

Advanced Network Analysis

A Local Structure Graph Model

Olga Chyzh [www.olgachyzh.com]

Readings

- Emily Casleton, Daniel Nordman, and Mark S. Kaiser. A local structure model for network analysis. *Statistics and Its Interface*, 2(10), 2017.
- Olga V. Chyzh and Mark S. Kaiser. Network analysis using a local structure graph model. *Political Analysis*, 27(4):397{414, 2019.

Motivation

- Do edges among red nodes affect the formation of edges among turquoise nodes?
- If nodes 6 and 7 formed an edge, would that make it more likely that nodes 3 and 4 form an edge?

The Estimator

- Suppose i is an potential edge in a network of potential edges (realized and unrealized);
- Then $s_i = (u_i, v_i)$ is i 's location in Cartesian space.
- Denote the binary random variable, $y(s_i) = y_i$, so that:

$$y(s_i) = \begin{cases} 1 & \text{if edge } s_i \text{ is present} \\ 0 & \text{if edge } s_i \text{ is absent.} \end{cases}$$

- Define i 's neighbors as $N_i = \{s_j : s_j \text{ is a neighbor of } s_i\}$.
- Make a Markov assumption of conditional spatial independence:

$$f(y(s_i) | \mathbf{y}(s_j) : s_j \neq s_i) = f(y(s_i) | \mathbf{y}(N_i))$$

- If connectivities between edges are continuous, then the Markov assumption is redundant.

The DV

##		edge_id	Y
##	[1,]	"12"	"1"
##	[2,]	"13"	"1"
##	[3,]	"14"	"1"
##	[4,]	"15"	"0"
##	[5,]	"16"	"0"
##	[6,]	"17"	"0"
##	[7,]	"21"	"0"
##	[8,]	"23"	"1"
##	[9,]	"24"	"0"
##	[10,]	"25"	"0"

Set Up the Connectivity Matrix, W

- Start with an adjacency matrix among all potential edge-pairs.
- Code two edges as connected if they connect opposite-colored pairs of same-colored nodes.

	54	56	57	61	62
12	0	1	1	0	0
13	0	1	1	0	0
14	0	1	1	0	0
15	0	0	0	0	0
16	0	0	0	0	0

Set Up the Connectivity Matrix, W

```
W[1:10, 25:35]
```

##		51	52	53	54	56	57	61	62	63	64	65
##	12	0	0	0	0	1	1	0	0	0	0	1
##	13	0	0	0	0	1	1	0	0	0	0	1
##	14	0	0	0	0	1	1	0	0	0	0	1
##	15	0	0	0	0	0	0	0	0	0	0	0
##	16	0	0	0	0	0	0	0	0	0	0	0
##	17	0	0	0	0	0	0	0	0	0	0	0
##	21	0	0	0	0	1	1	0	0	0	0	1
##	23	0	0	0	0	1	1	0	0	0	0	1
##	24	0	0	0	0	1	1	0	0	0	0	1
##	25	0	0	0	0	0	0	0	0	0	0	0

The Binary Conditional Distribution

$$P(Y_i = y_i | \mathbf{y}(N_i)) = \exp[A_i(\mathbf{y}(N_i))y_i - B_i(\mathbf{y}(N_i))], \quad (1)$$

where A_i is a natural parameter function and $B_i = \log[1 + \exp(A_i(\mathbf{y}(N_i)))]$, and $\mathbf{y}(N_i)$ is a vector of values of the binary random variables (edges) of i 's neighbors.

The Natural Parameter Function

$$A_i(\mathbf{y}(N_i)) = \log\left(\frac{\kappa_i}{1 - \kappa_i}\right) + \eta \sum_{j \in N_i} w_{ij} (y_j - \kappa_j), \quad (2)$$

where $\log\left(\frac{\kappa_i}{1 - \kappa_i}\right) = \mathbf{X}_i^T \boldsymbol{\beta}$, \mathbf{X}_i is a column vector of k exogenous covariates, $\boldsymbol{\beta}$ is a k by 1 vector of estimation parameters, w_{ij} is the ij^{th} element of a matrix of connectivities among edges $\texttt{\textbf{W}}$, and η is its parameter.

