BI documentation

Sebastian Sosa

2024-09-09

Table of contents

1

2 Introduction

2.1 1.1 Model set-up

We define a likelihood (e.g., a mathematical formula that specifies the plausibility of the data). The likelihood has parameters (e.g., adjustable inputs) for which we define priors (e.g., initial plausibility assignment for each possible value of the parameter). Considering a linear regression with an intercept (e.g., value when x is at zero, or at the mean if the data is centered), a slope (e.g., change value when x is incremented by one unit), and assuming the data is centered (as we will always concider in the next chapters):

* Toolpit available for each lines of equation

$$y \sim Normal(,)$$

$$\sim +x$$

$$\sim Normal(0,1)$$

$$\sim Normal(0,1)$$

$$\sim Uniform(0,1)$$

2.2 1.2 Model fitting

By using probability distributions for parameters, we can better tune the model by describing parameters with '*subequations*' and accounting for *correlated varying effects*, *Gaussian processes*, *measurement error*, and *missing data*.

In addition, we can use *Bayesian updating* using the *Bayesian theorem* to 'reshape' the prior distributions by considering every possible combination of values for µ and and scoring each combination by its relative plausibility in light of the data. These relative plausibilities are the posterior probabilities of each combination of values µ and : the *posterior distributions*. Various techniques can be used to approximate the mathematics that follows from the definition of Bayes' theorem: grid approximation, quadratic approximation, and Markov chain Monte Carlo ($MCMC$).

$$\frac{likelihood * Priors}{averagelikelihood}$$

2.3 1.3 Model 'diagnostic'

The posterior distribution can be described using percentile intervals (PI), the highest posterior density interval ($HPDI$), and point estimates. We can also sample the posterior distribution and generate *dummy data*, which can help check the model through *observations and p uncertainty propagation on the samples*. In some aspects, it is the opposite of a null model as it represents an expected model.

2.4 1.4 Link functions

We will see different families of regreessions that have different distribtions. For the moment we just need to know that those different distribtions required _link function (for each specific family we will discuss the corresponding link function):

Figure 2.1: img

2.5 Vocabulary

This method evaluate if variable we want to predict -the dependent variable (Y)- and the variable(s) that may affect(s)-independent variables (Xs)- this dependent variable is

2.6 Concideration s

When implementing Bayesian linear regression with TensorFlow Probability, it's important to consider the following: - Specifying appropriate prior distributions for the model parameters. - Choosing an appropriate likelihood function that captures the relationship between the inputs and outputs. - Selecting an inference method to approximate the posterior distribution over parameters, such as Markov chain Monte Carlo (MCMC) or variational inference.

3 Linear Regression for continuous vairable

3.1 General Principles

To study relationships between two continuous variables (e.g. heigth and weigth), we can use : *Linear regression approach*. Basically, we draw a line that cross the points clouds of the two tested variables. For this we need to have: 1) an intercept α which inform us about the starting point of the line, 2) a coefficient β which in inform us about the slope of the line and 3) a error term σ which inform us about spread of points between the line. We can interpret the intercept α as the mean for of Y for the smaller value of X, the coefficient β as how much Y increase for each increment of X, and σ as the error arround the prediction. So the coefficient β give the strength of the relationship between X and Y and σ the amount of error in the model.

Conciderations

- Bayesian linear regression considers uncertainty in the model parameters and provides a full posterior distribution over them. We thus need to decalre prior distribution for α , β and σ^2 .
- Ussually, we use *Normal* distribution for α , β and an *exponential* or *Uniform* distributiuon for σ.
- As we concider that data is standardized (see introduction) we use a distribution of mean 0 and of standard deviation of 1.
- σ is assumed to be normally distributed and is squared to force positive error and account for values bellow and above the line.
- Gaussian regression deals directly with continuous outcomes, estimating a linear relationship between predictors and the outcome variable without needing a link function. This simplifies interpretation, as coefficients represent direct changes in the outcome variable.

3.2 Example

Below is an example code snippet demonstrating Bayesian linear regression using Bayesian Inference (BI) package:

```
from .bi.main import*
# Setup device-----------------------------------------------
m = bi(platform='cpu')

# Import data ------------------------------------------------
m.data('../data/Howell1.csv', sep=';')
m.df = m.df[m.df.age > 18]
m.scale(['weight'])
m.data_to_model(['weight', 'height'])

# Define model -----------------------------------------------
def model(height, weight):
    alpha = dist.normal( 178, 20, name = 'alpha')
    beta = dist.normal(  0, 1, name = 'beta')
    sigma = dist.uniform( 0, 50, name = 'sigma')
    lk("Y", Normal(alpha + beta * weight , sigma), obs = height)

# Run mcmc ---------------------------------------------------
m.run(model)

# Summary ----------------------------------------------------
m.sampler.print_summary(0.89)
```

3.3 Mathematical Details

3.3.1 *Formula*

The following equation allow us to draw a line and is ths one that is most used in statistic clases:

$$Y = \alpha + \beta X + \sigma$$

Where: - Y is the target variable. - X is the input variable. - β is the regression coefficient. - α is the intercept term. - σ is the error term.

3.3.2 *Bayesian model*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$p(Y|\alpha, \beta, X) \sim Normal(\alpha + \beta X, \sigma)$$

$$p(\alpha) \sim Normal(0, 1)$$

$$p(\beta) \sim Normal(0, 1)$$

$$p(\sigma) \sim Uniform(0, 50)$$

Where: - $p(Y|X, \alpha, \beta)$ is the likelihood function (equivalent of the line equation). - $p(\beta)$ and $p(\alpha)$ are the prior distributions for the regression coefficients and intercept, respectivelly. - $p(\sigma)$, controll the variance of the likelihood.

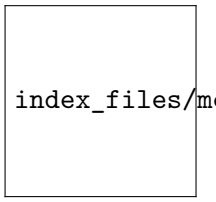
3.4 Reference(s)

McElreath (2018)

4 Multiple continuous variables model

4.1 General Principles

To study relationships between multiple continuous variables (e.g., effect of height and age on weight), we can use a Multiple Regression approach. Essentially, we extend the Linear Regression for continuous variable by adding a regression coefficient β for each continuous variables.



Considerations

- We have the same considerations as for Regression for continuous variable.
- We need a regression coefficient β for each independent variables .
- Model interpretation of the regression coefficients β is considered for a fixed value of the other dependent variables' regression coefficients —i.e., for a given age, a variation of 1 unit in height reflects the value of the regression coefficient β for height.

4.2 Example

Below is an example code snippet demonstrating Bayesian multiple regression using Bayesian Inference (BI) package:

```
from main import*

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

```
# Import data ------------------------------------------------
m.data('../data/Howell1.csv', sep=';')
m.df = m.df[m.df.age > 18]
m.scale(['weight', 'age'])
m.data_to_model(['weight', 'height', 'age'])

# Define model -----------------------------------------------
def model(height, weight, age):
    alpha = bi.dist.normal(0, 0.5, name = 'alpha')
    beta1 = bi.dist.normal(0, 0.5, name = 'beta1')
    beta2 = bi.dist.normal(0, 0.5, name = 'beta2')
    sigma = bi.dist.uniform( 1, name = 'sigma')

    lk("y", Normal(alpha + beta1 * weight + beta2 * age, sigma), obs=height)

