Welcome to General Assembly

- **WiFi** GA Guest
- **Password** yellowpencil

GENERAL ASSEMBLY

DATA SCIENCE

DAT11SYD

Lesson 16: Graph and Network Analysis

How can we model and analyse inter-relationships of entities?

Course Plan

Date	Class	Lesson	Who
Monday, 19 February 2018	Lesson 1	Introduction to Data Science	Paul
Wednesday, 21 February 2018	Lesson 2	Elements of Data Science	Paul
Monday, 26 February 2018	Lesson 3	Data Visualisation	Paul
Wednesday, 28 February 2018	Lesson 4	Linear Regression	Paul
Monday, 5 March 2018	No Class	* Paul ill	.--
Wednesday, 7 March 2018	Lesson 5	Logistic Regression	Paul
Monday, 12 March 2018	Lesson 6	Model Evaluation	Paul
Wednesday, 14 March 2018	Lesson 7	Regularisation	Paul
Monday, 19 March 2018	Lesson 8	Clustering	Paul
Wednesday, 21 March 2018	Lesson 9	Recommendations	Paul
Monday, 26 March 2018	Lesson 10	SQL + Productivity	Paul
Wednesday, 28 March 2018	Lesson 11	Decision Trees	Greg
Monday, 2 April 2018	No Class	Easter Monday	.--
Wednesday, 4 April 2018	Lesson 12	Ensembles	Greg
Monday, 9 April 2018	Lesson 13	Natural Language Programming	Greg
Wednesday, 11 April 2018	No Class	** Paul ill	.--
Monday, 16 April 2018	Lesson 14	Time Series + R	Paul
Wednesday, 18 April 2018	Lesson 15	Soft Skills + Cloud Computing	Paul
Monday, 23 April 2018	Lesson 16	Network Analysis	Paul
Wednesday, 25 April 2018	No Class	ANZAC Day	.--
Monday, 30 April 2018	Lesson 17	Neural Networks	Paul
Wednesday, 2 May 2018	Lesson 18	GA Data Science Alumni Panel / Final Project Help	Olivia / Paul
Monday, 7 May 2018	Lesson 19	* Final Projects Presentations	Paul
Wednesday, 9 May 2018	Lesson 20	** Final Projects Presentations	Paul

Lesson 16 - Review

FINAL PROJECT

- Final Project split into 4-parts: [Review with James 5mins]
    - (a) Real-world Problem Identification [Lesson 14]
    - (b) Data Cleaning [Lesson 15]
    - (c) Model & Validation [Lesson 16]
    - (d) Presentation & Storytelling [Lesson 17]

Lesson 18 – any final queries
Lesson 19 & 20 – Presenting final project back to class

Git & GitHub – 1 Pager Guide!

(Part B) EVERY CLASS:

At the START of the class, you'll need to sync the latest materials from the COURSE repo:

(1) Make sure you are in the dat11syd directory:

```
cd ~/workspace/dat11syd
```

(2) Make sure to select the "master" branch of your repo:

```
git checkout master
```

(3) Fetch the latest changes from the UPSTREAM repo (i.e the course repo)

```
git fetch upstream
```

(4) Merge the changes from the upstream repo to your master branch:

```
git merge upstream/master
```

DURING the class:

(5) Before editing, either copy files to your "students/" folder, or rename them

At the END of every class:

(6) Make sure you are in the dat11syd directory:

```
cd ~/workspace/dat11syd
```

(7) Add any files that you've updated to your git registry:

```
git add -A
```

(8) Commit the changes with a sensible comment:

```
git commit -m "my updates for lesson 7"
```

(9) Push your changes to your PERSONAL repo:

```
git push origin master
```

DONE!!!!!

Agenda

1. Maths Recap
2. What are networks?
3. Graph theory
4. Graph metrics
5. Directed graphs
6. Graph transforms
7. Node closeness
8. Network analysis in Python
9. Lab

Learning Objectives

By the end of this lesson you should be able to ...

- Explain what a network is

- Discuss basic graph theory

- Describe properties and metrics of graphs

- Perform network analysis using Python

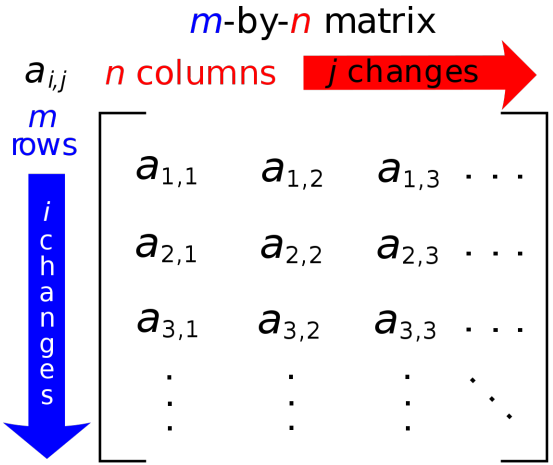
Let’s get started...

