CMSC5733 Social Computing

Tutorial V: Link Analysis and Spectral Clustering

Shenglin Zhao

The Chinese University of Hong Kong

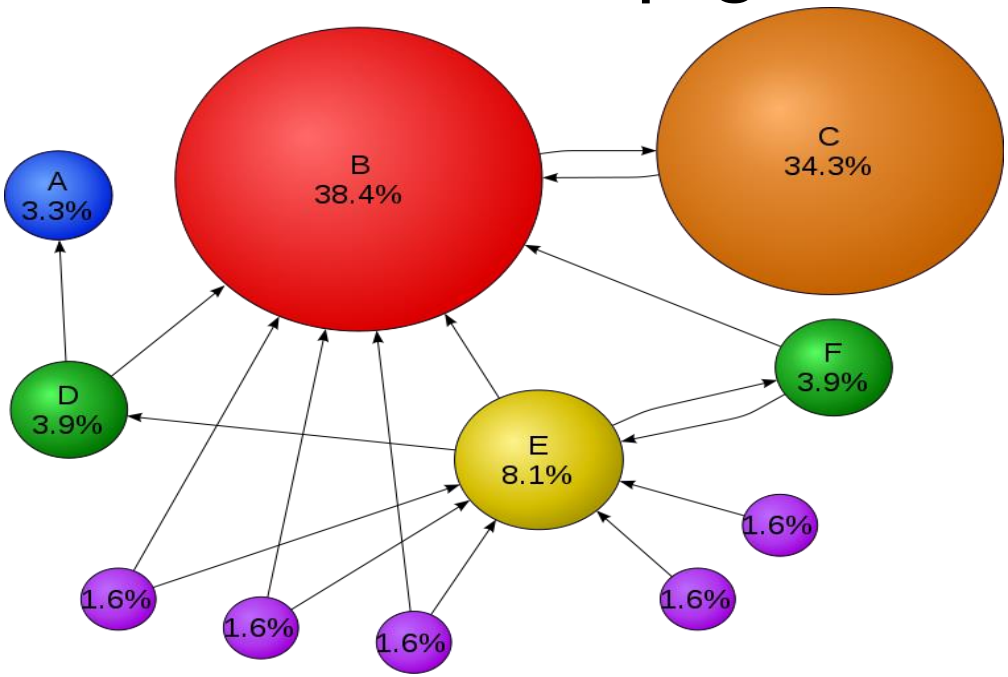
slzhao@cse.cuhk.edu.hk

Overview

- Link Analysis
  - PageRank
  - Hints
  - Implement via NetworkX
- Spectral Clustering
  - Laplacian Matrix
  - Spectral Clustering Algorithm
  - Implement via Numpy & Scipy

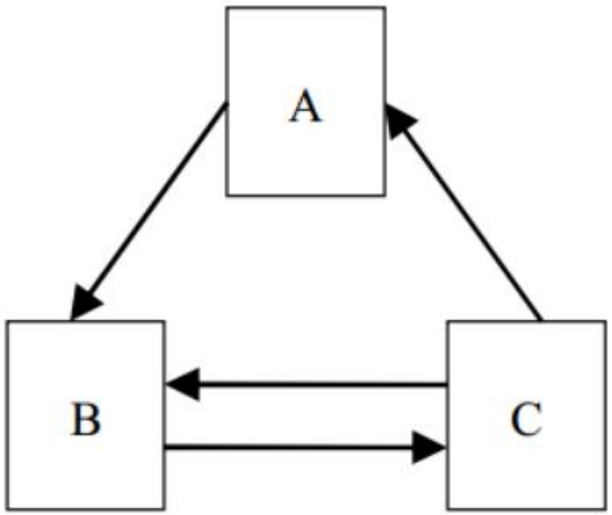
PageRank

- **PageRank** is an algorithm used by Google Search to rank websites in their search engine results. PageRank is a way of measuring the importance of website pages.



PageRank

$$PR(A) = \frac{1-d}{N} + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right).$$



The number of web pages $N = 3$
The damping parameter $d = 0.7$

$PR(A) = (1 - d) \times (1 / N) + d \times (PR(C) / 2)$
$PR(B) = (1 - d) \times (1 / N) + d \times (PR(A) / 1 + PR(C) / 2)$
$PR(C) = (1 - d) \times (1 / N) + d \times (PR(B) / 1)$

So

$PR(A) = 0.1 + 0.35 \times PR(C)$
$PR(B) = 0.1 + 0.70 \times PR(A) + 0.35 \times PR(C)$
$PR(C) = 0.1 + 0.70 \times PR(B)$

By solving the above system of linear equations, we get

$PR(A) = 0.2314$
$PR(B) = 0.3933$
$PR(C) = 0.3753$

PageRank in NetworkX

PageRank

PageRank analysis of graph structure.

pagerank(G[, alpha, personalization, ...])	Return the PageRank of the nodes in the graph.
pagerank_numpy(G[, alpha, personalization, ...])	Return the PageRank of the nodes in the graph.
pagerank_scipy(G[, alpha, personalization, ...])	Return the PageRank of the nodes in the graph.
google_matrix(G[, alpha, personalization, ...])	Return the Google matrix of the graph.

Compute in Numpy

```
In [13]: import numpy as np

In [14]: a = np.matrix((1, 0, -0.35,-.7, 1, -.35, 0, -.7, 1))

In [15]: a = a.reshape((3,3))

In [16]: a
Out[16]:
matrix([[ 1.  ,  0.  , -0.35],
        [-0.7 ,  1.  , -0.35],
        [ 0.  , -0.7 ,  1.  ]])

In [17]: b = np.array((0.1,0.1, 0.1))

In [18]: b
Out[18]: array([ 0.1,  0.1,  0.1])

In [19]: np.linalg.solve(a, b)
Out[19]: array([ 0.23136247,  0.3933162 ,  0.37532134])
```

PageRank in NetworkX

pagerank(*G, alpha=0.85, personalization=None, max_iter=100, tol=1e-08, nstart=None, weight='weight'*)

Return the PageRank of the nodes in the graph.

PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links. It was originally designed as an algorithm to rank web pages.

Parameters : **G** : graph

A NetworkX graph

alpha : float, optional

Damping parameter for PageRank, default=0.85

personalization: **dict, optional** :

The "personalization vector" consisting of a dictionary with a key for every graph node and nonzero personalization value for each node.

max_iter : integer, optional

Maximum number of iterations in power method eigenvalue solver.

tol : float, optional

Error tolerance used to check convergence in power method solver.

nstart : dictionary, optional

Starting value of PageRank iteration for each node.

weight : key, optional

Edge data key to use as weight. If None weights are set to 1.