- When $y_j > \kappa_j$, then the dependence term makes a positive contribution to $A_i(\mathbf{y}(N_i))$ ---complementary processes;
- When $y_j < \kappa_j$, then the dependence term makes a negative contribution to $A_i(\mathbf{y}(N_i))$ ---substitution-type processes;
- Key condition: $w_{ij} = w_{ji}$.
- Model does not require (prohibits) row-standardization of \mathbf{w} .

Estimation

$$\log PL = \sum_i \{y_i \log(p_i) + (1 - y_i) \log(1 - p_i)\}, \quad (3)$$

where:

$$p_i = \frac{\exp[A_i(y(N_i))]}{1 + \exp[A_i(y(N_i))]} \quad (4)$$

Let's Program This

```
#Likelihood
loglik<-function(par,W,Y){
  b0<-par[1]
  eta<-par[2]
  xbeta<-b0
  kappa<-exp(xbeta)/(1+exp(xbeta)) #logit of Xb
  A_i=log(kappa/(1-kappa))+eta*W%*%(Y-kappa) #Eqn 2
  p_i<- exp(A_i)/(1+exp(A_i)) #Eqn 1, also Eqn 4
  PL<-Y*log(p_i)+(1-Y)*log(1-p_i) #Eqn 3
  ell <- -sum(PL)
  #cat("ell",ell, fill=TRUE)
  return(ell)
}
```

Let's Estimate

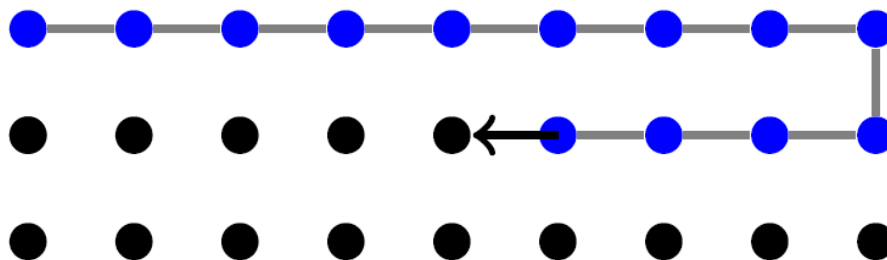
```
m1<-optim(par=c(0,0),loglik,W=Wmat,Y=Y)
m1
```

```
## $par
## [1] -3.1572208  0.8955375
##
## $value
## [1] 13.72339
##
## $counts
## function gradient
##          95          NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Estimating Standard Errors

Need to simulate Y based on our parameter estimates:

- Start from a vector of initial $y_0 = \{y_{01}, y_{02}, \dots, y_{0n}\}$ drawn from a binomial distribution;
- Moving row-wise, for $i = 1, \dots, n$, individually simulate/update each observation as a function of previously simulated observations:



- n individual updates provide 1 full Gibbs iteration
- Use the result of step 2 as the new initial values and repeat step 2 to obtain M simulated networks y (can burn-in, thin, etc.).

Simulate Networks Based on m1

1. Function `spatbin.genone` simulates/updates a value for y for a single observation--step 2
2. Function `spatbin.onegibbs` applies `spatbin.genone` to update every observation of y --step 3
3. Function `spatbin.genfield` applies `spatbin.onegibbs` to generate M networks --step 4

```
spatbin.genone<-function(coeffs,w,curys){  
  b0<-coeffs[1]  
  eta<-coeffs[2]  
  xbeta<- b0  
  kappa<-exp(xbeta)/(1+exp(xbeta))  
  A_i=log(kappa/(1-kappa))+eta*w%*%(curys-kappa)  
  p_i<- exp(A_i)/(1+exp(A_i))  
  y<- rbinom(n=length(curys), size=1, prob=p_i)  
  return(y)  
}
```

