Exercise

SAOM

Advanced Social Network Analysis

Background

In this exercise, I want you to use some of the data I presented in the first session. The data is about a friendship network in a Dutch school class. The data were collected between September 2003 and June 2004.

There are 26 students who were followed over their first year at secondary school during which friendship networks as well as other data were assessed at four time points at intervals of three months. There were 17 girls and 9 boys in the class, aged 11-13 at the beginning of the school year. Network data were assessed by asking students to indicate up to twelve classmates which they considered good friends.

Ultimately, I want you to run a network dynamics (only) model that replicates Model 0 (Table 1) in Snijders et al. (2010). The question is what drives network change in this school class?

More information

http://www.stats.ox.ac.uk/~snijders/siena/tutorial2010_data.htm

Data

http://www.stats.ox.ac.uk/~snijders/siena/klas12b.zip

You can find the data in the data folder that I prepared for this workshop.

Exercise 1

- Load the network datasets `klas12b-net-1.dat`, `klas12b-net-2.dat`, `klas12b-net-3.dat` and `klas12b-net-4.dat` from the data folder.
    - `read.table(...)` is a good start
- Transform the data.frame objects you just generated in matrix objects and save these matrices as `net1`, `net2`, `net3` and `net4`.
    - Use the command `as.matrix(...)`
- The networks are coded as follows: Friendship: 0 = no, 1 = yes, 9 = missing, 10 = not a member of the classroom (structural zero). In RSiena network values can only be 0,1,10,11 (structural ones).
- Therefore, you need to deal with the missing code 9. Here, it makes sense to set it to "structural zero".
    - a) You can subset a network with this syntax: `net1[net1==9]`
    - b) You can assign whatever is on the right side to whatever is on the left side or an arrow: `a <- 10`
- Therefore, you need to deal with the missing code 9. Here, it makes sense to set it to "structural zero".
- Next, load some additional data saved in `klas12b-demographics`.The file `klas12b-demographics.dat` contains the following four variables saved in four columns.
    - Sex (1 = girl, 2 = boy)
    - Age (years)
    - Ethnicity (1 = Dutch, 2 = other, 0 = missing)
    - Religion (1 = Christian, 2 = non-religious, 3 = non-Christian religion, 0 = missing)
- Generate a new object `sex.d` which holds the first column of the data.frame you just loaded.
    - You can select the first column of a data.frame called `data` with `data[,1]`
- Lastly, load the dyadic covariate saved in `klas12-b-primary.dat`. This file contains variables indicating if two students went to the same primary school. Save this data as a matrix and call it `primary.d`
    - Same primary school: 0 = no, 1 = yes.
- Great, if you made it so far and have some time left try to plot the networks represented by the matrices `net1, net2, net3, net4`.
    - Use `gplot(...)` to plot a network in matrix format

Exercise 2

- Before you can run a RSiena analyses, you need to create a sienaDependent object where you declare the sequence of networks (`net1… net4`) as dependent variable. Call this new object `friendship`
    - Check the code we used in the lab session for the syntax you need to use.
    - If you do not know the dimensions of the networks you can find that out with e.g. `dim(net1)`
- We want to use `sex.d` as independent variable. Create a constant covariate (`coCovar`) object for `sex.d` and call it `sex`
    - Look-up the code we used in the lab session to do similar things.
- We also want to use `primary.d` as independent variable. Remember, `primary.d` is a matrix (you should have transformed it in to this format). Create a constant dyadic covariate (`coDyadCovar`) object for `primary.d` and call it `primary`
    - The syntax to do this is very similar to the one when you create a constant covariate.
- Great, now we have all dependent and independent variable objects that we need. You just need to put it all together and create a sienaData object and call it `yourData`.
    - Again, check out the code we used in the lab. You just need to add all dependent and independent variables to the arguments list.
    - The command you need to use is `sienaDataCreate`
- Finally, create a sienaEffects objet from this sienaData object and call it `yourEff`
    - The simple command is `getEffects(…)`

Exercise 3

- We are nearly there to run a model. Everything is in place.
- Now, the only thing left to do is to specify the model you want to estimate. You do this by altering the `yourEff` object you just created. The best way to include an effect is using `includeEffects(…)`
- Okay, we want to replicate model 0 in table 1 of Snijders et al. (2010)
- There are a bunch of network effects you need to include. The short names for these effects are: `recip, transTrip, transTies, cycle3, inPopSqrt, outPopSqrt, outActSqrt.`
    - 💡 You can include all of these effects in one go. See how we did this in the lab session.
- Next, include `egoX, altX,` and `sameX` effects for `sex.` Remember, `egoX` just models that individuals with a high score on X send more ties and `altX` models that indivdiuals with a high score on X receive more ties. And `sameX` models that ties between two individuals who have the same value on X are more likely to form/maintain ties.
    - 💡 We used these effects already in the lab session. Basically, you just need to include them in the args list of `inlcudeEffects(…)` and specify `interaction1 = "sex"`
- Lastly, let us include a simple dyadic covariate effect of primary. With this effect we want to model that ties might be more likely to form/maintained when two individuals already went to the same primary school together. We did not use such an effect in the lab session before. This is the relevant syntax. The general form is always the same. You just need to know the short name of an effect (here: X) and specify it further with interaction1 (and sometimes also interaction2).
  `yourEff <- includeEffects(yourEff, X, interaction1 = "primary")`

- Yeah!! And now… run the model
    - 💡 `yourAlgorithm <- sienaAlgorithmCreate(projname ='Exercise')`
    - 💡 `yourAns <- siena07(yourAlgorithm, data = yourData, effects = yourEff)`
- Look at your results `yourAns` and be excited that you just replicated Snijders et al. (2010)!