Aula 3

Introdução ao Python - Parte III - NumPy

Introdução

- O *Numerical Python* (*NumPy*) é um pacote para a linguagem Python que suporta *arrays* e matrizes multidimensionais
- Além disso, possui uma série de funções para realizar operações matemáticas envolvendo essas estruturas
- Além das funções, também possui uma série de *sugar sintaxes*, a qual permite usar operadores matemáticas para fazer operações em matrizes e arrays

Introdução

- Outras razões da popularização do `NumPy` são:
  - Sintaxe e funções semelhantes ao *software* proprietário Matlab
  - Velocidades de processamento (2x mais rápido que o processamento em listas)
  - É utilizado em outras bibliotecas famosas como a `OpenCV` (visão computacional), `Keras` (*deep learning*) e o `Scikit-Learn` (aprendizado de máquina)
  - Aproximadamente 450 bibliotecas utilizam o `NumPy`

Importando

- Para utilizar o `NumPy` deve-se importar o pacote `numpy`
- Normalmente utiliza-se o alias `np`
- Portanto:

```
In [1]: import numpy as np
```

Inicializando

- O `NumPy` provê várias funções para criar *arrays*
- A mais comum delas é a função `array`
- Essa função recebe uma lista, lista de listas, e assim por diante, e retorna uma *array* multidimensional da biblioteca `NumPy`

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

In [3]: integers

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

In [4]: floats = np.array([0.0, 0.1, 0.2, 0.3, 0.4])

In [5]: floats

Out[5]: array([0. , 0.1, 0.2, 0.3, 0.4])
```

Inicializando

- Outras formas comuns de inicializar são:

In [2]:
```
#inicializando array com zeros
arr_zeros1 = np.zeros(10)
arr_zeros1
```

Out[2]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [4]:
```
#inicializando array com zeros (bidimensional)
arr_zeros2 = np.zeros((2,5))
arr_zeros2
```

Out[4]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [5]:
```
#inicializando com um range de valores
arr_range = np.arange(10)
arr_range
```

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

In [7]:
```
arr_empty = np.empty(10)
arr_empty
```

Out[7]:
```
array([6.93924935e-310, 6.93924935e-310, 0.00000000e+000, 0.00000000e+000,
       0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
       0.00000000e+000, 0.00000000e+000])
```

In [8]:
```
#inicializando um array com valores 5 linearmente espaçados entre 1 e 50
arr_lin = np.linspace(1,50, 5)
arr_lin
```

Out[8]:
```
array([ 1.  , 13.25, 25.5 , 37.75, 50.  ])
```

In [9]:
```
#inicializando um array com 5 inteiros aleatório de 0 a 10
arr_rand_int = np.random.randint(10,size=5)
arr_rand_int
```

Out[9]:
```
array([3, 9, 7, 2, 0])
```

Atributos

- Os objetos possui uma série de atributos para descrever o conteúdo armazenado

```
In [6]: integers.dtype #Retorna o tipo de dado armazenado no vetor
Out[6]: dtype('int64')

In [7]: floats.dtype
Out[7]: dtype('float64')

In [8]: integers.ndim #Nº de dimensões
Out[8]: 2

In [10]: integers.size # Total de elementos armazenados
Out[10]: 6

In [11]: floats.ndim
Out[11]: 1

In [12]: floats.size
Out[12]: 5

In [13]: integers.shape #Tupla contendo o nº de linhas e colunas
Out[13]: (2, 3)

In [14]: floats.shape
Out[14]: (5,)
```

Atributos

- Vale ressaltar que é possível alterar os atributos de um array
- O mais comum e requerido por muitas aplicações é alterar o *shape* do array
- Por exemplo, se você possui um vetor mas a função só aceita uma matriz, é possível alterar o shape `(x,)` para `(x,1)`
- O reshape só pode ser aplicado se o número de dimensões continuar o mesmo

Atributos

In [21]:
```
#criando um array unidimensional
array1 = np.arange(15)
array1
```

Out[21]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [22]:
```
array1.shape
```

Out[22]:
```
(15,)
```

In [23]:
```
#retornado o array1 com shape de  3 linhas e 5 colunas
array1.reshape(3,5)
```

Out[23]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

In [24]:
```
#retornado o array1 com shape de uma linha e 5 colunas
array1.reshape(1,15)
```

Out[24]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14]])
```

