Módulo de adaptación

Master en Business Intelligence y Big Data

PROFESOR/A
Antonio Sarasa Cabezuelo

Escuela de organización industrial

www.eoi.es

Numpy

Numpy

- Es una extensión de Python para dar soporte a operaciones sobre vectores y matrices.
- En Numpy estos objetos se denominan “ndarray” y se caracterizan porque todos los elementos de la matriz son del mismo tipo y están indexados por una tupla de números positivos.
- Observar que numpy.array no es lo mismo que la librería estandar de Python array.array, dado que esta solo gestiona array de una dimensión y ofrece menos funcionalidad.
- Cada “narray” tiene un conjunto de atributos que le caracterizan:
  - ndarray.ndim: Número de dimensiones del array.
  - ndarray.dtype: Describe el tipo de elementos del array

Numpy

- ndarray.shape: Dimensiones del array. Se trata de una tuple de enteros que indicant el tamaño del array en cada dimension. Por ejemplo para una matriz de n filas y m columnas, sus dimensiones serían (n,m) y el número de dimensiones sería 2.
- ndarray.size: Número total de elementos del array(producto de los elementos de la dimensión).
- ndarray.itemsize: El tamaño en bytes de cada elemento del array.
- ndarray.data: Es el buffer conteniendo los elementos actuals del array. Normalmente no se usa este atributo pues se accede a los elementos directamente mediante los índices.

Numpy

```
>>> from numpy  import *
>>> a = arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int32'
>>> a.itemsize
4
>>> a.size
15
>>> type(a)
numpy.ndarray
>>> b = array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
numpy.ndarray
```

Numpy

- Existen varias formas de crear un array:
  - Desde una lista o tuple de Python usando la función array. En este caso el tipo del array resultante se deduce del tipo de elementos de las secuencias.

```
>>> from numpy  import *
>>> a = array( [2,3,4] )
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int32')
>>> b = array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

Numpy

- Algunas observaciones:
  - Un error frecuente consiste en usar la función array con un conjunto de argumentos numéricos en vez de una secuencia de números(lista o tupla).
  - Cuando a la función array se le pasa como argumento una secuencia de secuencias, entonces genera un array de tantas dimensiones como secuencias. Por ejemplo una secuencia de secuencias sería un array bidimensional.

```
>>> b = array( [ (1.5,2,3), (4,5,6) ] )
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

Numpy

– El tipo del array puede ser especificado en el momento de la creación del mismo.

```
>>> c = array( [ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Numpy

- Otra forma de crear un array es usando un conjunto de funciones de Numpy que únicamente requieren conocer el tamaño del array pero no su contenido:
  - zeros crea un array lleno de ceros.
  - ones crea un array lleno de unos.
  - empty creaun array cuyo contenido es aleatorio.
- Por defecto en todos los casos, el atributo dttype del array es float64.

Numpy

```
>>> zeros( (3,4) )
array([[0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.],
       [0.,  0.,  0.,  0.]])
>>> ones( (2,3,4), dtype=int16 )
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
>>> empty( (2,3) )
array([[  3.73603959e-262,   6.02658058e-154,   6.55490914e-260],
       [  5.30498948e-313,   3.14673309e-307,   1.00000000e+000]])
```

Numpy

- Numpy tiene una función analoga a la función range de Python denominada arange que permite generar secuencias de números. Toma como parámetros el rango de los números a generar y la distancia entre ellos.

```
>>> arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> arange( 0, 2, 0.3 )
array([ 0. ,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8])
```

- Cuando se usan argumentos reales, es mejor usar la función linspace que recibe como argumento el número de elementos en vez de la distancia entre ellos.

```
>>> linspace( 0, 2, 9 )
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ])
>>> x = linspace( 0, 2*pi, 100 )
>>> f = sin(x)
```

Numpy

- Cuando se imprime un array:
  - Unidimensional, se imprime como una fila.
  - Bidimensional se imprime como una matriz.
  - Tridimensional se imprime como una lista de matrices.
  - Y así sucesivamente.

