STA 141A
Fundamentals of Statistical Data Science

Fall 2016

Instructor: Debashis Paul

Lecture 12

Working with graphs in R

- Relational data, or network data, that consist of information on the presence or absence of a relationship or connection between pairs of objects or entities, are typically expressed in the form of a **graph** consisting of a set of nodes (or vertices) and a collection of edges (directed or undirected) connecting pairs of nodes (included self-edge connecting a node with itself).
- We introduce the **igraph** package in R to organize, process and visualize graphical or network data.
- Reference: Kolaczyk, E. D. and Csardi, G. (2014). "Statistical Analysis of Network Data with R". Springer.

Basic terminology

- A graph consists of a set of nodes or vertices and a collection of edges (directed or undirected) joining pairs of nodes.
- adjacent vertices : two nodes u and v of a graph are "adjacent" if there is an edge connecting them
- neighbor : if nodes u and v are adjacent, then they are neighbors
- adjacent edges : two edges e1 and e2, say, are adjacent if they share a common node
- incident node : a node v is incident on an edge e of the graph if v is an endpoint of e
- degree : degree of node v is the number of edges incident on the node v
- regular graph : a regular graph is a graph in which every node has the same degree
- connected graph : a graph is connected if any two points can be joined by a path (a sequence of edges that are pairwise adjacent)

Terminology (cont.)

- complete graph : a graph is complete if every node is connected to all the other nodes
- clique : a clique is a subgraph of an undirected graph that is complete
- tree : a tree is an undirected graph in which any two vertices are connected by exactly one path; in other words, a tree is a connected undirected graph with no cycle
- forest : a graph consisting of disjoint union of trees is called a forest
- directed acyclic graph (DAG) : a DAG is a directed graph that has no cycle
- bipartite graph : a bipartite graph G = (V,E) is a graph with the property that the vertex set V can be partitioned into disjoint sets V1 and V2, such that each edge has one endpoint in V1 and the other in V2

Creating undirected graph

```
library(igraph)
gr1 = graph.formula(1-2, 1-3, 3-4, 3-5, 4-5, 4-6, 5-6, 6-7)
V(gr1)  # returns the set of vertices of gr1
E(gr1)  # returns the set of edges of gr1
str(gr1) # expresses the graph in a compact form, listing nodes that each node is connected to
plot(gr1)  # visual display of graph gr1
V(gr1)$name # returns the names of nodes of graph gr1
```

Creating directed graph

```
gr2 = graph.formula(a -+ b, a ++ c,  c ++ d, d +- e, f++ g)
# the sign + is used to indicate directionality
# the expression a -+ b is interpreted as "there is an edge from node a to node b"
V(gr2)  # returns the set of vertices of gr2
E(gr2)  # returns the set of directed edges of gr2
V(gr2)$name = c("Alex","Bob","Clo","Dan","Erin","Fang","Guan")
# assign names to the nodes of graph
V(gr2)$gender = c("M","M","F","M","F","M","F")  # gender of each node (person)
E(gr2)$distance = c(1,2,1,1,4,4,2,1)  # a measure of distance between nodes (persons)
plot(gr2)  # visual display of graph gr2  (notice the arrows)
```

Statistics on graphs

```
degree(gr1)  #  distribution of degree of the nodes of graph gr1
degree(gr2,mode="in")  # distribution of in-degree of graph gr2
degree(gr2,mode="out") # distribution of out-degree of graph gr2
is.connected(gr1)  # check if the graph is connected
clusters(gr2) # find the disconnected subgraphs of the graph gr2
get.edgelist(gr2) # obtain the (directed) edges in matrix format
neighbors(gr2,"Alex") # obtain the neighbors of node Alex
get.adjacency(gr2) # obtain the adjacency matrix of graph gr2 (in a sparse matrix representation)
matrix(get.adjacency(gr2),nrow=vcount(gr2),dimnames=list(V(gr2)$name,V(gr2)$name)) # in standard form
```

Operations on graphs

```
gr3 = induced.subgraph(gr2,1:5)
# extracts induced subgraph of gr2 corresponding to nodes 1 to 5
gr4 = gr2 - vertices(c(1,3))  # removes vertices 1 (Alex) and 3 (Clo) and edges incident to them
gr5 = gr2 + vertices(c("Ron","Mary")) # adds vertices "Ron" and "Mary"
gr6 = gr5 + edges(c(1,4),c(6,8),c(4,9),c(9,4))  # adds four directed edges (indexed by vertex indices)
gr7 = graph.union(gr3,gr4) # union of graphs gr3 and gr4
# (edges are included if they are present an at least one of the graphs)
graph.union(gr1,gr3) # returns error message since gr1 is undirected and gr3 is directed
```

Graph visualization

- Geometrical organization of the nodes, based on their neighborhood information, is a critical component of visualization of any graph
- We can use different layouts such as, "lattice", "circular" or many special types

```
g2.lat = graph.lattice(c(3,3,3)) # a 3 x 3 x 3 lattice graph (27 vertices)
plot(g2.lat) # usual "lattice" layout
plot(g2.lat, layout = layout.circle)  # "circular" layout
plot(g2.lat, layout = layout.star)  # "star" layout
plot(g2.lat, layout = layout.kamada.kawai)  # "Kamada-Kawai" layout (lattice-like)
plot(g2.lat, layout = layout.reingold.tilford)  # "Reingold-Tilford" layout  (tree-like)
```

Creating a graph from adjacency matrix

- graph_from_adjacency_matrix( ) is a flexible function for creating graphs from adjacency matrices.

```
# Typical usage:
# graph_from_adjacency_matrix(adjmatrix,
#       mode = c("directed", "undirected", "max", "min", "upper", "lower", "plus"), weighted = NULL)
adjm = matrix(sample(0:1, 100, replace=TRUE, prob=c(0.9,0.1)), nc=10)
g1 = graph_from_adjacency_matrix( adjm )
adjm = matrix(sample(0:5, 100, replace=TRUE, prob=c(0.9,0.02,0.02,0.02,0.02,0.02)), nc=10)
g2 = graph_from_adjacency_matrix(adjm, weighted=TRUE)
```