Network Models

A network represents the relationships (links) between entities (nodes). These links can be weighted (weighted network) or unweighted (binary network), directed (directed network) or undirected (undirected network). Regardless of their type, networks generate links shared by nodes, leading to data dependency when modeling the network. One proposed solution is to model network links with random intercepts and slopes. By adding such parameters to the model, we can account for the correlations between node link relationships.

Considerations

> **Caution**
>
> - The particularity here is that varying intercepts and slopes are generated for both nodal effects and dyadic effects . These varying intercepts and slopes are identical to those described in previous chapters and will therefore not be detailed further. Only the random-centered version of the varying slopes will be described here.

Example

Below is an example code snippet demonstrating a Bayesian network model with a sender-receiver effect:

Python

```
from BI import bi

# Setup device----------------------------------------------------
from BI import bi
# Setup device----------------------------------------------------
```

```
m = bi(platform='cpu')

m.data_on_model = dict(
    idx = idx,
    Any = Any-1,
    Merica = Merica-1,
    Quantum = Quantum-1,
    result_outcomes = m.net.mat_to_edgl(data['outcomes']),
    kinship = m.net.mat_to_edgl(kinship),
    focal_individual_predictors = data['individual_predictors'],
    target_individual_predictors = data['individual_predictors'],
    exposure_mat = data['exposure']
)


def model(idx, result_outcomes,
    exposure,
    kinship,
    focal_individual_predictors, target_individual_predictors,
    Any, Merica, Quantum):
      # Block ---------------------------------------
      B_any = m.net.block_model(Any,1)
      B_Merica = m.net.block_model(Merica,3)
      B_Quantum = m.net.block_model(Quantum,2)

      ## SR shape =  N individuals---------------------------------------
      sr =  m.net.sender_receiver(focal_individual_predictors,target_individual_predictors)

      # Dyadic shape = N dyads---------------------------------------
      dr = m.net.dyadic_effect(dyadic_predictors)

      m.dist.poisson(jnp.exp(B_any + B_Merica + B_Quantum + sender_receiver + dr), obs = resu



m.fit(model)
summary = m.summary()
summary.loc[['focal_effects[0]', 'target_effects[0]', 'dyad_effects[0]']]

m.fit(model2)
summary = m.summary()
summary
```

R

```r
library(BI)
# Setup platform-------------------------------------------------
m=importbi(platform='cpu')

# Import data ---------------------------------------------------

load(paste(system.file(package = "BI"),'/data/STRAND sim sr only.Rdata', sep = ''))

ids = 0:(model_dat$N_id-1)
idx = m$net$vec_node_to_edgle(jnp$stack(jnp$array(list(ids, ids)), axis = -as.integer(1)))

keys <- c("idx",
          'idxShape',
          "result_outcomes",
          'focal_individual_predictors',
          'target_individual_predictors')

values <- list(
  idx,
  idx$shape[[1]],
  m$net$mat_to_edgl(model_dat$outcomes[,,1]),
  jnp$array(model_dat$individual_predictors)$reshape(as.integer(1),as.integer(50)),
  jnp$array(model_dat$individual_predictors)$reshape(as.integer(1),as.integer(50))
)
data = py_dict(keys, values, convert = TRUE)
m$data_on_model=data

# Define model --------------------------------------------------
model <- function(idx, idxShape, result_outcomes,focal_individual_predictors, target_individu
  N_id = 50
  x=0.1/jnp$sqrt(N_id)
  tmp=jnp$log(x / (1 - x))

  ## Block ---------------------------------------
  B = bi.dist.normal(tmp, 2.5, shape=c(1), name = 'block')

  #SR ---------------------------------------
  sr =  m$net$sender_receiver(focal_individual_predictors,target_individual_predictors)

  ### Dyadic---------------------------------------
```

```
  #dr, dr_raw, dr_sigma, dr_L = m.net.dyadic_random_effects(idx.shape[0], cholesky_density =
  dr = m$net$dyadic_effect(shape = c(idxShape))

  ## SR ---------------------------------------
  m$poisson(jnp$exp(B + sr + dr), obs=result_outcomes)
}

# Run MCMC -----------------------------------------------
m$run(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
summary =m$summary()

summary[rownames(summary) %in% c('focal_effects[0]', 'target_effects[0]', 'block[0]'),]
```

Mathematical Details

Main Formula

The simple model that can be built to model link weights between nodes i and j can be defined using a Poisson distribution:

$$G_{ij} \sim Poisson(Y_{ij})$$

$$log(Y_{ij}) = \lambda_i + \pi_j + \delta_{ij}$$

where:

