Cheatography

Network Analysis with Python and NetworkX Cheat Sheet

by RJ Murray (murenei) via cheatography.com/58736/cs/15946/

Basic graph manipulation

```
import networkx as nx
```

```
G=nx.G raph()
```

`G=nx.M ult iGr aph()`	Create a graph allowing parallel edges
`G.add_ edg es_ fro m([(0, 1),(0, 2),(1, 3),(2, 4)]`	Create graph from edges
`nx.dra w_n etw orkx(G)`	Draw the graph
`G.add_ nod e(' A', rol e=' man ager')`	Add a node
`G.add_ edg e(' A', 'B' ,re lation = 'friend')`	Add an edge
`G.node ['A '][ 'role'] = 'team member'`	Set attribute of a node
`G.node ['A'],G.edge [(' A', 'B')]`	View attributes of node, edge
`G.edges(),G.nodes()`	Show edges, nodes
`list(G.ed ges())`	Return as list instead of EdgeView class
`G.node s(d ata =True),G.edge s(d ata =True)`	Include node/edge attributes
`G.node s(d ata ='r ela tion)`	Return specific attribute

Creating graphs from data

Bipartite graphs (cont)

`bipart ite.se ts(B)`	Get each set of nodes of bipartite graph
`bipart ite.pr oje cte d_g raph(B, X)`	Bipartite projected graph - nodes with bipartite friends in common
`P=bipa rti te.w ei ght ed_ pro jec ted _gr aph(B, X)`	projected graph with weights (number of friends in common)

Network Connectivity

```
G=nx.r ead _ad jli st( 'G_ adj lis t.txt', nodety pe=int)
```
Create from adjacency list

```
G=nx.G rap h(G _mat)
```
Create from matrix (np.array)

```
G=nx.r ead _ed gel ist ('G _ed gel ist.txt', data=[ ('W eight', int)])
```
Create from edgelist

```
G=nx.f rom _pa nda s_d ata fra me( G_df, 'n1', 'n2', edge_a ttr ='w eight')
```
Create from df

Adjacency list format
0 1 2 3 5
1 3 6 ...

Edgelist format:
0 1 14
0 2 17

Bipartite graphs

```
from networ kx.a lg orithms import bipartite
```

`bipart ite.is _bi par tite(B)`	Check if graph B is bipartite
`bipart ite.is _bi par tit e_n ode _se t(B ,set)`	Check if set of nodes is bipartition of graph

`nx.clu ste rin g(G, node)`	Local clustering coefficient
`nx.ave rag e_c lus ter ing(G)`	Global clustering coefficient
`nx.tra nsi tiv ity(G)`	Transitivity (% of open triads)
`nx.sho rte st_ pat h(G ,n1,n2)`	Outputs the path itself
`nx.sho rte st_ pat h_l eng th( G,n 1,n2)`	
`T=nx.b fs_ tree(G, n1)`	Create breadth-first search tree from node n1
`nx.ave rag e_s hor tes t_p ath _le ngth(G)`	Average distance between all pairs of nodes
`nx.dia met er(G)`	Maximum distance between any pair of nodes
`nx.ecc ent ric ity(G)`	Returns each node's distance to furthest node
`nx.rad ius(G)`	Minimum eccentricity in the graph
`nx.per iph ery(G)`	Set of nodes where eccentricity=diameter
`nx.cen ter(G)`	Set of nodes where eccentricity=radius

Connectivity: Network Robustness

`nx.nod e_c onn ect ivi ty(G)`	Min nodes removed to disconnect a network
`nx.min imu m_n ode _cut()`	Which nodes?
`nx.edg e_c onn ect ivi ty(G)`	Min edges removed to disconnect a network
`nx.min imu m_e dge _cut(G)`	Which edges?
`nx.all _si mpl e_p ath s(G , n1,n2)`	Show all paths between two nodes

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 4th June, 2018.
Last updated 4th June, 2018.

Cheatography

Network Analysis with Python and NetworkX Cheat Sheet

by RJ Murray (murenei) via cheatography.com/58736/cs/15946/

Network Connectivity: Connected Components

`nx.is_ con nec ted(G)`	Is there a path between every pair of nodes?
`nx.num ber _co nne cte d_c omp on e nts(G)`	# separate components
`nx.nod e_c onn ect ed_ com pon en t(G, N)`	Which connected component does *N* belong to?
`nx.is_ str ong ly_ con nec ted(G)`	Is the network connected direction-ally?
`nx.is_ wea kly _co nne cted(G)`	Is the directed network connected if assumed undire-cted?