In [25]:
```
#mesmo efeito da célula acima
array1.reshape(1,-1)
```

Out[25]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14]])
```

Atributos

- Em caso de um `reshape` em um *array* multidimensional, pode-se especificar a ordem em que os elementos do *array* original serão percorridos para gerar o novo *array*

```
In [21]:
1 #criando um array unidimensional
2 array1 = np.arange(15)
3 array1

Out[21]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])

In [22]:
1 array1.shape

Out[22]: (15,)

In [23]:
1 #retornado o array1 com shape de  3 linhas e 5 colunas
2 array1.reshape(3,5)

Out[23]: array([[ 0,  1,  2,  3,  4],
                [ 5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14]])

In [24]:
1 #retornado o array1 com shape de uma linha e 5 colunas
2 array1.reshape(1,15)

Out[24]: array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14]])

In [25]:
1 #mesmo efeito da célula acima
2 array1.reshape(1,-1)

Out[25]: array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14]])
```

Acesso aos elementos

- O acesso ao elemento de um *array* é o mesmo do acesso à um elemento de uma lista

```
In [3]: grades = np.array([[87, 96, 70],
                           [100, 87, 90],
                           [94, 77, 90],
                           [100, 81, 82]])

In [4]: grades

Out[4]: array([[ 87,  96,  70],
               [100,  87,  90],
               [ 94,  77,  90],
               [100,  81,  82]])

In [5]: grades[1]

Out[5]: array([100,  87,  90])

In [6]: grades[0,1]

Out[6]: 96
```

Acesso aos elementos

- Operações de *slicing* também podem ser utilizadas

```
In [7]: #Retornando as duas primeiras linhas do array
        grades[0:2]
Out[7]: array([[ 87,  96,  70],
               [100,  87,  90]])

In [8]: #Retornado as linhas 1 e 3 do array
        grades[[1,3]]
Out[8]: array([[100,  87,  90],
               [100,  81,  82]])

In [10]: """Retornando a primeira coluna das 3 primeiras
            linhas do array"""
         grades[0:3,0]
Out[10]: array([ 87, 100,  94])
```

Acesso aos elementos

```
In [11]: #Selecionando tudo menos a última coluna
         grades[:,0:-1]

Out[11]: array([[ 87,  96],
                [100,  87],
                [ 94,  77],
                [100,  81]])

In [12]: grades[:,-1] #Selecionando só a última coluna

Out[12]: array([70, 90, 90, 82])

In [14]: #Especificando as colunas utilizando uma lista
         grades[:,[0,-1]]

Out[14]: array([[ 87,  70],
                [100,  90],
                [ 94,  90],
                [100,  82]])
```

Acesso aos elementos

- Também é possível passar um *array* de booleanos no índice, e todos os respectivos elementos cuja respectivo valor no *array* booleano for `true` serão retornados

```
In [45]:
1 array1 = np.random.randint(50, size=10)
2 array1

Out[45]: array([33, 18, 44, 13,  6, 28, 40, 40, 17, 30])

In [46]:
1 array1[[True, False, True, True, False, False, True, True, False, False]]

Out[46]: array([33, 44, 13, 40, 40])

In [47]:
1 resultado = array1 > 10
2 resultado

Out[47]: array([ True,  True,  True,  True, False,  True,  True,  True,  True,
                True])

In [48]:
1 array1[resultado]

Out[48]: array([33, 18, 44, 13, 28, 40, 40, 17, 30])
```

Percorrendo os Arrays

- Arrays são iteráveis, e, portanto, pode-se aplicar a estrutura `for` considerando essas estruturas

In [18]:
```
for linha in integers:
    for elemento in linha:
        print(elemento, sep=' ', end=' ')
    print()
```

```
1 2 3
4 5 6
```

In [21]:
```
for elemento in floats:
    print(elemento, sep=' ', end=' ')
```

```
0.0 0.1 0.2 0.3 0.4
```

Operações

- Uma das principais características do `NumPy` e que o aproximam de outros *softwares* proprietários, é a **facilidade de se realizar operações matemáticas** utilizando a estruturas de *arrays*

- Além da facilidade das operações matemáticas, operações de **comparação**, **transposição** e **sumarização** dos dados também podem ser realizadas com facilidade

Operações entre Escalar e *Array*