Returns : **pagerank** : dictionary

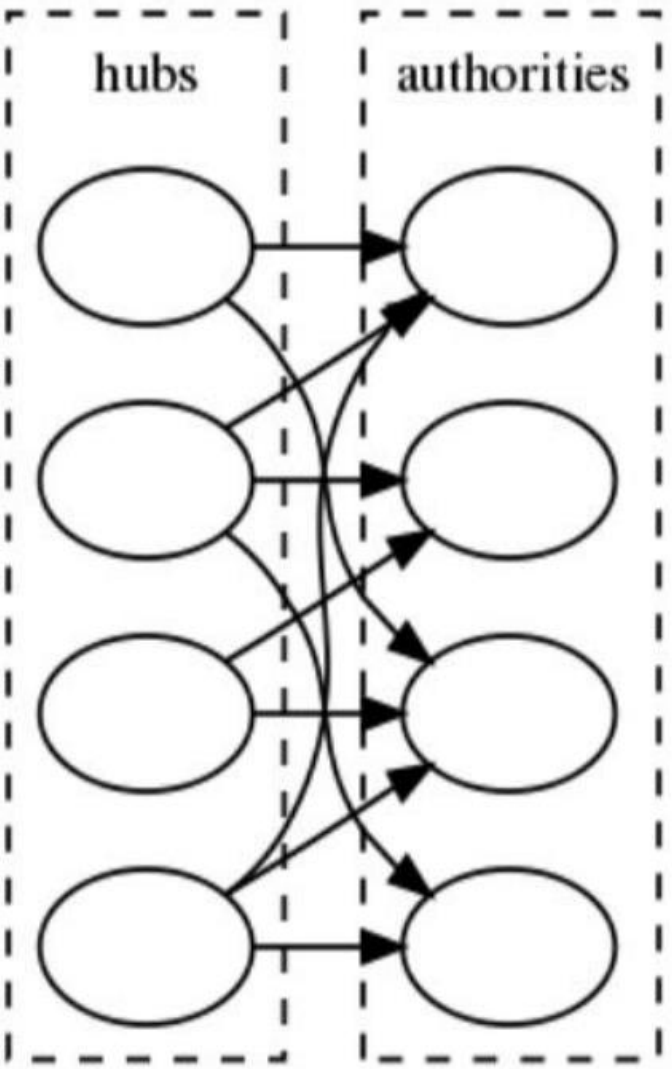
Dictionary of nodes with PageRank as value

Demo

```
import networkx as nx
import matplotlib.pyplot as plt

G=nx.gnp_random_graph(10,0.35)   #ER graph
pr=nx.pagerank(G,alpha=0.9)

nx.draw(G)
plt.savefig("pr.png")
```

HITS

- Hyperlink-Induced Topic Search (**HITS**; also known as **hubs and authorities**) is a link analysis algorithm that rates Web pages, developed by Jon Kleinberg.
- An authority is a page that many hubs link to
- A hub is a page that links to many authorities



HITS in NetworkX

Hits

Hubs and authorities analysis of graph structure.

hits(G[, max_iter, tol, nstart])	Return HITS hubs and authorities values for nodes.
hits_numpy(G)	Return HITS hubs and authorities values for nodes.
hits_scipy(G[, max_iter, tol])	Return HITS hubs and authorities values for nodes.
hub_matrix(G[, nodelist])	Return the HITS hub matrix.
authority_matrix(G[, nodelist])	Return the HITS authority matrix.

HITS in NetworkX

hits(*G*, *max_iter=100*, *tol=1e-08*, *nstart=None*)

Return HITS hubs and authorities values for nodes.

The HITS algorithm computes two numbers for a node. Authorities estimates the node value based on the incoming links. Hubs estimates the node value based on outgoing links.

Parameters **G** : graph

:

A NetworkX graph

max_iter : interger, optional

Maximum number of iterations in power method.

tol : float, optional

Error tolerance used to check convergence in power method iteration.

nstart : dictionary, optional

Starting value of each node for power method iteration.

Returns : **(hubs,authorities)** : two-tuple of dictionaries

Two dictionaries keyed by node containing the hub and authority values.

Demo

```
import networkx as nx
import matplotlib.pyplot as plt

#Return the GNR digraph with n nodes and redirection probability p.
G=nx.gnr_graph(20,0.15)

h,a=nx.hits(G)

nx.draw(G)
plt.savefig("HITS.png")
```

Spectral Clustering

- Algorithms that cluster points using eigenvectors of matrices derived from the data
- Obtain data representation in the low-dimensional space that can be easily clustered

Graph Notation

Given G = (V, E),

- Vertex set $V = \{v_1, \ldots, v_n\}$
- Weighted adiacency matrix $W = (w_{ij})_{i,j=1,\ldots,n}$
- Degree $d_i = \sum_{j=1}^{n} w_{ij}$
- Degree Matrix, diagonal matrix with degrees.

Laplacian Matrix

- The unnormalized graph Laplacian matrix is defined as

$$L = D - W.$$

Proposition 1 (Properties of L) *The matrix L satisfies the following properties:*

1. *For every vector $f \in \mathbb{R}^n$ we have*

$$f'Lf = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2.$$

2. *L is symmetric and positive semi-definite.*
3. *The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector $\mathbb{1}$.*
4. *L has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.*

Normalized Laplacian Matrix

- Two types:

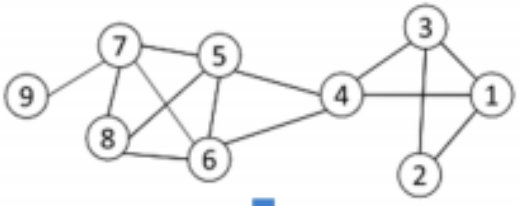
$$L_{\text{sym}} := D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$$
$$L_{\text{rw}} := D^{-1} L = I - D^{-1} W.$$

- L_{sym} is a symmetric matrix, and L_{rw} is related to a random walk.

Algorithm

- Compute the unnormalized Laplacian L.
- Compute the first k eigenvectors $u_1, \ldots, u_k$ of L.
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \ldots, u_k$ as columns.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
- Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

Demo



Two communities:
{1, 2, 3, 4} and {5, 6, 7, 8, 9}

k-means

$$D = diag(3, 2, 3, 4, 4, 4, 4, 3, 1)$$

$$\tilde{L} = D - A = \begin{bmatrix} 3 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 4 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 4 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix} \Rightarrow S = \begin{bmatrix} 0.33 & -0.38 \\ 0.33 & -0.48 \\ 0.33 & -0.38 \\ 0.33 & -0.12 \\ 0.33 & 0.16 \\ 0.33 & 0.16 \\ 0.33 & 0.30 \\ 0.33 & 0.24 \\ 0.33 & 0.51 \end{bmatrix}$$

Figure: From http://dmml.asu.edu/cdm/slides/chapter3.pdf

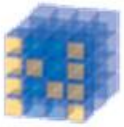
Hands-on Spectral Clustering

- NetworkX
- Scipy
- Numpy

Scipy & Numpy

Website: www.scipy.org

SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:



NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

IP[y]:
IPython

IPython
Enhanced Interactive Console

Sympy
Symbolic mathematics

pandas
Data structures & analysis

Install

Installing the SciPy Stack¶

These are instructions for installing *the full SciPy stack*. For installing individual packages, such as NumPy and SciPy, see *Individual binary and source packages* below.