# Run mcmc ---------------------------------------------------
m.run(model)

# Summary ----------------------------------------------------
m.sampler.print_summary(0.89)
```

4.3 Mathematical Details

4.3.1 *Formula*

We model the relationship between the independent variables $(X_1, X_2, ..., X_n)$ and the dependent variable Y using the following equation:

$$= \alpha + \beta_{11} + \beta_{22} + ... + \beta_{n+\sigma}$$

Where:

- Y is the dependent variable.
- α is the intercept term.
- X_1, X_2, ..., X_n are the independent variables.
- β_1, β_2, ..., β_n are the regression coefficients.
- $sigma$ is the error term.

4.3.2 *Bayesian model*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$(\alpha, \beta, X) \sim Normal(\alpha + \sum_{k}^{n} \beta_k X, \text{š})$$

$$p(\alpha) \sim Normal(0, 1)$$

$$p(\beta_i) \sim Normal(0, 1)$$

$$p() \sim Uniform(0, 50)$$

Where:

- $p(Y|, \alpha, \beta)$ is the likelihood function.
- $p(\alpha)$ is prior distributions for the intercept
- $p(\beta_k)$ are the prior distributions for the regression coefficients k distinct regression coefficients.
- $p(\sigma)$ is the prior distribution for the standard deviation, ensuring - it is positive.

4.4 Reference(s)

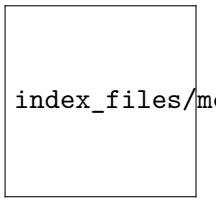
McElreath (2018)

5 Interaction Term between Two Continuous Variables

5.1 General Principles

To study relationships between two independent continuous variables and their interaction effect on a dependent variable (e.g., temperature and humidity affecting energy consumption), we can use Regression Analysis with Interaction Terms. In this approach, we extend the simple linear regression model to include an interaction term (a multiplication) between the two continuous variables.

Parallel lines indicate that there is no interaction effect while different slopes suggest that one might be present. Below is the plot for Food x Condiment. The crossed lines on the graph suggest that there is an interaction effect, which the significant p-value for the Food*Condiment term confirms. The graph shows that enjoyment levels are higher for chocolate sauce when the food is ice cream. Conversely, satisfaction levels are higher for mustard when the food is a hot dog. If you put mustard on ice cream or chocolate sauce on hot dogs, you won't be happy!



Considerations

- We have the same considerations as for Regression for continuous variable.
- Model relationship between Y and R to vary as a function of A. you explicitly model the hypothesis that the slope between Y and R depends—is conditional—upon A.
- For continuous interactions, the intercept becomes the grand mean of the outcome variable. This ease of interpretation alone is a good reason to center predictor variables.
- Estimate interpretation is more difficult as estimate of non-interaction terms become expected change in Y when R increases by one unit and A is at its average value

and estimate of interaction terms are expected change in the influence of A on Y when increasing R by one unit and expected change in the influence of R on Y when increasing A by one unit.

- Triptych plots are very handy for understanding the impact of interactions.

5.2 Example

Below is an example code snippet demonstrating Bayesian regression with an interaction term between two continuous variables with the Bayesian Inference (BI) package:

```
from main import*

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

# Import data -------------------------------------------------
m.data('../data/tulips.csv', sep=';')
m.scale(['blooms', 'water', 'shade'])
m.data_to_model(['blooms', 'water', 'shade'])

# Define model ------------------------------------------------
def model(blooms,shade, water):
    alpha = dist.normal(0.5, 0.25, name = 'alpha')
    sigma = dist.exponential(1, name = 'sigma')
    beta1 = dist.normal(0, 0.25, name = 'beta1')
    beta2 = dist.normal(0, 0.25, name = 'beta2')
    beta_interaction_ = dist.normal(0, 0.25, name = 'beta_interaction_')
    lk("y", Normal(a + beta1*water + beta2*shade + beta_interaction_*water*shade, sigma), obs

# Run mcmc ----------------------------------------------------
m.run(model)

# Summary -----------------------------------------------------
m.sampler.print_summary(0.89)
```

5.3 Mathematical Details

5.4 *Formula*

We model the relationship between the input features (X1 and X2) and the target variable (Y) using the following equation:

$$= \alpha + \beta_{11} + \beta_{22} + beta_{interaction12} +$$

σ

Where:

- Y is the target variable.
- α is the intercept term.
- X_1 and X_2 are the two independent continuous variables.
- β_1 and β_2 are the regression coefficients for X_1 and X_2, respectively.
- $\beta_{interaction}$ is the regression coefficient for the interaction term (X_1X_2).
- σ is the error term assumed to be normally distributed.

In this context, the interaction term $X_1 * X_2$ captures the joint effect of X_1 and X_2 on the target variable Y.

5.4.1 *Bayesian model*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$p(Y X_1, X_2, \beta_1, \beta_2, \beta_{interaction}) \sim Normal(\alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_{interaction} X_1 X_2,$$

$\sigma)$

$$(\alpha) \sim Normal(0, 1)$$

$$(\beta_1) \sim Normal(0, 1)$$

$$(\beta_2) \sim Normal(0, 1)$$

$$(\beta_{interaction}) \sim Normal(0, 1)$$

$$() \sim Exponential(1)$$

Where:

- $p(Y|X_1, X_2, \beta_1, \beta_2, \beta_{interaction})$ is the likelihood function.
- $p(\alpha)$ is the prior distribution for the intercept
- $p(\beta_1)$, $p(\beta_2)$ and $\beta_{interaction}$ are the prior distributions for the regression coefficients.
- $p(\sigma)$ is the prior distribution for the standard deviation, ensuring it is positive.

5.5 Reference(s)

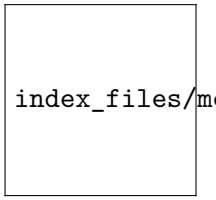
McElreath (2018)

6 Regression for Categorical Variables

6.1 General Principles

To study the relationship between a categorical independent variable and a continuous dependent variable, we use *Categorical model* wich apply *stratification*.

Stratification concist in modeling how the different categories of the independent variable affect the target continuous variable, by performing a regression for each categories and asing a regression coefficient for each categories. To realize the *stratification*, categorical variables are often encoded using one-hot encoding or converting categories to indeces.



Considerations

- We have the same considerations as for Regression for continuous variable.
- As we generate regression coefficients for each (k) category in the code, we need to specify a prior with a shape equal to the number of categories (see comments in the code).
- To compare differences between categories, we need to compute the distribution of the differences between categories, known as the contrast distribution. **Never compare the confidence intervals or p-values directly.**

6.2 Example

Below is an example code snippet demonstrating Bayesian regression with an independent categorical variable:

```
from main import*

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

# Import data -----------------------------------------------
m.data('../data/milk.csv', sep=';')
m.index(["clade"])
m.scale(['kcal_per_g'])
m.data_to_model(['kcal_per_g', "index_clade"])

# Define model ----------------------------------------------
def model(kcal_per_g, index_clade):
    beta = bi.dist.normal(0, 0.5, shape = (4,), name = 'beta') # we specify a vector of leng
    sigma = bi.dist.exponential( 1,  name = 'sigma')
    lk("y", Normal(beta[index_clade], s), obs= kcal_per_g)


# Run mcmc --------------------------------------------------
m.run(model)

# Summary ---------------------------------------------------
m.sampler.print_summary(0.89)
```

6.3 Mathematical Details

6.3.1 *Formula*

We model the relationship between the categorical input feature (X) and the target variable (Y) using the following equation:

$$= \alpha + \beta_k X_i + \sigma$$

Where:

- Y is the target variable.
- X is the encoded categorical input variable .
- α is the intercept term.
- β_k are the regression coefficient for each k categories.
- X is the independet varible

- σ is the error term .

We can interpret β_i as the effect of each category on Y relative to the baseline (usually one of the categories or the intercept).

6.3.2 *Bayesian model*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$p(Y|\alpha, \beta, X) \sim Normal(\alpha + \beta_k X, \sigma)$$

$$(\alpha) \sim Normal(0, 1)$$
$$(\beta_k) \sim Normal(0, 1)$$
$$() \sim Exponential(1)$$

Where:

- $p(_i\alpha_{,i}\,, \beta_i)$ is the likelihood function.
- $p(\alpha)$ is prior distributions for the intercept
- $p(\beta_k)$ are k prior distributions for k regression coefficients.
- $p(\sigma)$ is the prior distribution for the standard deviation, ensuring it is positive.

> **Notes**
>
> - We can apply multiple variables similarly as chapter 2: Multiple continuous Variables.
> - We can apply interaction terms similarly as chapter 3: Interaction between continuous variables.

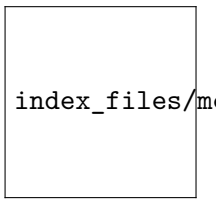
6.4 Reference(s)

McElreath (2018)

7 Binomial model

7.1 General Principles

To model the relationship between a binary outcome -e.g. such as success/failure, yes/no, or 1/0.- variable and one or more independent variables, we can use Binomial model.



Considerations

- We have the same considerations as for Regression for continuous variable.
- We have the firs link function *logit*. The *logit* link function in Bayesian binomial model converts the linear combination of predictor variables into probabilities, making it suitable for modeling binary outcomes. It helps estimate the relationship between predictors and the probability of success, ensuring results fall within the bounds of the binomial distribution.

7.2 Example

Below is an example code snippet demonstrating Bayesian binomial regression