```
In [22]: #Gerando um array de numeros inteiros de 1 a 4
         numbers = np.arange(1,5)

In [23]: numbers
Out[23]: array([1, 2, 3, 4])

In [68]: 5 * numbers
Out[68]: array([ 5, 10, 15, 20])

In [25]: 2 ** numbers
Out[25]: array([ 2,  4,  8, 16])

In [26]: 10 + numbers
Out[26]: array([11, 12, 13, 14])
```

Operações entre *Arrays*

```
In [31]: numbers2 = np.arange(5,9)

In [32]: numbers2
Out[32]: array([5, 6, 7, 8])

In [33]: numbers * numbers2
Out[33]: array([ 5, 12, 21, 32])

In [34]: numbers + numbers2
Out[34]: array([ 6,  8, 10, 12])
```

Operações

- Deve-se tomar cuidado com o operador $*$
- Mesmo que sua estrutura seja de matriz, ele irá considerar como se fosse um vetor para realizar a multiplicação (já que o operador para multiplicação de matrizes é o "$\cdot$")

```
In [82]: m1 = np.array([[2,1],[1,2]])

In [83]: m1
Out[83]: array([[2, 1],
                [1, 2]])

In [84]: inversa = np.linalg.inv(m1)

In [85]: inversa
Out[85]: array([[ 0.66666667, -0.33333333],
                [-0.33333333,  0.66666667]])

In [86]: m1 * inversa
Out[86]: array([[ 1.33333333, -0.33333333],
                [-0.33333333,  1.33333333]])

In [87]: m1.dot(inversa)
Out[87]: array([[1., 0.],
                [0., 1.]])
```

Comparações entre *Arrays*

```
In [36]: numbers < numbers2
Out[36]: array([ True,  True,  True,  True])

In [37]: numbers != numbers2
Out[37]: array([ True,  True,  True,  True])

In [39]: numbers == numbers2
Out[39]: array([False, False, False, False])
```

Transposição

```
In [15]: grades

Out[15]: array([[ 87,  96,  70],
                [100,  87,  90],
                [ 94,  77,  90],
                [100,  81,  82]])

In [16]: #Transpondo a matriz
         grades.T

Out[16]: array([[ 87, 100,  94, 100],
                [ 96,  87,  77,  81],
                [ 70,  90,  90,  82]])

In [17]: #Idem ao anterior
         grades.transpose()

Out[17]: array([[ 87, 100,  94, 100],
                [ 96,  87,  77,  81],
                [ 70,  90,  90,  82]])
```

Sumarização

```
In [52]: integers
Out[52]: array([[1, 2, 3],
                [4, 5, 6]])

In [53]: integers.max()
Out[53]: 6

In [54]: integers.mean()
Out[54]: 3.5
```

Sumarização

```
In [55]: #Tirando a média por linha
         integers.mean(axis=0)
Out[55]: array([2.5, 3.5, 4.5])

In [56]: #Tirando a média por coluna
         integers.mean(axis=1)
Out[56]: array([2., 5.])

