NumPy - Funções de classificação, pesquisa e contagem

Uma variedade de funções relacionadas à classificação estão disponíveis no NumPy. Essas funções de classificação implementam diferentes algoritmos de classificação, cada um deles caracterizado pela velocidade de execução, desempenho do pior caso, espaço de trabalho necessário e estabilidade dos algoritmos. A tabela a seguir mostra a comparação de três algoritmos de classificação.

tipo	velocidade	pior caso	área de trabalho	estábulo
'ordenação rápida'	1	O(n^2)	0	não
'mesclar classificação'	2	O(n*log(n))	~n/2	sim
'heapsort'	3	O(n*log(n))	0	não

numpy.sort()

A função sort() retorna uma cópia classificada do array de entrada. Possui os seguintes parâmetros -

```
numpy.sort(a, axis, kind, order)
```

Onde,

Sr. Não.	Parâmetro e Descrição
1	**a** Matriz a ser classificada
2	**eixo** O eixo ao longo do qual a matriz deve ser classificada. Se não houver, a matriz será nivelada, classificando no último eixo
3	**tipo** O padrão é classificação rápida
4	**ordem** Se a matriz contiver campos, a ordem dos campos a serem classificados

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[3,7],[9,1]])

print 'Our array is:'
print a
print '\n'

print 'Applying sort() function:'
print np.sort(a)
print '\n'

print 'Sort along axis 0:'
print np.sort(a, axis = 0)
print '\n'

# Order parameter in sort function
dt = np.dtype([('name', 'S10'),('age', int)])
a = np.array([("raju",21),("anil",25),("ravi", 17), ("amar",27)], dtype = dt)

print 'Our array is:'
print a
print '\n'

print 'Order by name:'
print np.sort(a, order = 'name')
```

Ele produzirá a seguinte saída -

```
Our array is:
[[3 7]
 [9 1]]

Applying sort() function:
[[3 7]
 [1 9]]

Sort along axis 0:
[[3 1]
 [9 7]]
```

```
Our array is:
[('raju', 21) ('anil', 25) ('ravi', 17) ('amar', 27)]

Order by name:
[('amar', 27) ('anil', 25) ('raju', 21) ('ravi', 17)]
```

numpy.argsort()

A função **numpy.argsort()** executa uma classificação indireta na matriz de entrada, ao longo de um determinado eixo e usando um tipo especificado de classificação para retornar a matriz de índices de dados. Esta matriz de índices é usada para construir a matriz classificada.

Exemplo

Demonstração ao vivo

```
import numpy as np
x = np.array([3, 1, 2])

print 'Our array is:'
print x
print '\n'

print 'Applying argsort() to x:'
y = np.argsort(x)
print y
print '\n'

print 'Reconstruct original array in sorted order:'
print x[y]
print '\n'

print 'Reconstruct the original array using loop:'
for i in y:
   print x[i],
```

Ele produzirá a seguinte saída -

```
Our array is:
[3 1 2]

Applying argsort() to x:
[1 2 0]
```

```
Reconstruct original array in sorted order:
[1 2 3]

Reconstruct the original array using loop:
1 2 3
```

numpy.lexsort()

A função executa uma classificação indireta usando uma sequência de chaves. As chaves podem ser vistas como uma coluna em uma planilha. A função retorna uma matriz de índices, com os quais os dados classificados podem ser obtidos. Observe que a última chave é a chave primária do tipo.

Exemplo

Demonstração ao vivo

```
import numpy as np

nm = ('raju','anil','ravi','amar')
dv = ('f.y.', 's.y.', 's.y.', 'f.y.')
ind = np.lexsort((dv,nm))

print 'Applying lexsort() function:'
print ind
print '\n'

print 'Use this index to get sorted data:'
print [nm[i] + ", " + dv[i] for i in ind]
```

Ele produzirá a seguinte saída -

```
Applying lexsort() function:
[3 1 0 2]