```
from main import*

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

# Import data ------------------------------------------------
m.data('../data/chimpanzees.csv', sep=';')
m.data_to_model(['pulled_left'])
```

```
# Define model ------------------------------------------------
def model(pulled_left):
    alpha = dist.normal( 0, 10)
    lk("y", Binomial(logits= alpha[actor] + beta1[side] + beta2[cond]), obs=pulled_left)

# Run mcmc ----------------------------------------------------
m.run(model, init_strategy = numpyro.infer.initialization.init_to_mean())

# Summary -----------------------------------------------------
m.sampler.print_summary(0.89)
```

7.3 Mathematical Details

7.3.1 *Formula*

We model the relationship between the independent variable (X) and the binary outcome variable (Y) using the following equation:

$$logit(Y) = \alpha + \beta$$

Where:

- Y is the probability of success (or the probability of the binary outcome being 1).
- X, is an independent variables.
- β is the regression coefficients.
- α is the intercept term.
- σ is the error term.
- $logit(Y)$ is the log-odds of success, calculated as the log of the odds ratio of success. Through this link function, the relationship between the independent variables and the log-odds of success is modeled linearly, allowing us to interpret the effect of each independent variables on the log-odds of success.

7.3.2 *Bayesian model (WIP)*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$Y \sim Binomial(= 1,)$$

$$logit(p) \sim \alpha + \beta X$$

$$(\alpha) \sim Normal(0, 1)$$
$$(\beta) \sim Normal(0, 1)$$

Where:

- $p(Y|\alpha, \beta,)$ is the likelihood function.
- $p(\beta)$ and $p(\alpha)$ are the prior distributions for the regression coefficients and intercept, respectivelly.
- $n = 1$ represents the number of trials in the binomial distribution (binary outcome).
- $logit(\alpha + \beta)$ is the *logit* link function that is equal to sigmoid function applied to the linear combination of predictors, mapping the log-odds to probabilities.

7.4 Notes

- We can apply multiple variables similarly as chapter 2.
- We can apply interaction terms similarly as chapter 3.
- We can apply caterogical variables similarly as chapter 4.
- Below is an example code snippet demonstrating Bayesian binomial model for multiple caterogical variables :

```
from main import*

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

# Import data --------------------------------------------------
m.data('../data/chimpanzees.csv', sep=';')
m.df["side"] = m.df.prosoc_left  # right 0, left 1
m.df["cond"] = m.df.condition  # no partner 0, partner 1
m.data_to_model(['pulled_left', "actor", "side", "cond"])

# Define model -------------------------------------------------
def model(pulled_left):
    alpha = bi.dist.normal(0, 10, shape = (7,), "alpha") # generating k intercept  #(one for
    beta1 = bi.dist.normal(0, 10, shape = (2,), "beta") # generating k regression coefficient
    beta2 = bi.dist.normal(0, 10, shape = (2,), "alpha") # generating k regression coefficie
    lk("y", Binomial(logits= alpha[actor] + beta1[side] + beta2[cond]), obs=pulled_left)

# Run mcmc -----------------------------------------------
```

```
m.run(model, init_strategy = numpyro.infer.initialization.init_to_mean())

# Summary ---------------------------------------------
m.sampler.print_summary(0.89)
```

7.5 Reference(s)

McElreath (2018)

8 Beta-Binomial model

8.1 General Principles

To model the relationship between a binary outcome variable representing success counts and one or more indepedent variables with overdispersion , we can use Beta-Binomial model.

We model the relationship between the predictor variables (X1, X2, . . . , Xn) and the probability of success (p) using the following equation:

Considerations

- We have the same considerations as for Binomial regression.
- A beta-binomial model assumes that each binomial count observation has its own probability of a success. The model estimates the distribution of probabilities of success across cases, instead of a single probability of success.
- A beta distribution has two parameters, an average probability p and a shape parameter .

8.2 Example

Below is an example code snippet demonstrating Bayesian Beta-Binomial regression:

```
from .bi.main import*#
# Setup device------------------------------------------------
m = bi()

# Import data ------------------------------------------------
m.data('../data/UCBadmit.csv', sep=';')
m.df["gid"] = (m.df["applicant.gender"] != "male").astype(int)
gid = jnp.array(m.df["gid"].astype('int32').values)
applications = jnp.array(m.df["applications"].astype('float32').values)
admit = jnp.array(m.df["admit"].astype('float32').values)
```

```
m.data_on_model = dict(
    gid = gid,
    applications = applications,
    admit =  admit
)

# Define model ------------------------------------------------
def model(gid, applications, admit):
    phi = dist.exponential(1, shape=[1], name = 'phi')
    alpha = dist.normal( 0., 1.5, shape=[2], name = 'alpha')
    theta = phi + 2
    pbar = jax.nn.sigmoid(alpha[gid])
    concentration1 = pbar*theta
    concentration0 = (1 - pbar) * theta
    lk("y", BetaBinomial(total_count = applications, concentration1 = concentration1, concent

# Run mcmc ----------------------------------------------------
m.run(model)

# Summary -----------------------------------------------------
m.sampler.print_summary(0.89)
```

8.3 Mathematical Details

8.4 *Formula*

$$logit() = \alpha + \beta X$$

Where:

- p is the probability of success.
- α the intercept term.
- β the regression term.
- X is an independent variables.
- logit(p) is the log-odds of success.

8.4.1 *Bayesian model (WIP)*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$p(Y|n, \overline{p}, \theta) \sim BetaBinomial(n, \overline{p}, \theta)$$

$$logit(\overline{p}) \sim \alpha + \beta X$$

$$(\alpha) \sim Normal(0, 1)$$

$$p(\theta) \sim HalfCaychy(0, 1)$$

Where:

- $p(Y|n, \overline{p}, \theta)$ is the likelihood function.
- n is the total count of trials.
- $\overline{p}$ is the average probability.
- $p(\theta)$ is the shape distribution term.
- (α) is the intercept term.
- (β) is regression coefficient term.

9 Poisson model

9.1 General Principles

To model the relationship between a count outcome variable -e.g. counts of events occurring in a fixed interval of time or space- and one or more independent variables, we can use Poisson model.

This is a special shape of the binomial distribution, it is useful because it model binomial events for which the number of trials n is unknown or uncountably large.

index_files/mediabag/Comparison-of-linear.png

Considerations

- We have the same considerations as for Regression for continuous variable.
- We have the second link function. The conventional link function for a Poisson model is the *log* link (it ensures that i is always positive).
- To invert the log link function and model linearly the relationship between the predictor variables and the log of the mean rate parameter we can apply the exponential function (see comment in code)

9.2 Example

Below is an example code snippet demonstrating Bayesian Poisson model

```
from main import*

# Setup device-----------------------------------------------
m = bi()
```

```
# Import data ------------------------------------------------
m.data('../data/Kline.csv', sep=';')
m.sale(["P"])
m.df["cid"] = (m.df.contact == "high").astype(int)

m.data_to_model(['total_tools', 'P', 'cid'])


# Define model -----------------------------------------------
def model(cid, P, total_tools):
    alpha = dist.normal(3, 0.5, name='alpha')
    beta = dist.normal(0, 0.2, name='beta')
    lk("y", Poisson(jnp.exp(alpha[cid] + beta[cid]*P)), obs=total_tools) # Exponential ensure

# Run mcmc ---------------------------------------------------
m.run(model)

# Summary ----------------------------------------------------
m.sampler.print_summary(0.89)
```

9.3 Mathematical Details

9.3.1 *Formula*

We model the relationship between the predictor variable (X) and the count outcome variable (Y) using the following equation:

$$log() = \alpha + \beta X$$

Where:

- λ is the mean rate parameter of the Poisson distribution (expected count).
- X is the predictor variables.
- β is the regression coefficients.
- α is the intercept term.
- $\log(\lambda)$ is the log of the mean rate parameter, ensuring it is positive.

9.3.2 *Bayesian model (WIP)*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$p(Y\alpha, \beta, X) \sim Poisson()$$

$$log(\lambda) \sim \alpha + \beta X$$
$$p(\alpha) \sim Normal(0, 1)$$

$$p(\beta) \sim Normal(0, 1)$$

Where:

- $p(Y|alpha, \beta, X)$ is the likelihood function.
- $p(\beta)$ and $p(\alpha)$ are the prior distributions for the regression coefficients and intercept.
- λ is the mean rate parameter of the Poisson distribution, modeled as the exponential function of the linear combination of predictors.

9.4 Notes

- We can apply multiple variables similarly as chapter 2.
- We can apply interaction terms similarly as chapter 3.
- We can apply caterogical variables similarly as chapter 4.

9.5 Reference(s)

McElreath (2018)

10 Gamma-Poisson model

10.1 General Principles

To model the relationship between a count outcome variable and one or more independent variables with overdispersion , we can use *Negative Binomial model*.

> Considerations
>
> - We have the same considerations as for Poisson model.
> - Overdispersion is handled because the Negative-binomial model assumes that each Poisson count observation has its own rate. This is an additional parameter specified in the model (in the code, it is `_log_days_`).

10.2 Example

```
# Simulate data ------------------------------------------------------------
import tensorflow_probability.substrates.jax.distributions as tfd
init_key, sample_key = random.split(random.PRNGKey(int(r.randint(0, 10000000))))
init_key = jnp.array(init_key)
num_days = 30
y = tfd.Poisson(rate=1.5).sample(seed = init_key, sample_shape=(num_days,))
num_weeks = 4
y_new = tfd.Poisson(rate=0.5 * 7).sample(seed = init_key, sample_shape=(num_weeks,))
y_all = np.concatenate([y, y_new])
exposure = np.concatenate([np.repeat(1, 30), np.repeat(7, 4)])
monastery = np.concatenate([np.repeat(0, 30), np.repeat(1, 4)])
d = pd.DataFrame.from_dict(dict(y=y_all, days=exposure, monastery=monastery))
d["log_days"] = d.days.pipe(np.log)

from main import*
# Setup device-----------------------------------------------
m = bi()
```

```
# import data ------------------------------------------------
m.data_on_model = dict(
    log_days = jnp.array(d.log_days.values), # rate of each counts data
    monastery = jnp.array(d.monastery.values),
    output = jnp.array(d.y.values)
)

# Define model -----------------------------------------------
def model(log_days, monastery, output):
    a = dist.normal(0, 1, shape=[1], name = 'a')
    b = dist.normal(0, 1, shape=[1], name = 'b')
    l = log_days + a +  b * monastery
    lk("y", Poisson(rate = l), obs=output)

# Run mcmc ---------------------------------------------------
m.run(model)

# Summary ----------------------------------------------------
m.sampler.print_summary(0.89)
```

10.3 Mathematical Details

10.3.1 *Formula*

We model the relationship between the independent variable X and the count outcome variable Y using the following equation:

$$log(\lambda) = exp(rates + \alpha + \beta)$$

Where:

- λ is the mean rate parameter of the negative binomial distribution (expected count).
- X is the predictor variables.
- β is the regression coefficients.
- α is the intercept term.
- $\log(\lambda)$ is the log of the mean rate parameter, ensuring it is positive.

10.3.2 *Bayesian model*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$(\alpha, \beta, X) \sim Poisson()$$

$$log() \sim rates + \alpha + \beta X$$

$$(\alpha) \sim Normal(0, 1)$$

$$(\beta) \sim Normal(0, 1)$$

Where:

- $p(Y|\alpha, \beta, X)$ is the likelihood function.
- $p(\beta)$ and $p(\alpha)$ are the prior distributions for the regression coefficients and intercept.
- $\lambda = rates + \alpha + \beta X$ is the mean rate parameter of the Poisson distribution assuming that each Poisson count observation has its own rate.

> **i** Notes
>
> - We can apply multiple variables similarly as chapter 2.
> - We can apply interaction terms similarly as chapter 3.
> - We can apply caterogical variables similarly as chapter 4.

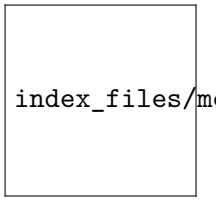
10.4 Reference(s)

McElreath (2018)

11 Multinomial model

11.1 General Principles

To model the relationship between a categorical outcome variable with more than two categories and one or more independent variables, we can use a *Multinomial* distribution.



Considerations

- We have the same considerations as for Regression for continuous variable.
- One way to interpret a multinomial is to consider that we need to build (K - 1) linear models, where (K) is the number of categories. Once we get the linear prediction for each category, we can convert these predictions to probabilities by building a simplex . To do this, we convert the regression outputs using the softmax function (see “nn.softmax” line in the code).
- The intercept captures the difference in the log-odds of the outcome categories, thus different categories need different intercepts.
- On the other hand as we assume that the effect of each predictor on the outcome is consistent across all categories, the regression coefficients are shared across categories.
- The relationship between the predictor variables and the log-odds of each category is modeled linearly, allowing us to interpret the effect of each predictor on the log-odds of each category.

11.2 Example

```
from main import*
# Simulated data-----------------------------------------------------------------------
# simulate career choices among 500 individuals
N = 500  # number of individuals
income = jnp.array([1, 2, 5])  # expected income of each career
score = 0.5 * income  # scores for each career, based on income

# next line converts scores to probabilities
p = jnp.array( jax.nn.softmax(score))

# now simulate choice
# outcome career holds event type values, not counts
career = bi.dist.categorical(p, shape = N, sample = True)
m.data_on_model = dict(
    income = income,
    career = career
)

# Define model ------------------------------------------------
def model(income, career):
    a = dist.normal(0, 1, shape= [2], name = 'a')
    b = dist.halfnormal(0.5, shape=[1], name = 'b')
    s_1 = a[0] + b * income[0]
    s_2 = a[1] + b * income[1]
    s_3 = a[0] + b * income[0]
    p = jax.nn.softmax(jnp.stack([s_1[0], s_2[0], s_3[0]]))
    lk("y", Categorical(probs =  p[career]), obs=career)

# Run sampler -------------------------------------------------
m.run(model)

# Summary -----------------------------------------------------
m.sampler.print_summary(0.89)
```

11.3 Mathematical Details

11.3.1 *Formula*

We model the relationship between the predictor variables (X1, X2, ..., Xn) and the categorical outcome variable (Y) using the following equation:

$$ logit(p_ik) = log(p_ik/(p_iK) = _k + _kX_i + _kk

$$

Where: - p_ik is the probability of the -th observation being in category .

- $_k$ are the regression coefficients for category .
- $_k$ is the intercept for category .
- X_i is the vector of predictor variables for the -th observation.
- The reference category is often chosen to simplify the model (Note that we didi not did it in the code).

We can express the Bayesian Multinomial model as follows:

If $KN, NN, andK$-simplex , then for yNK such that $Kk = 1yk = N$:

11.3.2 *Bayesian model*

In Bayesian multinomial modeling, the likelihood function of the data is specified using a multinomial distribution. The multinomial distribution models the counts of outcomes falling into different categories.For an outcome variable with categories, the multinomial likelihood function is:

$$Multinomial(y|) = \frac{N!}{\prod_{k=1}^{K} yk!} \prod_{k=1}^{K} \theta_k^{y_k}$$

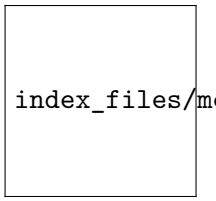
Where:

- $y = (y_1, y_2, \ldots, y_K)$ represents the counts of observations in each of the categories.
- N is the total number of observations or trials.
- $= (_1, _2, \ldots, _K)$ is a simplex of category probabilities, with representing the probability of category .
- $\frac{N!}{\prod_{k=1}^{K} yk!}$ is the multinomial coefficient accounts for the number of ways to arrange the observations into the categories. This coefficient ensures that the likelihood function properly accounts for the permutations of the counts across different categories. ## Reference(s) McElreath (2018)

12 Dirichlet model

12.1 General Principles

To model the relationship between a categorical outcome variable with more than two categories and one or more independent variables with overdispersion , we can use a *Dirichlet* distribution.



Considerations

- We have the same considerations as for Multinomial model.
- One major difference from the multinomial model is that the Dirichlet model doesn't require a simplex.

12.2 Example