```
spatbin.onegibbs<-function(coeffs,w,curys){  
  cnt<-0  
  n<-length(curys)  
  newys<-NULL  
  repeat{  
    cnt<-cnt+1  
    ny<-spatbin.genone(coeffs=coeffs,w=w,curys=curys)  
    curys[cnt]<-ny[cnt]  
    if(cnt==n) break  
  }  
  newys<-curys  
  return(newys)  
}
```

```
spatbin.genfield<-function(coeffs,w,y0s,M){  
  curys<-y0s  
  cnt<-0  
  res<-as.data.frame(y0s)  
  repeat{  
    cnt<-cnt+1  
    newys<-spatbin.onegibbs(coeffs=coeffs,w=w,curys=curys)  
    curys<-newys  
    res<-cbind(res,curys)  
    if(cnt==M) break  
  }  
  
  return(res)  
}
```



```
n<-length(Y)
y0s=rbinom(n=n, size=1, prob=.5)
sims<-spatbin.genfield(coeffs=m1$par,w=W,y0s=y0s,M=1000)
#Take every 10th simulated network, i.e. burnin=10, thinning=10
sims<-sims[,seq(from=10, to=ncol(sims),by=10)]
```

Obtaining Standard Errors

1. Estimate the model on simulated networks (after burnin and thinning);
2. The standard errors are the standard deviations of the estimated coefficients.

Obtaining Standard Errors

```
sim_est<-function(Y){  
  res<-optim(par=m1$par,loglik,W=W,Y=as.matrix(Y))  
  return(c(res$par,res$convergence))  
}  
  
library(parallel)  
sim_est<-do.call("rbind",mclapply(sims, sim_est))  
#Drop results if didn't converge (models that converged have converged)  
sim_est<-sim_est[sim_est[,3]==0,]  
  
#Get sds of the estimates:  
boot_se<-apply(sim_est,2,sd)  
mytable<-cbind("coeff"=m1$par,"se"=boot_se[-3],"z-value"=(m1$par/boot_se[-3])  
mytable
```

```
##           coeff           se      z-value  
## [1,] -3.1572208  13.80388 -0.22871980  
## [2,]  0.8955375 111.93631  0.00800042
```

Application: International Alliances

#Open the data:

```
data("ally_data")
ally_data$tot_trade<-log(ally_data$tot_trade+1)
ally_data<-ally_data[ally_data$year==2007,]
ally_data[1:5,]
```

##	ccode1	ccode2	edge	defense	mil_ratio	tot_trade	joint_dem	year
## 62	2	20	1	1	0.7231990	2.653716	1	2007
## 75	2	31	2	1	0.9936821	2.206597	1	2007
## 227	2	42	5	1	0.9553047	2.331096	1	2007
## 268	2	51	6	1	0.9852507	2.215345	1	2007
## 305	2	52	7	1	0.9901861	2.335300	1	2007

#Prepare W:

```
W2007 <- W
W2007[1:5,1:5]
```

##	edge_diff1	edge_diff2	edge_diff5	edge_diff6	edge_diff7
## [1,]	0.000000e+00	1.777676e-06	1.858189e-06	2.092406e-06	2.086917e-06
## [2,]	1.777676e-06	0.000000e+00	8.051235e-08	3.147301e-07	3.092406e-07
## [3,]	1.858189e-06	8.051235e-08	0.000000e+00	2.342178e-07	2.287283e-07
## [4,]	2.092406e-06	3.147301e-07	2.342178e-07	0.000000e+00	5.489462e-09

Likelihood (1 X)