Use this index to get sorted data:
['amar, f.y.', 'anil, s.y.', 'raju, f.y.', 'ravi, s.y.']
```

O módulo NumPy possui várias funções para pesquisar dentro de um array. Estão disponíveis funções para encontrar o máximo, o mínimo, bem como os elementos que satisfazem uma determinada condição.

numpy.argmax() e numpy.argmin()

Essas duas funções retornam os índices dos elementos máximo e mínimo, respectivamente, ao longo do eixo fornecido.

Exemplo

Demonstração ao vivo

```
import numpy as np
a = np.array([[30,40,70],[80,20,10],[50,90,60]])

print 'Our array is:'
print a
print '\n'

print 'Applying argmax() function:'
print np.argmax(a)
print '\n'

print 'Index of maximum number in flattened array'
print a.flatten()
print '\n'

print 'Array containing indices of maximum along axis 0:'
maxindex = np.argmax(a, axis = 0)
print maxindex
print '\n'

print 'Array containing indices of maximum along axis 1:'
maxindex = np.argmax(a, axis = 1)
print maxindex
print '\n'

print 'Applying argmin() function:'
minindex = np.argmin(a)
print minindex
print '\n'

print 'Flattened array:'
print a.flatten()[minindex]
print '\n'

print 'Flattened array along axis 0:'
minindex = np.argmin(a, axis = 0)
print minindex
print '\n'
```

```
print 'Flattened array along axis 1:'
minindex = np.argmin(a, axis = 1)
print minindex
```

Ele produzirá a seguinte saída -

```
Our array is:
[[30 40 70]
 [80 20 10]
 [50 90 60]]

Applying argmax() function:
7

Index of maximum number in flattened array
[30 40 70 80 20 10 50 90 60]

Array containing indices of maximum along axis 0:
[1 2 0]

Array containing indices of maximum along axis 1:
[2 0 1]

Applying argmin() function:
5

Flattened array:
10

Flattened array along axis 0:
[0 1 1]

Flattened array along axis 1:
[0 2 0]
```

numpy.diferente de zero()

A função **numpy.nonzero()** retorna os índices de elementos diferentes de zero na matriz de entrada.

Exemplo

```
import numpy as np
a = np.array([[30,40,0],[0,20,10],[50,0,60]])

print 'Our array is:'
print a
print '\n'

print 'Applying nonzero() function:'
print np.nonzero (a)
```

Demonstração ao vivo

Ele produzirá a seguinte saída -

```
Our array is:
[[30 40 0]
 [ 0 20 10]
 [50 0 60]]

Applying nonzero() function:
(array([0, 0, 1, 1, 2, 2]), array([0, 1, 1, 2, 0, 2]))
```

numpy.where()

A função where() retorna os índices dos elementos em uma matriz de entrada onde a condição fornecida é satisfeita.

Exemplo

```
import numpy as np
x = np.arange(9.).reshape(3, 3)

print 'Our array is:'
print x

print 'Indices of elements > 3'
y = np.where(x > 3)
print y

print 'Use these indices to get elements satisfying the condition'
print x[y]
```

Demonstração ao vivo

Ele produzirá a seguinte saída -

```
Our array is:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]

Indices of elements > 3
(array([1, 1, 2, 2, 2]), array([1, 2, 0, 1, 2]))

Use these indices to get elements satisfying the condition
[ 4. 5. 6. 7. 8.]
```

numpy.extract()

A função **extract()** retorna os elementos que satisfazem qualquer condição.

Demonstração ao vivo

```
import numpy as np
x = np.arange(9.).reshape(3, 3)

print 'Our array is:'
print x

# define a condition
condition = np.mod(x,2) == 0

print 'Element-wise value of condition'
print condition

print 'Extract elements using condition'
print np.extract(condition, x)
```

Ele produzirá a seguinte saída -

```
Our array is:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]

Element-wise value of condition
[[ True False True]
 [False True False]
```

```
 [ True False True]]

Extract elements using condition
[ 0. 2. 4. 6. 8.]
```