numpy

January 15, 2021

1 Module 0 - NumPy

A PDF version of this notebook is available at Module 0 - NumPy

1.1 NumPy (Numerical Python)

This module provides advanced features to store and operate big data sets.

The convention is to import the numpy module with the alias `np`. Note that you do not need to do this.

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

An object in Python is dynamic, meaning it can change variable type without defining previously. E.g.

In C, you need to define a variable `x` as an integer:

`int x = 4`

In Python, instead, you can assign the variable `x = 4`, and reassign it to `x = "Hello"` without errors. This results in Python objects containing much more information than objects in compiled languages such as C.

A list contains objects, each of which needs information on the object type, location, among other properties.

If all the objects of a list are of the same type, there is redundant information stored in the lists. In this particular case a more efficient is numpy array.

```
[2]: ## this is a numpy array of integers
     np.array([1,4,2,5,3])
```

```
[2]: array([1, 4, 2, 5, 3])
```

```
[3]: ## this is a numpy array of floats
     np.array([3.2, 2.1, 1.0, 3.0])
```

[3]:
```
array([3.2, 2.1, 1. , 3. ])
```

[4]:
```
## If one element is float, the numpy array converts the rest to floats
np.array([2.3, 1, 4, 6])
```

[4]:
```
array([2.3, 1. , 4. , 6. ])
```

[5]:
```
## You can state the data type using the dtype keyword
np.array([1,2,3,4], dtype = 'float32')
```

[5]:
```
array([1., 2., 3., 4.], dtype=float32)
```

[6]:
```
## If one element is a string, then all the elements are converted to strings
np.array(['a', 'b', 1])
```

[6]:
```
array(['a', 'b', '1'], dtype='<U1')
```

1.1.1 *Multidimensional Arrays and Matrices*

1.1.2 Vectors

[7]:
```
np.array([1,2,3])
```

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

1.1.3 Matrices

[8]:
```
## note that each row is a list
## each list has to be separated by a comma
## and all lists enclosed within a square brackets
np.array([[1,2,3],
          [3,4,5],
          [6,7,8]])
```

[8]:
```
array([[1, 2, 3],
       [3, 4, 5],
       [6, 7, 8]])
```

1.1.4 Tensors

[9]:

```
x3 = np.array([[[10, 11, 12], [13, 14, 15], [16, 17, 18]],
               [[20, 21, 22], [23, 24, 25], [26, 27, 28]],
               [[30, 31, 32], [33, 34, 35], [36, 37, 38]]])
x3
```

[9]:

```
array([[[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]],

       [[20, 21, 22],
        [23, 24, 25],
        [26, 27, 28]],

       [[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38]]])
```

[10]:

```
x3.shape
```

[10]:

```
(3, 3, 3)
```

[11]:

```
## We can generate a random 3d tensor by setting the size to a tuple of length 3
x4 = np.random.randint(10, size = (3,4,5))
x4
```

[11]:

```
array([[[4, 6, 3, 0, 2],
        [1, 7, 7, 8, 8],
        [1, 1, 6, 5, 9],
        [0, 1, 3, 3, 1]],

       [[8, 4, 1, 2, 1],
        [7, 8, 3, 2, 0],
        [2, 5, 6, 7, 5],
        [5, 7, 3, 3, 8]],

       [[7, 1, 6, 7, 4],
        [6, 6, 8, 7, 8],
        [2, 3, 7, 5, 0],
        [7, 8, 8, 0, 3]]])
```

[12]:

```
x4.shape
```

[12]:

```
(3, 4, 5)
```

1.1.5 *Generating data with NumPy*

```
[13]: ## arrays with zeros
      np.zeros(10)
```

```
[13]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
[14]: ## arrays of ones in 3x5 matrix (takes a tuple)
      np.ones((3,5), dtype = 'int')
```

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

```
[15]: ## matrix of constants
      np.full((3,5), 3.14)
```

```
[15]: array([[3.14, 3.14, 3.14, 3.14, 3.14],
             [3.14, 3.14, 3.14, 3.14, 3.14],
             [3.14, 3.14, 3.14, 3.14, 3.14]])
```

```
[16]: ## random uniform values 3x3 matrix
      np.set_printoptions(suppress=True) ## suppress exponential notation
      np.random.random((3,3))
```

```
[16]: array([[0.12056634, 0.58969043, 0.24187326],
             [0.49784566, 0.12539894, 0.72615596],
             [0.0835631 , 0.8799808 , 0.79407916]])
```

```
[17]: ## random normals (3x3) matrix mean 0, var 1
      np.random.normal(0,1,(3,3))
```

```
[17]: array([[ 0.81990423,  1.50828614, -0.57727806],
             [-0.31869337,  0.22796213,  0.87319467],
             [-1.30866627,  0.6974232 ,  0.48807457]])
```

1.1.6 *Numpy Array Attributes*

You can extract a numpy array attributes using . after the array name followed by the keyword.

```
[18]: ## dimensions
      x3.ndim
```

```
[18]: 3
```

```
[19]: ## shape
      x3.shape
```

```
[19]: (3, 3, 3)
```

```
[20]: ## size
      x3.size
```

```
[20]: 27
```

```
[21]: ## Type
      x3.dtype
```

```
[21]: dtype('int64')
```

1.1.7 *Indexing*

Indexing in NumPy is similar to that of lists and tuple with the acces to the ith value starting at zero.

```
[22]: ## random seed set to 630
      np.random.seed(630)

      ## 6 random integers max at 10
      x1 = np.random.randint(10, size = 6)
```

```
[23]: x1
```

```
[23]: array([8, 4, 0, 6, 3, 3])
```

```
[24]: ## first elememt
      x1[0]
```

```
[24]: 8
```

```
[25]: ## 3rd element
      x1[2]
```

```
[25]: 0
```

```
[26]: ## 3x2 matrix of random int maxed at 10
      x2 = np.random.randint(10, size = (3,2))
```

```
[27]: x2
```

```
[27]: array([[7, 1],
             [9, 5],
             [8, 7]])
```

[28]:

```
## we can now use double indexing with first index refering to rows and second␣
↪to columns
## e.g., first row, second column
x2[0,1]
```

[28]:

```
1
```

[ ]:

```
## we can modify a numpy array
x2[0,1] = 10
```

[37]:

```
x2
```

[37]:

```
array([[ 7, 10],
       [ 9,  5],
       [ 8,  7]])
```

1.1.8 *Slicing*

[38]:

```
## 4x3 random normals with mean 0 and var 1
x3 = np.random.normal(0, 1, size = (4,3))
```

[39]:

```
x3
```

[39]:

```
array([[-0.22438947, -0.55092701,  1.82260416],
       [ 0.02365925, -1.22770132, -0.41255385],
       [ 0.94216283,  0.61083174, -0.31554172],
       [-1.63817692, -0.11512212,  0.79362157]])
```

[40]:

```
## Rows 0, and 1, 2nd column
x3[0:2, 1]
```

[40]:

```
array([-0.55092701, -1.22770132])
```

[41]:

```
## first three rows (0, 1 and 2, not including 3), and third column
x3[:3, 2]
```

[41]:

```
array([ 1.82260416, -0.41255385, -0.31554172])
```

[42]:

```
## every other row, columns 0 and 1
x3[::2, 0:2]
```

[42]:

```
array([[-0.22438947, -0.55092701],
       [ 0.94216283,  0.61083174]])
```

[43]:

```
## reversed rows, all columns
x3[::-1,:]
```

[43]:

```
array([[-1.63817692, -0.11512212,  0.79362157],
       [ 0.94216283,  0.61083174, -0.31554172],
       [ 0.02365925, -1.22770132, -0.41255385],
       [-0.22438947, -0.55092701,  1.82260416]])
```

[44]:

```
## all rows, first column
x3[:,0]
```

[44]:

```
array([-0.22438947,  0.02365925,  0.94216283, -1.63817692])
```

1.1.9 *Reshaping Arrays*

Reshaping arrays is important in ML models for image recognition.

[45]:

```
## np.arange creates a seq of numbers
grid = np.arange(1,10)
grid
```

[45]:

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[46]:

```
## we can reshape to a 3x3 matrix with the reshape method. We need to pass a␣
 ↪tuple (3,3)
new_grid = grid.reshape((3,3))
new_grid
```

[46]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

1.1.10 NumPy Cheat Sheet

Pandas and Numpy are too extensive. We will be learning more about them throughout the semester. Below is a summary sheet from datacamp for NumPy.

[ ]:

```

```