import networkx as nx

import matplotlib.pyplot as plt

图中的每一个顶点Node都有一个关键的ID属性，用于唯一标识一个节点，ID属性可以整数或字符类型；顶点除了ID属性之外，还可以自定义其他的属性，其它属性一般也是字典

点和线的 ID:属性 是一组键值对，属性可以为空但ID必须要有

点id从0开始

读写

G = nx.read\_edgelist ("file.edges", nodetype=int) #读edge list

nx.write\_edgelist(G, "file.edges") #保存为edge list

G = nx.read\_weighted\_edgelist("wf.edges") #读weighted edge list

nx.write\_weighted\_edgelist(G, "wf.edges") #保存为weighted edge list

G=nx.read\_adjlist(‘file.adjlist’) #读adjacency list (始点,入点1,入点2)

G=nx.read\_gml("file.gml",label=attr) #读gml文件

nx.write\_gml(G, "file.gml") #写gml文件

图

## 创建图和绘制

G = nx.Graph() #创建空的无多重边无向图

G = nx.DiGraph() #创建空的无多重边有向图

G = nx.MultiGraph() #创建空的有多重边无向图

G = nx.MultiDiGraph() #创建空的有多重边有向图

G.to\_undirected() #转换成无向图

G.reverse() #有向图箭头转向

G2=G.copy() #复制图

nx.subgraph(G,点id元组) #创建子图

**创建交互式网络**

from pyvis.network import Network

nt = Network('500px', '500px')

nt.from\_nx(G)

nt.show('nx.html')

**显示基本网络图形**

nx.draw(G)

plt.show()

nx.draw\_networkx(G,pos=点位置dict, with\_labels=True,

arrows=bool,arrowsize, ax=plt.ax, edge\_color, edge\_cmap, edge\_vmin,edge\_vmax,

nodelist, edgelist, node\_size, node\_color, node\_shape, alpha,

cmap, vmin, vmax,

linewidths=点轮廓宽度, width=边宽度,

labels=dict(点id:label), font\_size, font\_color,label=图例标签)

## 查看宏观属性

nx.is\_directed(G) #是否有向

nx.is\_weighted(G) #是否有权

nx.info(G) #边数和点数

G.number\_of\_nodes() #总点数

G.number\_of\_edges() #总边数

G.nodes #查看所有点ID

G.edges #查看所有边

nx.density(G) #查看图密度

list(dict(G.degree()).values() #查看所有点的度

nx.adjacency\_matrix(G).todense() #邻接矩阵

list(nx.generate\_adjlist(G)) #邻接矢量

nx.incidence\_matrix(G) .todense() #点边矩阵

list(nx.generate\_edgelist(G)) #点边矢量

nx.diameter(G) #网络直径(网络中最大的点eccentricity)

nx.radius(G) #网络半径(网络中最小的点eccentricity)

**连通性**

nx.is\_connected(G) #判断无向图是否是连通的(是否有隔绝的点/component存在)

nx.is\_strongly\_connected(G) #判断有向图是否是strongly connected

nx.is\_weakly\_connected(G) #判断有向图是否是weakly connected

**Components**

comps=sorted(nx.connected\_components (G), key=len, reverse=True)

#以集合的形式按大小返回图中的component

comp=nx.subgraph(G, comps[i]) #取component

nx.number\_connected\_components(G) #Component个数

nx.weakly\_connected\_components(D) #所有弱连通的component

nx.strongly\_connected\_components(D) #所有强连通的component

**计算PageRank**

pagerank(G, alpha=0.85, personalization=None, max\_iter=100, tol=1e-06, nstart=None, weight='weight', dangling=None)

#无向边会被转换成2条有向边

#alpha是阻尼系数

#weight是选择边属性中的键值作为权重，缺省值为1

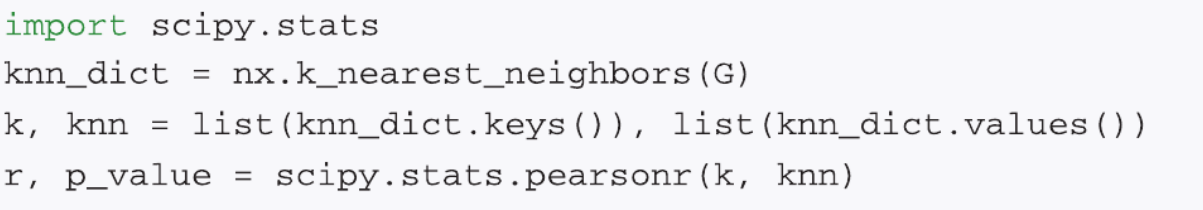
**同配性**

nx.attribute\_assortativity\_coefficient(G, attr) #计算网络属性的同配性(点属性为分类变量)

