Tabular Numeric Data

LECTURE 4

Organization of Lecture 4

- Creating Arrays
- Transposing and Reshaping
- Indexing and Slicing
- Broadcasting
- Demystifying Universal Functions
- Understanding Conditional Functions
- Aggregating and Order Arrays
- Treating Arrays as Sets
- Saving and Reading Arrays
- Generating a Synthetic Sine Wave?

Working with Tabular Numeric Data

- Often textual raw data is actually numbers
- Excel, **`CSV`** files and DB tables may contain TON of data
- *Core* Python is excellent text-processing too, not so good at numeric processing
- Import **`numpy`** (numeric Python) – interface to efficient, parallelizable functions for numerical operations.
- Module **`numpy`** provides an array (and toolbox of functions)
- Module **`numpy`** supports random numbers, data aggregation, linear algebra, Fourier transform, etc.
- For terabytes of numberic data, use **`h5py`** module

Creating arrays

- `numpy` arrays more compact and faster than Python lists
- `numpy` arrays are homogeneous
- Several ways to create arrays
  - function `array()`
  - functions `ones()`, `zeros()` and `empty()`
  - function `eye()` for identity
  - functions `arange()`
  - function `copy()`

Creating arrays cont.

- function `array`() creates array from array-like data (list, tuple or another array)
  - Actually creates link (not a copy) – think of it as a "view" of the underlying data
  - Want a copy – pass `copy=True` as parameter to array function
- functions `ones`(), `zeros`() and `empty`() construct arrays of all ones, all zeros, and all uninitialized entries, resp.
  - Requires a shape parameter – a list or tuple of array dimensions
- function `eye (N, M, k, dtype)` – contructs `NxM` identity matrix with ones ones on the `k`th diagonal and zeros elsewhere. If `M=None`, then `M=N`.
- Function `arange([start,] stop, [step,] dtype)` – creates regularly spaced numbers, `stop` can be smaller than `start`, but `step` will have to be < 0.
- function `copy()` – another way to copy

Creating Arrays Examples

```
import numpy as np
```

```
numbers = np.array(range(1,11), copy=True)
print (numbers, "\n")

ones = np.ones([2,4], dtype = np.float64)
print(ones, "\n")

empty = np.empty([2,3], dtype=int)
print(empty, "\n")

#array attributes
print(ones.shape, numbers.ndim, empty.dtype, "\n")

eye = np.eye (3, k=1, dtype = int)
print (eye, "\n")

np_numbers = np.arange(2,5, 0.25)
print (np_numbers, "\n")

np_inumbers = np_numbers.astype(int)
print (np_inumbers, "\n")

np_inumbers_copy = np_inumbers.copy()
print (np_inumbers_copy, "\n")
```

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

[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]

[[       45114048                0 140525901857456]
 [140525625547504 140526233703024 140526234064944]]

(2, 4) 1 int64

[[0 1 0]
 [0 0 1]
 [0 0 0]]

[2.   2.25 2.5  2.75 3.   3.25 3.5  3.75 4.   4.25 4.5  4.75]

[2 2 2 2 3 3 3 3 4 4 4 4]

[2 2 2 2 3 3 3 3 4 4 4 4]
```

Transposing and reshaping

- **numpy** arrays can change (already saw changing of type)
  - shape (dimensions)

```
sap = np.array(["MMM", "ABT", "ATT", "MSN", "NTFX", "WAL"])
sap2d = sap.reshape(2,3)
```

```
[['MMM' 'ABT' 'ATT']
 ['MSN' 'NTFX' 'WAL']]
```

  - Orientation (or transposed view)

```
print(sap2d.T)  #swaps 2D axes
```

```
[['MMM' 'MSN']
 ['ABT' 'NTFX']
 ['ATT' 'WAL']]
```

  - Swap two axes in multidimensional array

```
swapaxes(axis1, axis2)   #passing 0, 1 does what previous statement does, if 2D array
```

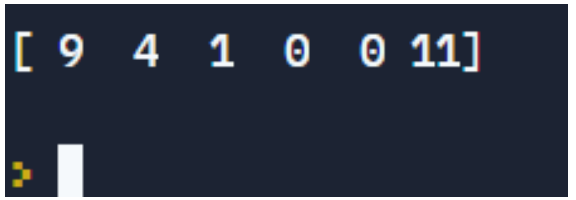
  - Swap multiple axes in multidimensional array

```
sap3d.transpose ((1, 2, 0))  #
```

Indexing and Slicing

- **`numpy arrays`** support same indexing and splicing operations as Python lists
- **`numpy arrays`** support Boolean indexing
  - Array of Boolean values can be used as an index and select items from another array where the Boolean value is True. Useful for *data cleaning*.

