [0, 1), com as dimensões do *array* especificadas como argumentos de entrada.

```
A=np.random.rand(3,2,4)                    # Array 3D

Out:
array([[[0.53561516, 0.42436643, 0.88596225, 0.34087257],
        [0.07481607, 0.38757027, 0.5925532 , 0.74853932]],

       [[0.48690417, 0.94873153, 0.49770214, 0.3775186 ],
        [0.60683769, 0.35918078, 0.91038508, 0.99875218]],

       [[0.19949367, 0.35299822, 0.1503897 , 0.42147179],
        [0.44092914, 0.32448023, 0.68568386, 0.55258676]]])
```

A função ***np.random.randint*** permite criar um array de valores aleatórios inteiros, onde os valores resultantes estarão compreendidos no intervalo [low, high).

```
A=np.random.randint(low=10, high=20, size=(3,5))

Out:
array([[15, 16, 17, 11, 19],
       [12, 14, 15, 19, 15],
       [14, 15, 17, 10, 10]])
```

Existem também diversas funções que permitem criar *arrays* de números aleatórios seguindo uma distribuição específica.
A função ***np.random.normal*** permite criar valores aleatórios com distribuição normal, podendo-se especificar a média (1º argumento de entrada) e o desvio padrão (2º argumento de entrada).

```
A=np.random.normal(1.0, 3.0, size=(3,2))
```

vi. Especificar o tipo de dados ao criar um *array* (argumento "***dtype=***").

```
A1=np.ones_like(A, dtype=float)
Out:
array([1., 1., 1., 1., 1.])

A=np.zeros(5, dtype=int)
Out:
array([0, 0, 0, 0, 0])
```

Num *array*, todos os elementos devem ser do mesmo tipo.

vii. Criar cópias de *arrays*.

```
A0=np.zeros(5, dtype=int)

A1=A0

A2=A0.copy()
```

3) Converter um *array*

i. Alterar as dimensões de um array (funções ***reshape***, ***transpose***, ***flatten***).

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

A=A.reshape((3,2))            # 2D
Out:
 array([[1, 2],
      [3, 4],
      [5, 6]])

A=A.transpose()               # Transposto
Out:
 array([[1, 3, 5],
      [2, 4, 6]])

A=A.flatten()                 # 1D
Out:
array([1, 3, 5, 2, 4, 6])
```

ii. Concatenar *arrays* (função ***concatenate***).

```
A0=np.zeros([2,2])
A1=np.ones([2,2])

np.concatenate((A0, A1), axis=0)
Out:
array([[0., 0.],
```

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

np.concatenate((A0, A1), axis=1)
Out:
array([[0., 0., 1., 1.],
    [0., 0., 1., 1.]])
```

iii. Converter um *array* numa lista (função ***tolist*).**

```
A=np.ones(6)
Out:
array([1., 1., 1., 1., 1., 1.])

A.tolist()
Out:
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

iv. Mudar o tipo dos dados contidos num *array* (função ***astype***).

```
 A=np.array([1.1, 2.2, 4.4, 8.8, 16.0], dtype=float)
Out:
array([ 1.1,  2.2,  4.4,  8.8, 16. ])

A=A.astype(np.uint)
Out:
array([ 1,  2,  4,  8, 16], dtype=uint64)
```

4) Extrair elementos de um *array*

i. Selecionar elementos a partir dos índices.

```
A=np.array([[1, 2, 3, 4], [11, 12, 13, 14]])

A[1,0]                  # Elemento da linha um coluna zero
Out:
11

A[:,1]                   # Seleciona toas as linhas, coluna um
Out:
array([ 2, 12])

A[:,:2]                 # Seleciona todas as linhas, e colunas anteriores à segunda
# Resposta:
array([[ 1,  2,  3],
    [11, 12, 13]])

A[0,-1]                 # Último elemento da linha zero
Out:
4

A[1,-2]                 # Penúltimo elemento da linha 1
Out:
13
```

Do mesmo modo que com o uso de listas, a indexação num *array* começa em 0.

ii. Selecionar elementos pelos seus valores

```
A=np.array([[1, 2, 3, 4], [11, 12, 13, 14]])

A[A>2]
Out:
array([ 3,  4, 11, 12, 13, 14])
```

5) Operações matemáticas e estatísticas.

i. Operações básicas, elemento a elemento.

```
A=np.array([[1, 2], [3, 4]])
B=np.array([[5, 6], [7, 0]])

A+B
Out:
array([[ 6, 8],
       [ 10,  4]])
A-B
Out:
array([[-4, -4],
    [-4,  4]])

A*B
Out:
array([[ 5, 12],
    [21,  0]])

A/B
Out:
array([[0.2      , 0.33333333],
    [0.42857143,       inf]])

B%A                          # resto da divisão
Out:
array([[0, 0],
    [1, 0]])

B**A                         # exponenciação
Out:
array([[  5,  36],
    [343,   0]])
```

Se os *arrays* não são do mesmo tamanho, mas são de tamanhos compatíveis, o menor *array* será expandido ao tamanho do *array* maior para realizar a operação elemento a elemento.

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

A+B
Out:
 array([[ 6,  8],
    [ 8, 10]])
```

Se os arrays não tiverem tamanhos compatíveis, aparecerá uma mensagem de erro.

ii. Funções que operam elemento a elemento.

Funções: abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, etc.

```
A=np.array([5, 6])
```

```
np.sqrt(A)
Out:
array([2.23606798, 2.44948974])
```

iii. Operações entre os elementos de um *array*.