nx.numeric\_assortativity\_coefficient(G, attr) #计算网络属性的同配性(点属性为数值变量)

nx.degree\_assortativity\_coefficient (G) #计算整个网络的点度同配性

**同配性假设检验**



**聚合系数**

nx.clustering(G) #所有点的聚合系数(返回字典)

nx.clustering(G,id) #单点的聚合系数

nx.average\_clustering(G) #网络的聚合系数

**其它**

nx.is\_tree(G) #判断图是否是树(删除任一边将拆分为2个component)

nx.triangles(G) #计算图中每个点的三角形数量(返回字典)

sum(nx.triangles(G).values)/3 #计算图中三角形数

list(nx.selfloop\_edges(G)) #查看图中的self-loop

## 社群

**检测某partition是否有效(包含网络中所有点，且每个点仅属于一个区块)**

nx.is\_partition(G,partition\_list) #partition\_list为一个元素是node集合的list

**Kernighan-Lin Network Bipartition(network partition, 节点交换+最小化cut size)**

nx.community.kernighan\_lin\_bisection(G)

**Girvan Newman社群算法(基于betweeness移除桥梁边)**

nx.community.girvan\_newman(G)

# Girvan Newman社群拆解，返回一个hierarchical partitions list

**Modularity Methods**

nx.community.quality.modularity(G, partition)

#计算某partition下的modularity值

nx.community.greedy\_modularity\_communitie(G)

#基于modularity和阶层聚类计算最佳partition方案

**Louvain社群算法(聚类+转换ego network)**

import community

community.best\_partition(G)

**标签传播 Label Propagation**

nx.community.asyn\_lpa\_communities (G)

**Zachary’s karate club Network(一个真实数据的社群网络，常用于评测社群发现算法)**

nx.karate\_club\_graph()

## 删除元素

G.clear() #清空图

G.remove\_node(node\_id) #删除一个点

G.remove\_nodes\_from(nodes\_list) #删除一堆点

G.remove\_edge(edge\_id) #删除一条边

G.remove\_edges\_from(edges\_list) #删除一堆边

## 改变布局方式

nx.draw(G, pos)

pos = nx.spring\_layout(G,iterations=200)

pos = nx.spectral\_layout(G)

pos = nx.circular\_layout(G)

pos = nx.kamada\_kawai\_layout(G)

pos = nx.random\_layout(G)

pos = nx.shell\_layout(G)

## 模板网络&网络模型

nx.complete\_graph(n) #全链接网络

nx.complete\_bipartite\_graph(n1,n2) #全连接二分网络

nx.cycle\_graph(n) #环状网络

nx.path\_graph(n) #链条网络

nx.star\_graph(n) #星形网络

**Random网络**

nx.gnm\_random\_graph(N,L=给定总边数,p=链接概率,directed=False)

**Small World网络**

nx.watts\_strogatz\_graph(N,k=初始k近邻数,p=边重塑概率)

**Configuration网络**

nx.configuration\_model(degree\_list)

#指定点度，链接随机生成

**BA Preferential Attachment网络**

nx.barabasi\_albert\_graph(N,m=每个新点所引入的边数)

#点度大的点更易吸引新点，旧点更易成为hub

**Stochastic Block Model**

nx.generators.stochastic\_block\_model(S=community\_size\_list, P=link\_prob\_matrix)

#将网络社群中的点的链接概率描述为一个矩阵

**Planted Partition Model**

nx.generators.planted\_partition\_graph(q=社群数, nc=社群大小, p\_int, p\_ext)

#此模型假设所有社群的尺寸都相同，只有同社群链接概率和异社群链接概率

**Zachary's Karate Club（一个自带两个社区的样本网络，用以评估社群检测算法）**

nx.karate\_club\_graph()