Maths Revision

Matrices ☺

- ***A Matrix is a 2-dimensional list of things, arranged into columns and rows.***

- ***The general form is called a "Tensor" (can have any dimensionality)***
  **** 1D Tensor is called a "Vector" such as:*** $\underline{c} = [x, y, z] = [2, 4, 7] = 2\underline{i} + 4\underline{j} + 7\underline{k}$
  **** 2D Tensor is called a "Matrix" (usually [rows x columns] given as "m x n")***



Lab – Markdown & Matrices

Networks

What are networks?

Examples?

Networks

Def: **Network**

1. an arrangement of intersecting horizontal and vertical lines.

   synonyms: web, criss-cross, grid, lattice, net, matrix, mesh, webbing, tracery, trellis;

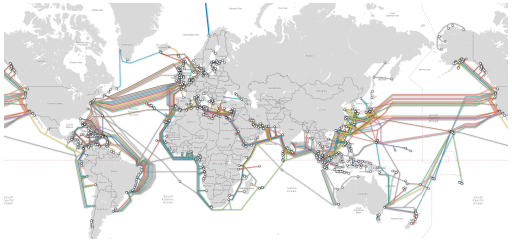
2. a group or system of interconnected people or things.

➢ *In data science, we tackle problems involving definition 2 using a conceptual model based on definition 1*

Networks

Examples

- social
- biological
- roads
- power grids
- infrastructure
- web pages
- mobile phone routing
- Internet routing
- vehicular flows



Social Networks

Graph Theory

Def: Graph

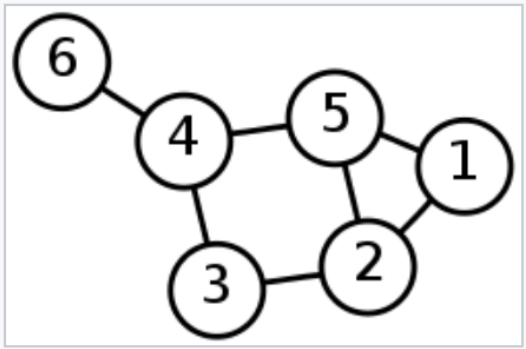
a mathematical structure used to model pairwise relations between objects

composed of

- **vertices** *or* **nodes**

connected by 1 or more

- **links** *or* **edges**



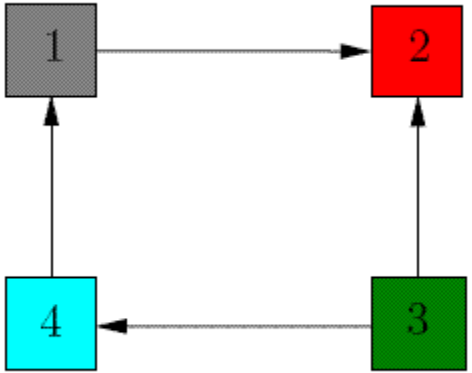
https://en.wikipedia.org/wiki/Graph_theory

Graph Theory

***Def:* Directed Graph**

a graph with defined transitions between nodes

- transitions can represent
  - flow of information
  - movement of quantities

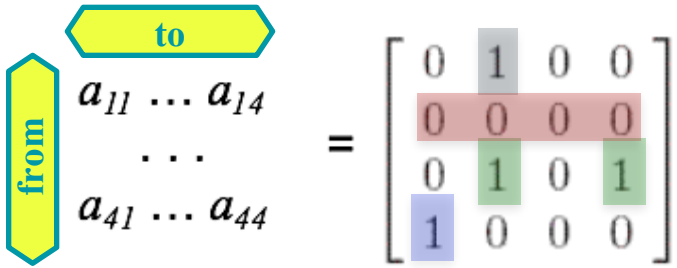


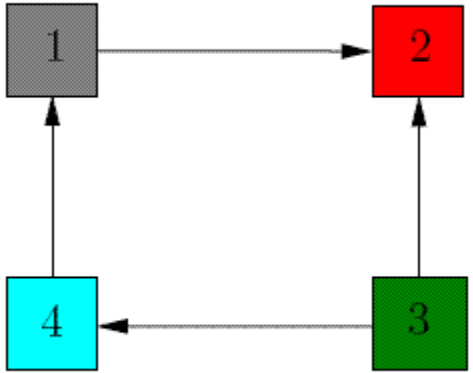
example:

