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):

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 variable

3.1 General Principles

To study relationships between two continuous variables (e.g., height and weight), we can use a linear regression approach. Essentially, we draw a line that passes through the point cloud of the two variables being tested. For this, we need to have:

1) An intercept α, which represents the origin of the line—the expected value of the dependent variable (height) when the independent variable (weight) is equal to zero.
2) A coefficient β, which informs us about the slope of the line. In other words, it tells us how much Y (height) increases for each increment of the independent variable (weight).
3) A variance term σ, which informs us about the spread of points around the line, i.e., the variance around the prediction.

3.2 Considerations

Caution

- Bayesian models consider model parameter uncertainty , allowing for the quantification of confidence or uncertainty through the parameters' posterior distribution . Therefore, we need to declare prior distributions for each model parameter, in your case for: α, β, and σ^2.
- Prior distributions are built following these considerations:
  - As the data is scaled (see introduction), we can use a Normal distribution for α and β, with a mean of 0 and a standard deviation of 1.
  - Since σ is strictly positive, we can use any distribution that is positively defined, such as the Exponential or Gamma distribution.
- 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.

3.3 Example

Below is an example code snippet demonstrating Bayesian linear regression using the Bayesian Inference (BI) package. Data consist of two continuous variables (height and weight), and the goal is to estimate the effect of weight on height.

3.4 Python

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

# Import Data & Data Manipulation ------------------------------------------------
m.data('../data/Howell1.csv', sep=';') # Import
m.df = m.df[m.df.age > 18] # Manipulate
m.scale(['weight']) # Scale
m.data_to_model(['weight', 'height']) # Send to model (convert to jax array)

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

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

# Summary ------------------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions
```

3.5 R

```r
library(reticulate)
setwd(paste0(getwd(), '/bi'))

# Setup device-----------------------------------------------
bi <- import("main")
load('STRAND sim sr dyad.Rdata')
m = bi$bi(platform='cpu')

# Import Data & Data Manipulation ----------------------------------------------
m$data('../data/Howell1.csv', sep=';')  # Import
m$df = m$df[m$df$age > 18,] # Manipulate
m$scale(list('weight')) # Scale
m$data_to_model(list('weight', 'height')) # Send to model (convert to jax array)

# Define model -----------------------------------------------
model <- function(height, weight){
  # Parameters priors distributions
  s = bi$dist$uniform(0, 50, name = 's', shape = tuple(as.integer(1)))
  a = bi$dist$normal(178, 20, name = 'a', shape = tuple(as.integer(1)))
  b = bi$dist$normal(0, 1, name = 'b', shape = tuple(as.integer(1)))
  # Likelihood
  bi$lk("y", bi$Normal(a + b * weight, s), obs = height)
}

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

# Summary -----------------------------------------------
m$sampler$print_summary(0.89) # Get posterior distributions
```

3.6 Mathematical Details

3.6.1 *Frequentist formulation*

The following equation allows us to draw a line:

$$Y_i = \alpha + \beta X_i + \sigma_i$$

Where:

- Y_i is the dependent variable for observation i.
- α is the intercept term.
- β is the regression coefficient.
- X_i is the input variable for observation i.
- σ_i is a vector of error terms for observation i.

3.6.2 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express a Bayesian version of this regression model using the following model:

$$Y_i \sim Normal(\alpha + \beta X_i, \sigma)$$

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

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

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

Where:

- Y_i is dependent variable for observation i.
- α and β are the regression coefficients and intercept parameters, respectively.
- X_i is the input variable for observation i.
- σ is a prior for the vairance term standard deviation of the normal distribution that describes the variance in the relationship between the dependent variable Y and the independent variable X.

3.7 Notes

> **Note**
>
> We can observe a difference between the *Frequentist* and the *Bayesian* formulation regarding the error term. Indeed, *Frequentist* formulation, the error term σ_i represents random fluctuations around the predicted values. This assumption leads to point estimates for α and β, without accounting for uncertainty in these estimates. In contrast, the *Bayesian* formulation treats σ as a parameter with its own prior distribution. This allows us to incorporate our uncertainty about the error term into the model.

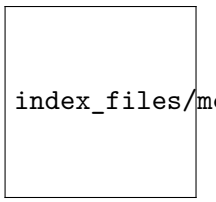
3.8 Reference(s)

McElreath (2018)

4 Multiple continuous variables model

4.1 General Principles

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


index_files/mediabag/0-dJqdzk1aMo2OQR7O.pdf

4.2 Considerations

Caution

- We have the same considerations as for the Regression for continuous variable.
- The model interpretation of the regression coefficients β_x is considered for fixed values of the other independent variable(s)' regression coefficients —i.e., for a given age, β_{weight} represents the expected change in the dependent variable variation (height) for each one-unit increase in weight, holding all other variable(s) constant (age).

4.3 Example

Below is an example code snippet demonstrating Bayesian multiple regression using the Bayesian Inference (BI) package. Data consist of three continuous variables (*height*, *weight*, *age*), and the goal is to estimate the effect of *weight* and *age* on *height*.

4.3.1 Python

```
from main import*

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

# Import Data & Data Manipulation -----------------------------------------------
m.data('../data/Howell1.csv', sep=';')  # Import
m.df = m.df[m.df.age > 18] # Manipulate
m.scale(['weight', 'age']) # Scale
m.data_to_model(['weight', 'height', 'age']) # Send to model (convert to jax array)

# Define model ------------------------------------------------
def model(height, weight, age):
    # Parameters priors distributions
    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(0,50, name = 'sigma')
    # Likelihood
    lk("y", Normal(alpha + beta1 * weight + beta2 * age, sigma), obs = height)

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

# Summary -----------------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions
```

4.3.2 R

```
library(reticulate)
# Setup device-------------------------------------------------
bi <- import("main")
m = bi$bi(platform='cpu')

# Import Data & Data Manipulation -----------------------------------------------
m$data('../data/Howell1.csv', sep=';') # Import
m$df = m$df[m$df$age > 18,] # Manipulate
m$scale(list('weight', 'age')) # Scale
m$data_to_model(list('weight', 'height', 'age')) # Send to model (convert to jax array)
```