```
>>> a = arange(6)
>>> print a
[0 1 2 3 4 5]
>>>
>>> b = arange(12).reshape(4,3)
>>> print b
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>>
>>> c = arange(24).reshape(2,3,4)
>>> print c
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Numpy

- Si un array es demasiado largo, solo se imprime la parte cental.

```
>>> print arange(10000)
[   0    1    2 ..., 9997 9998 9999]
>>>
>>> print arange(10000).reshape(100,100)
[[   0    1    2 ...,   97   98   99]
 [ 100  101  102 ...,  197  198  199]
 [ 200  201  202 ...,  297  298  299]
 ...,
 [9700 9701 9702 ..., 9797 9798 9799]
 [9800 9801 9802 ..., 9897 9898 9899]
 [9900 9901 9902 ..., 9997 9998 9999]]
```

- Este comportamiento se puede cambiar para que imprima todo el array entero mediante la sentencia:

  set_printoptions(threshold='nan')

Numpy

- Se pueden realizar operaciones aritméticas entre las matrices:
  - Las operaciones operan sobre los elementos de la misma posición, y como resultado crean un nuevo array:

```
>>> a = array( [20,30,40,50] )
>>> b = arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([True, True, False, False], dtype=bool)
```

Numpy

- Si se quiere hacer el product de dos matrices, y no el product element a element se usa el operador dot()

```
>>> A = array( [[1,1],
...             [0,1]] )
>>> B = array( [[2,0],
...             [3,4]] )
>>> A*B
array([[2, 0],
       [0, 4]])
>>> dot(A,B)
array([[5, 4],
       [3, 4]])
```

Numpy

- Los operadores += y *= modifican los arrays en vez de crear uno nuevo.

```
>>> a = ones((2,3), dtype=int)
>>> b = random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a
>>> b
array([[ 3.69092703,  3.8324276 ,  3.0114541 ],
       [ 3.18679111,  3.3039349 ,  3.37600289]])
>>> a += b
>>> a
array([[6, 6, 6],
       [6, 6, 6]])
```

Numpy

- Cuando se opera con arrays de diferente tipo, entonces el tipo del array resultante corresponde al más general.

```
>>> a = ones(3, dtype=int32)
>>> b = linspace(0,pi,3)
>>> b.dtype.name
'float64'
>>> c = a+b
>>> c
array([ 1.        ,  2.57079633,  4.14159265])
>>> c.dtype.name
'float64'
>>> d = exp(c*1j)
>>> d
array([ 0.54030231+0.84147098j, -0.84147098+0.54030231j,
       -0.54030231-0.84147098j])
>>> d.dtype.name
'complex128'
```

Numpy

- Muchas operaciones son implementadas como métodos de la clase ndarray, y por defecto se aplican como si el array fuera una lista de números. Sin embargo es posible especificar la dimensión sobre la que se quiere aplicar la operación.

```
>>> a = random.random((2,3))
>>> a
array([[ 0.6903007 ,  0.39168346,  0.16524769],
       [ 0.48819875,  0.77188505,  0.94792155]])
>>> a.sum()
3.4552372100521485
>>> a.min()
0.16524768654743593
>>> a.max()
0.9479215542670073
```

Numpy

```
>>> b = arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

Numpy

- En NumPy proporciona un conjunto de funciones matemáticas denominadas funciones universales tales como sin,cos,exp,…Estas funciones operan elemento a elemento y generan como resultado un nuevo array.

```
>>> B = arange(3)
>>> B
array([0, 1, 2])
>>> exp(B)
array([ 1.        ,  2.71828183,  7.3890561 ])
>>> sqrt(B)
array([ 0.        ,  1.        ,  1.41421356])
>>> C = array([2., -1., 4.])
>>> add(B, C)
array([ 2.,  0.,  6.])
```

Numpy

Función	Descripción
abs, fabs	Calcula el valor absoluto de enteros, números reales y números complejos.
sqrt	Calcular la raíz cuadrada de cada elemento
square	Calcula el cuadrado de cada elemento
exp	Calcula la exponencial de cada elemento
log, log10, log2, log1p	Logaritmos en distintas bases.
sign	Calcula el signo de cada elemento:1(+),0(0),-1(-)
ceil	Calcula el entero más pequeño mayor o igual que el elemento dado
floor	Calcula el entero más grande menor o igual que el elemento dado
rint	Redondea los elementos al entero más cercano
modf	Calcula las partes fraccional y entera de un array como arrays separados
isnan	Retorna un array de booleanos indicando si cada valor es un número o no
isfinite, isinf	Retorna un array de booleanos indicando si cada elemento es finito o infinito.
cos, cosh, sin, sinh, tan, tanh	Funciones trigonómetricas normales e hiperbólicas
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Funciones trigonométricas inversas
logical_not	Calcula el valor de verdad de la negación del array

Numpy

Función	Descripción
add	Añadir elementos a un array
subtract	Restar los elementos del primer array del segundo
multiply	Multiplicar elementos del array
divide, floor_divide	Dividir truncando o sin truncar
power	Hacer la potencia de los elementos del primer array a los exponentes dados en el segundo array
maximum, fmax	Elemento máximo consideran o no los nulos
minimum, fmin	Elemento mínimo consideran o no los nulos
mod	Módulo de los elementos
copysign	Copiar los signos de los valores del segundo argumento al primero
greater, greater_equal, less, less_equal, equal, not_equal	Operaciones de comparación lógicas: >, >=, <, <=, ==, !=
logical_and, logical_or, logical_xor	Calcula el valor de verdad de la operación lógica

Numpy

- Los arrays pueden ser indexados, se pueden seleccionar subrangos y se puede iterar sobre sus elementos:
  - Cuando se accede a un array por índice, se indica un número por cada dimensión.
  - Cuando se elige un subrango se indica por cada dimensión a:b:c dónde a indica dónde se comienza, b dónde se termina, y c el salto entre elementos

Numpy

- Ejemplo con arrays unidimensionales