node 1 has *in-degree* = 1 (inflow from node 4), *out-degree* = 1 (outflow to node 2)
node 2 has *in-degree* = 2 (inflows from nodes 1 and 3), *out-degree* = 0
node 3 has *in-degree* = 0, *out-degree* = 2 (outflows to nodes 2 and 4)
node 4 has *in-degree* = 1 (inflow from node 3), *out-degree* = 1 (outflow to node 1)

Adjacency Matrix A (for a directed graph)

a_{ij} represents a directed connection **from** node i **to** node j

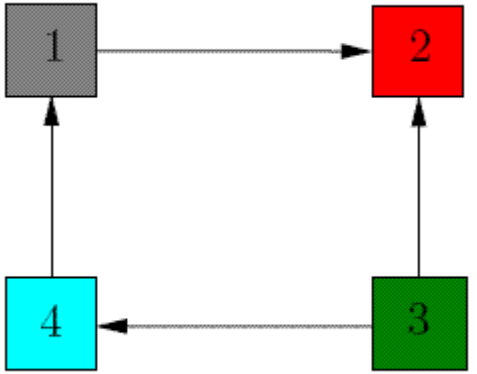




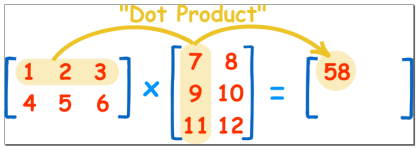
Graph Theory

Adjacency matrix for a 2-step path

$$A^2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

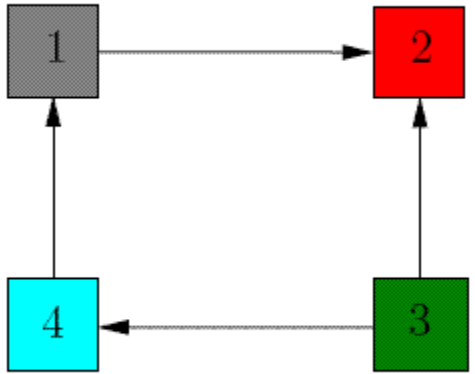


What about for *k* steps?



Adjacency matrix for a k-step path

is just A^k

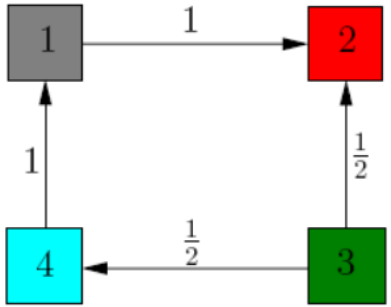


➢*adjacency matrices provide a powerful tool for computing paths, which we can make even more powerful by adding some concepts ...*

Transition Matrix *T*

an adjacency matrix that indicates quantitative flows between nodes

- the ***out degree*** of a node is the number of edges flowing out
- the ***in degree*** of a node is the number of edges flowing in
- t_{ij} represents the proportion flowing **from** node *i* **to** node *j*



$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}$$

Transition Matrix *T*

an adjacency matrix that indicates quantitative flows between nodes

- t_{ij} could also represent
  - an *absolute quantity* that *flows* from node *i* to node *j*
  - the *probability* of a *transition* from state *i* to state *j*

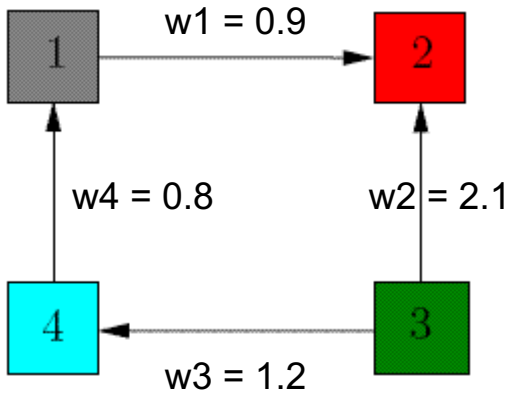
➢ *these generalisations allow an edge to have two directions (a 2-headed arrow):*

  - e.g. t_{24} = flow node 2 to 4
    t_{42} = flow node 4 to 2

Graph Theory

Weighted Edges

- assigning a weight to each edge of the graph allows more complex representations
- examples
  - proportion of items moving from one node to another
  - probability of a state transition from one node to another
  - distance, time, or cost associated with traversing a link between two nodes
  - compound weights
- applications
  - GPS programming
  - travel-planning
  - times / cost optimisations