Scientific Python distributions

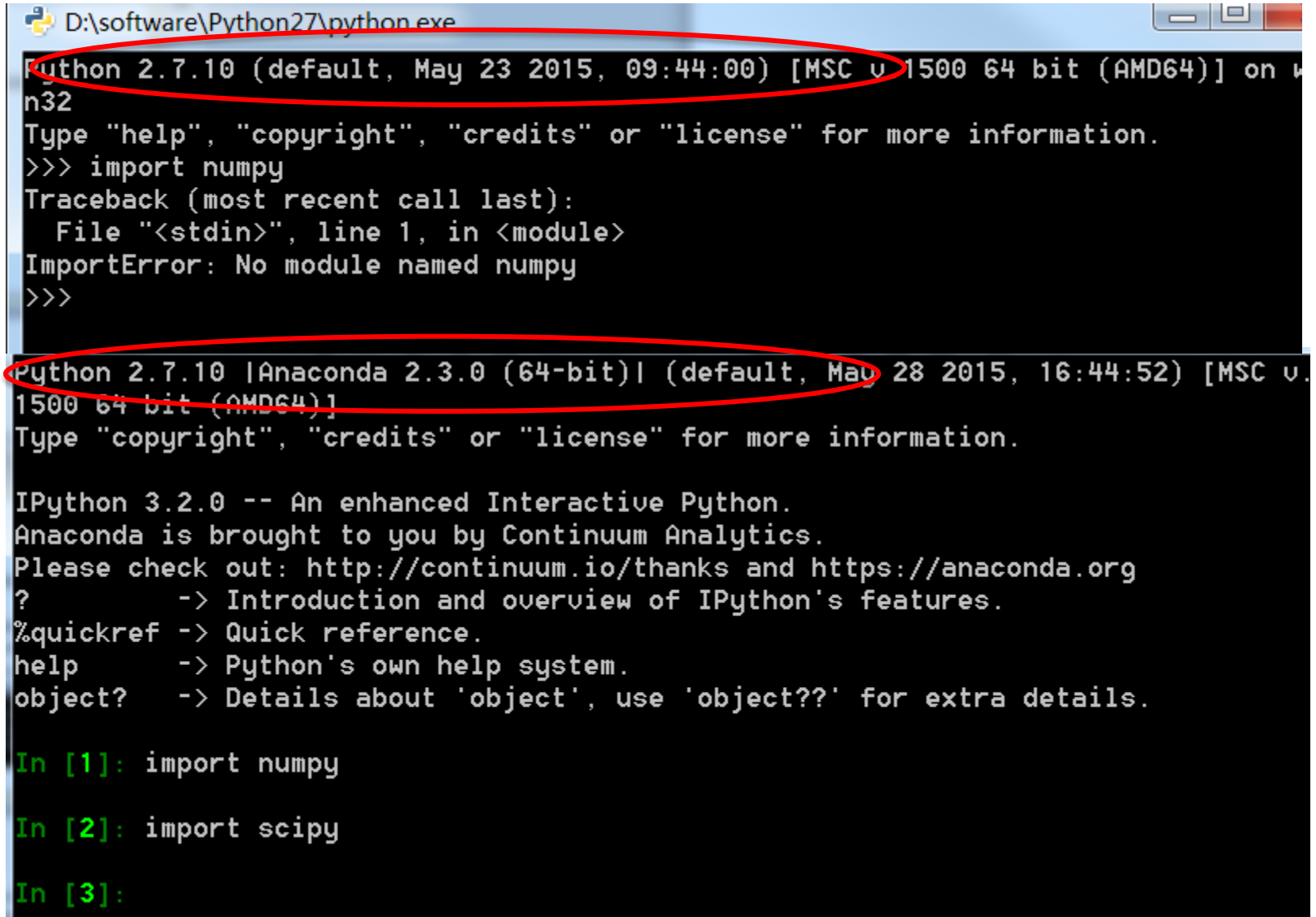
For most users, especially on Windows and Mac, the easiest way to install the packages of the SciPy stack is to download one of these Python distributions, which includes all the key packages:

- Anaconda: A free distribution for the SciPy stack. Supports Linux, Windows and Mac.
- Enthought Canopy: The free and commercial versions include the core SciPy stack packages. Supports Linux, Windows and Mac.
- Python(x,y): A free distribution including the SciPy stack, based around the Spyder IDE. Windows only.
- WinPython: A free distribution including the SciPy stack. Windows only.
- Pyzo: A free distribution based on Anaconda and the IEP interactive development environment. Supports Linux, Windows and Mac.

Install

- Check whether you successfully install the package
  - open python
  - import scipy
  - import numpy

Install


D:\software\Python27\python.exe
Python 2.7.10 (default, May 23 2015, 09:44:00) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ImportError: No module named numpy
>>>
Python 2.7.10 |Anaconda 2.3.0 (64-bit)| (default, May 28 2015, 16:44:52) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 3.2.0 -- An enhanced Interactive Python.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]: import numpy
In [2]: import scipy
In [3]:

Numpy

- NumPy is the fundamental package for scientific computing with Python. It contains among other things:
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities
- Website: http://www.numpy.org/

Numpy.linalg

- Matrix and vector products
- Decompositions
- Matrix eigenvalues
- Norms and other numbers
- Solving equations and inverting matrices
- Exceptions

Eigenvalue and Eigenvector

numpy.linalg.eig

numpy.linalg.**eig**(*a*) [source]

Compute the eigenvalues and right eigenvectors of a square array.

Parameters: a *: (..., M, M) array*
Matrices for which the eigenvalues and right eigenvectors will be computed

Returns: w *: (..., M) array*
The eigenvalues, each repeated according to its multiplicity. The eigenvalues are not necessarily ordered. The resulting array will be always be of complex type. When *a* is real the resulting eigenvalues will be real (0 imaginary part) or occur in conjugate pairs

v *: (..., M, M) array*
The normalized (unit "length") eigenvectors, such that the column `v[:,i]` is the eigenvector corresponding to the eigenvalue `w[i]`.

Raises: LinAlgError *:*
If the eigenvalue computation does not converge.

See also:

eigvalsh	eigenvalues of a symmetric or Hermitian (conjugate symmetric) array.
eigvals	eigenvalues of a non-symmetric array.

K-means

Clustering package (scipy.cluster)

scipy.cluster.vq

Clustering algorithms are useful in information theory, target detection, communications, compression, and other areas. The vq module only supports vector quantization and the k-means algorithms.

scipy.cluster.hierarchy

The hierarchy module provides functions for hierarchical and agglomerative clustering. Its features include generating hierarchical clusters from distance matrices, calculating statistics on clusters, cutting linkages to generate flat clusters, and visualizing clusters with dendrograms.