```r
# Define model ------------------------------------------------
model <- function(height, weight, age){
  # Parameters priors distributions
  alpha = bi$dist$normal( 0, 0.5, name = 'a', shape = tuple(as.integer(1)))
  beta1 = bi$dist$normal( 0, 0.5, name = 'b1', shape = tuple(as.integer(1)))
  beta2 = bi$dist$normal(  0, 0.5, name = 'b2', shape = tuple(as.integer(1)))
  sigma = bi$dist$uniform(0, 50, name = 's', shape = tuple(as.integer(1)))
  # Likelihood
  bi$lk("y", bi$Normal(alpha + beta1 * weight + beta2 * age, sigma), obs=height)
}

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

# Summary -----------------------------------------------------
m$sampler$print_summary(0.89) # Get posterior distributions
```

4.4 Mathematical Details

4.4.1 *Frequentist formulation*

We model the relationship between the independent variables $(X_{1i}, X_{2i}, ..., X_{ni})$ and the dependent variable Y using the following equation:

$$_i = \alpha + \beta_{11i} + \beta_{22i} + ... + \beta_{nni} + \sigma_i$$

Where:

- Y_i is the dependent variable for observation i.
- α is the intercept term.
- X_{1i}, X_{2i}, ..., X_{ni} are the values of the independent variables for observation i.
- β_1, β_2, ..., β_n are the regression coefficients.
- σ_i is a vector of error terms for observation i.

4.4.2 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian model as follows:

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

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

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

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

Where:

- Y_i is dependent variable for observation i.
- α is the prior distribution for the intercept.
- β_k are the prior distributions for the regression coefficients k distinct regression coefficients.
- X_{1i}, X_{2i}, ..., X_{ni} are the values of the independent variables for observation i.
- σ is the prior distribution for the standard deviation, ensuring it is positive.

4.5 Reference(s)

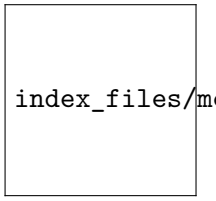
McElreath (2018)

5 Interaction terms

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 temperature x humidity. The crossed lines on the graph suggest that there is an interaction effect. The graph shows that energy consumption levels are higher for humidity when the temperature is high. Conversely, energy consumption levels are higher for temperature when humidity is low.



5.2 Considerations

Caution

- We have the same considerations as for Regression for continuous variable.
- Model the relationship between independent variable Y and dependent variable X_1 to vary as a function of dependent variable X_2. You explicitly model the hypothesis that the slope between Y and X_1 depends—is conditional—upon X_2.
- For continuous interactions with scaled data, the intercept becomes the grand mean of the outcome variable.
- The interpretation of estimates is more complex. The estimate of non-interaction

terms reflects the expected change in Y when X_1 increases by one unit, holding X_2 constant at its average value. The estimate of interaction terms represents how the effect of X_1 on Y changes depending on the value of X_2, and vice versa, showing how the relationship between the two variables influences the outcome Y.

- Triptych plots are very handy for understanding the impact of interactions, specially when more than two interactions are present.

5.3 Example

Below is an example code snippet demonstrating Bayesian regression with an interaction term between two continuous variables with the Bayesian Inference (BI) package. Data consist of three continuous variables (temperature, humidity, energy consumption), and the goal is to estimate the effect of interaction between temperature and humidity on energy consumption.

5.3.1 Python

```
from main import*

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

# Import Data & Data Manipulation ------------------------------------------------
m.data('../data/tulips.csv', sep=';') # Import
m.scale(['blooms', 'water', 'shade']) # Scale
m.data_to_model(['blooms', 'water', 'shade']) # Send to model (convert to jax array)

# Define model -------------------------------------------------
def model(blooms, shade, water):
    # Parameters priors distributions
    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_')
    # Likelihood
    lk("y", Normal(alpha + beta1 * water + beta2 * shade + beta_interaction_ * water * shade

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

```
m.run(model) # Optimize model parameters through MCMC sampling

# Summary ----------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions
```

5.3.2 R

```
library(reticulate)
bi <- import("main")

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

# Import Data & Data Manipulation ----------------------------------------------
m$data('../data/tulips.csv', sep=';') # Import
m$scale(list('blooms', 'water', 'shade')) # Scale
m$data_to_model(list('blooms', 'water', 'shade')) # Send to model (convert to jax array)

# Define model ----------------------------------------------
model <- function(blooms, water,shade){
  # Parameters priors distributions
  alpha = bi$dist$normal( 0.5, 0.25, name = 'a',shape= tuple(as.integer(1)))
  beta1 = bi$dist$normal( 0,  0.25, name = 'b1',shape= tuple(as.integer(1)))
  beta2 = bi$dist$normal(  0,  0.25, name = 'b2',shape= tuple(as.integer(1)))
  beta_interaction_ = bi$dist$normal(  0, 0.25, name = 'bint',shape= tuple(as.integer(1)))
  sigma = bi$dist$uniform(0, 50, name = 's',shape = tuple(as.integer(1)))
  # Likelihood
  bi$lk("y", bi$Normal(alpha + beta1*water + beta2*shade + beta_interaction_*water*shade, sig
}

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

# Summary ----------------------------------------------
m$sampler$print_summary(0.89) # Get posterior distributions
```

5.4 Mathematical Details

5.5 *Frequentist formulation*

We model the relationship between the input features (X_1 and X_2) and the target variable (Y) using the following equation:

$$_i = \alpha + \beta_{11i} + \beta_{22i} + \beta_{interaction1i2i} + \sigma$$

Where:

- Y_i is the dependent variable for observation i.
- α is the intercept term.
- X_{1i} and X_{2i} are the two values of the independent continuous variables for observation i.
- 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 $X_{1i} * X_{2i}$ captures the joint effect of X_{1i} and X_{2i} on the target variable Y_i.

5.5.1 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distribution as follows:

$$Y \sim Normal(\alpha + \beta_1 X_{1i} + \beta_2 X_{2i} + \beta_{interaction} X_1 1 i X_{2i}, \sigma)$$

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

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

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

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

$$\sim Exponential(1)$$

Where:

- Y_i is dependent variable for observation i.
- α is the prior distribution for the intercept.
- β_1, β_2, and $\beta_{interaction}$ are the prior distributions for the regression coefficients.
- X_{1i} and X_{2i} are the two values of the independent continuous variables for observation i.
- σ is the prior distribution for the standard deviation, ensuring it is positive.