```
#Likelihood
loglik<-function(par,X,W,Y){
  b0<-par[1]
  b1<-par[2]
  eta<-par[3]
  xbeta<-b0+b1*X
  kappa<-exp(xbeta)/(1+exp(xbeta)) #logit of Xb
  A_i=log(kappa/(1-kappa))+eta*W%*%(Y-kappa) #Eqn 2
  p_i<- exp(A_i)/(1+exp(A_i)) #Eqn 1, also Eqn 4
  PL<-Y*log(p_i)+(1-Y)*log(1-p_i) #Eqn 3
  ell <- -sum(PL)
  #cat("ell",ell, fill=TRUE)
  return(ell)
}
```

Let's Estimate

```
X=ally_data$tot_trade
Y=ally_data$defense
m1<-optim(par=c(0,0,0),loglik,X=X,W=W,Y=Y)
m1
```

```
## $par
## [1] -1.686244323 -0.007318607  0.936193299
##
## $value
## [1] 470.8606
##
## $counts
## function gradient
##      88      NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Standard Errors

```
spatbin.genone<-function(coeffs,X,w,curys){  
  b0<-coeffs[1]  
  b1<-coeffs[2]  
  eta<-coeffs[3]  
  xbeta<- b0+b1*X  
  kappa<-exp(xbeta)/(1+exp(xbeta))  
  A_i=log(kappa/(1-kappa))+eta*w%*%(curys-kappa)  
  p_i<- exp(A_i)/(1+exp(A_i))  
  y<- rbinom(n=length(curys), size=1, prob=p_i)  
  return(y)  
}
```

```
spatbin.onegibbs<-function(coeffs,X,w,curys){  
  cnt<-0  
  n<-length(curys)  
  newys<-NULL  
  repeat{  
    cnt<-cnt+1  
    ny<-spatbin.genone(coeffs=coeffs,X=X,w=w,curys=curys)  
    curys[cnt]<-ny[cnt]  
    if(cnt==n) break  
  }  
  newys<-curys  
  return(newys)  
}
```



```
spatbin.genfield<-function(coeffs,X,w,y0s,M){  
  curys<-y0s  
  cnt<-0  
  res<-as.data.frame(y0s)  
  repeat{  
    cnt<-cnt+1  
    newys<-spatbin.onegibbs(coeffs=coeffs,X=X,w=w,curys=curys)  
    curys<-newys  
    res<-cbind(res,curys)  
    if(cnt==M) break  
  }  
  
  return(res)  
}
```

Simulate 1000 Random Networks

```
n<-length(Y)
y0s=rbinom(n=n, size=1, prob=.5)
sims<-spatbin.genfield(coeffs=m1$par,X=X,w=W,y0s=y0s,M=1000)
#Take every 10th simulated network, i.e. burnin=10, thinning=10
sims<-sims[,seq(from=10, to=ncol(sims),by=10)]
saveRDS(sims, "sims.rds")
```

Estimate an LSGM on Each of the Simulated Networks

```
sims<-readRDS("data/sims.rds")
sim_est<-function(Y){
  res<-optim(par=m1$par,loglik,X=X,W=W,Y=as.matrix(Y))
  return(c(res$par,res$convergence))
}

library(parallel)
sim_est<-do.call("rbind",mclapply(sims, sim_est))
#Drop results if didn't converge (models that converged have converged)
sim_est<-sim_est[sim_est[,4]==0,]
saveRDS(sim_est,"sim_est.rds")
```

Calculate SEs and Make a Table

```
#Get sds of the estimates:
sim_est<-readRDS("data/sim_est.rds")
boot_se<-apply(sim_est,2,sd)
mytable<-cbind("coeff"=m1$par,"se"=boot_se[-4],"z-value"=(m1$par/boot
mytable
```

```
##           coeff           se      z-value
## [1,] -1.686244323 2.318147e-01 -7.2741041749
## [2,] -0.007318607 3.253597e-02 -0.2249389529
## [3,]  0.936193299 2.832822e+03  0.0003304808
```

Package LSGM

Note: the package is in beta testing

```
library(devtools)
install_github("ochyzh/lsgm")
library(lsgm)
data(W)
data(toy_data)
lsgm(Y=as.matrix(toy_data$Y),W=W,X=as.data.frame(toy_data$X))
```