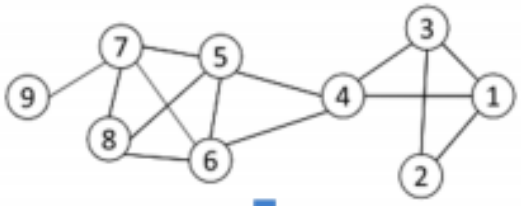
K-means

K-means clustering and vector quantization (scipy.cluster.vq)

Provides routines for k-means clustering, generating code books from k-means models, and quantizing vectors by comparing them with centroids in a code book.

whiten(obs[, check_finite])	Normalize a group of observations on a per feature basis.
vq(obs, code_book[, check_finite])	Assign codes from a code book to observations.
kmeans(obs, k_or_guess[, iter, thresh, ...])	Performs k-means on a set of observation vectors forming k clusters.
kmeans2(data, k[, iter, thresh, minit, ...])	Classify a set of observations into k clusters using the k-means algorithm.

Back to Example



Two communities:
{1, 2, 3, 4} and {5, 6, 7, 8, 9}

k-means

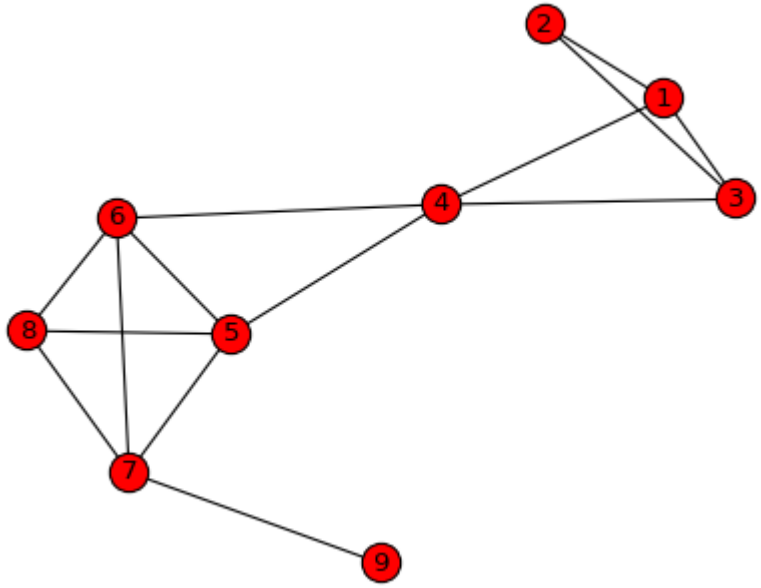
$$D = diag(3, 2, 3, 4, 4, 4, 4, 3, 1)$$

$$\tilde{L} = D - A = \begin{bmatrix} 3 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 4 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 4 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix} \Rightarrow S = \begin{bmatrix} 0.33 & -0.38 \\ 0.33 & -0.48 \\ 0.33 & -0.38 \\ 0.33 & -0.12 \\ 0.33 & 0.16 \\ 0.33 & 0.16 \\ 0.33 & 0.30 \\ 0.33 & 0.24 \\ 0.33 & 0.51 \end{bmatrix}$$

Figure: From http://dmml.asu.edu/cdm/slides/chapter3.pdf

- Draw the graph

```
In [18]: import networkx as nx

In [19]: import matplotlib.pyplot as plt

In [20]: G = nx.Graph()

In [21]: G.add_edges_from([(1,2), (1,3), (

In [22]: G.add_edges_from([(2,3), (3,4), (

In [23]: G.add_edges_from([(5,6), (5,7), (

In [24]: nx.draw_networkx(G)

In [25]: plt.show()
```



- Check Laplacian Matrix

```
In [29]: L = nx.laplacian_matrix(G)

In [30]: L.todense()
Out[30]:
matrix([[ 3, -1, -1, -1,  0,  0,  0,  0,  0],
        [-1,  2, -1,  0,  0,  0,  0,  0,  0],
        [-1, -1,  3, -1,  0,  0,  0,  0,  0],
        [-1,  0, -1,  4, -1, -1,  0,  0,  0],
        [ 0,  0,  0, -1,  4, -1, -1, -1,  0],
        [ 0,  0,  0, -1, -1,  4, -1, -1,  0],
        [ 0,  0,  0,  0, -1, -1,  4, -1, -1],
        [ 0,  0,  0,  0, -1, -1, -1,  3,  0],
        [ 0,  0,  0,  0,  0,  0, -1,  0,  1]])
```

- Compute eigenvalue and eigenvectors
- Find first two engenvalues

```
In [65]: eigv, eigvec = np.linalg.eig(L)

In [66]: eigv
Out[66]:
array([  5.88398314e+00,   4.82979118e+00,   4.00000000e+00,
         2.76811766e+00,   4.67884991e-16,   4.14773461e-01,
         1.10333455e+00,   4.00000000e+00,   5.00000000e+00])
```

- Find corresponding eigenvectors

```
In [67]: eigvec
Out[67]:
matrix([[  1.87686754e-01,  -1.60449411e-01,  -7.74596669e-01,
           2.65780041e-01,   3.33333333e-01,   3.82381445e-01,
           1.05185352e-01,  -4.91371422e-02,   3.59006071e-16],
        [ -9.66465338e-02,   1.13400177e-01,   1.29099445e-01,
          -6.92029497e-01,   3.33333333e-01,   4.82431294e-01,
           2.34614487e-01,  -2.79904197e-01,  -4.61778306e-16],
        [  1.87686754e-01,  -1.60449411e-01,   5.16397779e-01,
           2.65780041e-01,   3.33333333e-01,   3.82381445e-01,
           1.05185352e-01,   6.08945536e-01,   5.29222259e-16],
        [ -6.32325653e-01,   3.40638151e-01,   1.29099445e-01,
           4.87879153e-01,   3.33333333e-01,   1.23729921e-01,
          -1.40298417e-01,  -2.79904197e-01,  -7.99333883e-16],
        [  4.07958681e-01,   1.91201441e-02,   1.29099445e-01,
           3.47248145e-02,   3.33333333e-01,  -1.60581547e-01,
          -3.08384140e-01,  -2.79904197e-01,   7.07106781e-01],
        [  4.07958681e-01,   1.91201441e-02,   1.29099445e-01,
           3.47248145e-02,   3.33333333e-01,  -1.60581547e-01,
          -3.08384140e-01,  -2.79904197e-01,  -7.07106781e-01],
        [ -4.00006136e-01,  -7.82212741e-01,   9.49244096e-16,
          -1.46696684e-01,   3.33333333e-01,  -2.98987213e-01,
          -7.81916689e-02,   4.81408028e-18,   1.97654923e-15],
        [ -1.44214167e-01,   4.06588719e-01,  -2.58198890e-01,
          -3.33130398e-01,   3.33333333e-01,  -2.39882385e-01,
          -3.66411457e-01,   5.59808394e-01,  -1.05280789e-15],
        [  8.19016210e-02,   2.04244227e-01,  -2.08157148e-16,
           8.29677157e-02,   3.33333333e-01,  -5.10891413e-01,
           7.56684631e-01,   1.25594959e-16,  -6.04886245e-16]])
```

