Data Science with Python

Contents

Data Science

Numpy

Array Basics

```
import numpy as np

np.array([1,2,3])            # 1-D
np.array([[1,2,3], [4,5,6]]) # 2-D

np.zeroes((3,4))             # 3x4 array of 0.0s
np.zeroes((3,4), dtype=int)  # 3x4 array of 0s
np.ones((3,4))
```

```
np.full((3,4), fill_value=7) # fill with 7s
np.empty((3,4))              # uninitialized

np.eye(5, dtype=int)         # 5x5 identity matrix
np.arange(0,10,2)            # 0, 2, 4, 6, 8, 100
np.linspace(0, np.pi, 5)     # evenly spaced range with 5 elements

np.random.seed(0)
np.random.random((3,4))      # random from [0.0-1.0)
np.random.normal(0, 1, (3,4)) # normally distributed with mean 0, std 1
np.random.randint(-2, 10, (3,4)) # random from [-2-10)
```

- array attributes include:
  - ndim for num dimensions
  - shape for size in each dimensions
  - size for num elements
  - dtype for element type
- modify and access elements through slices:
  - `a[1,2]` or `b[0,-1]`
  - can also slice in different dimensions
    - `b[:,0]` grabs the first column as a list
    - `b[:,1:]` grabs the second and rest of columns as a 2-D list
    - `b[0,:]` grabs the first row as a list
- use *fancy indexing* to select multiple elements:
  - can also assign using fancy indexing
  - the result array shape is determined by the index array shape
    - **not** the shape of the original array
  - `idx = [2,5,7]`, `a[idx]` or `a[[2,5,7]]` grabs specified indices
- combining indexing types and slicing:
  - `a[:, [0,2]]` selects 1st and 3rd columns from all rows
- *reshape* arrays with `array.reshape(*dimensions)`
  - number of elements and elements stay the same
  - `a.reshape(3,3)` reshapes into a 3x3 array
  - can quickly create col / row vectors with `np.newaxis` keyword
    - `rowvec = d[np.newaxis, :]`
    - `colvec = d[:, np.newaxis]`
- *transpose* arrays with `array.T`

Array Operations

- `np.concatenate((arrs), axis=0)` takes n-dim arrays and returns an n-dim array
  - by default acts along axis 0 (vertically)
- `np.stack((arrs), axis=0)` takes n-dim arrays and returns an n+1-dim array
- `np.split(arr, breaks)` splits an array
  - `np.split(d, 2)` splits d into two halves along axis 0
  - `np.split(d, (2,3,5), axis=1)` splits d at break points along axis 1
- numpy supports fast vector operations (universal functions) with arrays:
  - ‘+ - * ** / // %‘ are all supported
    - `double = a*2`
  - as well as np.abs, np.cos, np.exp, np.log2
    - `np.log2(np.abs(a))`
- aggregations:
  - min, max, sum, mean, student
    - aggregation functions have corresponding methods
  - specify axis: 0 vertically, 1 horizontally
  - `a.min()`, `col_sums = b.sum(axis=0)`
- numpy performs *broadcasting* on binary operations on arrays that differ in shape:
  - `np.broadcast_arrays(*arrs)` checks what arguments were broadcasted to
  - eg. `np.arange(3) + np.array([4])` braodcasts the scalar to `np.array([4,4,4])`
    - then performs element-wise addition
- some broadcasting rules:
  - arrays with smaller ndims have a 1 prepended to their shapes
  - size in each dimension is the max of all input sizes
  - an input must match the output in a particular dimension, or be exactly 1 in size
- compare arrays element-wise, returning an array of booleans:
  - `c = a > b`, `c.all()` True if all True, `c.any()` True if any True
  - with broadcasting: `c = a > 0`
  - with summing: `np.sum(a < b)` (True corresponds with 1)
- combine boolean values with elementwise operators `& | ~`
  - `np.sum((0 < a) & (a < 10))`
- *mask* arrays to select or assign subsets:
  - `c = a > 0`, `a[c]` selects only positive elements
  - `a[~c] = 0` would zero out negative elements
- *sorting* arrays:
  - `np.sort(arr)` does not change array

        * can sort by axis
    - `arr.sort()` changes in place
    - `idx = np.argsort(arr)` returns the indices of sorted elements
        * eg. `arr[idx]` would print in order

Matrix Operations

- `np.matmul(*arrs)` or `a@b` performs *matrix* multiplication instead of elementwise
    - matrices shapes must be compatible for matrix multiplication