Common Graphs

`G=nx.k ara te_ clu b_g raph ()`	Karate club graph (social network)
`G=nx.p ath _gr aph(n)`	Path graph with n nodes
`G=nx.c omp let e_g raph(n)`	Complete graph on n nodes
`G=rand om_ reg ula r_g rap - h(d,n)`	Random d-regular graph on n-nodes

See NetworkX Graph Generators reference for more.
Also see “An Atlas of Graphs” by Read and Wilson (1998).

Influence Measures and Network Centralization

Influence Measures and Network Centralization (cont)

`nx.edg e_b etw een nes s_c ent ral ity(G)`	B e
`nx.edg e_b etw een nes s_c ent ral ity _su bse t( G ,{s ubset})`	B s o e

Normalization: Divide by number of pairs of nodes.

PageRank and Hubs & Authorities Algorithms

`nx.pag era nk(G, alpha=0.8)`	Scaled PageRank of G with dampening parameter
`h,a=nx.hi ts(G)`	HITS algorithm - outputs 2 dictio-naries (hubs, authorities)
`h,a=nx.hi ts( G,m ax_ ite r=1 0,n orm ali - zed =True)`	Constr-ained HITS and normalized by sum at each stage

Centrality measures make different assumptions about what it means to be a “central” node. Thus, they produce different rankings.

Network Evolution - Real-world Applications

`G.degree(),G.in_d egree(),G.out_ deg ree()`	Distributio of node degrees
Preferential Attachment Model	Results in power law -> many nodes with low degrees; few with high degrees

`dc=nx.d eg ree _ce ntr ali ty(G)`	Degree centrality for network
`dc[node]`	Degree centrality for a node
`nx.in_ deg ree _ce ntr ali ty(G),nx.out _de g re e_c ent ral ity(G)`	DC for directed networks
`cc=nx.c lo sen ess _ce ntr ali ty( G,n orm al i zed =True)`	Closeness centrality (normalised) for the network
`cc[node]`	Closeness centrality for an individual node
`bC=nx.b et wee nne ss_ cen tra lity(G)`	Betweenness centrality
`..., normal ize d=T rue ,...)`	Normalized betweenness centrality
`..., endpoi nts =False, ...)`	BC excluding endpoints
`..., K=10,...)`	BC approximated using random sample of K nodes
`nx.bet wee nne ss_ cen tra lit y_s ubs et( G, { sub set})`	BC calculated on subset

`G=bara bas i_a lbe rt_ gra ph(n,m)`	Preferential Attachment Model with *n* nodes and each new node attaching to *m* existing nodes
Small World model	High average degree (global clustering) and low average shortest path
`G=watt s_s tro gat z_g rap h(n ,k,p)`	Small World network of *n* nodes, connected to its *k* nearest neighbours, with chance *p* of rewiring
`G=conn ect ed_ wat ts_ str oga tz_ gra ph( n,k,p, t)`	*t* = max iterations to try to ensure connected graph
`G=newm an_ wat ts_ str oga tz_ gra ph( n,k ,p)`	*p* = probability of adding (no rewiring)
Link Prediction measures	How likely are 2 nodes to connect, given an existing network

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 4th June, 2018.
Last updated 4th June, 2018.

Network Evolution - Real-world Applications (cont)	
`nx.com mon _ne igh bor s(G ,n1,n2)`	Calc common neighbors of nodes *n1*, *n2*
`nx.jac car d_c oef fic ient(G)`	Normalised common neighbors measure
`nx.res our ce_ all oca tio n_i ndex (G)`	Calc RAI of all nodes not already connected by an edge
`nx.ada mic _ad ar_ ind ex(G)`	As per RAI but with log of degree of common neighbor
`nx.pre fer ent ial _at tac hme nt(G )`	Product of two nodes' degrees
Community Common Neighbors	Common neighbors but with bonus if they belong in same 'community'
`nx.cn_ sou nda raj an_ hop cro ft(n 1, n2)`	CCN score for *n1*, *n2*
`G.node ['A '][ 'co mmu nit y']=1`	Add community attribute to node
`nx.ra_ ind ex_ sou nda raj an_ hop - cro ft(G)`	Community Resource Allocation score

These scores give only an indication of whether 2 nodes are likely to connect.
To make a link prediction, you would use these scores as features in a classification ML model.

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 4th June, 2018.
Last updated 4th June, 2018.