Python

Handling vectors with NumPy

Ricco Rakotomalala

http://data-mining-tutorials.blogspot.fr/

Numpy ?

• NumPy (numerical python) is a package for scientific computing. It provides tools for handling n-dimensional arrays (especially vectors and matrices).

• The objects are all the same type into a NumPy arrays structure

• The package offers a large number of routines for fast access to data (e.g. search, extraction), for various manipulations (e.g. sorting), for calculations (e.g. statistical computing)

• Numpy arrays are more efficient (speed, volume management) than the usual Python collections (list, tuple).

• Numpy arrays are underlying to many packages dedicated to scientific computing in Python.

• Note that a vector is actually a 1 single dimension array

To go further, see the reference manual (used to prepare this slideshow).

http://docs.scipy.org/doc/numpy/reference/index.html

Creation on the fly, generation of a sequence, loading from a file

CREATING A NUMPY VECTOR

Array creation

First, we must import the module “numpy”

```
import numpy as np
```

np is the alias used for accessing to the routines of the package 'numpy '.

Converting Python array_like objects (e.g. list)

```
a = np.array([1.2,2.5,3.2,1.8])
```

[ ] is a list of values (float)

Information about the structure

```
#object type
print(type(a))  #<class ‘numpy.ndarray’>
#data type
print(a.dtype)  #float64
#number of dimensions
print(a.ndim)  #1 (we have 2 if it is a matrix, etc.)
#number of rows and columns
print(a.shape)  #(4,) → tuple! 4 elements for the 1st  dim (n°0)
#total number of elements
print(a.size)  #4, nb.rows  x nb.columns if a matrix
```

Setting the data type

Specifying the data type can be implicit or explicit

```
#creating a vector – implicit typing
a = np.array([1,2,4])
print(a.dtype)  #int32

#creating a vector – explicit typing – preferable !
a = np.array([1,2,4],dtype=float)
print(a.dtype)  #float64
print(a)  #[1.  2.  4.]

#a vector of Boolean values is possible
b = np.array([True,False,True,True], dtype=bool)
print(b) #[True  False  True True]
```

Creating an array with objects of non-standard type is possible

```
# the array value may be an object
a = np.array([{"Toto":(45,2000)},{"Tata":(34,1500)}])
print(a.dtype) #object
```

Creating sequence of numbers

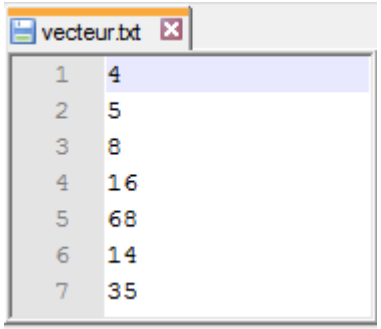
```
#evenly spaced values within a given interval (step = 1 here)
a = np.arange(start=0,stop=10)
print(a) #[0  1  2  3  4  5  6  7  8  9], the last value is excluded

#specifying the step property
a = np.arange(start=0,stop=10,step=2)
print(a) #[0  2  4  6  8]

#evenly spaced value, specify the number of elements
a = np.linspace(start=0,stop=10,num=5)
print(a) #[0.  2.5  5.  7.5  10.], the last value is included here

#repeating 5 times the value 1 – number  of values = 5 (1 dimension)
a = np.ones(shape=5)
print(a) # [1.  1.  1.  1.  1.]

#repeating 5 times (1 dimension) the value 3.2
a = np.full(shape=5,fill_value=3.2)
print(a) #[3.2  3.2  3.2  3.2  3.2]
```

Loading a vector from a data file

The values can be stored in a text file (loadtxt for reading, savetxt for writing)

```
#loading from a text file
#we can set the type of the data
a = np.loadtxt("vecteur.txt",dtype=float)
print(a) #[4.  5.  8.  16.  68.  14.  35.]
```

Only 1 column here



Note: If necessary, we change the default directory with the function **chdir()** from the **os** module (that must be imported)

We can convert a Python sequence type in a "numpy" array

```
#lst is a list of values (float)
lst = [1.2,3.1,4.5]
print(type(lst)) #<class 'list'>
#converting the list
a = np.asarray(lst,dtype=float)
print(type(a)) #<class 'numpy.ndarray'>
print(a) #[1.2  3.1  4.5]
```

Adding and removing elements

Add a value in last position

```
#a is a vector
a = np.array([1.2,2.5,3.2,1.8])
#append the value 10 into the vector a
a = np.append(a,10)
print(a) #[1.2  2.5  3.2  1.8  10.]
```

Remove a value from its index

```
#remove the value n°2
b = np.delete(a,2) #a range of indices can be used
print(b) #[1.2  2.5  1.8  10.]
```

Modify the size of a vector

```
a = np.array([1,2,3])
#adding two cells
#fills zero for the new cell
a.resize(new_shape=5)
print(a) #[1  2  3  0  0]
```

Concatenation of vectors

```
#concatenate 2 vectors
x = np.array([1,2,5,6])
y = np.array([2,1,7,4])
z = np.append(x,y)
print(z) #[1  2  5  6  2  1  7  4]
```