- Fourth and fifth columns

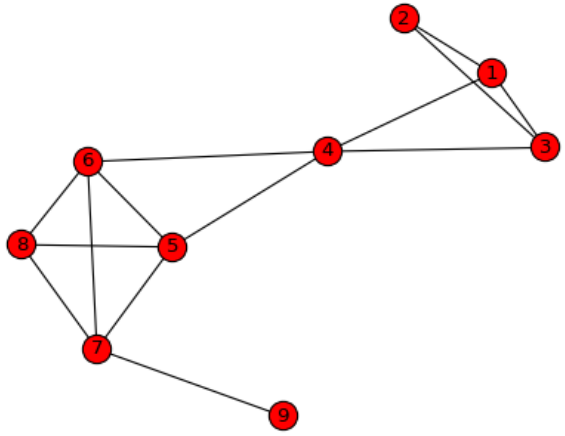
```
In [68]: v1 = eigvec[:, 4]

In [69]: v2 = eigvec[:, 5]

In [70]: v1
Out[70]:
matrix([[ 0.33333333],
        [ 0.33333333],
        [ 0.33333333],
        [ 0.33333333],
        [ 0.33333333],
        [ 0.33333333],
        [ 0.33333333],
        [ 0.33333333],
        [ 0.33333333]])

In [71]: v2
Out[71]:
matrix([[ 0.38238145],
        [ 0.48243129],
        [ 0.38238145],
        [ 0.12372992],
        [-0.16058155],
        [-0.16058155],
        [-0.29898721],
        [-0.23988238],
        [-0.51089141]])
```

- S is a 9*2 matrix, combined by the fourth and fifth eigenvectors

```
In [73]: S = np.column_stack((v1,v2))

In [74]: S
Out[74]:
matrix([[ 0.33333333,  0.38238145],
        [ 0.33333333,  0.48243129],
        [ 0.33333333,  0.38238145],
        [ 0.33333333,  0.12372992],
        [ 0.33333333, -0.16058155],
        [ 0.33333333, -0.16058155],
        [ 0.33333333, -0.29898721],
        [ 0.33333333, -0.23988238],
        [ 0.33333333, -0.51089141]])
```

- K-means clustering for S

```
In [78]: from scipy.cluster.vq import kmeans2

In [79]: c, l = kmeans2(S, 2)

In [80]: l
Out[80]: array([0, 0, 0, 0, 1, 1, 1, 1, 1])

In [81]: G.nodes()
Out[81]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```



More about Numpy

Overview of NumPy

N-D ARRAY (NDARRAY)

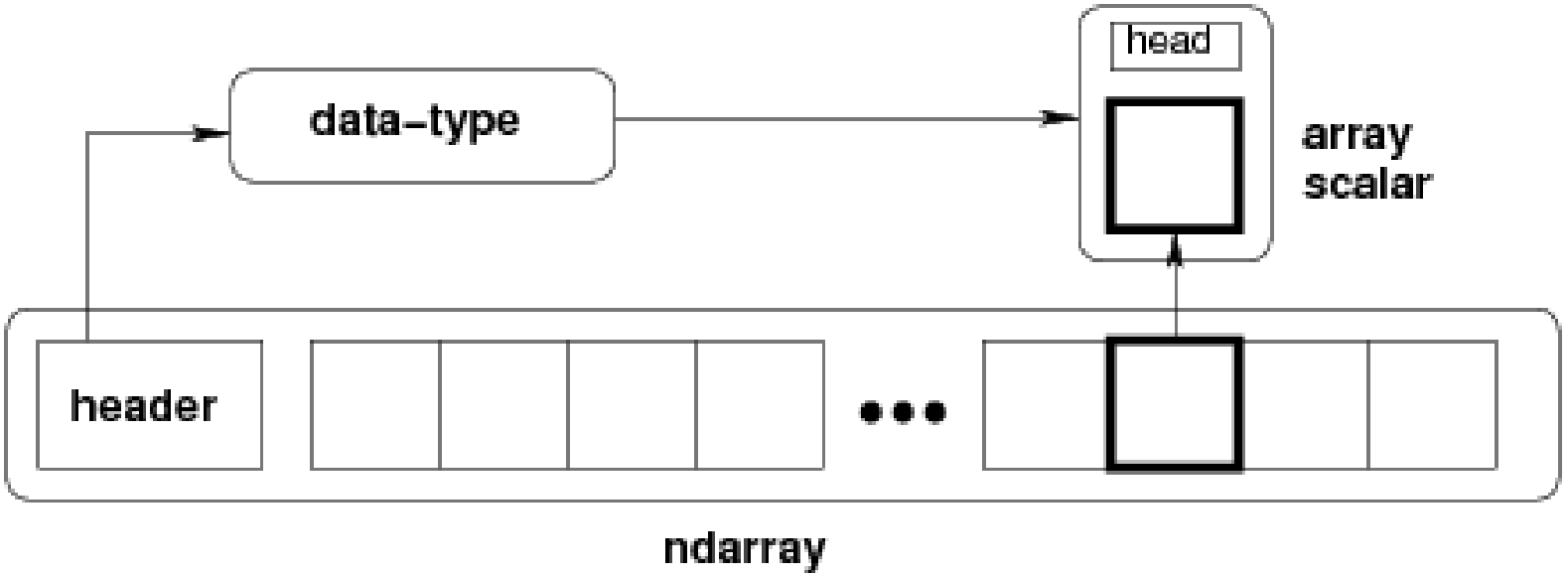
- N-dimensional array of rectangular data
- Element of the array can be C-structure or simple data-type.
- Fast algorithms on machine data-types (int, float, etc.)

UNIVERSAL FUNCTIONS (UFUNC)

- functions that operate element-by-element and return result
- fast-loops registered for each fundamental data-type
- $\sin(x) = [\sin(x_i)\ i=0..N]$
- $x+y = [x_i + y_i\ i=0..N]$

NumPy Array

A NumPy array is an N-dimensional homogeneous collection of “items” of the same “kind”. The kind can be any arbitrary structure and is specified using the data-type.



NumPy Array

A NumPy array is a homogeneous collection of "items" of the same "data-type" (dtype)