#存在点属性 ‘club’ 描述各点所属的社群

## Core-decomposition

nx.core\_number(G) #返回网络中每个节点的core层级

nx.k\_shell(G,k) #提取出k-shell

nx.k\_core(G,k) #提取出k-core

nx.k\_core(G) #提取出最后一层core

点

## 添加点

G.add\_node('a') #用id添加点a

#在向图中添加顶点时，除ID属性之外，也可以向顶点中增加自定义的属性，例如，名称属性，权重属性

G.add\_nodes\_from(nodes\_list) #批量添加点

G.add\_node(1,1) #用坐标来添加点

G.add\_node(1,name='n1',weight=1)

## 查看&设置点属性

list(G.nodes[any\_node\_id].keys()) #查看属性列表

G.\_node[id].update({key:value})

G.\_node[id][key]=value #更新点属性

del G.nodes[id][key] #删除点属性

G.nodes[id][ 'label']= 'text' #设置点label

G.nodes[id]['title'] = 'text' #设置点悬浮文本

G.nodes[id]['group'] = 1 #设置点类别(将以不同颜色体现)

G.nodes[id][ 'size']=10 #设置点大小

nx.set\_node\_attributes(G,attr\_dict,’keyname’) #批量设置点属性

部分函数如nx.degree(G)返回的不是dict，需要转换类型

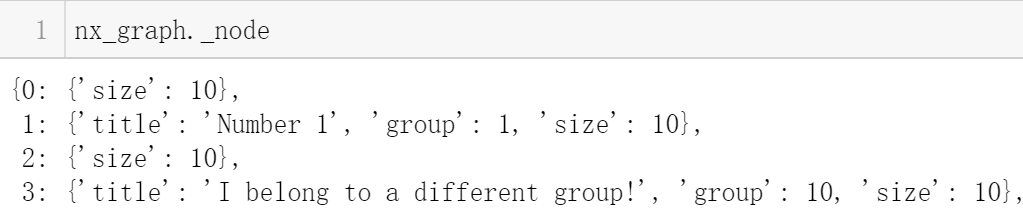
nx.get\_node\_attributes(G,'keyname') #批量查看点属性(返回字典)

## 计算点属性

G.has\_node(id) #查询是否有指定点

G.nodes #查看所有点的ID

G\_node #以字典形式返回所有点的键值对



G\_node(id) #通过ID查看某个点的属性

G.neighbors(id) #查看点的邻点

G.predecessors(id) #查看点的来点

G.successors(id) #查看点的success

G[id] #查看指定点

dict(G.degree()) #查看所有点的度

dict(G.in\_degree()) #查看所有点入度

dict(G.out\_degree()) #查看所有点出度

G.in\_degree(id) #查看点的入度

G.out\_degree(id) #查看点的出度

G.degree(id,weight=’weight’) #查看点的权度

nx.betweenness\_centrality(G) #查看所有点的中介中心度(返回字典)

nx.betweenness\_centrality(G)[node] #查看点的接近中介中心度

nx.closeness\_centrality(G) #查看所有点的接近中心度(返回字典), 对于有向图默认是in-closeness centrality

nx.closeness\_centrality(G,u=node) #查看点的接近中心度

nx.closeness\_centrality(G.reverse()) #对于无向图，计算out-closeness centrality

nx.pagerank(D) #查看所有点的PageRank

nx.eccentricity(G) #计算所有点eccentricity(到各点最短路径长度最大值)

## 查看点对路径

nx.has\_path(G, id1, id2) #判断两点是否连通

nx.shortest\_path(G, id1, id2) #返回两点最短路径

nx.shortest\_path\_length(G, id1, id2) #返回两点最短路径长度

nx.all\_simple\_paths(G, id1, id2) #返回两点所有路径

nx.shortest\_path(G, id) #返回网络中所有点到该点的最短路径

nx.shortest\_path\_length(G, id) #返回网络中所有点到该点的最短路径长度

nx.shortest\_path(G) #返回网络中所有点对的最短路径

nx.shortest\_path\_length (G) #返回网络中所有点对的最短路径长度

nx.average\_shortest\_path\_length(G) #返回网络平均最短路径长度(网络中不能有互相隔绝的点对)

上述方法可额外添加超参数‘weight’来计算加权路径

e.g.

nx.shortest\_path\_length(G, id1, id2,’weight’) #返回两点最短加权路径长度

## 绘图设置点

nx.draw\_networkx\_nodes(G, pos,

nodelist=None,

node\_size=300,

node\_color='r',

node\_shape='o',

alpha=1.0,

cmap=None,

vmin=None,

vmax=None,

ax=None,

linewidths=None,

label=None,

\*\*kwds):

