Stochastic Block Models

Within networks, nodes can belong to different categories, and these categories can potentially affect the propensity for node interactions. For example, nodes can have different sex categories, and the propensity to interact with nodes of the same sex can be higher than with nodes of different sexes. To model the propensity for interaction between nodes based on the categories they belong to, we can use a stochastic block model approach.

Considerations

> Caution
>
> - We consider predefined groups here, with the goal of evaluating the propensity for interaction between nodes within each group.
> - In addition to the block model(s) being tested, we need to include a block where all individuals are considered as belonging to the same group (`Any` in the example). This allows us to assess whether interaction tendencies differ between groups or if the propensity to interact is uniform across all individuals.

Example

Below is an example code snippet demonstrating a Bayesian network model using the stochastic block model approach. The data is identical to the Network model example, with the addition of covariates *Any*, *Merica*, and *Quantum*, representing the block membership of each node. This example is based on Ross, McElreath, and Redhead (2024).

Python

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

# Setup device----------------------------------------------
m = bi(platform='cpu', rand_seed = False)
# 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)

category = m.dist.categorical(jnp.array([.25,.25,.25,.25]), sample = True, shape  = (N,))
N_grp, N_by_grp = jnp.unique(category, return_counts=True)
N_grp = N_grp.shape[0]

def sim_network(kinship, individual_predictor,category):
  # Intercept
  B_intercept = m.net.block_model(jnp.full((N,),0), 1, N, sample = True)
  B_category = m.net.block_model(category, N_grp, N_by_grp, sample = True)

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

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



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


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

# 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,
    category = category
)


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

    # Block ----------------------------------------
    B_intercept = m.net.block_model(jnp.full((N_id,),0), 1, N_id, name = "B_intercept")
    B_category = m.net.block_model(category, N_grp, N_by_grp, name = "B_category")

    ## 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 = B_intercept + B_category + sr + dr, obs=network)

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

```
jax.local_device_count 16

  0%|          | 0/1000 [00:00<?, ?it/s]warmup:   0%|          | 1/1000 [00:04<1:07:10,  4.03
arviz - WARNING - Shape validation failed: input_shape: (1, 500), minimum_shape: (chains=2, d
/home/sosa/work/.venv/lib/python3.12/site-packages/arviz/stats/diagnostics.py:991: RuntimeWar

invalid value encountered in scalar divide

/home/sosa/work/.venv/lib/python3.12/site-packages/arviz/stats/diagnostics.py:991: RuntimeWar

invalid value encountered in scalar divide
```

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r
b_B_category[0, 0]	-2.51	1.99	-5.39	1.06	0.09	0.07	498.22	330.41	N
b_B_category[0, 1]	-6.86	2.39	-10.46	-3.16	0.08	0.11	861.69	514.39	N
b_B_category[0, 2]	-6.91	2.28	-10.06	-2.99	0.10	0.11	568.97	311.32	N
b_B_category[0, 3]	-6.45	2.32	-9.66	-2.64	0.08	0.11	822.69	398.72	N
b_B_category[1, 0]	-7.04	2.45	-10.76	-3.22	0.10	0.10	564.03	393.66	N
...	...	...	...	...	...	...	...	...	..
sr_rf[48, 1]	-0.04	1.23	-1.98	1.49	0.05	0.11	549.16	423.28	N
sr_rf[49, 0]	-0.13	1.07	-1.45	1.39	0.04	0.08	740.57	454.12	N
sr_rf[49, 1]	0.75	1.26	-0.84	2.69	0.06	0.08	377.84	371.34	N
sr_sigma[0]	0.91	0.69	0.01	1.86	0.03	0.02	321.37	310.17	N
sr_sigma[1]	0.99	0.77	0.01	2.04	0.04	0.03	232.24	138.28	N

R

```
![](travaux-routiers.png){fig-align="center"}
```

Julia

```
# Setup device-----------------------------------------------
using BayesianInference

# Setup device-----------------------------------------------
m = importBI(platform="cpu", rand_seed = false)

# 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)

category = m.dist.categorical(jnp.array([.25,.25,.25,.25]), sample = true, shape  = (N,))
N_grp, N_by_grp = jnp.unique(category, return_counts=true)
N_grp = N_grp.shape[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,
    category = category
)


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

    # Block ---------------------------------------
    B_intercept = m.net.block_model(jnp.full((N_id,),0), 1, N_id, name = "B_intercept")
    B_category = m.net.block_model(category, N_grp, N_by_grp, name = "B_category")

    ## 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 = B_intercept + B_category + sr + dr, obs=network)
end

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

Mathematical Details

Main Formula

The model's block structure can be represented by the following formula. Note that the sender-receiver and dyadic effects are not represented here, as they are already accounted for in the Network model chapter:

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

$$\log(Y_{ij}) = B_{k(i),k(j)}$$

where:

- B is a matrix of intercept parameters unique to the interaction of categories. For example, if there are three groups, then B will be a 3x3 matrix where each element give the rate an individual in group k interacting with an individual in group l.
- We use the function k, to return the group identity (i.e., the block) of individual i.

Defining formula sub-equations and prior distributions

To account for all link rates between categories, we can define a square matrix B as follows: the off-diagonal elements represent the link rates between categories i and j, while the diagonal elements represent the link rates within category i.

$$B_{i,j} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,j} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i,1} & a_{i,2} & \cdots & a_{i,j} \end{bmatrix}$$

As we consider the link probability within categories to be higher than the link probabilities between categories, we define different priors for the diagonal and the off-diagonal. Priors

should also depend on sample size, N, so that the resultant network density approximates empirical networks. Basic priors could be:

$$a_{k \to k} \sim \text{Normal}\left(\text{Logit}\left(\frac{0.1}{\sqrt{N_k}}\right), 1.5\right)$$

$$a_{k \to \tilde{k}} \sim \text{Normal}\left(\text{Logit}\left(\frac{0.01}{0.5\sqrt{N_k} + 0.5\sqrt{N_{\tilde{k}}}}\right), 1.5\right)$$

where:

- $k \to k$ indicates a diagonal element.
- $k \to \tilde{k}$ indicates an off-diagonal element.

Note(s)

> **Note**
>
> - By defining this block model within our network model, we are estimating assortativity and disassortativity for categorical variables.
> - Similarly, for continuous variables, we can generate a block model that includes all continuous variables.

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.