```
#Simulated data ------------------------------
import seaborn as sns
import numpy as np
from jax import random
from jax.nn import softmax
import jax.numpy as jnp
import numpyro as numpyro
import numpyro.distributions as dist
from numpyro.infer import MCMC, NUTS, Predictive

################################################################################
```

```
############ SIMULATING MULTINOMIAL DATA WITH SOFTMAX LINK FUNCTION ############
def mysoftmax(x):
    exp_x = np.exp(x - np.max(x))
    return exp_x / np.sum(exp_x, axis=0)

K = 3
N = 100
N_obs = 2
sigma_random = 0.6


############################################################
#################### Fixed effect Sim ######################
#a = np.random.normal(0, 1, K)
a = np.array([3,1,1]) # Forcing a values


# Factors-------------------------
NY = 4
NV = 8

Y2 = np.full((NV, NY), np.nan)
means = np.random.normal(0, 1, NY)
offsets = np.random.normal(0, 1, NV)
for i in range(NV):
  for k in range(NY):
    Y2[i,k] = means[k] + offsets[i]


b_individual = np.random.normal(0, 1, (N, K))
mu = b_individual + a


# Declare an empty Matrix to fill with data
Y = np.empty((N * N_obs, K))

# Declare an empty vector to fill with IDs
id = []

# Loop over each individual
for i in range(N):
    # Simulate N_obs draws from the multinomial
```

```
    Y[i*N_obs:(i+1)*N_obs, :] = np.apply_along_axis(lambda x: np.random.multinomial(100, nn.s
    # Assign ID vector
    id += [i] * N_obs


N = N*N_obs
K = K
ni = N
y = jnp.array(Y, dtype=jnp.int32).reshape(N, K)
i_ID = jnp.array(id)



# Deifne model ----------------------------
from main import*
m = bi()
def model(K, ni, y, i_ID):
    a = normal('a', [K], 0,1)
    Sigma_individual = exponential('Sigma_individual', [ni], 1 )
    L_individual = lkjcholesky('L_individual', [], ni, 1) # Implies a uniform distribution ov
    z_individual = normal('z_individual', [ni,K], 0, 1)
    alpha = random_centered(Sigma_individual, L_individual, z_individual)
    lk = jnp.exp(a + alpha[i_ID])
    sample("y", DirichletMultinomial(lk, int(100)), obs=y)

m.data_on_model = dict(
    K = K,
    ni = ni,
    y = y,
    i_ID = i_ID
)

# Run sampler ------------------------------------------------
m.run(model)

# Summary ----------------------------------------------------
m.sampler.print_summary(0.89)
```

12.3 Mathematical Details

12.3.1 *Formula*

12.3.2 *Bayesian model*

We can express the Bayesian regression model accounting for prior distribution as follows:

12.4 Reference(s)

McElreath (2018)

13 Zero inflated

13.1 General Principles

Zero-Inflated Regression models are used when the outcome variable is a count variable with an excess of zero counts. These models combine a count model (e.g., Poisson or Negative Binomial) with a separate model for predicting the probability of excess zeros.

> Considerations
>
> In Bayesian Zero-Inflated regression, we consider uncertainty in the model parameters and provide a full posterior distribution over them. We need to declare prior distributions for $W\{1\pi\}, W\_\{2\pi\}, \ldots, W\_\{n\pi\}$

13.2 Example

Below is an example code snippet demonstrating Bayesian Zero-Inflated Poisson regression using TensorFlow Probability:

```
from jax.scipy.special import expit
r.seed(42)
from main import*

# Simulated data-------------------------------------------------
prob_drink = 0.2  # 20% of days
rate_work = 1     # average 1 manuscript per day

# sample one year of production
N = 365

np.random.seed(365)
drink = np.random.binomial(1, prob_drink, N)
y = (1 - drink) * np.random.poisson(rate_work, N)
```

```
# Setup device-------------------------------------------------
m = bi(platform='cpu')
m.data_on_model = dict(
    y = jnp.array(y)
)

# Define model -------------------------------------------------
def model(y):
    al = dist.normal( 1, 0.5, name = 'al')
    ap = dist.normal( -1.5, 1,name = 'ap')
    p = expit(ap)
    lambda_ = jnp.exp(al)
    lk("y", ZeroInflatedPoisson(p, lambda_), obs=y)

# Run mcmc -----------------------------------------------------
m.run(model)

# Summary ------------------------------------------------------
m.sampler.print_summary(0.89)
```

13.3 Mathematical Details

13.4 *Formula*

We model the relationship between the predictor variables X and the count outcome variable Y using two components: 1. A logistic regression model to predict the probability of an excess zero. 2. A count model (e.g., Poisson or Negative Binomial) to predict the count outcome.

The overall model can be represented as follows:

$$\begin{aligned}
&\text{logit}(\pi) = \alpha_\pi + \beta_\pi X \\
&\log(\lambda) = \alpha_\lambda + \beta_\lambda X \\
&Y \sim \begin{cases} 0 & \text{with probability } \pi \\ \text{CountModel}(\lambda) & \text{with probability } (1-\pi) \end{cases}
\end{aligned}$$

Where: - π is the probability of an excess zero. - λ is the mean rate parameter of the count model. - α_π and β_π are respectivelly, the intercept and the regression coefficient for the logistic model. - α_λ and β_λ are respectivelly, the regression coefficient for the the count model. - X is the independent variables.

13.4.1 *Bayesian model*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$(\lambda) \sim ZIPoisson(\pi, \lambda)$$

$$logit(\pi) = \alpha_\pi + \beta_\pi X$$

$$log(\lambda) = \alpha_\lambda + \beta_\lambda X$$

$$p(\alpha_\pi) \sim Normal(0, 1)$$

$$p(\beta_\pi) \sim Normal(0, 1)$$

$$p(\alpha_\lambda) \sim Normal(0, 1)$$

$$p(\beta_\lambda) \sim Normal(0, 1)$$

Where: - π is the probability of an excess zero. - λ is the mean rate parameter of the count model. - α_π and β_π are respectivelly, the intercept and the regression coefficient for the logistic model. - α_λ and β_λ are respectivelly, the regression coefficient for the the count model. - X is the independent variables.

13.5 Reference(s)

McElreath (2018)

14 Varying intercepts

14.1 General Principles

To model the relationship between predictor variables and an independent variable while allowing for different intercepts across groups or clusters, we can use a *Varying Intercepts* model. This approach is particularly useful when data is grouped (e.g., by subject, location, or time period) and we expect the baseline level of the outcome to vary across these groups.

Considerations

- We have the same considerations as for Regression for continuous variable.
- The main idea of varying intercepts is to generate an intercept for each group, allowing each group to start at different levels. Thus, the intercept $ β\$isdefinedbasedonthek* declared groups.
- Each intercept have is own *Normal distribution* -i.e. a hyper-prior -. In the code below, the *hyper-prior* is `_a_bar_`.

14.2 Example

Below is an example code snippet demonstrating Bayesian regression with varying intercepts:

```
from main import*

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

# Import data ----------------------------------------------------
m.data('../data/reedfrogs.csv', sep=';')
m.df["tank"] = np.arange(m.df.shape[0])
tank = jnp.array(m.df["tank"].astype('int32').values)
density = jnp.array(m.df["density"].astype('float32').values)
surv = jnp.array(m.df["surv"].astype('int32').values)
```