In [57]: integers.mean(axis=1)[0]
Out[57]: 2.0
```

Gravação de Arquivos

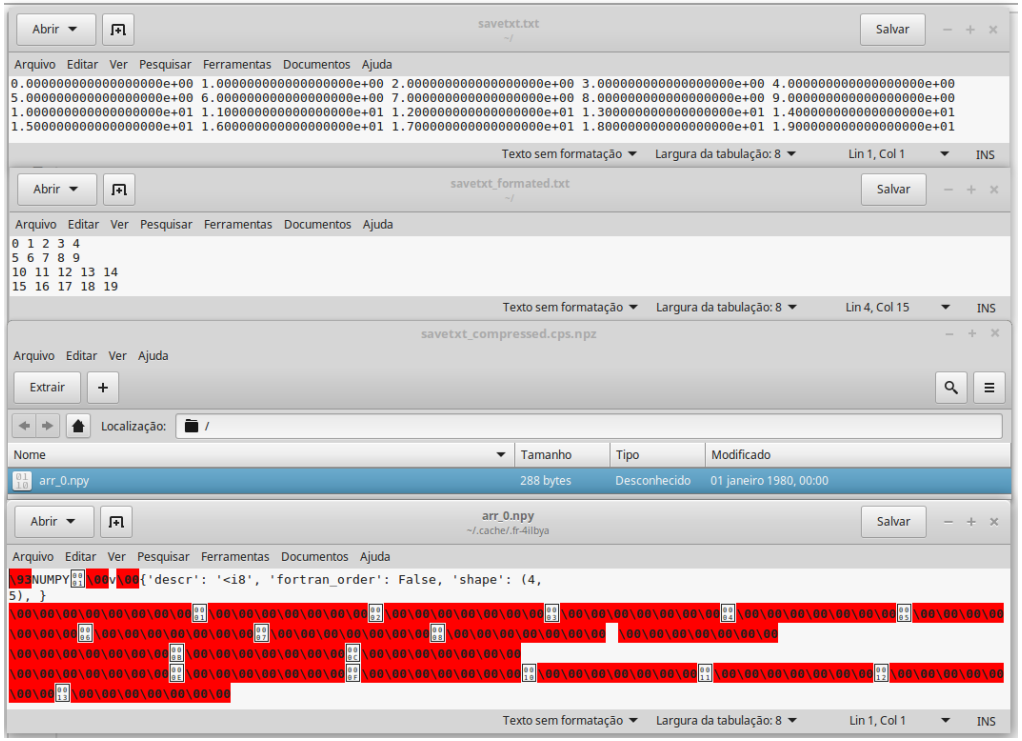
- O módulo `NumPy` também possui utilitários para fazer gravações e leituras de arrays
- No caso da leitura, após feita, já é gerado um objeto de `array` do `NumPy`
- É possível salvar em modo texto, modo binário e binário comprimido
- Também é possível especificar delimitadores dos elementos de um array, cabeçalhos e rodapés no caso do salvamento em modo texto

Gravação de Arquivos

```
1 np.savetxt('savetxt.txt',array1)
```

```
1 np.savetxt('savetxt_formated.txt', array1, fmt='%d')
```

```
1 np.savez_compressed('savetxt_compressed.cps', array1)
```



Leitura de Arquivos

- Assim como na gravação, também há métodos para se fazer a leitura de arrays em modo texto, binário ou binário comprimido

```
In [63]:
1 array2 = np.loadtxt('savetxt.txt', dtype=np.int16)
2 array2

Out[63]: array([[ 0,  1,  2,  3,  4],
                [ 5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14],
                [15, 16, 17, 18, 19]], dtype=int16)

In [66]:
1 array2 = np.load('savetxt_compressed.cps.npz')
2 array2

Out[66]: <numpy.lib.npyio.NpzFile at 0x7fbd59265208>

In [67]:
1 array2.files

Out[67]: ['arr_0']

In [68]:
1 array2['arr_0']

Out[68]: array([[ 0,  1,  2,  3,  4],
                [ 5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14],
                [15, 16, 17, 18, 19]])
```

Concatenando Arrays

- É possível utilizar o método `concat` para concatenar *arrays*
- Por padrão, a concatenação é feita feita verticalmente, porém, é possível alterar o eixo da concatenação

```
In [85]:
1 arr1 = np.array([[1,2,3],[4,5,6]])
2 arr2 = np.array([[7,8,9],[10,11,12]])

In [86]:
1 arr1

Out[86]: array([[1, 2, 3],
                [4, 5, 6]])

In [87]:
1 arr2

Out[87]: array([[ 7,  8,  9],
                [10, 11, 12]])

In [88]:
1 np.concatenate((arr1,arr2))

Out[88]: array([[ 1,  2,  3],
                [ 4,  5,  6],
                [ 7,  8,  9],
                [10, 11, 12]])

In [89]:
1 np.concatenate((arr1,arr2), axis=0)

Out[89]: array([[ 1,  2,  3],
                [ 4,  5,  6],
                [ 7,  8,  9],
                [10, 11, 12]])

In [90]:
1 np.concatenate((arr1,arr2), axis=1)

Out[90]: array([[ 1,  2,  3,  7,  8,  9],
                [ 4,  5,  6, 10, 11, 12]])
```

Concatenando Arrays

- Também é possível utilizar o método `stack`, ou os métodos `vstack` e `hstack` para realizar as concatenações verticalmente e horizontalmente respectivamente

