Numpy para Adaboost

Sistemas Inteligentes, curso 21/22

Departamento DCCIA. Universidad de Alicante

Numpy para Adaboost

Los conceptos de vectorización, indexación y transmisión de NumPy son los estándares de facto de la computación de vectores y matrices en la actualidad https://numpy.org/. Para más información y documentación ver https://numpy.org/doc/

Qué veremos

- Crear arrays de múltiples dimensiones
- Operaciones simples con arrays
- Operaciones complejas con arrays
- Seleccionar zonas de una matriz
- Construir una matriz mediante submatrices

Para usar numpy...

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

Crear arrays de múltiples dimensiones

Creando arrays vacios

- np.zeros
- np.ones

In [2]:
```
np.zeros(5)
```

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

In [3]:
```
np.zeros((3,3))
```

Out[3]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [4]:
```
np.ones(5)
```

Out[4]:
```
array([1., 1., 1., 1., 1.])
```

¿Y si queremos que sea un array de ochos?

In [5]:
```
8*np.ones(5)
```

Out[5]:
```
array([8., 8., 8., 8., 8.])
```

Creando arrays a partir de una lista

In [6]:
```
mis_datos = ((0,0), (0,1), (2,3), (5,7))
print(mis_datos)
```

```
((0, 0), (0, 1), (2, 3), (5, 7))
```

In [7]:
```
np.asarray(mis_datos)
```

Out[7]:
```
array([[0, 0],
       [0, 1],
       [2, 3],
       [5, 7]])
```

Operaciones simples con arrays

- Suma, resta y multiplicación
- Suma de todos los elementos
- Transpuesta

In [8]:
```
A = np.asarray(((0,1),(1,0)))
B = np.asarray(((8,8),(5,5)))
```

In [9]:
```
2*A
```

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

In [10]:
```
A*B
```

Out[10]:
```
array([[0, 8],
       [5, 0]])
```

In [11]:
```
-A
```

Out[11]:
```
array([[ 0, -1],
       [-1,  0]])
```

In [12]:
```
B-A
```

Out[12]:
```
array([[8, 7],
       [4, 5]])
```

In [13]:
```
np.sum(A)
```

Out[13]:
```
2
```

In [14]:
```
A.transpose()
```

Out[14]:
```
array([[0, 1],
       [1, 0]])
```

Operaciones complejas con arrays

- Tamaño de un array
- Iterar mediante un bucle
- Vectorizar operaciones

In [15]:
```
A = np.asarray(((2,2,4),(3,4,5)))
```

In [16]:
```
A.shape
```

Out[16]:
```
(2, 3)
```

Varias maneras de calcular la suma de `A`

In [17]:
```
sum=0
(alto,ancho) = A.shape
for i in range(alto):
    for j in range(ancho):
        sum += A[i][j]
print(sum)
```

```
20
```

In [18]:
```
sum = 0
for fila in A:
    for val in fila:
        sum += val
print(sum)
```

```
20
```

In [19]:
```
np.sum(A)
```

Out[19]:
```
20
```

Aplicando una función a cada elemento

In [20]:
```
A = np.asarray(((2.0,2.0,4.0),(3.0,4.0,5.0)))

(alto,ancho) = A.shape
for i in range(alto):
    for j in range(ancho):
        A[i][j] = A[i][j]*(A[i][j]/2.0)
A
```

Out[20]:
```
array([[ 2. ,  2. ,  8. ],
       [ 4.5,  8. , 12.5]])
```

In [21]:
```
A = np.asarray(((2.0,2.0,4.0),(3.0,4.0,5.0)))

def myfunc(v):
    return v*(v/2.0)

myfunc(A)
```

Out[21]:
```
array([[ 2. ,  2. ,  8. ],
       [ 4.5,  8. , 12.5]])
```

Seleccionar zonas de una matriz

- Acceder a elementos
- Filas y columnas
- Vectores booleanos

https://numpy.org/doc/stable/reference/arrays.indexing.html

In [22]:
```
A = np.asarray(((1,2,3),(4,5,6),(7,8,9),(10,11,12)))
print(A)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

In [23]:
```
A[0]
```

Out[23]:
```
array([1, 2, 3])
```

In [24]:
```
A[0][0]
```

Out[24]:
```
1
```

In [25]:
```
A[1]
```

Out[25]: array([4, 5, 6])

In [26]:
```
A[:,1]
```

Out[26]:
```
array([ 2,  5,  8, 11])
```

In [27]:
```
A[:,2]
```

Out[27]:
```
array([ 3,  6,  9, 12])
```

In [28]:
```
A[1,:]
```

Out[28]:
```
array([4, 5, 6])
```

In [29]:
```
A[:,1:3]
```

Out[29]:
```
array([[ 2,  3],
       [ 5,  6],
       [ 8,  9],
       [11, 12]])
```

¿Cómo puedo obtener los elementos cuyo valor es mayor de 4?

In [30]:
```
res = []
for c in A:
  for v in c:
    if v>4:
        res.append(v)

res
```

Out[30]:
```
[5, 6, 7, 8, 9, 10, 11, 12]
```

¿Es la manera más eficiente?

In [31]:
```
A>4
```

Out[31]:
```
array([[False, False, False],
       [False,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

In [32]:
```
filtro = A>4
```

In [33]:
```
A[filtro]
```

Out[33]:
```
array([ 5,  6,  7,  8,  9, 10, 11, 12])
```