Algumas funções:

```
np.sum(A)                # soma
np.prod(A)               # produto
A.mean()                 # média
A.var()                  # variância
A.std()                  # desvio padrão
A.min()                  # valor mínimo
A.max()                  # valor máximo
A.argmin()               # índice correspondente ao valor mínimo
A.argmax()               # índice correspondente ao valor máximo
np.corrcoef(A)           # coeficiente de correlação
np.cov(A)                # covariância
```

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

np.sum(A)
Out:
21

A.sum(axis=0)
Out:
array([5, 7, 9])

A.max()
Out:
6

A.argmax()
Out:
5
```

Em *arrays* 2D o argumento "axis=1" ou "axis=2" permite indicar em qual dos eixo será realizada a operação.

6) Transformação dos elementos de um array.

Algumas funções:

```
A.sort()                 # ordena os elementos do array de menor a maior
A.clip(min,max)          # converte os elementos a valores compreendidos entre min e max
np.where(cond, if V, if F) # aplica transformações a cada valor em função da condição ser V ou F
np.floor(A)              # aproxima cada elemento do array ao inteiro inferior
np.ceil(A)               # aproxima cada elemento do array ao inteiro superior
np.rint(A)               # aproxima cada elemento do array ao inteiro mais próximo
np.unique(A)             # cada valor aparece uma única vez
A.fill()                 # substitui todos os elementos pelo valor indicado
```

```
A=np.arange(20,1,-3)
Out:
array([20, 17, 14, 11,  8,  5,  2])

A.sort()
Out:
```

```
array([ 2,  5,  8, 11, 14, 17, 20])

A=A.clip(6, 15)
Out:
array([ 6,  6,  8, 11, 14, 15, 15])

A=np.where(A>12, A/2, A+1)
Out:
array([ 7. ,  7. ,  9. , 12. ,  7. ,  7.5,  7.5])

A[3]=15
Out:
array([ 7. ,  7. ,  9. , 15. ,  7. ,  7.5,  7.5])

A=np.unique(A)
Out:
array([ 7. ,  7.5,  9. , 15. ])

A=np.ceil(A)
Out:
array([ 7.,  8.,  9., 15.])

A.fill(3)
Out:
array([3., 3., 3., 3.])

np.append(A, [4, 4], axis=0)
Out:
 array([3., 3., 3., 3., 4., 4.])
```

7) Operações matriciais.

Algumas funções:

```
np.dot()            # produto matricial
np.transpose()      # transposta
np.linalg.inv()     # inversa
np.linalg.det()     # determinante
```

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

np.dot(A,B)
Out:
array([[14, 14],
       [32, 32]])
```

8) Comparações envolvendo *arrays*, elemento a elemento.

Operadores <, <=, >, >=, == e !=

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

A>B
Out:
array([False,  True, False, False])

A==B
```

```
Out:
array([ True, False, False,  True])

A!=B
Out:
array([False,  True,  True, False])

A>2
Out:
array([False, False,  True,  True])
```

Operadores lógicos: logical_and, logical_or, logical_not, logical_xor

```
A=np.array([True, True, False], bool)
B=np.array([True, False, False], bool)

np.logical_and(A,B)
Out:
array([ True, False, False])
```

9) Examinar elementos num *array*.

```
A.nonzero()          # indica os índices dos elementos diferentes de zero
np.isnan(A)          # retorna um array booleano indicando para cada elemento se é NaN
                     (True) ou não (False)
np.isfinite(A)       #  retorna um array booleano indicando para cada elemento se é finito
                     (True) ou não (False)
np.isinf(A)          #  retorna um array booleano indicando para cada elemento se é infinito
                     (True) ou não (False)
any(A)               # True se algum elemento é verdadeiro
all(A)               # True se todos os elementos são verdadeiros
np.isin(_ , A)       # True se o array contém determinado elemento
```

```
A=np.array([np.pi, 0, np.inf, np.nan, np.e])
Out:
array([3.14159265, 0. , inf, nan, 2.71828183])

np.isinf(A)
Out:
array([False, False,  True, False, False])

np.isnan(A)
Out:
array([False, False, False,  True, False])

A.nonzero()
Out:
(array([0, 2, 3, 4]),)

np.isfinite(A)
Out:
array([ True,  True, False, False,  True])

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

np.isin(3, A)
Out:
array(True)
```

```
3 in A
Out:
True

7 in A
Out:
False

A=np.array([True, True, False], bool)

any(A)
Out:
True

all(A)
Out:
False
```

10) Iterações com *arrays*.

```
A=np.array([1, 2, 3, 4])  # 1D

x=0
for a in A:
     x=x+a

Out:
10

A=np.array([[1, 2],[ 3, 4]])   #2D

for a in A:
   print(a)

Out:
[1 2]
[3 4]

x=0
for (a, b) in A:
   x=x+a+b

Out:
10
```

11) Propriedades de um *array*

Algumas funções:

A.shape	# tamanho do array
A.dtype	# tipo de dados contidos no array
len(A)	# comprimento do array

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

A.shape
Out:
```

```
(2, 3)

A.dtype
Out:
dtype('int64')

A=np.ones(5)
len(A)
Out:
5               # arrays 1D: comprimento do vetor

A=np.ones([2,3])
len(A)
Out:
2               # arrays 2D: comprimento do primeiro eixo
```