**pos**：dictionary

将节点作为键和位置作为值的字典。

位置应该是长度为2的序列。

**ax**：Matplotlib Axes对象，可选

在指定的Matplotlib轴中绘制图形。

**nodelist**：list，可选

只绘制指定的节点（默认G.nodes（））

**node\_size**：标量或数组

节点大小（默认值= 300）。如果指定了数组，它必须是

与点头长度相同。

**node\_color**：颜色字符串或浮点数组

节点颜色。可以是单颜色格式字符串（default ='r'），

或者具有与点头相同长度的颜色序列。

如果指定了数值，它们将被映射到

颜色使用cmap和vmin，vmax参数。看到

matplotlib.scatter更多详细信息。

**node\_shape**：string

节点的形状。规格为matplotlib.scatter

标记，'so ^> v <dph8'（默认='o'）之一。

**alpha**：float

节点透明度（默认值= 1.0）

**cmap**：Matplotlib色(mang)图

色彩映射节点的强度（默认=无）

cmap=plt.cm.get\_cmap('Greens') #选择色板

node\_color=[cmap(u/(0.01+max(value\_list))) for u in value\_list]

#value\_list可以是degree列表, centrality列表等

**vmin，vmax**：float

节点色彩映射缩放的最小值和最大值（默认值=无）

**width**：[无|标量|序列]

符号边框的线宽（默认值= 1.0）

**label**：[无|串]

图例的标签

**分社群上色**

1. 创建一个partition\_map = dict(), 记录每个节点所属社群编号
2. 将[partition\_map[n] for n in G.nodes] 传入nx.draw的node\_color参数

边

## 添加边

G.add\_edge(id1,id2) #添加边,起点为id1，终点为id2

G.add\_edges\_from(edge\_list) #批量添加边

G.add\_edge(id1,id2,weight=s) #添加权边

G.add\_weighted\_edges\_from ([(id1,id2,权重)]) #批量添加权边，第三个输入量为权值，注意输入的一条边是元组

G[id1][id2].update({key: value})

G[id1][id2][key]=value #更新边属性

del G[id1][id2][key] #删除边属性

## 查看&设置边属性

list(G.edges[any\_edge\_id].keys()) #查看属性列表

G.edges #查看所有边

G.edges(data=True) #查看所有边及边属性

G[id1][id2]

G.get\_edge\_data(id1,id2) #查看两个点间边的属性

for (i,j) in G.edges: 遍历无权图边

for (i,j,w) in G.edges(data=’weight’): 遍历权图权重边

nx.edge\_betweenness\_centrality(G) #查看所有边的中介中心度

nx.set\_edge\_attributes(G,attr\_dict,’keyname’) #批量设置边属性

## 绘图设置边

def draw\_networkx\_edges(G, pos,

edgelist=None,

width=1.0,

edge\_color='k',

style='solid',

alpha=1.0,

edge\_cmap=None,

edge\_vmin=None,

edge\_vmax=None,

ax=None,

arrows=True,

label=None,

\*\*kwds):

G：图表

一个networkx图

pos：dictionary

将节点作为键和位置作为值的字典。

位置应该是长度为2的序列。

edgelist：边缘元组的集合

只绘制指定的边（默认= G.edges（））

width：float或float数组

边线宽度（默认值= 1.0）

edge\_color：颜色字符串或浮点数组

边缘颜色。可以是单颜色格式字符串（default ='r'），

或者具有与edgelist相同长度的颜色序列。

如果指定了数值，它们将被映射到

颜色使用edge\_cmap和edge\_vmin，edge\_vmax参数。

style：string

边线样式（默认='solid'）（实线|虚线|点线，dashdot）

alpha：float

边缘透明度（默认值= 1.0）

edge\_ cmap：Matplotlib色彩映射

用于映射边缘强度的色彩映射（默认值=无）

edge\_vmin，edge\_vmax：float

边缘色(mang)图缩放的最小值和最大值（默认值=无）

ax：Matplotlib Axes对象，可选

在指定的Matplotlib轴中绘制图形。

arrows：bool，optional（default = True）

对于有向图，如果为真，则绘制箭头。

label：图例的标签