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.Graph()`	
`G=nx.MultiGraph()`	Create a graph allowing parallel edges
`G.add_edges_from([(0, 1),(0, 2),(1, 3),(2, 4)]`	Create graph from edges
`nx.draw_networkx(G)`	Draw the graph
`G.add_node('A',role='manager')`	Add a node
`G.add_edge('A','B',relation = '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.edges())`	Return as list instead of EdgeView class
`G.nodes(data=True), G.edges(data=True)`	Include node/edge attributes
`G.nodes(data='relation)`	Return specific attribute

Creating graphs from data

`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
`bipartite.sets(B)`	Get each set of nodes of bipartite graph
`bipartite.projected_graph(B, X)`	Bipartite projected graph - nodes with bipartite friends in common
`P=bipartite.weighted_projected_graph(B, X)`	projected graph with weights (number of friends in common)

Network Connectivity

`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_connected(G)`	Is there a path between every pair of nodes?
`nx.number_connected_components(G)`	# separate components
`nx.node_connected_component(G, N)`	Which connected component does *N* belong to?
`nx.is_strongly_connected(G)`	Is the network connected directionally?
`nx.is_weakly_connected(G)`	Is the directed network connected if assumed undirected?

Common Graphs

`G=nx.karate_club_graph()`	Karate club graph (social network)
`G=nx.path_graph(n)`	Path graph with n nodes
`G=nx.complete_graph(n)`	Complete graph on n nodes
`G=random_regular_graph(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

`dc=nx.degree_centrality(G)`	Degree centrality for network
`dc[node]`	Degree centrality for a node
`nx.in_degree_centrality(G)`, `nx.out_degree_centrality(G)`	DC for directed networks
`cc=nx.closeness_centrality(G,normalized=True)`	Closeness centrality (normalised) for the network
`cc[node]`	Closeness centrality for an individual node
`bC=nx.betweenness_centrality(G)`	Betweenness centrality
`..., normalized=True,...)`	Normalized betweenness centrality
`..., endpoints=False, ...)`	BC excluding endpoints
`..., K=10,...)`	BC approximated using random sample of K nodes
`nx.betweenness_centrality_subset(G,{subset})`	BC calculated on subset
`nx.edge_betweenness_centrality(G)`	BC on edges
`nx.edge_betweenness_centrality_subset(G,{subset})`	BC on subset of edges

Normalization: Divide by number of pairs of nodes.

PageRank and Hubs & Authorities Algorithms

`nx.pagerank(G, alpha=0.8)`	Scaled PageRank of G with dampening parameter
`h,a=nx.hits(G)`	HITS algorithm - outputs 2 dictionaries (hubs, authorities)
`h,a=nx.hits(G,max_iter=10,normalized=True)`	Constrained 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_degree()`, `G.out_degree()`	Distribution of node degrees
Preferential Attachment Model	Results in power law -> many nodes with low degrees; few with high degrees
`G=barabasi_albert_graph(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=watts_strogatz_graph(n,k,p)`	Small World network of *n* nodes, connected to its *k* nearest neighbours, with chance *p* of rewiring
`G=connected_watts_strogatz_graph(n,k,p, t)`	*t* = max iterations to try to ensure connected graph
`G=newman_watts_strogatz_graph(n,k,p)`	*p* = probability of adding (not rewiring)
Link Prediction measures	How likely are 2 nodes to connect, given an existing network
`nx.common_neighbors(G,n1,n2)`	Calc common neighbors of nodes *n1*, *n2*
`nx.jaccard_coefficient(G)`	Normalised common neighbors measure
`nx.resource_allocation_index(G)`	Calc RAI of all nodes not already connected by an edge
`nx.adamic_adar_index(G)`	As per RAI but with log of degree of common neighbor
`nx.preferential_attachment(G)`	Product of two nodes' degrees

Network Evolution - Real-world Applications (cont)

Community Common Neighbors	Common neighbors but with bonus if they belong in same 'community'
`nx.cn_soundarajan_hopcroft(n1, n2)`	CCN score for *n1*, *n2*
`G.node['A']['community']=1`	Add community attribute to node
`nx.ra_index_soundarajan_hopcroft(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.