```
>>> a = arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> a[:6:2] = -1000
-1000
>>> a
array([-1000,     1, -1000,    27, -1000,   125,   216,   343,   512,   729])
>>> a[ : :-1]
array([  729,   512,   343,   216,   125, -1000,    27, -1000,     1, -1000])
```

Numpy

- Ejemplo con arrays multidimensionales

```
>>> b =
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2,3]
23
>>> b[0:5, 1]
array([ 1, 11, 21, 31, 41])
>>> b[ : ,1]
array([ 1, 11, 21, 31, 41])
>>> b[1:3, : ]
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

Numpy

- Cuando se accede a un array y se indican menos índices que el número de dimensiones entonces se consideran todos los valores de las dimensiones no indicadas.

```
>>> b[-1]
array([40, 41, 42, 43])
```

- También es posible usar "…" para representar el resto de dimensiones.

```
>>> c = array( [ [[  0,  1,  2],
...               [ 10, 12, 13]],
...
...              [[100,101,102],
...               [110,112,113]] ] )
>>> c.shape
(2, 2, 3)
>>> c[1,...]
array([[100, 101, 102],
       [110, 112, 113]])
>>> c[...,2]
array([[  2,  13],
       [102, 113]])
```

Numpy

- Para iterar se utiliza “for”:

```
>>> for i in a:
...         print i**(1/3.),
...
nan 1.0 nan 3.0 nan 5.0 6.0 7.0 8.0 9.0
>>> for row in b:
...         print row
...
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

Numpy

- También es posible usar en las iteraciones el iterador “flat” que permite iterar sobre todos los elementos del array.

```
>>> for element in b.flat:
...         print element,
...
0 1 2 3 10 11 12 13 20 21 22 23 30 31 32 33 40 41 42 43
```

Numpy

- El tamaño de un array viene dado por el número de elementos de cada dimensión. Sin embargo el tamaño puede ser cambiado:
  - Una opción consiste en aplanar la matriz mediante el commando ravel() y a continuación usar la función reshape() para establecer las nuevas dimensiones.

```
>>> a =
array([[ 7.,  5.,  9.,  3.],
       [ 7.,  2.,  7.,  8.],
       [ 6.,  8.,  3.,  2.]])
>>> a.shape
(3, 4)
>>> a.ravel()
array([ 7.,  5.,  9.,  3.,  7.,  2.,  7.,  8.,  6.,  8.,  3.,  2.])
>>> a.shape = (6, 2)
>>> a.transpose()
array([[ 7.,  9.,  7.,  7.,  6.,  3.],
       [ 5.,  3.,  2.,  8.,  8.,  2.]])
```

Numpy

- Otra opción consiste en usar la función resize(), indicando el nuevo tamaño.

```
>>> a
array([[ 7.,  5.],
       [ 9.,  3.],
       [ 7.,  2.],
       [ 7.,  8.],
       [ 6.,  8.],
       [ 3.,  2.]])
>>> a.resize((2,6))
>>> a
array([[ 7.,  5.,  9.,  3.,  7.,  2.],
       [ 7.,  8.,  6.,  8.,  3.,  2.]])
```

- Otra opción sería usar la función reshape() que devuelve un nuevo array con el tamaño cambiado.

Numpy

- Numpy permite apilar arrays tanto por columnas como por filas. Para ello se usan las funciones vstack() y hstack().

```
>>> a
array([[ 1.,  1.],
       [ 5.,  8.]])
>>> b
array([[ 3.,  3.],
       [ 6.,  0.]])
>>> vstack((a,b))
array([[ 1.,  1.],
       [ 5.,  8.],
       [ 3.,  3.],
       [ 6.,  0.]])
>>> hstack((a,b))
array([[ 1.,  1.,  3.,  3.],
       [ 5.,  8.,  6.,  0.]])
```

- Existen funciones equivalentes a vstack() y hstack() pero que solo pueden usarse para arrays de una dimensión: column_stack(), row_stack(), r_() y c_()

Numpy

- A veces puede ser interesante dividir un array, para lo cual Numpy proporciona la función hsplit(), vsplit() y array_split() que permiten dividir un array por su dimensión horizontal, vertical o especificando que dimensión se desea respectivamente.

```
>>> a =
array([[ 8.,  8.,  3.,  9.,  0.,  4.,  3.,  0.,  0.,  6.,  4.,  4.],
       [ 0.,  3.,  2.,  9.,  6.,  0.,  4.,  5.,  7.,  5.,  1.,  4.]])
>>> hsplit(a,3)   :
[array([[ 8.,  8.,  3.,  9.],
       [ 0.,  3.,  2.,  9.]]), array([[ 0.,  4.,  3.,  0.],
       [ 6.,  0.,  4.,  5.]]), array([[ 0.,  6.,  4.,  4.],
       [ 7.,  5.,  1.,  4.]])]
>>> hsplit(a,(3,4))
[array([[ 8.,  8.,  3.],
       [ 0.,  3.,  2.]]), array([[ 9.],
       [ 9.]]), array([[ 0.,  4.,  3.,  0.,  0.,  6.,  4.,  4.],
       [ 6.,  0.,  4.,  5.,  7.,  5.,  1.,  4.]])]
```

Numpy

- Cuando se trabaja con arrays a veces interesa que las operaciones afecten al array y en otras ocasiones interesa que el array se mantenga intacto:
  - Las asignaciones no hacen copia del array o sus datos. Se trata del mismo array. Lo mismo ocurre con las llamadas a funciones.

```
>>> a = arange(12)
>>> b = a
>>> b is a
True
>>> b.shape = 3,4
>>> a.shape
(3, 4)
```

Numpy

- Otra posibilidad es crear una vista con el método view(), creándose un nuevo array que toma los datos del array original, pero no se trata del mismo array, y no se ve afectado por las operaciones que se hagan sobre la vista.

```
>>> c = a.view()
>>> c is a
False
>>> c.shape = 2,6
>>> a.shape
(3, 4)
>>> c[0,4] = 1234
>>> a
array([[   0,    1,    2,    3],
       [1234,    5,    6,    7],
       [   8,    9,   10,   11]])
```

Numpy

- Por ultimo con el método copy() se realiza una copia independiente del array.

```
>>> d = a.copy()
>>> d is a
False
>>> d[0,0] = 9999
>>> a
array([[   0,   10,   10,    3],
       [1234,   10,   10,    7],
       [   8,   10,   10,   11]])
```

Numpy

- Otras posibilidades de Numpy:
  - Operaciones sobre arrays de booleanos. Los métodos any and all, permiten chequear si algún valor es cierto o si todos los valores son ciertos respectivamente.

```
In [149]: bools=array([False, False, True,True])

In [150]: bools.any()
Out[150]: True

In [151]: bools.all()
Out[151]: False
```

Numpy

- Ordenación de arrays usando el método sort(). En el caso de arrays multidimensionales hay que indicar la dimensión sobre la que se ordena.

```
In [153]: arr=array([0.609,0.462, 0.098,0.123,0.987,0.0185,-1.345,-0.567])

In [154]: arr.sort()

In [155]: arr

Out[155]: array([-1.345 , -0.567 ,  0.0185,  0.098 ,  0.123 ,  0.462 ,  0.609 ,
                  0.987 ])

In [156]: b=array([[   0,    1,    2,    3],
                   [1234,    5,    6,    7],
                   [   8,    9,   10,   11]])

In [157]: b.sort(1)

In [158]: b

Out[158]: array([[   0,    1,    2,    3],
                 [   5,    6,    7, 1234],
                 [   8,    9,   10,   11]])
```

Numpy

- Es posible manipular polinomios utilizando la libreria polynomials.