Graph Metrics

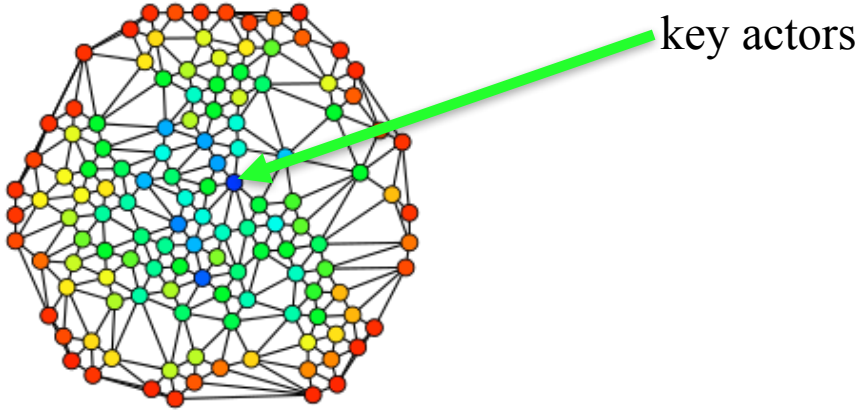
What measurable properties does a graph have?

- ?
- ?

Graph Metrics

What measurable properties does a graph/network have?

- who/what the key actors (most important nodes) are in the network?
- how would we calculate importance / centrality?



Graph Metrics

Calculating Closeness:

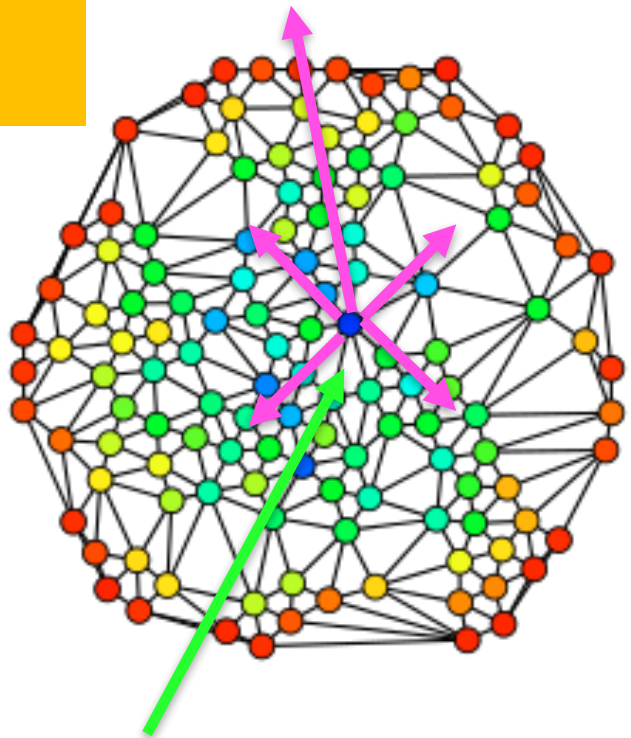
1. Calc shortest path to every other node s_{ij}

2. Farness = sum(shortest paths s_{ij})

3. Closeness = (N-1) / Farness

- ***Degree*** Centrality
  - "Popularity"
  - number of edges at a node

- ***Closeness*** Centrality
  - the reciprocal of the sum of the shortest path distances from one node to all n-1 other nodes
  - the sum of distances depends on the number of nodes in the graph;
    so, closeness is normalized by the sum of minimum possible distances n-1.
  - higher values of closeness indicate higher centrality