Indexing with indices or Boolean array

EXTRACTING VALUES

Indexed access – v = np.array([1.2,7.4,4.2,8.5,6.3])

```
#printing all the values
print(v)
#or
print(v[:]) # note the role of :  ; here, from start to end

#indexed access - first value
print(v[0]) # 1.2 – the first index is 0 (zero)

#last value
print(v[v.size-1]) #6.3, v.size is okay because v is a vector

#contiguous indices
print(v[1:3]) # [7.4  4.2]

#extreme values, start to 3 (not included)
print(v[:3]) # [1.2  7.4  4.2]

#extreme values, 2 to end
print(v[2:]) # [4.2  8.5  6.3]

#negative indices
print(v[-1]) # 6.3, last value

#negative indices
print(v[-3:]) # [4.2  8.5  6.3], 3 last values
```

Note : Apart from singletons, the generated vectors are of type numpy.ndarray

Indexed access – Generic approach - v = np.array([1.2,7.4,4.2,8.5,6.3])

Generic writing of indices is : first:last:step

last is not included

```
#value n°1 to n°3 with a step = 1
print(v[1:4:1) # [7.4, 4.2, 8.5]

#step = 1 is implicit
print(v[1:4]) # [7.4, 4.2, 8.5]

#n°0 to n°2 with a step = 2
print(v[0:3:2]) # [1.2, 4.2]

#the step can be negative, n°3 to n°1 with a step = -1
print (v[3:0:-1]) # [8.5, 4.2, 7.4]

#we can use this idea (negative step) to reverse a vector
print(v[::-1]) # [6.3, 8.5, 4.2, 7.4, 1.2]
```

```
#extraction with a vector of Booleans
#if b too short, the remainder is considered False
b = np.array([False,True,False,True,False],dtype=bool)
print(v[b]) # [7.4  8.5]

#one can use a condition for extraction
print(v[v < 7]) # [1.2  4.2  6.3]

#because a condition generates a vector of Booleans
b = v < 7
print(b) # [True  False  True  False  True]
print(type(b)) # <class 'numpy.ndarray'>

#one can use also the extract() function
print(np.extract(v < 7, v)) # [1.2  4.2  6.3]
```

```
#get the max value
print(np.max(v)) # 8.5

#find the index of the max value
print(np.argmax(v)) # 3

#sort the values
print(np.sort(v)) # [1.2  4.2  6.3  7.4  8.5]

#get the indices that would sort the values
print(np.argsort(v)) # [0  2  4  1  3]

#unique elements of the vector
a = np.array([1,2,2,1,1,2])
print(np.unique(a)) # [1  2]
```

Note : The equivalent exists for min()

STATISTICAL ROUTINES

```
#mean
print(np.mean(v)) # 5.52

#median
print(np.median(v)) # 6.3

#variance
print(np.var(v)) # 6.6856

#percentile
print(np.percentile(v,50)) #6.3 (50% = médiane)

#sum
print(np.sum(v)) # 27.6

#cumulative sum
print(np.cumsum(v)) # [1.2  8.6  12.8  21.3  27.6]
```

The statistical functions are not numerous, we will need SciPy (and other)

Calculations between vectors – “Elementwise” operations

```
#two vectors : x and y
x = np.array([1.2,1.3,1.0])
y = np.array([2.1,0.8,1.3])
#multiplication
print(x*y) # [2.52  1.04  1.3]
#addition
print(x+y) # [3.3  2.1  2.3]
#multiplication by a scalar
print(2*x) # [2.4  2.6  2. ]

#comparison of vectors
x = np.array([1,2,5,6])
y = np.array([2,1,7,4])
b = x > y
print(b) # [False  True  False  True]

#logical operations
a = np.array([True,True,False,True],dtype=bool)
b = np.array([True,False,True,False],dtype=bool)
#AND operator
np.logical_and(a,b) # [True  False  False  False]
#XOR operator (exclusive or)
np.logical_xor(a,b) # [False  True  True  True]
```

The calculations are made in the element wise fashion - We have the same principle under R.

The list of functions is long.
See :
http://docs.scipy.org/doc/numpy/reference/routines.logic.html

```
x = np.array([1.2,1.3,1.0])
y = np.array([2.1,0.8,1.3])

#dot product of two vectors
z = np.vdot(x,y)
print(z) # 4.86

#or, equivalently
print(np.sum(x*y))  # 4.86

#vector norm
n = np.linalg.norm(x)
print(n) # 2.03

#or, equivalently
import math
print(math.sqrt(np.sum(x**2))) # 2.03
```

The functions for matrix operations exist, some of them can be applied to vectors

A vector of values (especially integer) can be considered as a set of values.

```
#set routines
x = np.array([1,2,5,6])
y = np.array([2,1,7,4])

#intersection
print(np.intersect1d(x,y)) # [1  2]

#union – this is not a concatenation
print(np.union1d(x,y)) # [1  2  4  5  6  7]

#difference i.e. values in x but not in y
print(np.setdiff1d(x,y)) # [5  6]
```

References

Course materials (in French)
http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html

Python website
Welcome to Python - https://www.python.org/
Python **3.4.3** documentation - https://docs.python.org/3/index.html

NumPy Manual
Numpy User Guide and Numpy Reference

POLLS (KDnuggets)
Data Mining / Analytics Tools Used
Python, 4th in 2015
Primary programming language for Analytics, Data Mining, Data Science tasks
Python, 2nd in 2015 (next R)