```
In [162]: from numpy.polynomial import Polynomial as P

In [163]: polinomio=P([1,2,3])

In [164]: print(polinomio)
poly([ 1.  2.  3.])

In [166]: print(polinomio+polinomio)
poly([ 2.  4.  6.])
```

Numpy

- Sobre los arrays es posible realizar diferentes tipos de productos entre arrays:
  - Producto interno , usando la función inner()
  - Producto externo, usando la función outer()
  - Producto cruzado , usando la función cross()

Numpy

```
In [167]: a= array([2,34,4])

In [168]: b=array([2,5,6])

In [169]: z=inner(a,b)

In [170]: z
Out[170]: 198

In [171]: z=outer(a,b)

In [172]: z
Out[172]: array([[  4,  10,  12],
                 [ 68, 170, 204],
                 [  8,  20,  24]])

In [173]: z=cross(a,b)

In [175]: z
Out[175]: array([184,  -4, -58])
```

Numpy

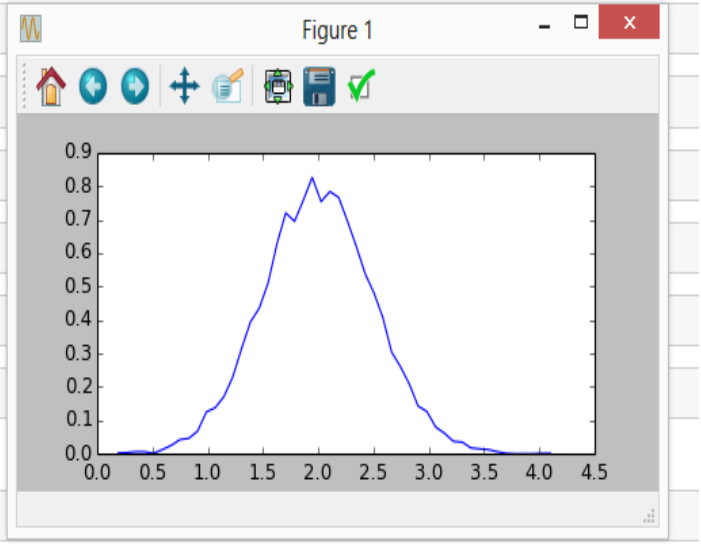
- Para definer una matriz se puede optar por crear un array multidimensional o bien utilizar la libreria matrix que tiene los mismos métodos que array.

```
In [176]: m=matrix([[2,3],[3,2]])

In [177]: m

Out[177]: matrix([[2, 3],
                  [3, 2]])

In [178]: n=matrix([[1,2],[-5,2]])

In [179]: n

Out[179]: matrix([[ 1,  2],
                  [-5,  2]])

In [180]: n+m

Out[180]: matrix([[ 3,  5],
                  [-2,  4]])
```

Numpy

- Numpy proporciona una función que puede ser aplicada a un array y retorna un par de vectores los valores y rangos del histograma, pero no los representa gráficamente. Para ello es necesario usar la función plot de la libreria matplotlib.

```
In [209]: import numpy

In [210]: import pylab

In [212]: mu, sigma = 2, 0.5

In [213]: v = numpy.random.normal(mu,sigma,10000)

In [214]: (n, bins) = numpy.histogram(v, bins=50, normed=True)

In [215]: pylab.plot(.5*(bins[1:]+bins[:-1]), n)
Out[215]: [<matplotlib.lines.Line2D at 0xa5d4d30>]

In [*]: pylab.show()
```



Numpy

- En las siguientes transparencias se van a mostrar agrupados algunas funciones útiles que proporciona Numpy para realizar operaciones matemáticas, algrebra lineal, operaciones de conjuntos,…

Numpy

- Métodos matemáticos de Numpy

Metodo	Descripción
sum	Suma todos los elementos de un array.
mean	Media aritmética.
std, var	Desviación estándar y varianza
min, max	Mínimo y máximo
argmin, argmax	Índices del mínimo y del máximo elemento.
cumsum	Suma acumulativa de elementos empezando por 0
cumprod	Producto acumulativo de elementos empezando por 1

Numpy

- Algunos ejemplos:

```
In [181]: import numpy.random as R

In [182]: arr=R.randn(5,4)