5.6 Reference(s)

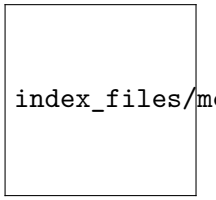
McElreath (2018)

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 a *Categorical model* which applies *stratification*.

Stratification involves modeling how the k different categories of the independent variable affect the target continuous variable by performing a regression for each k category and assigning a regression coefficient for each category. To implement stratification, categorical variables are often encoded using one-hot encoding or by converting categories to indices .



6.2 Considerations

> Caution
>
> - We have the same considerations as for Regression for continuous variable.
> - As we generate regression coefficients for each k category, we need to specify a prior with a shape equal to the number of categories k in the code (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 confidence intervals or p-values directly**.

6.3 Example

Below is an example code snippet demonstrating Bayesian regression with an independent categorical variable using the Bayesian Inference (BI) package. The data consist of one continuous dependent variable (*kcal_per_g*) representing the caloric value of milk per gram, and a categorical independent variable representing species clade membership. The goal is to estimate the differences in milk calories between clades.

6.3.1 Python

```
from main import*

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

# Import Data & Data Manipulation ------------------------------------------------
m.data('../data/milk.csv', sep=';') # Import
m.index(["clade"]) # Manipulate
m.scale(['kcal_per_g']) # Scale
m.data_to_model(['kcal_per_g', "index_clade"]) # Send to model (convert to jax array)

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


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

# Summary ------------------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions
```

6.3.2 R

```
library(reticulate)
bi <- import("main")
```

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

# Import Data & Data Manipulation ----------------------------------------------
m$data('../data/milk.csv', sep=';')
m$scale(list('kcal_per_g')) # Manipulate
m$index(list('clade')) # Scale
m$data_to_model(list('kcal_per_g', 'index_clade')) # Send to model (convert to jax array)

# Define model -----------------------------------------------
model <- function(kcal_per_g, index_clade){
  # Parameters priors distributions
  beta = bi$dist$normal( 0, 0.5, name = 'beta',shape= tuple(as.integer(1)))
  sigma = bi$dist$exponential(1, name = 's',shape = tuple(as.integer(1)))
  # Likelihood
  bi$lk("Y", bi$Normal(beta[index_clade], sigma), obs=kcal_per_g)
}

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

# Summary -----------------------------------------------
m$sampler$print_summary(0.89) # Get posterior distributions
```

6.4 Mathematical Details

6.4.1 *Frequentist formulation*

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

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

Where:

- Y_i is dependent variable for observation i.
- α is the intercept term.
- β_k are the regression coefficients for each k category.

- X_i is the encoded categorical input variable for observation i.
- 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).

6.4.2 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distribution as follows:

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

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

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

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

Where:

- Y_i is dependent variable for observation i.
- α is the prior distribution for the intercept.
- β_k are k prior distributions for k regression coefficients.
- X_i is the encoded categorical input variable for observation i.
- σ is the prior distribution for the standard deviation, ensuring it is positive.

6.5 Notes

> **Note**
>
> - We can apply multiple variables similarly as in Chapter 2: Multiple Continuous Variables.
> - We can apply interaction terms similarly as in Chapter 3: Interaction between Continuous Variables.

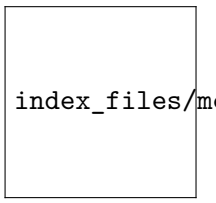
6.6 Reference(s)

McElreath (2018)

7 Binomial Model

7.1 General Principles

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



7.2 Considerations

> Caution
>
> - We have the same considerations as for Regression for continuous variable.
> - We have the first link function *logit*. The *logit* link function in the 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.

7.3 Example

Below is an example code snippet demonstrating Bayesian binomial regression using the Bayesian Inference (BI) package. The data consist of one binary dependent variable (*pulled_left*), which represents the side individuals pulled. The goal is to evaluate the probability of pulling the left side.

7.3.1 Python

```python
from main import*

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

# Import Data & Data Manipulation ------------------------------------------------
m.data('../data/chimpanzees.csv', sep=';') # Import
m.data_to_model(['pulled_left', 'actor', 'side', 'cond']) # Send to model (convert to jax ar

# Define model -------------------------------------------------
def model(pulled_left, actor, side, cond):
    # Parameters priors distributions
    alpha = dist.normal(0, 10)
    # Likelihood
    lk("y", Binomial(logits=alpha), obs=pulled_left)

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

# Summary ------------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions
```

7.3.2 R

```r
library(reticulate)
bi <- import("main")
# Setup device--------------------------------------------------
m = bi$bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
m$data('../data/chimpanzees.csv', sep=';') # Import
m$data_to_model(list('pulled_left')) # Send to model (convert to jax array)

# Define model -------------------------------------------------
model <- function(pulled_left){
   # Parameters priors distributions
  alpha = bi$dist$normal( 0, 10, name = 'alpha')
  # Likelihood
  bi$lk("Y", bi$Binomial(logits = alpha), obs=pulled_left)
```

```
}

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

# Summary ------------------------------------------------
m$sampler$print_summary(0.89) # Get posterior distributions
```

7.4 Mathematical Details

7.4.1 *Frequentist formulation*

We model the relationship between the independent variable (X_i) and the binary dependent variable (Y_i) using the following equation:

$$logit(Y_i) = \alpha + \beta X_i$$

Where:

- Y_i is the probability of success (i.e., the probability of the binary outcome being 1) for observation i.
- α is the intercept term.
- β is the regression coefficient.
- X_i is the value of the independent variable for observation i.
- $logit(Y_i)$ 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 variable on the log-odds of success for observation i.

7.4.2 *Bayesian formulation*

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

$$Y_i \sim Binomial(n = 1, p)$$

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

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

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

Where:

- Y is the probability of success (i.e., the probability of the binary outcome being 1) for observation i.
- $n = 1$ represents the number of trials in the binomial distribution (binary outcome).
- β and α are the prior distributions for the regression coefficients and intercept, respectively.
- *logit* 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 variable on the log-odds of success for observation i.

7.5 Notes

> **Note**
>
> - We can apply multiple variables similarly as in chapter 2.
> - We can apply interaction terms similarly as in chapter 3.
> - We can apply categorical variables similarly as in chapter 4.
> - Below is an example code snippet demonstrating a Bayesian binomial model for multiple categorical variables using the Bayesian Inference (BI) package. The data consist of one binary dependent variable (*pulled_left*), which represents the side individuals pulled, and three independent variables (*actor*, *side*, *cond*). The goal is to evaluate, for each individual, the probability of pulling the left side, accounting for whether the individual is left-handed or right-handed, as well as the different conditions.

```
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,), name="alpha")  # generating k intercepts (one
    beta1 = bi.dist.normal(0, 10, shape=(2,), name="beta")  # generating k regression coefi
    beta2 = bi.dist.normal(0, 10, shape=(2,), name="beta")  # generating k regression coefi
    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()) # Optimize model pa

# Summary ----------------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions

:::

## Reference(s)
@mcelreath2018statistical



`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4ifQ== -->`{=html}

```{=html}
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4iLCJib29rSXRlbVR5cGUiOiJjaGFwdGVyIiwiYm9v
```
```

8 Beta-Binomial Model

8.1 General Principles

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

8.2 Considerations

Caution

- We have the same considerations as for Binomial regression.
- A Beta-Binomial model assumes that each binomial count observation has its own probability of 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: the rates for each probabilities and a shape parameter . influence how probabilities are distributed between 0 and 1. Specifically, it consists of two parameters, γ and η , which determine the concentration of probability around 0 and 1. The values of and shape the distribution:
  - If both are greater than 1, the distribution is bell-shaped and centered around the mean .
  - If $\gamma < 1$ and $\eta < 1$, the distribution is U-shaped, indicating that outcomes are more likely to be near 0 or 1.
  - If $= \gamma$ and $\eta = 1$, the Beta distribution is uniform.

Thus, the shape parameters γ η provide flexibility in modeling various types of prior beliefs about probabilities.

8.3 Example

Below is an example code snippet demonstrating Bayesian Beta-Binomial regression using the Bayesian Inference (BI) package. The data consist of:

1) One binary dependent variable (admit), which represents candidates' admission status.

2) One independent categorical variable representing individuals' gender (gid).

3) Additionally, we have the number of applications (applications) per individual, which will be used to account for independent rates.

The goal is to evaluate whether the probability of admission is different between genders, while accounting for differences in the number of applications between genders.

8.3.1 Python

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

# Import Data & Data Manipulation ----------------------------------------------
m.data('../data/UCBadmit.csv', sep=';') # Import
# Data type (int, float is important, specially for indices)
m.df["gid"] = (m.df["applicant.gender"] != "male").astype(int) # Manipulate
gid = jnp.array(m.df["gid"].astype('int32').values) # Manipulate
applications = jnp.array(m.df["applications"].astype('float32').values) # Manipulate
admit = jnp.array(m.df["admit"].astype('float32').values) # Manipulate

# Send to model (convert to jax array)
m.data_on_model = dict(
    gid=gid,
    applications=applications,
    admit=admit
)

# Define model ------------------------------------------------
def model(gid, applications, admit):
    # Parameters priors distributions
    phi = dist.exponential(1, shape=[1], name='phi')
    alpha = dist.normal(0., 1.5, shape=[2], name='alpha')
    t = phi + 2
    pbar = jax.nn.sigmoid(alpha[gid])
    gamma = pbar * t
    eta = (1 - pbar) * t
    # Likelihood
    lk("y", BetaBinomial(total_count=applications, concentration1=gamma, concentration0=eta

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

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

8.3.2 R

```r
library(reticulate)
bi <- import("main")

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

# Import Data & Data Manipulation ---------------------------------------------
m$data('../data/UCBadmit.csv', sep=';') # Import
m$df["gid"] = as.integer(ifelse(m$df["applicant.gender"] == "male", 0, 1)) # Manipulate
m$data_to_model(list('gid', 'applications', 'admit' )) # Send to model (convert to jax arra

# Define model ----------------------------------------------
model <- function(gid, applications, admit){
  # Parameters priors distributions
  phi = bi$dist$exponential(1, name = 'phi')
  alpha = bi$dist$normal(0., 1.5, shape= tuple(as.integer(2)), name='alpha')
  t = phi + 2
  pbar = bi$jax$nn$sigmoid(alpha[gid])
  gamma = pbar * t
  eta = (1 - pbar) * t
  # Likelihood
  bi$lk("y", bi$BetaBinomial(total_count=applications, concentration1=gamma, concentration0
}

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

# Summary -------------------------------------------------
m$sampler$print_summary(0.89) # Get posterior distributions
```

8.4 Mathematical Details

8.4.1 *Frequentist formulation*

???

8.4.2 *Bayesian Model*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distribution as follows:

$$Y_i \sim BetaBinomial(n_i, \gamma_i, \eta_i)$$

$$\gamma_i = \overline{\rho}\tau$$

$$\eta_i = (1 - \overline{\rho})\tau$$

$$\overline{\rho} = logit(\alpha_i)$$

$$\tau = \phi + 2$$

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

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

Where:

- Y_i is the ount of successes for the i-th observation, which follows a beta-binomial distribution with n_i trials.
- γ_i represents the concentration parameter for the number of successes, derived from the probability of success and scaled by τ.
- η_i represents the concentration parameter for failures, derived from the probability of failure $(1 - \overline{\rho})$ and also scaled by τ.
- $\overline{\rho}$ is the probability of success for the i-th observation. The logit function transforms the linear predictor (which can take any real value) into a probability value between 0 and 1.
- τ is derived from and is used as a scaling factor for the shape parameters and .
- is a vector of parameters, each representing the effect of the group variable *i on the success probability.
- is a random variable following an exponential distribution with a rate of 1.

8.5 Reference(s)

McElreath (2018)

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

index_files/mediabag/Comparison-of-linear.png

9.2 Considerations

Caution

- 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 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).

9.3 Example

Below is an example code snippet demonstrating a Bayesian Poisson model using the Bayesian Inference (BI) package. Data consist of:

1) A continuous dependent variable *total_tools*, which represents the number of tools produced by a civilization.
2) A continuous independent variable *population* representing population size.

3) A categorical independent variable *cid* representing different civilizations.

The goal is to estimate the production of tools based on population size, accounting for each civilization.

9.3.1 Python

```
from main import*

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

# Import Data & Data Manipulation -----------------------------------------------
m.data('../data/Kline.csv', sep=';') # Import
m.sale(["population"])  # Scale
m.df["cid"] = (m.df.contact == "high").astype(int) # Manipulate
m.data_to_model(['total_tools', 'population', 'cid']) # Send to model


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

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

# Summary -----------------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions
```

9.3.2 R

```
library(reticulate)
bi <- import("main")
# Setup device--------------------------------------------------
m = bi$bi(platform='cpu')

# Import Data & Data Manipulation ------------------------------------------------
m$data('../data/Kline.csv', sep=';') # Import
m$scale(list('population'))# Scale
m$df["cid"] =  as.integer(ifelse(m$df$contact == "high", 1, 0)) # Manipulate
m$data_to_model(list('total_tools', 'population', 'cid' )) # Send to model (convert to jax


# Define model ------------------------------------------------
model <- function(total_tools, population, cid){
  # Parameters priors distributions
  alpha = bi$dist$normal(3, 0.5, name='alpha', shape = tuple(as.integer(1)))
  beta = bi$dist$normal(0, 0.2, name='beta', shape = tuple(as.integer(1)))
  l = bi$jnp$exp(alpha[cid] + beta[cid]*population)
  # Likelihood
  bi$lk("y", bi$Poisson(l), obs=total_tools)
}

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

# Summary ------------------------------------------------
m$sampler$print_summary(0.89) # Get posterior distributions
```

9.4 Mathematical Details

9.4.1 *Frequentist formulation*

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

$$\log(\lambda_i) = \alpha + \beta X_i$$

Where:

- λ_i is the mean rate parameter of the Poisson distribution (expected count) for observation i, modeled as the exponential function of the linear combination of predictors.

- $\log(\lambda_i)$ is the log of the mean rate parameter for observation i, ensuring it is positive.
- β is the regression coefficient.
- α is the intercept term.
- X_i is the value for the independent variables for observation i.

9.4.2 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distribution as follows:

$$Y \sim Poisson(\lambda_i)$$

$$\log(\lambda_i) \sim \alpha + \beta X_i$$

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

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

Where:

- Y_i is the dependent variable for observation i.
- λ_i is the mean rate parameter of the Poisson distribution for observation i, modeled as the exponential function of the linear combination of predictors.
- $\log(\lambda_i)$ is the log of the mean rate parameter for observation i.
- α and β are the prior distributions for the intercept and the regression coefficients, respectivelly .
- λ is the mean rate parameter of the Poisson distribution, modeled as the exponential function of the linear combination of predictors.
- X_i is the value for the independent variables for observation i.

9.5 Notes

> **Note**
>
> - We can apply multiple variables similarly as in chapter 2.
> - We can apply interaction terms similarly as in chapter 3.
> - We can apply categorical variables similarly as in chapter 4.

9.6 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 the *Negative Binomial model*.

10.2 Considerations

Caution

- We have the same considerations as for the 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`).

10.3 Example

Below is an example code snippet demonstrating Bayesian Gamma-Poisson model using the Bayesian Inference (BI) package:

10.3.1 Python

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

# Import Data & Data Manipulation -----------------------------------------------
m.data('../data/Sim dat Gamma poisson.csv', sep=',')  # Import
m.data_to_model(['log_days', 'monastery', 'y']) # Send to model (convert to jax array)

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

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

# Summary -----------------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions
```

10.3.2 R

```r
library(reticulate)
bi <- import("main")

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

# Import data -------------------------------------------------
m$data('Sim dat Gamma poisson.csv', sep=',') # Import
m$data_to_model(list('log_days', 'monastery', 'y' )) # Send to model (convert to jax array)

# Define model ------------------------------------------------
model <- function(log_days, monastery, y){
  # Parameters priors distributions
  alpha = bi$dist$normal(0, 1, name='alpha')
  beta = bi$dist$normal(0, 0.2, name='beta')
  l = log_days + alpha + beta * monastery
  # Likelihood
  bi$lk("y", bi$Poisson(rate=l), obs=y)
}

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

# Summary -----------------------------------------------------
m$sampler$print_summary(0.89) # Get posterior distributions
```

10.4 Mathematical Details

10.4.1 *Frequentist formulation*

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

$$\log(\lambda_i) = \exp(\text{rates}_i + \alpha + \beta X_i)$$

Where:

- λ_i is the mean rate parameter of the negative binomial distribution (expected count) for observation i.
- $\log(\lambda_i)$ is the log of the mean rate parameter, ensuring it is positive for observation i.

- α is the intercept term.
- β is the regression coefficients.
- X_i is the value of the predictor variable for observation i.

10.4.2 *Bayesian model*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distribution as follows:

$$Y_i \sim \text{Poisson}(\lambda)$$

$$\log(\lambda_i) \sim \text{rates}_i + \alpha + \beta X_i$$

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

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

Where:

- Y_i is dependent variable for observation i.
- λ_i is the mean rate parameter of the Poisson distribution for observation i, assuming that each Poisson count observation has its own $rate_i$.
- $\log(\lambda_i)$ is the log of the mean rate parameterfor observation i, ensuring it is positive.
- α is the intercept term.
- β is the regression coefficients.
- X_i is the value of the predictor variable for observation i.

10.5 Notes

> **Note**
>
> - We can apply multiple variables similarly as in chapter 2.
> - We can apply interaction terms similarly as in chapter 3.
> - We can apply categorical variables similarly as in chapter 4.

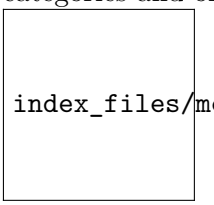
10.6 Reference(s)

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* model.



11.2 Considerations

Caution

- We have the same considerations as for Regression for continuous variable.
- One way to interpret a multinomial model 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 the "nn.softmax" line in the code).
- The intercept α 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 β 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.

11.3 Example

Below is an example code snippet demonstrating Bayesian multinomial model using the Bayesian Inference (BI) package:

11.3.1 Python

```
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
)
d.to_csv('Sim data multinomial.csv')

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

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

# Summary -----------------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions
```

11.3.2 R

```r
library(reticulate)
bi <- import("main")

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

# Import Data & Data Manipulation -----------------------------------------------
m$data('Sim data multinomial.csv', sep=',') # Import

keys <- c("income", "career")
income = unique(m$df$income)
income = income[order(income)]
values <- list(as.integer(income), as.integer(m$df$career))
m.data_on_model = py_dict(keys, values, convert = TRUE)
m.data_on_model

# Define model ------------------------------------------------
model <- function(income, career){
  # Parameters priors distributions
  alpha = bi$dist$normal(0, 1, name='alpha', shape = tuple(as.integer(2)))
  beta = bi$dist$halfnormal(0.5, name='beta', shape = tuple(as.integer(1)))
  s_1 = alpha[0] + beta * income[1]
  s_2 = alpha[1] + beta * income[1]
  s_3 = 0 # reference category
  p = bi$jax$nn$softmax(bi$jnp$stack(list(s_1[0], s_2[0], s_3[0])))
  # Likelihood
  bi$lk("y", bi$Categorical(probs=p[career]), obs=career)
}

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

# Summary ------------------------------------------------
m$sampler$print_summary(0.89)# Get posterior distributions
```

11.4 Mathematical Details

11.4.1 *Frequentist formulation*

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

$$logit(p_ik) =_k +X_i$$

Where:

- p_ik is the probability of the -th observation being in category .
- $_k$ is the intercept for category .
- is the regression coefficients common to all categories.
- X_i is the vector of independent variables for the -th observation.
- A reference category is often chosen to simplify the model.

11.4.2 *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:

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

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 $_k$ representing the probability of category .
- $\frac{N!}{\frac{K}{k=1}yk!}$ is the multinomial coefficient that 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.
-

11.5 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* model.

index_files/mediabag/Multinomial-Logistic.jpg

12.2 Considerations

Caution

- We have the same considerations as for the Multinomial model.
- One major difference from the multinomial model is that the Dirichlet model doesn't require a simplex but rather strictly positive values. We can thus exponentiate the outputs from the categorical regressions instead of using the softmax function.
-

12.3 Example

```
# 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
)
d.to_csv('Sim data multinomial.csv')

# Define model ------------------------------------------------
def model(income, career):
    # Parameters priors distributions
    alpha = dist.normal(0, 1, shape=[2], name='a')
    beta = dist.halfnormal(0.5, shape=[1], name='b')
    s_1 = alpha[0] + beta * income[0]
    s_2 = alpha[1] + beta * income[1]
    s_3 = alpha[0] + beta * income[0]
    p = jax.nn.exp(jnp.stack([s_1[0], s_2[0], s_3[0]]))
    # Likelihood
    lk("y", DirichletMultinomial(probs=p[career]), obs=career)
# Run sampler -------------------------------------------------
m.run(model)

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

12.4 Mathematical Details

12.4.1 *Formula*

12.4.2 *Bayesian Model*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distribution as follows:

12.5 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.

13.2 Considerations

Caution

- 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 $W_{1\pi}, W_{2\pi}, ..., W_{n\pi}$, $W_{1\lambda}, W_{2\lambda}, ..., W_{n\lambda}$, b_π, and b_λ.

13.3 Example

Below is an example code snippet demonstrating Bayesian Zero-Inflated Poisson regression using the Bayesian Inference (BI) package. The data represent the production of books in a monastery (y), which is affected by the number of days that individuals work, as well as the number of days individuals drink.

13.3.1 Python

```
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)
```

13.3.2 R

```
library(reticulate)
bi <- import("main")

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

# Define parameters
prob_drink = 0.2  # 20% of days
rate_work = 1     # average 1 manuscript per day
# sample one year of production
N = as.integer(365)
drink = bi$dist$binomial(total_count = as.integer(1), probs = prob_drink, shape = tuple(N),
y = (1 - drink) *  bi$dist$poisson(rate_work, shape = tuple(N), sample = T)
data = list()
data$y = y
m$data_on_model = data
# Import data -------------------------------------------------

# Define model ------------------------------------------------
model <- function(y){
  al = bi$dist$normal(1, 0.5, name='al')
  ap = bi$dist$normal(-1, 1, name='ap')
  p = bi$jax$scipy$special$expit(al)
  lambda_ = bi$jnp$exp(al)
  bi$lk("y", bi$ZeroInflatedPoisson(p, lambda_), obs=y)
}

# Run MCMC ----------------------------------------------------
m$run(model)

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

14 Mathematical Details

14.0.1 *Frequentist formulation*

We model the relationship between the independent variable 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:

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

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 respectively the intercept and the regression coefficient for the logistic model.
- α_λ and β_λ are respectively the intercept and the regression coefficient for the count model.
- X_i is the independent variables value for observation i.

14.0.2 *Bayesian formulation*

In the Bayesian formulation, we define each parameter with priors . We can express the Bayesian regression model accounting for prior distribution as follows:

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

$$\text{logit}(\pi) = \alpha_\pi + \beta_\pi X$$

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

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

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

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

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

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 respectively the intercept and the regression coefficient for the logistic model.
- α_λ and β_λ are respectively the intercept and the regression coefficient for the count model.
- X_i is the independent variables value for observation i.

14.1 Reference(s)

McElreath (2018)

15 Varying intercepts

15.1 General Principles

To model the relationship between dependent 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.

15.2 Considerations

Caution

- 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 is defined based on the k declared groups.
- Each intercept have is own prior -i.e. a hyper-prior

-In the code below, the *hyper-prior* is `a_bar`.

15.3 Example

Below is an example code snippet demonstrating Bayesian regression with varying intercepts using the Bayesian Inference (BI) package. The data consist of a dependent variable representing individuals' survival (*surv*) and an independent categorical variable (*tank*), which indicates the tank where the individual was born, with a total of 48 tanks.

15.4 Python

```
from main import*

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

# Import Data & Data Manipulation ----------------------------------------------
m.data('../data/reedfrogs.csv', sep=';') # Import
m.df["tank"] = np.arange(m.df.shape[0]) # Manipulate
tank = jnp.array(m.df["tank"].astype('int32').values) # Manipulate
density = jnp.array(m.df["density"].astype('float32').values) # Manipulate
surv = jnp.array(m.df["surv"].astype('int32').values) # Manipulate
m.data_on_model = dict(
    tank = tank,
    surv = surv
) # Send to model (convert to jax array)

# Define model ----------------------------------------------
def model(tank, surv):
  # Parameters priors distributions
    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])
    # Likelihood
    lk("y", Binomial(total_count = density, logits = p), obs=surv)

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

# Summary ----------------------------------------------
m.sampler.print_summary(0.89) # Get posterior distributions
```

15.5 R

```
library(reticulate)
bi <- import("main")

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

# Import Data & Data Manipulation -----------------------------------------------
m$data('../data/reedfrogs.csv', sep=';')  # Import
m$df$tank = c(0:(nrow(m$df)-1)) # Manipulate
m$data_to_model(list('tank', 'surv', 'density')) # Manipulate
m$data_on_model$tank = m$data_on_model$tank$astype(bi$jnp$int32) # Manipulate
m$data_on_model$surv = m$data_on_model$surv$astype(bi$jnp$int32) # Manipulate

# Define model ------------------------------------------------
model <- function(tank, surv, density){
  # Parameters priors distributions
  sigma = bi$dist$exponential( 1,  name = 'sigma')
  a_bar = bi$dist$normal(0, 1.5, name='a_bar')
  alpha = bi$dist$normal(a_bar, sigma, name='alpha', shape = tuple(as.integer(48)))
  p = alpha[tank]
  # Likelihood
  bi$lk("y", bi$Binomial(total_count = density, logits = p), obs=surv)
}

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

# Summary -----------------------------------------------------
m$sampler$print_summary(0.89) # Get posterior distributions
```

15.6 Mathematical Details

15.6.1 *Frequentist formulation*

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.

15.6.2 *Bayesian model*

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

$$Y_{ik} = Normal(\mu_{ik}, \sigma)$$
$$\mu_{ik} = \alpha_j + \beta X_{ik} + \sigma$$
$$\alpha_k \sim \text{Normal}(\mu_{\alpha_k}, \sigma_{\alpha_k})$$
$$\beta \sim \text{Normal}(0, 1)$$
$$\sigma \sim \text{Exponential}(1)$$
$$\mu_{\alpha_k} \sim \text{Normal}(0, 1)$$
$$\sigma_{\alpha_k} \sim \text{Exponential}(1)$$

Where:

- Y_{ij} is the likelihood function for the outcome variable.
- α_k is the varying intercepts across groups.
- μ_{α_k} is the overall mean intercept.
- σ_{α_k} is the variance of the intercepts across groups.
- β is the prior distributions for the regression coefficients.
- σ is the prior distributions for the error term.

15.7 Notes

Note

- 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.

15.8 Reference(s)

McElreath (2018)

16 Varying slopes

16.1 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.

16.2 Considerations

Caution

- 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.

16.3 Example

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

16.3.1 Simulated data

16.4 Python

```
from main import*
# Setup device-----------------------------------------------
m = bi(platform='cpu')
m.data('Sim data multivariatenormal.csv', sep=',')
m.data.to_model(['cafe', 'wait', 'N_cafes'])

def model(cafe, wait, N_cafes):
    a = dist.normal(5, 2, name = 'a') #Standard Normal Vector
    b = dist.normal(-1, 0.5, name = 'b') # Standard Normal Vector
    sigma_cafe = dist.exponential(1, shape=[2], name = 'sigma_cafe')
    sigma = dist.exponential( 1, name = 'sigma')
    Rho = dist.lkj(2, 2, name = 'Rho') # Cholesky Factor
    # Applies the correlation structure between the variables.
    # Scaling the Cholesky Factor by the standard deviations.
    cov = jnp.outer(sigma_cafe, sigma_cafe) * Rho # In a multivariate normal distribution,
    #This operation applies the correlation and standard deviation structure (encoded in so
    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)
```

16.5 R

16.6 Mathematical Details

16.7 Mathematical Details

16.7.1 *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.
- σ is the error term, assumed to be strictly positive.

16.7.2 *Bayesian model*

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

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

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 using the Cholesky Factor .

The full cetered version of the model is thus :

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

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

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

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

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

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

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

$$L \sim LKJcorr(2)$$

16.8 Notes

Note

- 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:

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

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

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

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

16.9 Reference(s)

McElreath (2018)

17 Gaussian Processes

17.1 General Principles

Through varying intercepts and slopes we have seen how quantify some of the unique features that generate variation across clusters and covariance among the observations within each cluster. But through the covariance matrix that is used to account for correlation between clusters, we are assuming inherently linear relationships between clusters. What if we want to model the relationship between two variables that are not linearly related? In this case, we can use a Gaussian Process (GP) to model the relationship between two variables. Basically, a GP is a a varying slope model with a covariance matrix where each element of the matrix is a kernel function .

17.2 Considerations

Caution

- To capture complex, non-linear relationships in data where the underlying function is smooth but has an unknown functional form, GPs use a kernel
- GPs assume normally distributed errors and may not be appropriate for all types of noise
- The choice of kernel hyperparameters can significantly impact results, thus GPs require choosing an appropriate kernel function that captures the expected behavior of your data.
- Though kernel definition we can incorporate domain knowledge.
- They scale poorly with dataset size ($O(n^3)$ complexity) due to matrix operations, thus memory requirements can be substantial for large datasets which as lead neural networks to be used instead to resolve large non linear problems.

17.3 Example

Below is an example code snippet demonstrating Gaussian Process regression:

17.4 Python

```
import time as tm
from main import*

Kline2 = pd.read_csv('resources/data/Kline2.csv', sep=";")
islandsDistMatrix = pd.read_csv('resources/data/islandsDistMatrix.csv'"", index_col=0)
d = Kline2
d["society"] = range(1, 11)  # index observations


dat_list = dict(
    T=d.total_tools.values,
    P=d.population.values,
    society=d.society.values - 1,
    Dmat=islandsDistMatrix.values,
)

# setup platform-----------------------------------------------
from main import*
m = bi(platform='cpu')


def model(Dmat, P, society, T):
    a = bi.dsit.exponential(1, name = 'a')
    b = bi.dsit.exponential(1, name = 'b')
    g = bi.dsit.exponential(1, name = 'g')
    etasq = bi.dsit.exponential(2, 'etasq')
    rhosq = bi.dsit.exponential(0.5 'rhosq')

    # non-centered Gaussian Process prior
    SIGMA = cov_GPL2(Dmat, etasq, rhosq, 0.01)
    L_SIGMA = jnp.linalg.cholesky(SIGMA)
    z = normal('z', [10], 0, 1)
    k = (L_SIGMA @ z[..., None])[..., 0]
    lambda_ = a * P**b / g * jnp.exp(k[society])
    sample("T", Poisson(lambda_), obs=T)

# Run sampler ------------------------------------------------

m14_8.run(model)

m14_8.sampler.print_summary(0.89)
```

17.5 R

```
R implementation would go here#
```

17.6 Mathematical Details

17.6.1 *Formula*

The following equation allows us to evaluate the relationship between dependent variable Y and independent variable X while incorporating a GP for variable Z :

$$Y_i = \alpha + \beta X_i + \gamma_{Z_i}$$

where: - Y_i is the i-th value for the dependent variable Y.

- α is the intercept term.
- β is the regression coefficient term.
- X_i is the i-th value for independent variable X.
- γ_{Z_i} is the gaussian process i-th value for independent variable Z.

The GP γ_{Z_i} follow a multivariate normal distribution:

$$\begin{pmatrix} Z_1 \\ \vdots \\ Z_n \end{pmatrix} \sim MVNormal \left(\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, K \right)$$

where:

- $(Z_1, ..., Z_n)$ represents a collection of all values of the random variable Z.
- $(0, ..., 0)$ represents the mean vector of the multivariate normal distribution of the same size as the number of random variable and set to zero .
- K is the covariance matrix of the random variable Z. Each element Kij of the matrix is given by the kernel function evaluated at the corresponding points: $K_{ij} = k(Z_i, Z_j)$

$$K = \begin{pmatrix} k(Z_1, Z_1) & k(Z_1, Z_2) & \cdots & k(Z_1, Z_n) \\ k(Z_2, Z_1) & k(Z_2, Z_2) & \cdots & k(Z_2, Z_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(Z_n, Z_1) & k(Z_n, Z_2) & \cdots & k(Z_n, Z_n) \end{pmatrix}$$

- Multiple kernel function exist and will be discussed in the Note(s) section. But the most common one is the quadratic kernel:

$$K_{ij} = \eta^2 exp(-p^2 D_{ij}^2) + \delta_{ij}\sigma^2$$

Where:

- η is the signal variance, representing the overall variance of the outputs of the Gaussian process. It scales the influence of the kernel function. A larger η^2 indicates a wider range of values the function can take.
- p determines the rate of decline.
- D_{ij} is the distance between the i-th and j-th points.
- δ_{ij} is the Kronecker delta taking a value of zero when $i = j$, allowing to included in the calculation the self-covariance.
- σ^2 is the noise variance, which accounts for the observation noise in the data. It represents the uncertainty or variability in the measurements or outputs at each point. The term effectively adds this noise variance only when $i =$, ensuring that the diagonal elements of the covariance matrix represent the total variance at each input point.

17.7 *Bayesian model*

In the Bayesian formulation, we define each parameter with priors . We can express a Bayesian version of this GP using the following model:

$$Y_i = \alpha + \beta X_i + \gamma_{Z_i}$$

$$\gamma \sim MVNormal\left(\begin{pmatrix}0\\ \vdots \\ 0\end{pmatrix}, K\right)$$

$$K_{ij} = \eta^2 exp(-p^2 D_{ij}^2) + \delta_{ij}\sigma^2$$

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

$$\eta^2 \sim HalfCauchy(0, 1)$$

$$p^2 \sim HalfCauchy(0, 1)$$

where:

- Y_i is the i-th value for the dependent variable Y.
- α is the intercept term with prior $Normal(0, 1)$.
- β is the regression coefficient term with prior $Normal(0, 1)$.
- X_i is the i-th value for independent variable X.
- γ_{Z_i} is the gaussian process i-th value for independent variable Z.
- γ is the latent function modeled by the GP.
- K_{ij} is the kernel function evaluated at the corresponding points: $K_{ij} = k(Z_i, Z_j)$ with priors HalfCauchy(0,1) for η^2 and p^2 to ensure positive values.

17.8 Notes

Note

Common kernel functions include:

- *Radial Basis Function* (RBF) or Squared Exponential Kernel:

$$k(x, x') = \sigma^2 \exp\left(-\frac{||x - x'||^2}{2l^2}\right)$$

- *Rational Quadratic Kernel*, this kernel is equivalent to adding together many RBF kernels with different length scales:

$$k(x, x') = \sigma^2 \left(1 + \frac{||x - x'||^2}{2l^2}\right)^{-\alpha}$$

- *Periodic kernel* allows to model functions which repeat themselves exactly:

$$k(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi||x - x'||/p)}{l^2}\right)$$

- *Locally Periodic Kernel*:

$$k(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi||x - x'||/p)}{l^2}\right) \exp\left(-\frac{||x - x'||^2}{2l^2}\right)$$

- GPs can be extended to classification problems using link functions Multi-output problems using matrix-valued kernels Deep learning through Deep Kernel Learning
- Computational tricks for large datasets include:
  - Sparse approximations (e.g., FITC, VFE)
  - Inducing points methods
  - Random Fourier features

17.9 Reference(s)

McElreath (2018)
https://www.cs.toronto.edu/~duvenaud/cookbook/

18 Measuring error

18.1 General Principles

18.2 Considerations

Caution

18.3 Example

Below is an example code snippet demonstrating Bayesian Measuring error model using the Bayesian Inference (BI) package:

18.4 Python

18.5 Mathematical Details

18.5.1 *Frequentist formulation*

18.5.2 *Bayesian formulation*

18.6 Notes

Note

18.7 Reference(s)

McElreath (2018)

19 Missing data

19.1 General Principles

19.2 Considerations

Caution

19.3 Example

Below is an example code snippet demonstrating Bayesian Missing data model using the Bayesian Inference (BI) package:

19.4 Python

19.5 Mathematical Details

19.5.1 *Frequentist formulation*

19.5.2 *Bayesian formulation*

19.6 Notes

Note

19.7 Reference(s)

McElreath (2018)

20 Latent Variables

20.1 General Principles

In some scenarios, the observed data does not directly reflect the underlying structure or factors influencing the outcome. Instead, *latent variables* — variables that are not directly observed but inferred from the data — can help model this hidden structure. These latent variables capture unobserved factors that affect the relationship between predictors (X) and the outcome (Y).
We model the relationship between the predictor variables (X) and the outcome variable (Y) with a latent variable (Z) as follows:

$$Y = f(X, Z) + \epsilon$$

Where: - Y is the observed outcome variable. - X is the observed predictor variable(s). - Z is the latent (unobserved) variable, which we aim to infer. - *f(X, