numpy

Ben Bolker

18:27 26 February 2017

Installation

I am hoping that everyone has successfully followed the instructions for installing Anaconda+Python. If you have, then you should be able to just `import numpy as np` to get access to `numpy`. If you have not, you should try to resolve the issue as soon as possible; if absolutely necessary you can run code in PythonAnywhere.

Arrays

The `array()` is numpy's main data structure.

- `np.array()` constructs an array from an existing Python object
- creates a *homogeneous* order set of numeric values (e.g. floating point (`float64`) or integer (`int64`)
- 1-D: vectors
  - vectorized/elementwise arithmetic operators!
    - arithmetic now works on *elements*, not on the list: `a+1` creates a new array with 1 added to every element; `a+=1` adds 1 to every element
  - create from list or tuple (specify type explicitly; `np.array(x,dtype)` where `type` is `int`, `str`, `float`, `complex` ...)
  - the `.dtype` **attribute** tells you the type of an array

```
import numpy as np
```

```
a2 = np.array([1,2],type="float")
print(a2.dtype)
```

- array slicing and indexing of 1-D arrays works the same way as lists/tuples/strings
- arrays are **mutable** like lists/dictionaries, so we can set elements (e.g. `a[1]=0`)
- or use the `.copy()` method (works for lists etc. too!)
- create arrays directly:
  - `np.arange(start,stop,step,dtype)`: like `range`
  - `np.zeros()`: all zeros
  - `np.ones()`: all ones
  - `np.linspace(start, stop, num=50, endpoint, dtype)` (similar to `np.arange`, but specify number of elements rather than step size):
- from `help("np.arange")`:

  When using a non-integer step, such as 0.1, the results will often not be consistent. It is better to use `linspace` for these cases.

Multi-dimensional arrays

- create by passing a list of lists/tuple of tuples
- index via `a[i,j]` rather than `a[i][j]`
- arithmetic is still elementwise; in particular, multiplication is elementwise (*Hadamard product*)
- multi-dimensional slicing:
  - e.g. `a[:,0:2]` takes the first two columns
  - how would you get odd rows?

  - flip the order of columns?
- dimensions: `a.shape` (**note** no parentheses! `shape` is an **attribute**, like `a.dtype`
- `a.fill()`: fill an array with a single (*scalar*) value
- `a.reshape(shape)`: change dimensions; **row-major order**
- `a.transpose()`
- `a.flatten()`: convert to a 1-D array
- `np.concatenate((a1,a2,a3),axis=0)`: combine arrays
- arrays that do not match in the number of dimensions will be **broadcasted** (i.e. by Python to perform mathematical operations.

Matrices

2-D arrays with special features

- `dot` function: regular matrix multiplication
- `np.eye()`: identity matrix
- `np.diag(v,k=0)` diagonal matrix
- `np.linalg` submodule

Logical arrays

- vectorized *logical*/comparison operations and indexing
- e.g. `a>0` gives array of `bool`
- `a[a>0]` selects only positive elements of `a`
- `any` and `all` functions

Many more useful array methods

`.sum()`, `.prod()`, `.max()`, `.min()`, `.argmax()`, `.argmin()`, `.mean()` (note `axis` argument), `sd()`, `rank()` ...

examples

- (32) take a 2-dimensional array and scale it so the columns have mean 0 and standard deviation 1
- (18) Create a 3x3 matrix with row values ranging from 0 to 2 (various choices)
- (33) sort an array by the nth column

to do in class

- calculate the mean of the squares of the natural numbers up to 7
- create a 5 x 5 array with row values ranging from 0 to 1 by 0.2
- create a 3 x 7 array containing the values 0 to 20 and a 7 x 3 array containing the values 0 to 20 and matrix-multiply them: result should be

```
array([[ 273,  294,  315],
   [ 714,  784,  854],
   [1155, 1274, 1393]])
```

References:

- official docs
- useful unofficial intro
- numpy examples