- Y_{ij} is the weight of the link between i and j.
- λ_i is the sender effect .
- π_j is the receiver effect .
- δ_{ij} is the dyadic effect .
-

Defining formula sub-equations and prior distributions

λ_i and π_j are varying intercepts and slopes identical to those described in previous chapters and are defined through the following equations:

$$\begin{pmatrix} \lambda_i \\ \pi_j \end{pmatrix} \sim MultivariateNormal\left(\begin{pmatrix} \sigma_\lambda \\ \sigma_\pi \end{pmatrix} \circ \left(L * \begin{pmatrix} \hat{\lambda}_i \\ \hat{\pi}_i \end{pmatrix} \right) \right)$$

$$\sigma_\lambda \sim Exponential(1)$$

$$\sigma_\pi \sim Exponential(1)$$

$$L \sim LKJ(2)$$

Similarly, for each dyad we can define varying intercepts and slopes to account for the correlation between the propensity to emit and receive links of a dyad:

$$\begin{pmatrix} \delta_{ij} \\ \delta_{ji} \end{pmatrix} \sim MultivariateNormal\left(\begin{pmatrix} \sigma_\delta \\ \sigma_\delta \end{pmatrix} \circ \left(L_\delta * \begin{pmatrix} \hat{\delta}_{ij} \\ \hat{\delta}_{ji} \end{pmatrix} \right) \right)$$

$$\sigma_\delta \sim Exponential(1)$$

$$\sigma_\delta \sim Exponential(1)$$

$$L_\delta \sim LKJ(2)$$

Note(s)

> **Note**
>
> - Note that any additional covariates can be summed with a regression coefficient to λ_i, π_j and δ_{ij}. Of course, for λ_i and π_j, as they represent nodal effects, these covariates need to be nodal characteristics (e.g., sex, age), whereas for δ_{ij}, as it represents dyadic effects, these covariates need to be dyadic characteristics (e.g., genetic distances). Considering the previous example, given a vector of nodal characteristics, *individual_predictors*, and a matrix of dyadic characteristics, *kinship*, we can incorporate these covariates into the sender-receiver and dyadic effects, respectively, as

follows:

```
def model2(idx, result_outcomes, dyad_effects, focal_individual_predictors, target_individu
    N_id = ids.shape[0]

    # Sender Receiver effect (SR), its shape is equal to N_id ----------------------------
    ## Covariates for SR
    sr_terms, focal_effects, target_effects = m.net.nodes_terms(focal_individual_predictors

    ## Varying intercept and slope for SR
    sr_rf, sr_raw, sr_sigma, sr_L = m.net.nodes_random_effects(N_id, cholesky_density = 2)

    sender_receiver = sr_terms + sr_rf

    # Dyadic effect (D), its shape is equal to n dyads -----------------------------------
    ## Covariates for D
    dr_terms, dyad_effects = m.net.dyadic_terms(dyad_effects)

    ## Varying intercept and slope for D
    rf, dr_raw, dr_sigma, dr_L = m.net.dyadic_random_effects(sender_receiver.shape[0], chol
    dr = dr_terms + rf

    lk('Y', Poisson(jnp.exp( sender_receiver + dr ), is_sparse = False), obs=result_outcome

m.data_on_model = dict(
    idx = idx,
    result_outcomes = m.net.mat_to_edgl(data['outcomes']),
    dyad_effects = m.net.prepare_dyadic_effect(kinship), # Can be a jax array of multiple c
    focal_individual_predictors = data['individual_predictors'],
    target_individual_predictors = data['individual_predictors']
)

m.fit(model2)
summary = m.summary()
summary.loc[['focal_effects[0]', 'target_effects[0]', 'dyad_effects[0]']]
```

- We can apply multiple variables as in chapter 2: Multiple Continuous Variables.
- We can apply interaction terms as in chapter 3: Interaction Between Continuous Variables.
- Network links can be modeled using Bernoulli, Binomial, Poisson, or zero-inflated Poisson distributions. So, by replacing the Poisson distribution with a binomial

distribution, we can model the existence or absence of a link — i.e., model binary networks.

- If the network is undirected, then accounting for the correlation between the propensity to emit and receive links is not necessary, and the terms λ_i, π_j, and δ_{ij} are no longer required. (Is it correct?)
- In the following chapters, we will see how to incorporate additional network effects into the model to account for network structural properties (e.g., clusters, assortativity, triadic closure, etc.).