```
In [91]: 1 arr1

Out[91]: array([[1, 2, 3],
                [4, 5, 6]])

In [92]: 1 arr2

Out[92]: array([[ 7,  8,  9],
                [10, 11, 12]])

In [93]: 1 np.stack((arr1,arr2))

Out[93]: array([[[ 1,  2,  3],
                 [ 4,  5,  6]],

                [[ 7,  8,  9],
                 [10, 11, 12]]])

In [98]: 1 np.vstack((arr1,arr2))

Out[98]: array([[ 1,  2,  3],
                [ 4,  5,  6],
                [ 7,  8,  9],
                [10, 11, 12]])

In [99]: 1 np.hstack((arr1,arr2))

Out[99]: array([[ 1,  2,  3,  7,  8,  9],
                [ 4,  5,  6, 10, 11, 12]])
```

Concatenando Arrays

- Pode-se também utilizar o método `append` para fazer concatenações, porém, há distorções nos *shapes* originais dos *arrays*

```
In [100]:  1 arr1

Out[100]: array([[1, 2, 3],
                 [4, 5, 6]])

In [101]:  1 arr2

Out[101]: array([[ 7,  8,  9],
                 [10, 11, 12]])

In [102]:  1 np.append(arr1, arr2)

Out[102]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9,
          10, 11, 12])

In [103]:  1 np.append(arr1, arr2, axis=1)

Out[103]: array([[ 1,  2,  3,  7,  8,  9],
                 [ 4,  5,  6, 10, 11, 12]])
```

Dividindo Arrays

- Pode-se utilizar o método *vsplit* para realizar segmentações verticais em um *array*
- O método aceita tanto um número fíxo de segmentos quanto uma lista, a qual indicará o intervalo de índices em cada divisão

```
In [104]:  1 arr = np.arange(20).reshape(4,5)

In [105]:  1 arr

Out[105]: array([[ 0,  1,  2,  3,  4],
                 [ 5,  6,  7,  8,  9],
                 [10, 11, 12, 13, 14],
                 [15, 16, 17, 18, 19]])

In [108]:  1 np.vsplit(arr,2)

Out[108]: [array([[0, 1, 2, 3, 4],
                  [5, 6, 7, 8, 9]]),
           array([[10, 11, 12, 13, 14],
                  [15, 16, 17, 18, 19]])]

In [110]:  1 np.vsplit(arr,[1,3])

Out[110]: [array([[0, 1, 2, 3, 4]]),
           array([[ 5,  6,  7,  8,  9],
                  [10, 11, 12, 13, 14]]),
           array([[15, 16, 17, 18, 19]])]
```

Dividindo Arrays

- Da mesma forma que o `vsplit`, pode-se utilizar o `hsplit` para realizar sementações verticais em um *array*

```
In [112]: 1 arr

Out[112]: array([[ 0,  1,  2,  3,  4],
                 [ 5,  6,  7,  8,  9],
                 [10, 11, 12, 13, 14],
                 [15, 16, 17, 18, 19]])

In [117]: 1 np.hsplit(arr, [1,2])

Out[117]: [array([[ 0],
                  [ 5],
                  [10],
                  [15]]),
           array([[ 1],
                  [ 6],
                  [11],
                  [16]]),
           array([[ 2,  3,  4],
                  [ 7,  8,  9],
                  [12, 13, 14],
                  [17, 18, 19]])]
```

Material Complementar

- Wikipedia – NumPy

  `https://en.wikipedia.org/wiki/NumPy`

- Numpy — Array Creation

  `https://www.geeksforgeeks.org/numpy-array-creation/`

- `numpy.loadtxt`

  `https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.loadtxt.html#numpy.loadtxt`

Material Complementar

- Learn NUMPY in 5 minutes - BEST Python Library!

  `https://www.youtube.com/watch?v=xECXZ3tyONo&t=4s`

- Python NumPy Tutorial for Beginners

  `https://www.youtube.com/watch?v=QUT1VHiLmmI`

- Quickstart tutorial

  `https://numpy.org/doc/stable/user/quickstart.html`

- NumPy Joining Array

  `https://www.w3schools.com/python/numpy_array_join.asp`

Imagem do Dia

Tópicos em Inteligência Artificial
http://ava.ufms.br/

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br