```
m.data_on_model = dict(
    tank = tank,
    surv = surv
)

# Define model ------------------------------------------------
def model(tank, surv):
    sigma = dist.exponential( 1,  name = 'sigma')
    a_bar = dist.normal( 0., 1.5, name = 'a_bar')
    alpha = dist.normal( a_bar, sigma, shape= [48], name = 'alpha')
    p = jnp.squeeze(alpha[tank])[0]
    lk("y", Binomial(total_count = density, logits = p), obs=surv)

# Run mcmc ----------------------------------------------------
m.run(model)

# Summary -----------------------------------------------------
m.sampler.print_summary(0.89)
```

14.3 Mathematical Details

14.3.1 *Formula*

We model the relationship between the independent variables X and the outcome variable Y with varying intercepts α for each group k using the following equation:

$$Y_{ik} = \alpha_k + \beta X_{ik} + \sigma$$

Where: - Y_{ik} is the outcome variable for observation i in group k. - α_k is the varying intercept for group k. - X_{ik} is the independent variables for observation i in group k. - β is the regression coefficients term. - σ is the error term, typically assumed to be normally distributed and positive.

14.3.2 *Bayesian model*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$p(Y_{ik}|\mu_{ik}, \sigma) = Normal(\mu_{ik}, \sigma)$$

$$\mu_{ik} = \alpha_j + \beta X_{ik} + \sigma$$

$$\alpha_k \sim \text{Normal}(\mu_{alpha_k}, \sigma_{alpha_k})$$

$$p(\beta) \sim \text{Normal}(0, 1)$$

$$p(\sigma) \sim \text{Exponential}(1)$$

$$p(\mu_{alpha_k}) \sim \text{Normal}(0, 1)$$

$$p(\sigma_{alpha_k}) \sim \text{Exponential}(1)$$

Where: - $p(Y_{ij}|\mu_{ij}, \sigma)$ is the likelihood function for the outcome variable. - α_k is the varying intercepts across groups. - μ_alpha_k is the overall mean intercept. - σ_alpha_k is the variance of the intercepts across groups. - $p(\beta)$ is the prior distributions for the regression coefficients. - $p(\sigma)$ is the prior distributions for the error term.

> **Notes**
>
> - We can apply multiple variables similarly as chapter 2.
> - We can apply interaction terms similarly as chapter 3.
> - We can apply caterogical variables similarly as chapter 4.
> - We can apply varying intercepts with any distribution developped in previous chapters.

14.4 Reference(s)

McElreath (2018)

15 Varying slopes

15.1 General Principles

To model the relationship between predictor variables and an independent variable while allowing for varying effects across groups or clusters, we use a *Varying slopes* model.

This approach is useful when we expect the relationship between predictors and the independent variable to differ across groups (e.g., different slopes for different subjects, locations, or time periods).This allow every unit in the data to have its own unique response to any treatment or exposure or event, while also improving estimates via pooling.

Considerations

- We have the same considerations as for 12. Varying interceps.
- The idea is pretty similar to categorical models, where a slope is specified for each category. However, here, we also estimate relationships between different groups. This leads to a different mathematical approach, as to model these relationships between groups, we model a matrix of covariance .
- The covariance matrix requiere a correlation matris distribution which is modeleld using a $LKJcorr$ distribution that hold a parameter . is ussually set to 2 to define a weakly informative prior that is skeptical of extreme correlations near 1 or 1. When we use LKJ- corr(1), the prior is flat over all valid correlation matrices. When the value is greater than 1, then extreme correlations are less likely.
- The Half-Cauchy distribution is used when modeling the covariance matrix to specify strictly positive values for the diagonal of the covariance matrix, ensuring positive variances.

15.2 Example

Below is an example code snippet demonstrating Bayesian regression with varying effects:

15.2.1 Simulated data

```
from main import*
# Setup device-----------------------------------------------
m = bi(platform='cpu')

a = 3.5  # average morning wait time
b = -1  # average difference afternoon wait time
sigma_a = 1  # std dev in intercepts
sigma_b = 0.5  # std dev in slopes
rho = -0.7  # correlation between intercepts and slopes
Mu = jnp.array([a, b])
cov_ab = sigma_a * sigma_b * rho
Sigma = jnp.array([[sigma_a**2, cov_ab], [cov_ab, sigma_b**2]])
jnp.array([1, 2, 3, 4]).reshape(2, 2).T
sigmas = jnp.array([sigma_a, sigma_b])  # standard deviations
Rho = jnp.array([[1, rho], [rho, 1]])  # correlation matrix

# now matrix multiply to get covariance matrix
Sigma = jnp.diag(sigmas) @ Rho @ jnp.diag(sigmas)

N_cafes = 20
seed = random.PRNGKey(5)  # used to replicate example
vary_effects = bi.dist.multivariatenormal(Mu, Sigma, shape=(N_cafes,), sample = True)
a_cafe = vary_effects[:, 0]
b_cafe = vary_effects[:, 1]

seed = random.PRNGKey(22)
N_visits = 10
afternoon = jnp.tile(jnp.arange(2), N_visits * N_cafes // 2)
cafe_id = jnp.repeat(jnp.arange(N_cafes), N_visits)
mu = a_cafe[cafe_id] + b_cafe[cafe_id] * afternoon
sigma = 0.5  # std dev within cafes
wait = dist.normal(mu, sigma, sample = True)
d = pd.DataFrame(dict(cafe=cafe_id, afternoon=afternoon, wait=wait))
m.data_to_model(['cafe_id', "afternoon", "wait"])
```

15.2.1.1 Model definition

```
def model(cafe, wait, N_cafes):
    a = dist.normal(5, 2, name = 'a')
    b = dist.normal(-1, 0.5, name = 'b')
    sigma_cafe = dist.exponential(1, shape=[2], name = 'sigma_cafe')
    sigma = dist.exponential( 1, name = 'sigma')
    Rho = dist.lkj(2, 2, name = 'Rho')
    cov = jnp.outer(sigma_cafe, sigma_cafe) * Rho
    a_cafe_b_cafe = dist.multivariatenormal(jnp.stack([a, b]), cov, shape = [N_cafes], name =

    a_cafe, b_cafe = a_cafe_b_cafe[:, 0], a_cafe_b_cafe[:, 1]
    mu = a_cafe[cafe] + b_cafe[cafe] * afternoon
    lk("y", Normal(mu, sigma), obs=wait)

# Run mcmc ------------------------------------------------
m.run(model)

# Summary -------------------------------------------------
m.sampler.print_summary(0.89)
```

15.3 Mathematical Details

15.3.1 *Formula*

$$\begin{pmatrix} \sigma_\alpha^2 & \sigma_\alpha\sigma_{\beta\rho} \\ \sigma_\alpha\sigma_{\beta\rho} & \sigma_\beta^2 \end{pmatrix}$$

where : - σ_α^2 is the variance of intercepts. - σ_β^2 is the covariance of intercepts & slopes. - $\sigma_\alpha\sigma_{\beta\rho}$ is the covariance between intercepts and slopes -i.e. the product of the two standard deviations-.

15.4 Mathematical Details

15.4.1 *Formula*

We model the relationship between the independent variable X and the outcome variable Y with varying intercepts (α) and varying slopes (β) for each group (k) using the following equation:

$$Y_{ik} = \alpha_k + \beta_k X_{ik} + \sigma$$

Where: - Y_{ik} is the outcome variable for observation i in group k. - X_{ik} is the independent variables for observation i in group k. - α_k is the varying intercept for group k. - β_k is the varying regression coefficients for group k. - $\$\sigma\$istheerrorterm, assumedtobestrictlypositive$.

15.4.2 *Bayesian model*

We can express the Bayesian regression model accounting for prior distribution as follows:

$$p(Y_{ik}|\mu_{ik}, \sigma) \sim \text{Normal}(\mu_{ik}, \sigma)$$
$$\mu_{ik} = \alpha_k + \beta_k X_{ik} + \sigma$$
$$\alpha_k \sim Normal(0, 1)$$
$$\beta_k \sim Normal(0, 1)$$
$$\sigma \sim Exponential(0, 1)$$

The varying intercepts slopes (α_k) and (β_k) are modeled using a *Multivariate Normal distribution*:

$$\begin{pmatrix} \alpha_k \\ \beta_k \end{pmatrix} \sim \text{MultivariateNormal}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma_\alpha^2 & \sigma_\pi \sigma_{\alpha\rho} \\ \sigma_\alpha \sigma_{\pi\rho} & \sigma_\pi \end{pmatrix}\right)$$

Where: - $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$, is the prior for average intercept. - $\begin{pmatrix} \sigma_\alpha^2 & \sigma_\pi \sigma_{\alpha\rho} \\ \sigma_\alpha \sigma_{\pi\rho} & \sigma_\pi \end{pmatrix}$ is is the covariance matrix which specifies the variance and covariance of α_k and β_k, where: - σ_α^2 The variance of α_k. - σ_π^2 The variance of β_k. - $\sigma_\pi \sigma_{\alpha\rho}$ and $\sigma_\alpha \sigma_{\pi\rho}$ The covariance between α_k and β_k

For computational reasons, it is often better to implement a centered version of the varying intercept that is equivalent to the *Multivariate Normal distribution* approach:

$$\begin{pmatrix} \alpha_k \\ \beta_k \end{pmatrix} \sim \begin{pmatrix} \sigma_\alpha \\ \sigma_\pi \end{pmatrix} \circ L * \begin{pmatrix} \widehat{\alpha}_k \\ \widehat{\pi}_k \end{pmatrix}$$

- Where:
  - $\sigma_\alpha \sim Exponential(1)$ bewing the prior standard deviation among intercepts.
  - $\sigma_\beta \sim Exponential(1)$ bewing the prior standard deviation among slopes.
  - $L \sim LKJcorr()$ bewing the prior for the correlation matrix.

The full cetered version of the model is thus :

$$p(Y_i|\mu_k, \sigma) \sim \text{Normal}(\mu_k, \sigma)$$

$$\mu_k = \alpha_k + \beta_i X_i$$

$$\begin{pmatrix} \alpha_k \\ \beta_k \end{pmatrix} \sim \begin{pmatrix} \sigma_\alpha \\ \sigma_\pi \end{pmatrix} \circ L * \begin{pmatrix} \widehat{\alpha}_k \\ \widehat{\pi}_k \end{pmatrix}$$

$$\alpha \sim Normal(0, 1)$$
$$\beta \sim Normal(0, 1)$$
$$\sigma_\alpha \sim Exponential(1)$$
$$\sigma_\pi \sim Exponential(1)$$
$$L \sim LKJcorr(2)$$

Notes

- We can apply multivariate model similarly as chapter 2. In this case, we apply the same principle, but with a covariance matrix of a dimension equal to the number of varying slopes we define. For example, if we want to generate random slopes for i actors in a model with two independent variables X_1 and X_2, we can define the formula as follows:

$$p(Y_i|\mu_i, \sigma) \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha_i + \beta_{1i} X_{1i} + \beta_{1i} X_{2i}$$

$$\begin{pmatrix} \alpha_i \\ \beta_{1i} \\ \beta_{2i} \end{pmatrix} \sim \begin{pmatrix} \sigma_\alpha \\ \sigma_\pi \\ \sigma_\gamma \end{pmatrix} \circ L \cdot \begin{pmatrix} \widehat{\alpha}_k \\ \widehat{\pi}_k \\ \widehat{\gamma}_k \end{pmatrix}$$

$$\sigma_\alpha \sim Exponential(1)$$
$$\sigma_\pi \sim Exponential(1)$$
$$\sigma_\gamma \sim Exponential(1)$$
$$L \sim LKJcorr(2)$$

- We can apply interaction terms similarly as chapter 3.
- We can apply caterogical variables similarly as chapter 4.
- We can apply varying slopes with any distribution presented in previous chapters.
- For more than two varying effects we apply the same principel but with a covariance matrix for each varying effect that are summed to gernerat the varying intercept and slope. For exmaple, if we want to generate random slopes for i actors, and k groups we can define the formula as follow:

$$p(Y_i|\mu_i, \sigma) \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha_i + \beta_i X_i$$
$$\alpha_i = \alpha + \alpha_{actor[i]} + \alpha_{group[i]}$$
$$\beta_i = \beta + \beta_{actor[i]} + \beta_{group[i]}$$

$$\alpha \sim Normal(0, 1)$$
$$\beta \sim Normal(0, 1)$$

$$\begin{pmatrix} \alpha_{\text{actor}} \\ \beta_{\text{actor}} \end{pmatrix} \sim \begin{pmatrix} \sigma_{\alpha a} \\ \sigma_{\pi a} \end{pmatrix} \circ L_a \cdot \begin{pmatrix} \widehat{\alpha}_{ka} \\ \widehat{\pi}_{ka} \end{pmatrix}$$

$$\sigma_{\alpha a} \sim Exponential(1)$$
$$\sigma_{\pi a} \sim Exponential(1)$$
$$L_a \sim LKJcorr(2)$$

$$\begin{pmatrix} \alpha_{\text{group}} \\ \beta_{\text{group}} \end{pmatrix} \sim \begin{pmatrix} \sigma_{\alpha g} \\ \sigma_{\pi g} \end{pmatrix} \circ L_g \cdot \begin{pmatrix} \widehat{\alpha}_{kg} \\ \widehat{\pi}_{kg} \end{pmatrix}$$

$$\sigma_{\alpha g} \sim Exponential(1)$$
$$\sigma_{\pi g} \sim Exponential(1)$$
$$L_g \sim LKJcorr(2)$$

- Bellow the formula and the code snipset for a Binomial multivariate model with interaction between two independent variables X_1 and X_2 and multiples varying effects for each actor and each group:

$$p(Y_i|n, p_i) \sim \text{Binomial}(n = 1, p_i)$$

$$logitp_i = \alpha_i + (\beta_{1i} + \beta_{2i}X_{2i})X_{1i}$$

$$\alpha_i = \alpha + \alpha_{actor[i]} + \alpha_{group[i]}$$

$$\beta_{1i} = \beta + \beta_{1actor[i]} + \beta_{group[i]}$$

$$\beta_{2i} = \beta + \beta_{2actor[i]} + \beta_{2group[i]}$$

$$\alpha \sim Normal(0, 1)$$

$$\beta \sim Normal(0, 1)$$

$$\begin{pmatrix} \alpha_{\text{actor}} \\ \beta_{1\,\text{actor}} \\ \beta_{2\,\text{actor}} \end{pmatrix} \sim \begin{pmatrix} \sigma_{\alpha a} \\ \sigma_{\pi a} \\ \sigma_{\gamma a} \end{pmatrix} \circ L_a \cdot \begin{pmatrix} \widehat{\alpha}_{ka} \\ \widehat{\pi}_{ka} \\ \widehat{\gamma}_{ka} \end{pmatrix}$$

$$\sigma_{\alpha a} \sim Exponential(1)$$

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

$$\sigma_{\gamma a} \sim Exponential(1)$$

$$L_a \sim LKJcorr(2)$$

$$\begin{pmatrix} \alpha_{\text{group}} \\ \beta_{1\,\text{group}} \\ \beta_{2\,\text{group}} \end{pmatrix} \sim \begin{pmatrix} \sigma_{\alpha g} \\ \sigma_{\pi g} \\ \sigma_{\gamma g} \end{pmatrix} \circ L_g \cdot \begin{pmatrix} \widehat{\alpha}_{kg} \\ \widehat{\pi}_{kg} \\ \widehat{\gamma}_{kg} \end{pmatrix}$$

$$\sigma_{\alpha g} \sim Exponential(1)$$

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

$$\sigma_{\gamma g} \sim Exponential(1)$$

$$L_g \sim LKJcorr(2)$$

```
from main import*
# Setup device-----------------------------------------------
m = bi(platform='cpu')
# Import data
m.read_csv("../data/chimpanzees.csv", sep=";")
m.df["block_id"] = m.df.block
m.df["treatment"] = 1 + m.df.prosoc_left + 2 * m.df.condition
m.data_to_model(['pulled_left', 'treatment', 'actor', 'block_id'])


def model(tid, actor, block_id, L=None, link=False):
    # fixed priors
    g = dist.normal(0, 1, name = 'g', shape = (4,))
    sigma_actor = dist.exponential(1, name = 'sigma_actor', shape = (4,))
    L_Rho_actor = dist.lkjcholesky(4, 2, name = "L_Rho_actor")
    sigma_block = dist.exponential(1, name = "sigma_block", shape = (4,))
    L_Rho_block = dist.lkjcholesky(4, 2, name = "L_Rho_block")

    # adaptive priors - non-centered
    z_actor = dist.normal(0, 1, name = "z_actor", shape = (4,7))
    z_block = dist.normal(0, 1, name = "z_block", shape = (4,3))
    alpha = deterministic(
        "alpha", ((sigma_actor[..., None] * L_Rho_actor) @ z_actor).T
    )
    beta = deterministic(
        "beta", ((sigma_block[..., None] * L_Rho_block) @ z_block).T
    )

    logit_p = g[tid] + alpha[actor, tid] + beta[block_id, tid]
    dist("L", dist.Binomial(logits=logit_p), obs=L)

    # compute ordinary correlation matrixes from Cholesky factors
    if link:
        deterministic("Rho_actor", L_Rho_actor @ L_Rho_actor.T)
        deterministic("Rho_block", L_Rho_block @ L_Rho_block.T)
        deterministic("p", expit(logit_p))

# Run mcmc ------------------------------------------------
m.run(model)

# Summary -------------------------------------------------
m.sampler.print_summary(0.89)
```

15.5 Reference(s)

McElreath (2018)

16

17

18

19 Modeling Network

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 effects. By adding such parameters to the model, we can account for the correlations between node link relationships.

Considerations

- The particularity here is that varying intercepts and slopes are generated for both nodal effects and [dyadic effects](# " link-related categorical (e.g., same group membership) or continuous (e.g., genetic distances) characteristics between nodes"). Those the varying intercepts and slopes are identical to those described in previous chapters and will therefore not be detailed further and only the random centered version of the varying slopes will be described here.

19.1 Example

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

```
# Building model and sampling it ------------------
ids = jnp.arange(0,data['N_id'][0])
idx = bi.net.vec_node_to_edgle(jnp.stack([ids, ids], axis = -1))

@jit
def logit(x):
    return jnp.log(x / (1 - x))

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

    # Sender Receiver effect (SR) its shape is equal to N_id ---------------------------------
```

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

    # Dyadic effect (D) its shape is equal to n dyads ----------------------------------------
    ## Varying intercept and slope for D
    rf, dr_raw, dr_sigma, dr_L = bi.net.dyadic_random_effects(sender_receiver.shape[0], chole
    dr =  rf

    lk('Y', Poisson(jnp.exp( sender_receiver + dr )), obs=result_outcomes)

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

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

19.2 Mathematical Details

19.2.1 *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 links between i and j.
- λ_i is the sender effect
- π_j is the receiver effect
- δ_{ij} is the dyadic effect .

19.2.2 *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 define 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 dyads we can define varying intercepts and slopes to account for correlation between the propensity to emit and receive links of a dyad :

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

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

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

$$L \sim LKJ(2)$$

i Note(s)

- 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, π_j as they represent nodal effects those covariates need to be nodal characteristics (e.g., sex, age) whereas for $delta_{ij}$ as it represents dyadic effects those covariates need to be dyadic characteristics (e.g., genetic distances). Concidering previous example given a vector of nodal characteristics *individual_predictors* and a a matrix of dyadic characteristics *kinship* we can incorporate those covariates in the sender-receiver effect and dyadic effect respectivelly as fellow:

```
# Building model and sampling it ------------------
ids = jnp.arange(0,data['N_id'][0])
idx = bi.net.vec_node_to_edgle(jnp.stack([ids, ids], axis = -1))

@jit
def logit(x):
    return jnp.log(x / (1 - x))

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 = bi.net.nodes_terms(focal_individual_predictor

    ## Varying intercept and slope for SR
    sr_rf, sr_raw, sr_sigma, sr_L = bi.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 = bi.net.dyadic_terms(dyad_effects)

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

    lk('Y', Poisson(jnp.exp( sender_receiver + dr )), obs=result_outcomes)

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

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

- We can apply multiple variables similarly as chapter 2: Multiple continuous Variables.
- We can apply interaction terms similarly as 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 link — i.e., model binary networks.
- If the network is undirected, then accounting for correlation between propensity to emit and receive links is not necessary, and the terms λ_i, π_j , and δij 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.).

20 Network with block model

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.

> **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(s) model 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.

20.1 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.

```
def model3(idx, result_outcomes, kinship, focal_individual_predictors, target_individual_pred
    N_id = ids.shape[0]

    # Block ----------------------------------------
    B_any, b_any, b_ij_any, b_ii_any = bi.net.block_model(Any,1, name_b_ij = 'b_ij_Any', name
    B_Merica, b_Merica, b_ij_Merica, b_ii_Merica = bi.net.block_model(Merica,  3, name_b_ij =
    B_Quantum, b_Quantum, b_ij_Quantum, b_ii_Quantum = bi.net.block_model(Quantum, 2, name_b_

    ## SR ----------------------------------------
    sr_terms, focal_effects, target_effects = bi.net.nodes_terms(focal_individual_predictors
    sr_rf, sr_raw, sr_sigma, sr_L = bi.net.nodes_random_effects(sr_terms.shape[0], cholesky_
```

```
    sender = sr_terms[idx[:,0],0] + sr_terms[idx[:,1],1] + sr_rf[idx[:,0],0]
    receiver =  sr_terms[idx[:,1],0] + sr_terms[idx[:,0],1] + sr_rf[idx[:,1],1]
    sender_receiver = jnp.stack([sender, receiver], axis = 1)

    # Dyadic-------------------------------------
    dr_terms, dyad_effects = bi.net.dyadic_terms(kinship[:,0], kinship[:,1])
    rf, dr_raw, dr_sigma, dr_L = bi.net.dyadic_random_effects(idx.shape[0], cholesky_density
    dr = dr_terms + rf

    lk('Y', Poisson(jnp.exp(B_any + B_Merica + B_Quantum + sender_receiver + dr )), obs=resul

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

m.run(model3)
summary = m.summary()
summary.loc[['focal_effects[0]', 'target_effects[0]', 'dyad_effects[0]']]
```

20.2 Mathematical Details

20.2.1 *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:

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

$$log(Y_{ij}) = B_{ij} + B_{ji}$$

where: - B_{ij} is the link probability between category i to j. - B_{ji} is the link probability between category j to i.

20.2.2 *Defining formula sub-equations and prior distributions*

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

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

Where: - $B[i,j]$ is the link probability between category i and j when $i \neq j$. - $B[i,j]$ is the link probability within category i when $i = j$.

As we concider link probability within categorioes to be higher to links probabilites 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:

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

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

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

> **i** Note(s)
>
> - By defining this block model within our network model, we are estimating assortativity and disassortativity (#" the tendency of nodes to connect with dissimilar nodes") for categorical variables.
> - Similarly, for continuous variables, we can generate a block model that includes all continuous variables.

21

22

23

24

25 Multilayer Networks

25.1 General Principles

A multilayer network is a broader term that refers to networks composed of multiple layers, where each layer may have its own set of nodes and edges, potentially representing different types of entities and interactions. This means the nodes can be the same or different across layers.

25.2 Considerations

- We have the same considerations as for Network model. ## Example

25.3 Mathematical Details

25.3.1 *Formula*

25.3.2 *Bayesian model (WIP)*

26 Notes

- We can apply sender-receiver model 19.1. Sender receiver network model..
- We can apply block model 19.2. Block model network.qmd.

McElreath, Richard. 2018. *Statistical Rethinking: A Bayesian Course with Examples in r and Stan*. Chapman; Hall/CRC.