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. This example is based on Ross, McElreath, and Redhead (2024).

Python

```
# Setup device-----------------------------------------------
from BayesForge import bf
import jax.numpy as jnp

# Setup device-----------------------------------------------
m = bf(platform='cpu')
```

```
# Simulate data ------------------------------------------------
N = 50
individual_predictor = m.dist.normal(0,1, shape = (N,1), sample = True)

kinship = m.dist.bernoulli(0.3, shape = (N,N), sample = True)
kinship = kinship.at[jnp.diag_indices(N)].set(0)

def sim_network(kinship, individual_predictor):
  # Intercept
  alpha = m.dist.normal(0,1, sample = True)

  # SR
  sr = m.net.sender_receiver(individual_predictor, individual_predictor, s_mu = 0.4, r_mu = -

  # D
  DR = m.net.dyadic_effect(kinship, d_sd=2.5, sample = True)

  return m.dist.bernoulli(logits = alpha + sr + DR, sample = True)


network = sim_network(m.net.mat_to_edgl(kinship), individual_predictor)

# Predictive model ------------------------------------------------

m.data_on_model = dict(
    network = network,
    dyadic_predictors = m.net.mat_to_edgl(kinship),
    focal_individual_predictors = individual_predictor,
    target_individual_predictors = individual_predictor
)


def model(network, dyadic_predictors, focal_individual_predictors, target_individual_predicto
    N_id = network.shape[0]

    # Block ---------------------------------------
    alpha = m.dist.normal(0,1, sample = True)

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

```
      s_mu = 0.4, r_mu = -0.4
    )

    # Dyadic shape = N dyads-------------------------------------
    dr = m.net.dyadic_effect(dyadic_predictors, d_sd=2.5) # Diadic effect intercept only

    m.dist.bernoulli(logits = alpha + sr + dr, obs=network)

m.fit(model, progress_bar=False)
```

```
bf v 0.0.48 package loaded
jax.local_device_count 32
```

```
E0526 16:16:02.028513 1060526 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
E0526 16:16:02.030119 1060526 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
```

R

```
library(BayesForge)
# Setup platform-----------------------------------------------
m=importBF(platform='cpu')

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

m$data_on_model = list()
m$data_on_model$N_id = length(ids)
m$data_on_model$network = m$net$mat_to_edgl(model_dat$outcomes[,,1])
m$data_on_model$N_dyads = m$net$mat_to_edgl(model_dat$outcomes[,,1])$shape[[1]]
m$data_on_model$focal_individual_predictors = jnp$array(model_dat$individual_predictors)
m$data_on_model$target_individual_predictors =jnp$array(model_dat$individual_predictors)

# Define model ------------------------------------------------
model <- function( N_id, N_dyads, network, focal_individual_predictors, target_individual_pre


  ## Block ---------------------------------------
  B = bf.dist.normal(0, 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(N_dyads))

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

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

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

Julia

```
# Setup device-------------------------------------------------
using BayesForge

# Setup device-------------------------------------------------
m = importBF(platform="cpu")

# Simulate data ------------------------------------------------
N = 50
individual_predictor = m.dist.normal(0,1, shape = (N,1), sample = true)

kinship = m.dist.bernoulli(0.3, shape = (N,N), sample = true)
kinship = kinship.at[jnp.diag_indices(N)].set(0)

function sim_network(kinship, individual_predictor)
  # Intercept
  alpha = m.dist.normal(0,1, sample = true)

  # SR
  sr = m.net.sender_receiver(individual_predictor, individual_predictor, s_mu = 0.4, r_mu =

  # D
  DR = m.net.dyadic_effect(kinship, d_sd=2.5, sample = true)
```

```
  return m.dist.bernoulli(logits = alpha + sr + DR, sample = true)

end

network = sim_network(m.net.mat_to_edgl(kinship), individual_predictor)


# Predictive model ------------------------------------------------

m.data_on_model = pydict(
    network = network,
    dyadic_predictors = m.net.mat_to_edgl(kinship),
    focal_individual_predictors = individual_predictor,
    target_individual_predictors = individual_predictor
)


@BF function model(network, dyadic_predictors, focal_individual_predictors, target_individual
    N_id = network.shape[0]

    # Block ---------------------------------------
    alpha = m.dist.normal(0,1, name = "alpha")

    ## SR shape =  N individuals---------------------------------------
    sr =  m.net.sender_receiver(
      focal_individual_predictors,
      target_individual_predictors,
      s_mu = 0.4, r_mu = -0.4
    )

    # Dyadic shape = N dyads---------------------------------------
    dr = m.net.dyadic_effect(dyadic_predictors, d_sd=2.5) # Diadic effect intercept only

    m.dist.bernoulli(logits = alpha + sr + dr, obs=network)
end

m.fit(model, num_samples = 500, num_warmup = 500, num_chains = 1, thinning = 1)
```

> **Caution**
>
> Event if you don't have dyadic effect, or block model effect, they need to be define to create intercepts (means) for those effects

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:

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

$$log(Y_{ij}) = \alpha + \lambda_i + \pi_j + \delta_{ij} + \beta_1 X_i + \beta_2 X_j + \beta_3 Q_{ij}$$

where:

- Y_{ij} is the weight of the link between i and j.
- λ_i is the sender random effect .
- π_j is the receiver random effect .
- δ_{ij} is the dyadic random effect .
- β_1 is the effect of an individuals i level feature on the emission of a link (i.e., out-strength).
- β_2 is the effect of an individuals j level feature on the receiving a link (i.e., in-strength).
- β_3 is the effect of an dyadic characteristic between i and j on the likelihood of a tie.

Defining formula sub-equations and prior distributions

The sender and receiver random effects are similar to those described in chapter 15: Varying intercepts, but they are defined here using a joint prior so as to estimate the correlation within individuals to emit and receive a link:

$$\begin{pmatrix} \lambda_i \\ \pi_i \end{pmatrix} = \begin{pmatrix} \sigma_\lambda \\ \sigma_\pi \end{pmatrix} \circ \left(L \begin{pmatrix} \hat{\lambda}_i \\ \hat{\pi}_i \end{pmatrix} \right)$$

$$\sigma_\lambda \sim \text{Exponential}(1)$$

$$\sigma_\pi \sim \text{Exponential}(1)$$

$$L \sim \text{LKJ}(2)$$

$$\hat{\lambda}_i \sim \text{Normal}(0, 1)$$

$$\hat{\pi}_i \sim \text{Normal}(0, 1)$$

Similarly, for each dyad we can define a joint prior to estimate correlation between i–j links and j–i links:

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

$$\sigma_\delta \sim \text{Exponential}(1)$$

$$L_\delta \sim \text{LKJ}(2)$$

$$\hat{\delta}_{ij} \sim \text{Normal}(0, 1)$$

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.
> -
> - 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 (for proportions), Binomial (for unweighted network), Poisson or zero-inflated Poisson distributions (for count). In BF, you just need to set the correct likelihood distributions. For example, if you want to model the number of interactions between nodes, you can use the Poisson distribution. If you want to model the existence or absence of a link, you can use the Bernoulli distribution.
- 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?)
- To account for exposure on a poisson model treat exposure as a nodal characteristic with its own parameter effect (i.e., regression coefficient). Their is several function that will help you to convert vectors or matrices in edge list format to have compatible data structure for the model (see API reference for BF.net.vec_to_edgl and BF.net.mat_to_edgl). F
-
- 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.).

Reference(s)

Ross, Cody T, Richard McElreath, and Daniel Redhead. 2024. "Modelling Animal Network Data in r Using STRAND." *Journal of Animal Ecology* 93 (3): 254–66.