```
>>> import numpy as N
>>> a =
N.array([[1,2,3],[4,5,6]],float)
>>> print a
[[1. 2. 3.]
 [4. 5. 6.]]
>>> print a.shape, "\n", a.itemsize
(2, 3)
8
>>> print a.dtype, a.dtype.type
'<float64' <type 'numpy.float64'>
>>> type(a[0,0])
<type 'numpy.float64'>
>>> type(a[0,0]) is type(a[1,2])
True
```

Introducing NumPy Arrays

SIMPLE ARRAY CREATION

```
>>> a = array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
```

CHECKING THE TYPE

```
>>> type(a)
<type 'numpy.ndarray'>
```

NUMERIC 'TYPE' OF ELEMENTS

```
>>> a.dtype
dtype('int32')
```

BYTES PER ELEMENT

```
>>> a.itemsize # per element
4
```

ARRAY SHAPE

```
# shape returns a tuple
# listing the length of the
# array along each dimension.
>>> a.shape
(4,)
>>> shape(a)
(4,)
```

ARRAY SIZE

```
# size reports the entire
# number of elements in an
# array.
>>> a.size
4
```

Introducing NumPy Arrays

ARRAY COPY

```
# create a copy of the array
>>> b = a.copy()
>>> b
array([0, 1, 2, 3])
```

CONVERSION TO LIST

```
# convert a numpy array to a
# python list.
>>> a.tolist()
[0, 1, 2, 3]

# For 1D arrays, list also
# works equivalently, but
# is slower.
>>> list(a)
[0, 1, 2, 3]
```

Setting Array Elements

ARRAY INDEXING

```
>>> a[0]
0
>>> a[0] = 10
>>> a
[10, 1, 2, 3]
```

FILL

```
# set all values in an array.
>>> a.fill(0)
>>> a
[0, 0, 0, 0]

# This also works, but may
# be slower.
>>> a[:] = 1
>>> a
[1, 1, 1, 1]
```

BEWARE OF TYPE COERSION

```
>>> a.dtype
dtype('int32')

# assigning a float to into # an int32 array
will
# truncate decimal part.
>>> a[0] = 10.6
>>> a
[10, 1, 2, 3]

# fill has the same behavior
>>> a.fill(-4.8)
>>> a
[-4, -4, -4, -4]
```

Multi-Dimensional Arrays

MULTI-DIMENSIONAL ARRAYS

```
>>> a = array([[ 0, 1, 2, 3],
               [10,11,12,13]])
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,13]])
```

(ROWS,COLUMNS)

```
>>> a.shape
(2, 4)
```

ELEMENT COUNT

```
>>> a.size
8
```

NUMBER OF DIMENSIONS

```
>>> a.ndims
2
```

GET/SET ELEMENTS

```
>>> a[1,3]
13
```


```
>>> a[1,3] = -1
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,-1]])
```

ADDRESS FIRST ROW USING SINGLE INDEX

```
>>> a[1]
array([10, 11, 12, -1])
```

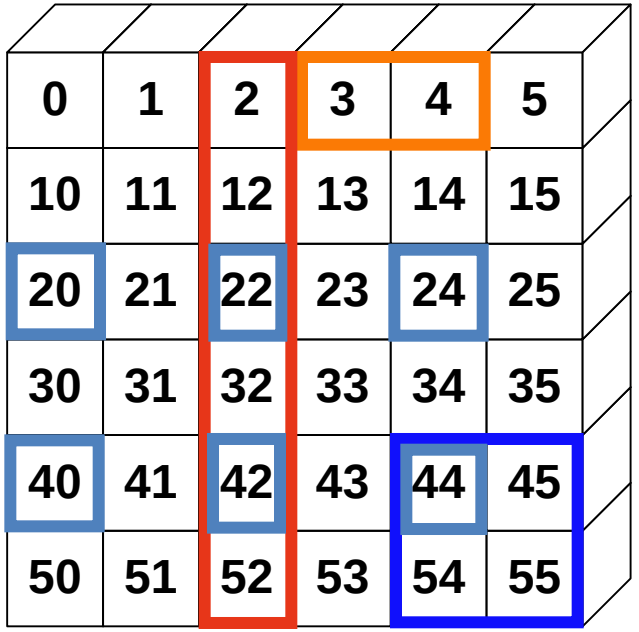
Array Slicing

SLICING WORKS MUCH LIKE STANDARD PYTHON SLICING

```
>>> a[0,3:5]
array([3, 4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])
```

STRIDES ARE ALSO POSSIBLE

```
>>> a[2::2,::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



Slices Are References

Slices are references to memory in original array. Changing values in a slice also changes the original array.

```
>>> a = array((0,1,2,3,4))

# create a slice containing only the
# last element of a
>>> b = a[2:4]
>>> b[0] = 10

# changing b changed a!
>>> a
array([ 1,  2, 10, 3, 4])
```

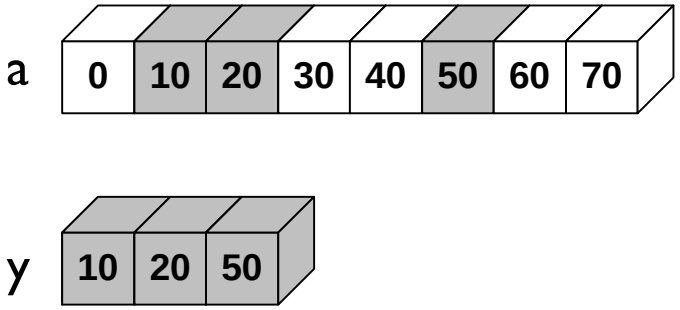
Fancy Indexing

INDEXING BY POSITION

```
>>> a = arange(0,80,10)


# fancy indexing
>>> y = a[[1, 2, -3]]
>>> print y
[10 20 50]

# using take
>>> y = take(a,[1,2,-3])
>>> print y
[10 20 50]
```

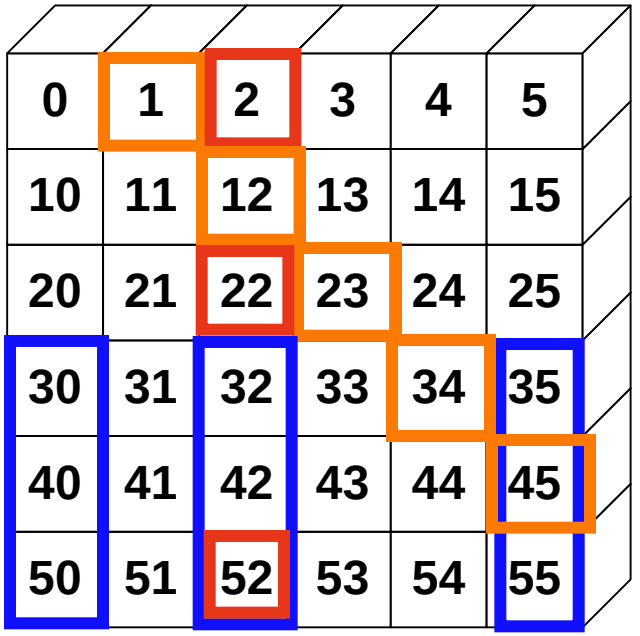
INDEXING WITH BOOLEANS

```
>>> mask = array([0,1,1,0,0,1,0,0],
...              dtype=bool)

# fancy indexing
>>> y = a[mask]
>>> print y
[10,20,50]

# using compress
>>> y = compress(mask, a)
>>> print y
[10,20,50]
```



Fancy Indexing in 2D

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45]])
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```



Unlike slicing, fancy indexing creates copies instead of views into original arrays.

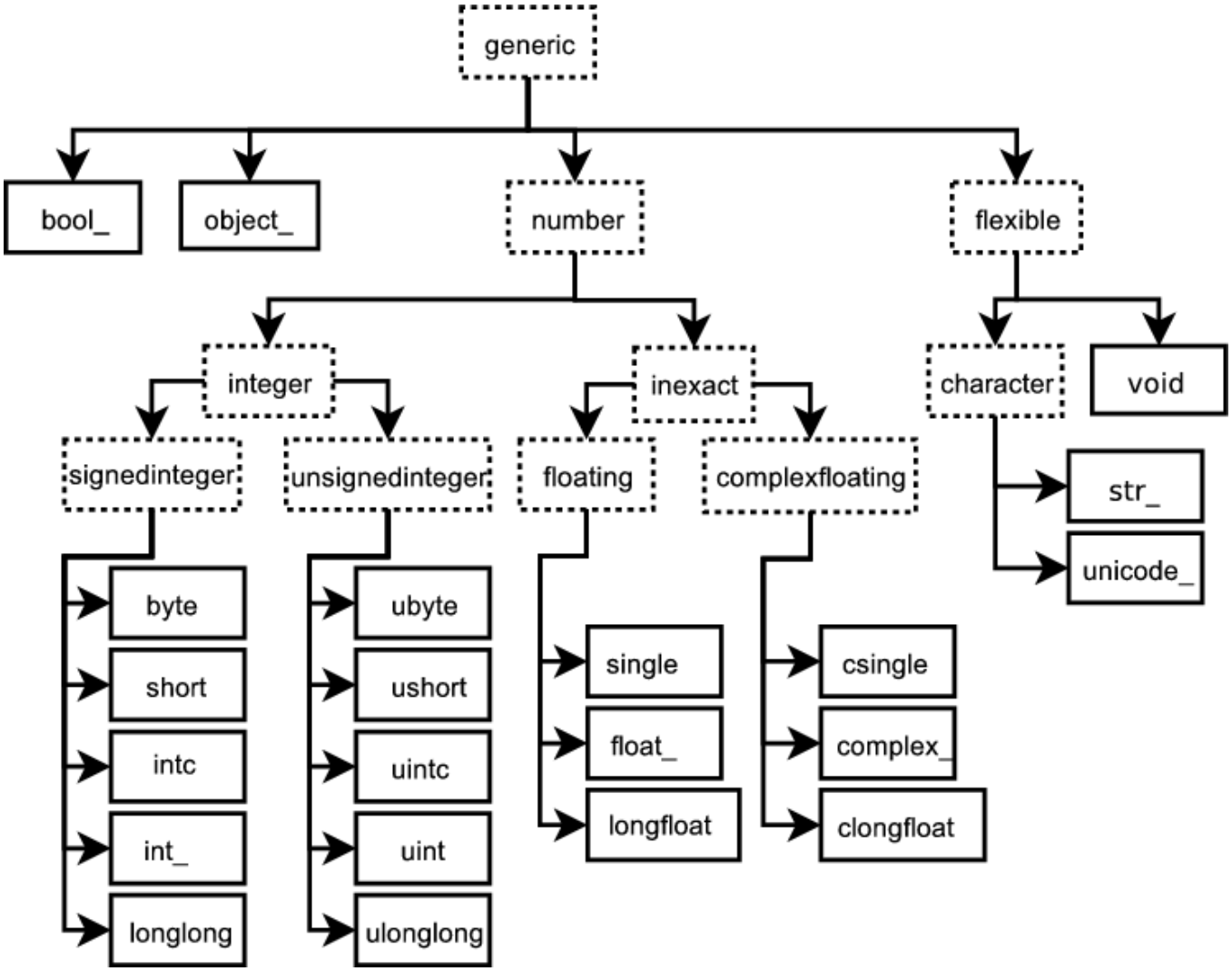
Data-types

- There are two related concepts of "type"
  - The data-type object (dtype)
  - The Python "type" of the object created from a single array item (hierarchy of scalar types)
- The **dtype** object provides the details of how to interpret the memory for an item. It's an instance of a single dtype class.
- The "type" of the extracted elements are true Python classes that exist in a hierarchy of Python classes
- Every dtype object has a type attribute which provides the Python object returned when an element is selected from the array

NumPy dtypes

Basic Type	Available NumPy types	Comments
Boolean	`bool`	Elements are 1 byte in size
Integer	`int8, int16, int32, int64, int128, int`	int defaults to the size of int in C for the platform
Unsigned Integer	`uint8, uint16, uint32, uint64, uint128, uint`	uint defaults to the size of unsigned int in C for the platform
Float	`float32, float64, float, longfloat,`	Float is always a double precision floating point value (64 bits). longfloat represents large precision floats. Its size is platform dependent.
Complex	`complex64, complex128, complex`	The real and complex elements of a complex64 are each represented by a single precision (32 bit) value for a total size of 64 bits.
Strings	`str, unicode`	Unicode is always UTF32 (UCS4)
Object	`object`	Represent items in array as Python objects.
Records	`void`	Used for arbitrary data structures in record arrays.

Built-in “scalar” types



Data-type object (dtype)

- There are 21 "built-in" (static) data-type objects
- New (dynamic) data-type objects are created to handle
  - Alteration of the byteorder
  - Change in the element size (for string, unicode, and void built-ins)
  - Addition of fields
  - Change of the type object (C-structure arrays)
- Creation of data-types is quite flexible.
- New user-defined "built-in" data-types can also be added (but must be done in C and involves filling a function-pointer table)

Data-type fields

- An item can include fields of different data-types.
- A field is described by a data-type object and a byte offset --- this definition allows nested records.
- The array construction command interprets tuple elements as field entries.

```
>>> dt = N.dtype(“i4,f8,a5”)
>>> print dt.fields
{'f0': (dtype('int32'), 0), 'f1':
(dtype('float64'), 4), 'f2': (dtype('|S5'), 12)}
>>> a = N.array([(1,2.0,”Hello”), (2,3.0,”World”)],
dtype=dt)
>>> print a['f3']
[Hello World]
```

Array Calculation Methods

SUM FUNCTION

```
>>> a = array([[1,2,3],
               [4,5,6]], float)

# Sum defaults to summing along
# axis=0.
>>> sum(a)
21.

# supply the keyword axis to
# sum along the 0th axis.
>>> sum(a, axis=0)
array([5., 7., 9.])

# supply the keyword axis to
# sum along the last axis.
>>> sum(a, axis=-1)
array([6., 15.])
```

SUM ARRAY METHOD

```
# The a.sum() defaults to
# summing *all* array values
>>> a.sum()
21.

# Supply an axis argument to
# sum along a specific axis.
>>> a.sum(axis=0)
array([5., 7., 9.])
```

PRODUCT

```
# product along columns.
>>> a.prod(axis=0)
array([ 4., 10., 18.])

# functional form.
>>> prod(a, axis=0)
array([ 4., 10., 18.])
```

Min/Max

MIN

```
>>> a = array([2.,3.,0.,1.]) >>> a.min(axis=0)
0.
# use Numpy's amin() instead
# of Python's builtin min()
# for speed operations on
# multi-dimensional arrays.
>>> amin(a, axis=0)
0.
```

MAX

```
>>> a = array([2.,1.,0.,3.]) >>> a.max(axis=0)
3.

# functional form
>>> amax(a, axis=0)
3.
```

ARGMIN

```
# Find index of minimum value.
>>> a.argmin(axis=0)
2
# functional form
>>> argmin(a, axis=0)
2
```

ARGMAX

```
# Find index of maximum value.
>>> a.argmax(axis=0)
1
# functional form
>>> argmax(a, axis=0)
1
```

Statistics Array Methods

MEAN

```
>>> a = array([[1,2,3],
              [4,5,6]], float)

# mean value of each column
>>> a.mean(axis=0)
array([ 2.5,  3.5,  4.5])
>>> mean(a, axis=0)
array([ 2.5,  3.5,  4.5])
>>> average(a, axis=0)
array([ 2.5,  3.5,  4.5])

# average can also calculate
# a weighted average
>>> average(a, weights=[1,2],
...         axis=0)
array([ 3., 4., 5.])
```

STANDARD DEV./VARIANCE

```
# Standard Deviation
>>> a.std(axis=0)
array([ 1.5,  1.5,  1.5])

#Variance
>>> a.var(axis=0)
array([2.25, 2.25, 2.25])
>>> var(a, axis=0)
array([2.25, 2.25, 2.25])
```

Other Array Methods

CLIP

```
# Limit values to a range

>>> a = array([[1,2,3],
               [4,5,6]], float)

# Set values < 3 equal to 3.
# Set values > 5 equal to 5.
>>> a.clip(3,5)
>>> a
array([[ 3.,  3.,  3.],
       [ 4.,  5.,  5.]])
```

ROUND

```
# Round values in an array.
# Numpy rounds to even, so
# 1.5 and 2.5 both round to 2.
>>> a = array([1.35, 2.5, 1.5])
>>> a.round()
array([ 1.,  2.,  2.])

# Round to first decimal place.
>>> a.round(decimals=1)
array([ 1.4,  2.5,  1.5])
```

POINT TO POINT

```
# Calculate max – min for
# array along columns
>>> a.ptp(axis=0)
array([ 3.0,  3.0,  3.0])
# max – min for entire array.
>>> a.ptp(axis=None)
5.0
```

Universal Functions

- ufuncs are objects that rapidly evaluate a function element-by-element over an array.
- Core piece is a 1-d loop written in C that performs the operation over the largest dimension of the array
- For 1-d arrays it is equivalent to but much faster than list comprehension

```
>>> type(N.exp)
<type 'numpy.ufunc'>
>>> x = array([1,2,3,4,5])
>>> print N.exp(x)
[   2.71828183    7.3890561    20.08553692
54.59815003  148.4131591 ]
>>> print [math.exp(val) for val in x]
[2.7182818284590451,
7.3890560989306504,20.085536923187668,
 54.598150033144236,148.4131591025766]
```

Mathematic Binary Operators

a + b → add(a,b)
a - b → subtract(a,b)
a % b → remainder(a,b)

a * b → multiply(a,b)
a / b → divide(a,b)
a ** b → power(a,b)

MULTIPLY BY A SCALAR

```
>>> a = array((1,2))
>>> a*3.
array([3., 6.])
```

ELEMENT BY ELEMENT ADDITION

```
>>> a = array([1,2])
>>> b = array([3,4])
>>> a + b
array([4, 6])
```

ADDITION USING AN OPERATOR FUNCTION

```
>>> add(a,b)
array([4, 6])
```

IN PLACE OPERATION

```
# Overwrite contents of a.
# Saves array creation
# overhead
>>> add(a,b,a) # a += b
array([4, 6])
>>> a
array([4, 6])
```

Comparison and Logical Operators

equal (==)	not_equal (!=)	greater (>)
greater_equal (>=)	less (<)	less_equal (<=)
logical_and	logical_or	logical_xor
logical_not		

2D EXAMPLE

```
>>> a = array(((1,2,3,4),(2,3,4,5)))
>>> b = array(((1,2,5,4),(1,3,4,5)))
>>> a == b
array([[True, True, False, True],
       [False, True, True, True]])
# functional equivalent
>>> equal(a,b)
array([[True, True, False, True],
       [False, True, True, True]])
```

Bitwise Operators

`bitwise_and (&)`	`invert (~)`	right_shift(a,shifts)
`bitwise_or (|)`	`bitwise_xor`	left_shift (a,shifts)

BITWISE EXAMPLES

```
>>> a = array((1,2,4,8))
>>> b = array((16,32,64,128))
>>> bitwise_or(a,b)
array([ 17,  34,  68, 136])

# bit inversion
>>> a = array((1,2,3,4), uint8)
>>> invert(a)
array([254, 253, 252, 251], dtype=uint8)

# left shift operation
>>> left_shift(a,3)
array([ 8, 16, 24, 32], dtype=uint8)
```

Trig and Other Functions

TRIGONOMETRIC

```
sin(x)        sinh(x)
cos(x)        cosh(x)
arccos(x)     arccosh(x)
arctan(x)     arctanh(x)
arcsin(x)     arcsinh(x)
arctan2(x,y)
```

OTHERS

```
exp(x)        log(x)
log10(x)      sqrt(x)
absolute(x)   conjugate(x)
negative(x)   ceil(x)
floor(x)      fabs(x)
hypot(x,y)    fmod(x,y)
maximum(x,y)  minimum(x,y)
```

hypot(x,y)

Element by element distance calculation using $\sqrt{x^2 + y^2}$

References

- http://www.slideshare.net/hnly228078/spectral-clustering-tutorial
- Von Luxburg, Ulrike. "A tutorial on spectral clustering." Statistics and computing 17.4 (2007): 395-416
- https://en.wikipedia.org/wiki/PageRank
- https://en.wikipedia.org/wiki/HITS_algorithm