Estatística Aplicada à Computação/Telemática

Tutorial Básico de Numpy

Nesse ponto, é esperado que você tenha feito a instalação do anaconda 3 em sua máquina e passado pelo tutorial de Jupyter Notebook. Se não o fez, retorne e faça, ou não terá sucesso nesse tutorial.

A ideia aqui é servir de auto estudo. Dessa forma, espera-se que você crie um notebook novo e execute todos os comandos as seguir, tentando entender cada saída, e responda a todos os exercícios espalhados no próprio notebook que você acabou de criar, anotando suas dúvidas, quando tiver (após consultar as referências da biblioteca, lógico), para que que essas sejam discutidas em sala de aula. De outro jeito, não haverá aprendizado

Começamos, como discutido em sala, pela importação do módulo,

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

Exercise 1 Existem outras formas de fazer essa importação? Liste-as e discuta suas vantagens e desvantagens.

Exercise 2 Explique os resultados das execuções de todas as células do tutorial, com suas palavras, logo após cada resultado.

CRIANDO ARRAYS

Existem várias formas de criar arrays, pode ser através do método np.array(), usando uma lista ou tupla do python:

In [ ]:
```
arr = np.array([-2, 4.3, 7, 9])

my_tuple = (8, -5, 0, 3.2)
arr2 = np.array(my_tuple)

print('arr = ', arr)
print('arr2 = ', arr2)
```

Exercise 3 É curioso notar que todos os elementos de *arr* se tornaram *float* após a criação do *array*, mesmo que nem todos os elementos da lista fossem desse tipo. Por que isso ocorreu? E se um desses elementos fosse um *string*, qual seria a saída?

Usando o método arange (similar ao range do python)

In [ ]:
```
arr = np.arange(10)
print(arr, '\n')

arr2 = np.arange(1, 32, 2).reshape(4, 4) # reshape muda o formato da array - nesse caso de
print(arr2)
```

Temos também outro método, o linspace

In [ ]:
```
# Método linspace()

arr = np.linspace(1, 100, 10) # de 1 a 100, contendo 10 valores
print(arr, '\n')
```

Exercise 4 Quais as diferenças entre os métodos *arange* e *linspace* para criar arrays?

In [ ]:
```
# Array vazia com o método empty()

arr2 = np.empty((6,4)) # normalmente a saída é lixo de memória ou zeros
print(arr2, '\n')


# Np.Eye

arr3 = np.eye(4) # eye cria uma array de duas dimensões com 1 na da diagnoal e 0 nos outro
print(arr3)
```

Usando os métodos 'zeros()' e 'ones()' para criar arrays de 0 e 1, respectivamente:

In [ ]:
```
arr = np.zeros(8) # array contendo 8 'zeros'
print (arr)

arr2 = np.ones((4,3)) # array contendo 12 'uns', no formato 4 x 3
print(arr2)
```

ATRIBUTOS DAS ARRAYS

Existem vários atributos, mas o três principais são .shape , .size e .dtype

In [ ]:
```
arr = np.linspace(.5, 50, 10)

print('.shape: ', arr.shape)    # imprime o formato da array
print('.size: ', arr.size)      # imprime o tamanho da array
print('.dtype: ', arr.dtype)    # imprime o tipo de dado dos elementos
```

Outros atributos:

In [ ]:
```
print('.ndim: ', arr.ndim)          # número de dimensões
print('.itemsize: ', arr.size)      # tamanho de cada elemento em bytes
```

Exercise 5 O método *len()* é usado para mostrar o tamanho de uma lista. Esse método também funciona para *arrays*? Explique.

MÉTODOS

Existem inúmeros métodos que nos ajudam a trabalhar com numpy arrays, vou mostrar os mais usados:

In [ ]:
```
arr = np.arange(1,10)
print('array original: ', arr, '\n')
```

Como já vimos 'reshape()' muda o formato da array, MAS o número de elementos dos dois formatos tem que coincidir, exemplo:

In [ ]:
```
arr2 = arr.reshape((3,3))
print ('formato 3x3: \n' , arr2, '\n')
```

In [ ]:
```
arr.max() # retorna o valor máximo
```

In [ ]:
```
print(arr.argmax()) # retorna o índice que contém o valor máximo
arr[8] == arr[arr.argmax()]
```

In [ ]:
```
arr.min() # retorna o mínimo
```

In [ ]:
```
print(arr.argmin()) # retorna o índice que contém o valor máximo
arr[0] == arr[arr.argmin()]
```

In [ ]:
```
arr.mean() # retorna a média
```

In [ ]:
```
arr.std() # retorna o desvio padrão
```

In [ ]:
```
arr.sum() #retorna a soma de todos os elementos da array
```

In [ ]:
```
### FUNÇÕES UNIVERSAIS ###

np.sqrt(arr) # retorna um array com a raiz quadrada de cada elemento
```

In [ ]:
```
np.sin(arr) # retorna um array com o seno de cada elemento
```

In [ ]:
```
np.cos(arr) # retorna um array com o cosseno de cada elemento
```

In [ ]:
```
np.exp(arr) # retorna um array com o exponencial cada elemento (e^x)
```

OPERAÇÕES BÁSICAS

É bem fácil realizar operações matemáticas com arrays

In [ ]:
```
# vamos criar as arrays A e B para fazer operações com elas

A = np.arange(10)
B = np.arange(10,38,3)

print (' A: ', A, '\n', 'B: ', B)
```

In [ ]:
```
somar = A + B
subtrair = A - B
dividir = A / B
multiplicar = A * B

print('somar: ', somar )
print('subtrair: ', subtrair )
print('dividir: ', dividir )
print('multiplicar: ', multiplicar)
```

Perceba que se você dividir por zero, numpy não vai dar erro (o código vai rodar), porém vai emitir uma advertência!

In [ ]:
```
B/A     # a array A contém um zero como primeiro elemento
```

BROADCAST = transmissão

In [ ]:
```
''' Você pode fazer operações envolvendo arrays e um único elemento, dessa forma numpy ite
do elemento único com todos os componentes da array, exemplo: '''

A * 100    # retorna um array com a multiplicação de todos elementos da array por 100
```

Exercise 6 Explique, com suas palavras, o conceito de vetorização do *numpy*.

INDEXAÇÃO E FATIAMENTO (SLICING)

O acesso de elementos de uma array, funciona de forma similar ao das listas do Python:

In [ ]:
```
# criar uma array com o quadrado dos números de 1 a 10
arr = np.arange(1, 10) **2
arr
```

In [ ]:
```
# acessando o primeiro elemento
print('1º elemento: ', arr[0])    # lembrar que índice começa do 0

# acessando o quinto elemento
print ('5º elemento: ', arr[4])   # O quinto elemento é acessado pelo índice 4
```

In [ ]:
```
# Em um array de duas dimensões (ou mais) existem duas formas de acessar o índice:

arr = arr.reshape((3,3))
print (arr, '\n')

# Usando colchetes duplos [][]
print ('Acessando o elemento 25: ', arr[1][1])

# Ou minha preferência pessoal - separando índices por vírgula
print ('Acessando o elemento 25: ', arr[1,1])

# acessando o último elemento usando o -1
arr[-1, -1]
```

In [ ]:
```
# Para acessar fatias de elementos, usamos o slice [i:i], exemplo

# linha: acessar da linha de índice 0 a linha de índice 2 (excluindo esta) = [0:2]
# coluna: acessar da coluna de índice 0 a coluna de 3 (excluindo esta) = [0:3] - no nosso
print(arr[0:2 , 0:3],'\n')

# outra forma de obter o mesmo resultado é usando [:], que pega todos os elementos
print(arr[0:2 , 0:3])
```

In [ ]:
```
# Podemos criar uma nova array com slice de outra

arr2 = arr[ : , 1:3]
arr2
```

In [ ]:
```
# Podemor atribuir novos elementos a nossa array, usando slice

arr[1:2, :] = -99
arr
```

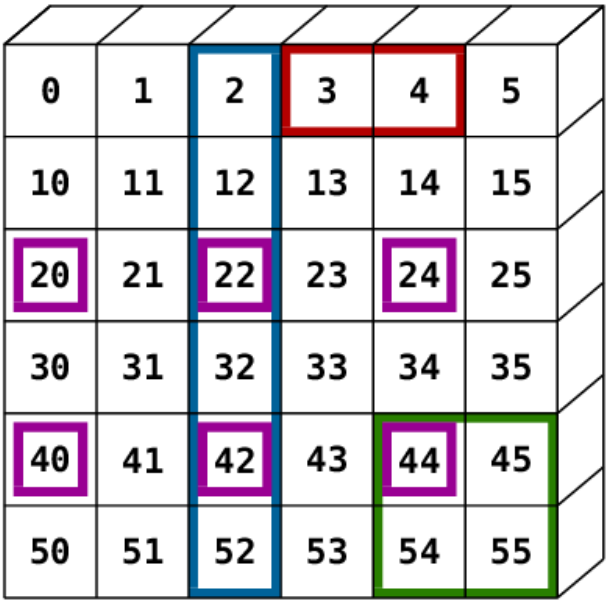
Uma imagem vale mais que mil palavras:

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

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



SELEÇÃO CONDICIONAL:

Uma forma interessante de selecionar elementos baseado em alguma condição

In [ ]:
```
A = np.arange(10)
print ('A: ', A)
```

In [ ]:
```
A > 5 # retorna uma array de variáveis booleanas de acordo com a condição
```

In [ ]:
```
# Também odemos criar uma array a partir dessa condição
B = A > 5
```

In [ ]:
```
# Isso nos dá 2 opções para fazer seleção condicional, passando a condição ou passando o a

print (A [A > 5])

print (A [B])

# em ambos os casos só retorna os valores que admitem essa condição
```