```
dirty = np.array([9, 4, 1, -1, -2, 11])
whos_dirty = dirty < 0 #boolean array returned
dirty[whos_dirty] = 0 #replaces negative values with 0
print (dirty, "\n")
```



Broadcasting

- Basically *vectorized operations on arrays*
- As long as two arrays are same shape, can add, multiply,…etc.
- Without **`numpy`**, need a for loop or list comprehension

```
a = np.arange(4)
b = np.arange(1,5)
print (a, "+", b, " = ", a+b, "\n")
```

```
[0 1 2 3] + [1 2 3 4]  =  [1 3 5 7]
```

- Note: * operator in numpy, multiplies; in Core Python, it replicates.

Universal Functions Example

- Stock prices for 8 companies recorded after and before a weekend
- Want to know which stock prices fell over the weekend

```
stocks = np.array([140.49, 0.97, 40.68, 41.53, 55.7, 57.21, 98.2, 99.19, \
                   109.96, 111.47, 35.71, 36.27, 87.85, 89.11, 30.22, 30.91])
stocks = stocks.reshape(8,2).T #2 rows of 8 with after prices in row 1
print (stocks)
```

```
[[140.49  40.68  55.7   98.2  109.96  35.71  87.85  30.22]
 [  0.97  41.53  57.21  99.19 111.47  36.27  89.11  30.91]]
```

- Now apply ufunc greater to both rows, perform column-wise comparison and find out symbol of interest using Boolean indexing

```
fall = np.greater(stocks[0], stocks[1])
print (sap[fall], "\n")
```

```
['MMM']
```

IEEE 754 floating-point standards

- numpy provides symbols for
  - Positive infinity (`inf`)
  - Non-a-number (nan) - useful as placeholder for missing data
- *ufuncs* `isnan()` useful for locating outcasts in data

```
#set one of values to nan
stocks[1,0] = np.nan
print (np.isnan(stocks), "\n")

#repair damage?
stocks[np.isnan(stocks)]=0
print (stocks)
```

```
[[False False False False False False False False]
 [ True False False False False False False False]]

[[140.49  40.68  55.7   98.2  109.96  35.71  87.85  30.22]
 [  0.    41.53  57.21  99.19 111.47  36.27  89.11  30.91]]
```

Understanding Conditional Functions

- `where(c, a, b)` takes a Boolean array c and two other arrays a and b and returns an array d, where if c[i], d[i] = a[i], else b[i]
  - Similar to ternary operator ? In C++
  - All 3 arrays must be same shape
- `any()` and `all()` return `True`, if any or all array elements are True, resp.
- `nonzero()` returns the indexes of all non-zero elements

Example of where function

- Interested in only stocks that changed substantially (by more than $1.00/share)? Replace “small” changes with 0’s, locate non-zero elements and use their indexes as a “smart index” into array of stock symbols.

```
changes = np.where(np.abs(stocks[1]-stocks[0]) > 1.00, stocks[1] - stocks[0], 0)
print (changes, "\n")
print (sap[np.nonzero(changes)])
```

```
[-139.52    0.      1.51    0.      1.51    0.      1.26    0.  ]

['MMM' 'ATT' 'NTFX' 'TARG']
```

Aggregating and Order Arrays

- You start with large amounts of data and you distill them by
  - Binning
  - Averaging
  - accumulating, etc.
- Until you get a small, easily presentable and interpretable data set.
- numpy provides functions `mean()`, `sum()`, `std()`, `min()` and `max()`
- numpy provides functions `cumsum()` and `cumprod()`
  - Useful for additive and multiplicative data (e.g. interest and compound interest)
  - Be careful if any array element is 0, because cumprod() corresponding element will be 0, as well as all subsequent elements.
- numpy provides function sort() – overwrites original array (make a copy first?)

Treating Arrays as Sets

- Order not important? Numpy knows how to treat arrays as sets.
- function `unique(arr)` returns array of all unique elements in arr
- functions `union1D()` and `intersect1D()` calculates set union and intersection of two one-dimensional arrays. Arrays do not have to be same size.
  - native Python set operators & and | are about 2x faster than `numpy`

```
dna = "AGTCCGCGAATACAGGCTCGGT"
dna_as_array = np.array(list(dna))
print (np.unique(dna_as_array))
```

```
['A' 'C' 'G' 'T']
```

Saving and Reading Arrays

- Probably won't use arrays on their own
- Arrays powerful backend to pandas, network and machine learning
- Create arrays from data at low-level and deliver to higher-level tools
  - `save(file, arr)` and `load(file, arr)`
- `numpy` also has loadtxt() and savetxt(), which loads tabular data from a text file and saves an array to a text file
- `numpy` also zips and unzips files if file ends with `.gz`

More Examples

- https://stackabuse.com/numpy-tutorial-a-simple-example-based-guide/