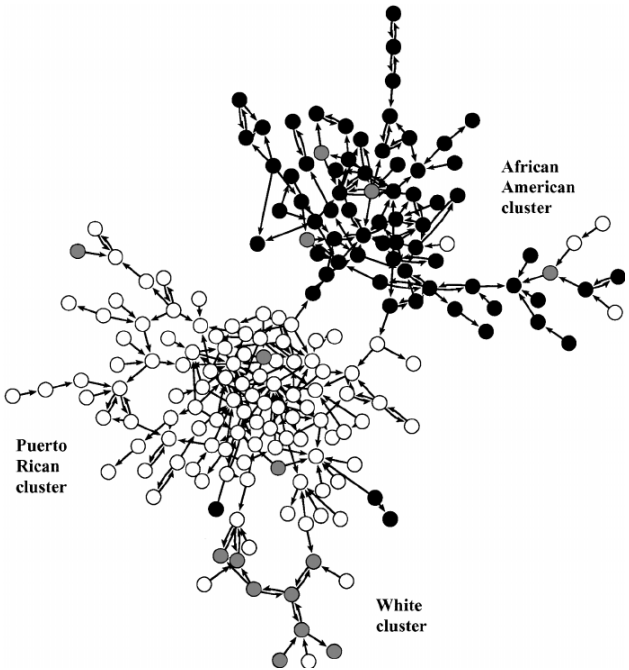


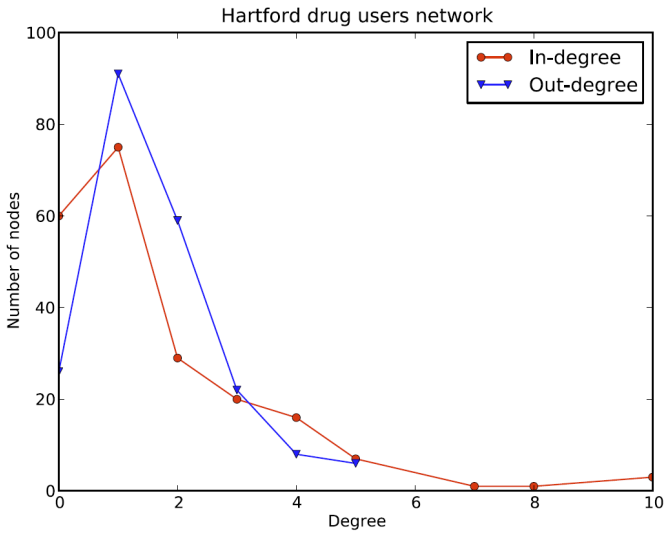
Graph Metrics

- ***Betweenness*** Centrality
  - the sum of the fraction of all-pairs shortest paths that pass through the node v
  - (i.e. is “v” a “hub”? Is v on the path between lots of things? - CBD etc)

- ***Eigenvector*** Centrality
  - computes the centrality for a node from the centrality of its neighbours

Graph Metrics





Weeks, Margaret & Clair, Scott & Borgatti, Stephen & Radda, Kim & Schensul, Jean. (2002). Social Networks of Drug Users in High-Risk Sites: Finding the Connections. AIDS and Behavior. 6. 193-206.

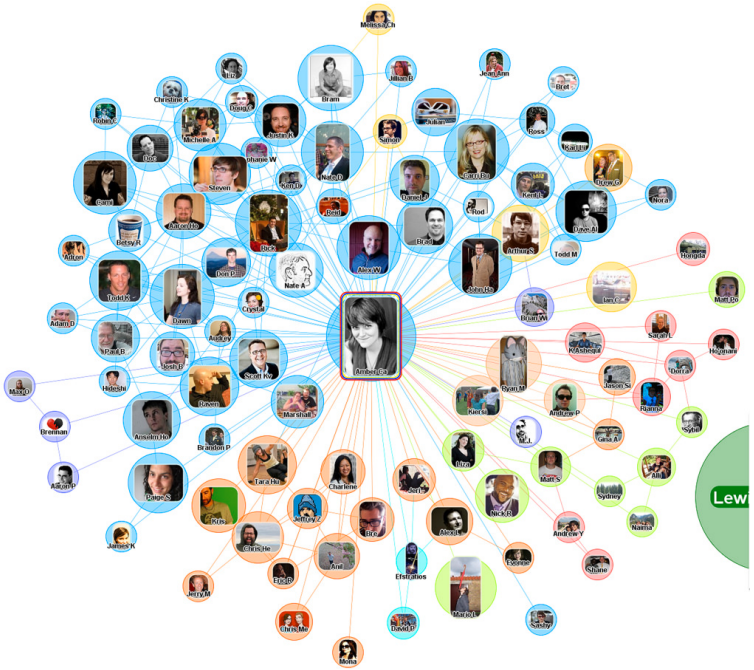
Grouping within Networks

How would we detect communities within a network?

e.g. Facebook friends

- we met them at different places and stages in our lives
- work, school, travelling, other friends Facebook friends, etc.

  How might we find sub-groupings (communities) among them?



Grouping within Networks

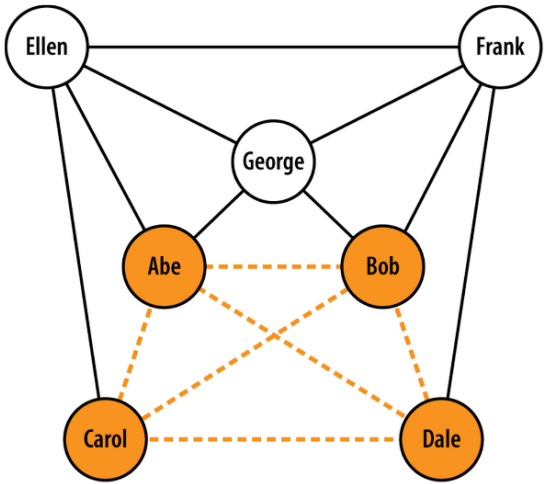
- the criteria for finding communities is similar to that for finding clusters, but applied to edges rather than nodes

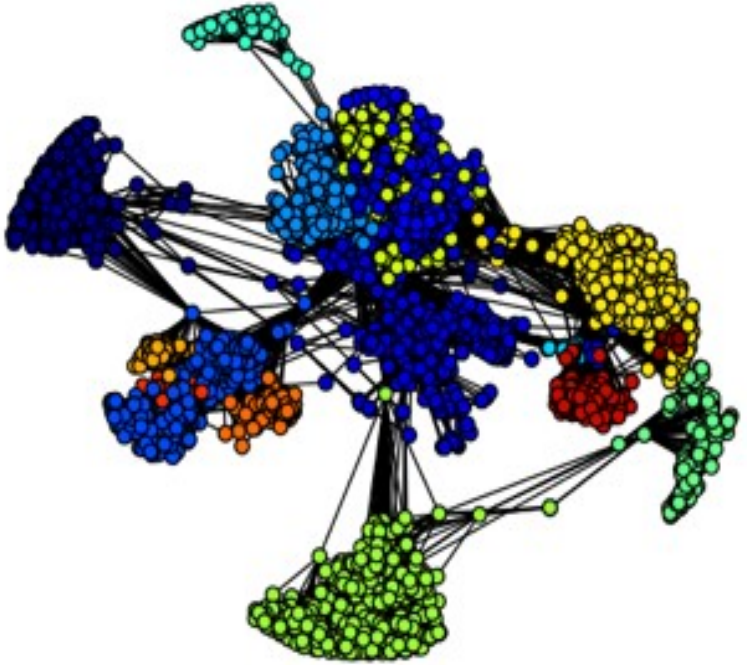
- want to maximize intra-community edges while minimizing inter-community edges

- formally, the algorithm tries to maximize the modularity of the network:
  - or
    (the fraction of edges that fall within the community)
    minus
    (the expected fraction of edges from a random distribution)

Grouping within Networks

'communities'
or
'cliques'





https://github.com/ptwobrussell/Mining-the-Social-Web-2nd-Edition

Practical Network Analysis

Network Analysis in Python

- custom compiled code *plus* Python
  - Boost Graph
  - igraph
  - Graphviz
- Python-based
  - NetworkX

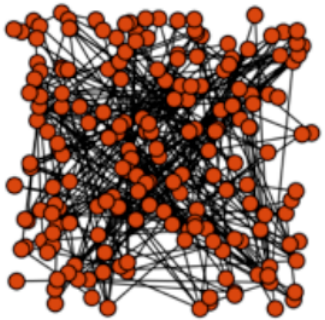
Network Analysis using NetworkX

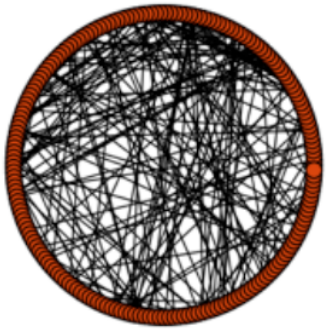
- data structures
  - for representing many types of networks, graphs
- flexibility
  - ideal for representing networks found in many different fields
- easy interface to existing code bases
  - legacy algorithms in C, C++, and FORTRAN
- productivity
  - user can focus on computational network modelling rather than software tool development

Network Visualisation using NetworkX

- okay for small graphs

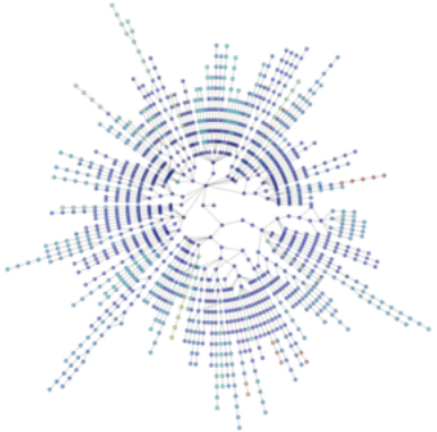
```
>>> import pylab as plt #import Matplotlib plotting interface
>>> g = nx.erdos_renyi_graph(100,0.15)
>>> nx.draw(g)
>>> nx.draw_random(g)
>>> nx.draw_circular(g)
>>> nx.draw_spectral(g)
>>> plt.savefig('graph.png')
```

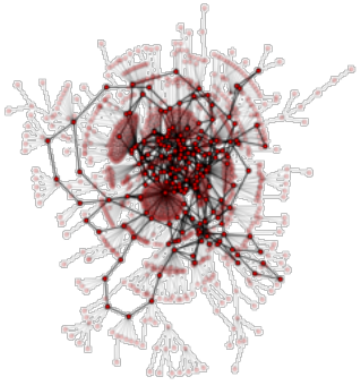


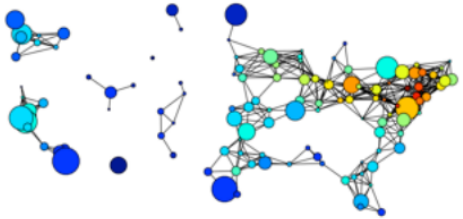


Network Visualisation using NetworkX

- can export network data and draw with other programs
  e.g GraphViz, matplotlib







Network Analysis using NetworkX

Creating and populating a graph

```
# Instantiate a graph:
import networkx as nx
g = nx.Graph()

# Add one node:
g.add_node(1) # method of nx.Graph

# Add a list of nodes:
g.add_nodes_from([2 ,3])

# Add a container of nodes
h = nx.path_graph(10)
g.add_nodes_from(h) # g now contains the nodes of h

# Remove a specific node:
g.remove_node(2)
```

A node can be any hashable object:

strings, numbers, files, functions, etc.

```
import math
g.add_node(math.cos) # cosine function
fh = open('tmp.txt','w') # file handle
g.add_node(fh)
print g.nodes()

[<built-in function cos>, <open file 'tmp.txt', mode 'w' at 0x30dc38>]

g.add_node('a')
```

Adding edges to a graph

```
# Single edge
g.add_edge(1,2)
e = (2,3)
g.add_edge(*e) # unpack edge tuple

# List of edges
g.add_edges_from([(1 ,2) ,(1 ,3)])

# Container of edges
g.add_edges_from(h.edges())

# In contrast, you can remove any edge of the graph
g.remove_edge(1,2)
```

Network Analysis using NetworkX

Properties of a graph

```
g.number_of_nodes() # also g.order()
4

g.number_of_edges() # also g.size()
2

g.nodes()
[1, 2, 3, 'a']

g.edges()
[(1, 2), (1, 3)]

g.neighbors(1)
[2, 3]

g.degree(1)
2
```

Graph, Node, and Edge Attributes

```
G = nx.Graph(day="Friday")
G.graph['day'] = "Monday"
```

Node Attributes

On *access* (only), the nodes of a NetworkX graph behave like the primary keys of a Python dict

```
g.add_node(1, time='5pm')
g.node[1]['time']
'5pm'

g.node[1]
{'time': '5pm'}

G.add_nodes_from([3], time='2pm')
G.nodes[1]['room'] = 714
G.nodes.data()
NodeDataView({1: {'room': 714, 'time': '5pm'}, 3: {'time': '2pm'}})
```

Network Analysis using NetworkX

Edge Attributes

```
g.add_edge(1, 2, weight=4.0)
g[1][2]
{'weight': 4.0}

g[1][2]['weight'] = 5.0 # change the weight of an existing edge
g[1][2]
{'weight': 5.0}
```

examples:

- modelling road networks
  - weight = distance
- modelling social networks
  - weight = strength of association

Note and Edge Iterators

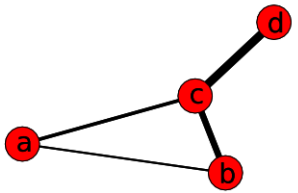
```
for node in g.nodes():
    print node, g.degree(node)


for n1, n2, attr in g.edges(data=True): # unpacking
    print n1, n2, attr['weight']
```

