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.read_adjlist('G_adjlist.txt', nodetype=int)`	Create from adjacency list
`G=nx.Graph(G_mat)`	Create from matrix (np.array)
`G=nx.read_edgelist('G_edgelist.txt', data=[('Weight', int)])`	Create from edgelist
`G=nx.from_pandas_dataframe(G_df, 'n1', 'n2', edge_attr='weight')`	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 networkx.algorithms import bipartite`

`bipartite.is_bipartite(B)`	Check if graph B is bipartite
`bipartite.is_bipartite_node_set(B,set)`	Check if set of nodes is bipartition of graph

`nx.clustering(G, node)`	Local clustering coefficient
`nx.average_clustering(G)`	Global clustering coefficient
`nx.transitivity(G)`	Transitivity (% of open triads)
`nx.shortest_path(G,n1,n2)`	Outputs the path itself
`nx.shortest_path_length(G,n1,n2)`	
`T=nx.bfs_tree(G, n1)`	Create breadth-first search tree from node n1
`nx.average_shortest_path_length(G)`	Average distance between all pairs of nodes
`nx.diameter(G)`	Maximum distance between any pair of nodes
`nx.eccentricity(G)`	Returns each node's distance to furthest node
`nx.radius(G)`	Minimum eccentricity in the graph
`nx.periphery(G)`	Set of nodes where eccentricity=diameter
`nx.center(G)`	Set of nodes where eccentricity=radius

Connectivity: Network Robustness

`nx.node_connectivity(G)`	Min nodes removed to disconnect a network
`nx.minimum_node_cut()`	Which nodes?
`nx.edge_connectivity(G)`	Min edges removed to disconnect a network
`nx.minimum_edge_cut(G)`	Which edges?
`nx.all_simple_paths(G, n1,n2)`	Show all paths between two nodes

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)`	E e
`nx.edg e_b etw een nes s_c ent ral ity _su bse t( G ,{s ubset})`	E 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()`	Distribution 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 (norma- lised) for the network
`cc[node]`	Closeness centrality for an individual node
`bC=nx.b et wee nne ss_ cen tra lity(G)`	Betwee- nness centrality
`..., normal ize d=T rue ,...)`	Normalized betwee- nness centrality
`..., endpoi nts =False, ...)`	BC excluding endpoints
`..., K=10,...)`	BC approx- imated 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)`	Preferentia Attachmen 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 neighb- ours, with chance *p* c 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* = probab ility of adding (no rewiring)
Link Prediction measures	How likely are 2 nodes to connect, given an existing network

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.