In [183]: arr.mean()
Out[183]: 0.13742699092643684

In [184]: arr.sum()
Out[184]: 2.7485398185287369

In [186]: arr.mean(axis=1)
Out[186]: array([ 0.20443882,  0.46320307, -0.98564746,  0.6424256 ,  0.36271492])
```

Numpy

- Operaciones sobre conjuntos en Numpy

Método	Descripción
unique(x)	Conjunto ordenado de elementos únicos en x
intersect1d(x, y)	Conjunto ordenado de elementos comunes en x e y
union1d(x, y)	Conjunto ordenado de la unión de elementos de x e y
in1d(x, y)	Array de booleanos que indican si cada elemento de x está contenido en y.
setdiff1d(x, y)	Conjunto diferencia de elementos que están en x y no en y
setxor1d(x, y)	Conjunto de la diferencia simétrica. Elementos que están en uno u otro, pero no en ambos.

Numpy

- Algunos ejemplos:

```
In [188]: nombres1=array(["Juan","Luisa","Juan","Antonio","Pepe", "José","Daniel","Eva"])

In [189]: nombres2=array(["Juan","Maria","Juan","Alfonso","Pepe", "Laura","Daniel","Eva"])

In [190]: unique(nombres1)

Out[190]: array(['Antonio', 'Daniel', 'Eva', 'Jos\xc3\xa9', 'Juan', 'Luisa', 'Pepe'],
                dtype='|S7')

In [191]: intersect1d(nombres1,nombres2)

Out[191]: array(['Daniel', 'Eva', 'Juan', 'Pepe'],
                dtype='|S7')

In [192]: union1d(nombres1,nombres2)

Out[192]: array(['Alfonso', 'Antonio', 'Daniel', 'Eva', 'Jos\xc3\xa9', 'Juan',
                 'Laura', 'Luisa', 'Maria', 'Pepe'],
                dtype='|S7')
```

Numpy

- Operaciones de algebra lineal en Numpy(libreria numpy.linalg)

Función	Descripción
diag	Retorna los elementos de la diagonal de una matriz cuadrada como un array de 1 dimensión o convierte una matriza de una dimensión es un matriz cuadrada con cero en la diagonal.
dot	Multiplicación de matrices
trace	Cálcula la suma de los elementos de la diagonal
det	Cálcula el determinante de una matriz cuadrada
eig	Cálcula los autovalores y autovectores de una matriz cuadrada
inv	Calcula la inversa de una matriz cuadrada
pinv	Calcula la pseudoinversa de Moore-Penrose de una matriz cuadrada
qr	Cálcula la descomposión QR
svd	Cálcula la descomposición en valores singulares (SVD)
solve	Resuelve el sistema lineal Ax = b para x dónde A es una matriz cuadrada
lstsq	Cálcula la solución de mínimos cuadrados de y = Xb

Numpy

- Algunos ejemplos:

```
In [193]: from numpy.linalg import inv,qr

In [194]: x=array([[1,2,3],[4,5,6]])

In [200]: y=array([[6,7],[4,8],[9,20]])

In [203]: z=x.dot(y)

In [204]: inv(z)

Out[204]: array([[-0.44131455,  0.19483568],
                 [ 0.23004695, -0.09624413]])

In [205]: q,r=qr(z)

In [206]: r

Out[206]: array([[-106.23088063, -205.4675615 ],
                 [   0.        ,   -4.01013338]])
```

Numpy

- Aleatoriedad en Numpy(librería numpy.random)

Function	Description
seed	Semilla para el generador de números aleatorios
permutation	Retorna una permutación aleatoria de una secuencia o retorna un rango de valores permutado.
shuffle	Permuta aleatoriamente una secuencia
rand	Dibuja casos de una distribución uniforme
randint	Dibuja enteros aleatorios desde un rango dado.
randn	Dibuja casos de una distribución normal con media 0 y desviación estándar 1.
binomial	Dibuja casos de una distribución binomial
normal	Dibuja casos de una distribución normal
beta	Dibuja casos de una distribución beta
chisquare	Dibuja casos de una distribución chi-square
gamma	Dibuja casos de una distribución Gamma
uniform	Dibuja casos de una distribución uniforme [0, 1)

Numpy

- Algunos ejemplos:

```
In [207]: casos=R.normal(size=(4,4))

In [208]: casos

Out[208]: array([[-1.87280505, -1.25777016, -1.35338977, -1.55918987],
                 [ 0.46970949,  2.21909482, -0.08786561,  0.49332454],
                 [ 0.20774166, -0.07832509, -1.16346918, -1.70062967],
                 [ 0.41671736,  0.1496858 , -0.54470983,  0.14271311]])
```

Numpy

- Para profundizar en:

  http://docs.scipy.org/doc/