Dijkstra's algorithm https://youtu.be/_lHSawdgXpI

- finds the shortest path in a weighted or unweighted network:

```
>>> import networkx as nx
>>> g = nx.Graph()
>>> g.add_edge('a','b',weight=0.1)
>>> g.add_edge('b','c',weight=1.5)
>>> g.add_edge('a','c',weight=1.0)
>>> g.add_edge('c','d',weight=2.2)
>>> print nx.shortest_path(g,'b','d')
 ['b', 'c', 'd']
>>> print nx.shortest_path(g,'b','d',weighted=True)
 ['b', 'a', 'c', 'd']
```



Directed Graphs

```
dg = nx.DiGraph()
dg.add_weighted_edges_from([(1, 4, 0.5), (3, 1, 0.75)])

dg.out_degree(1, weighted=True)
0.5

dg.degree(1, weighted=True)
1.25

dg.successors(1)
[4]

dg.predecessors(1)
[3]
```

examples:

- the link structure of a website
- natural language modelling
- business process modelling
- assembly line control

Multigraphs

- allow multiple edges between any pair of nodes

```
    mg = nx.MultiGraph()
mg.add_weighted_edges_from([(1,2,.5), (1,2,.75), (2,3,.5)])
mg.degree(weighted=True)
{1: 1.25, 2: 1.75, 3: 0.5}
```

Conversion to undirected graph

- some algorithms don't support *directed* or *multiple* edges
- to use these on a directed graph or multigraph it must first be converted to an undirected graph

```
Graph.to_undirected()
```

Graph Operators

`subgraph(G, nbunch)`	induce subgraph of G on nodes in nbunch (an iterable)
`union(G1,G2)`	graph union
`disjoint_union(G1,G2)`	graph union assuming all nodes are different
`cartesian_product(G1,G2)`	return Cartesian product graph
`compose(G1,G2)`	combine graphs identifying nodes common to both
`complement(G)`	graph complement
`create_empty_copy(G)`	return an empty copy of the same graph class
`convert_to_undirected(G)`	return an undirected representation of G
`convert_to_directed(G)`	return a directed representation of G

Network Analysis using NetworkX

Graph Generators

```
petersen=nx.petersen_graph()                                    # small
famous graphs
tutte=nx.tutte_graph()
maze=nx.sedgewick_maze_graph()
tet=nx.tetrahedral_graph()

K_5=nx.complete_graph(5)                                        # classic
graphs
K_3_5=nx.complete_bipartite_graph(3,5)
barbell=nx.barbell_graph(10,10)
lollipop=nx.lollipop_graph(10,20)

er=nx.erdos_renyi_graph(100,0.15)                             # random graphs
ws=nx.watts_strogatz_graph(30,3,0.1)
ba=nx.barabasi_albert_graph(100,5)
red=nx.random_lobster(100,0.9,0.9)
```

Network Analysis using NetworkX

General read/write format

```
>>> g = nx.read_format("path/to/file.txt",...options...)
>>> nx.write_format(g,"path/to/file.txt",...options...)
```

Read and write edge lists

```
g = nx.read_edgelist(path,comments='#',create_using=None,
delimiter=' ',nodetype=None,data=True,edgetype=None,encoding='utf-8')
nx.write_edgelist(g,path,comments='#',
delimiter=' ',data=True,encoding='utf-8')
```

Formats

- Node pairs with no data:

  ```
  1 2
  ```

- Python dictionary as data:

  ```
  1 2 {'weight':7, 'color':'green'}
  ```

- Arbitrary data:

  ```
  1 2 7 green
  ```

Lab:

Graph and Network Analysis

with NetworkX and Facebook SDK

Summary

- brief introduction to graphs and networks
  - types of graphs
  - properties of graphs / networks
  - metrics for nodes and paths
- exposure to NetworkX
  - creating and visualising graphs
  - computing graph properties and metrics
  - basic algorithms

Task List

Read the following articles:

http://www.wired.com/2016/01/googles-go-victory-is-just-a-glimpse-of-how-powerful-ai-will-be/

http://www.wired.com/2014/01/geoffrey-hinton-deep-learning https://sites.google.com/site/deepernn/home/blog/briefsummaryofthepaneldiscussionatdlworkshopicml2015

Install TensorFlow

Q&A

Additional Resources

- NetworkX
  - http://networkx.lanl.gov/
  - https://networkx.github.io/documentation
- GraphViz
  - http://www.graphviz.org/
- Jupyter Notebooks from *Mining the Social Web*
  - https://github.com/ptwobrussell/Mining-the-Social-Web-2nd-Edition
    (warning: APIs have changed since last updates)
- Dataset for *Hartford Drug Users*
  - https://sites/google.com/site/ucinetsoftware/datasets/covert-networks/drugnet

Thank You!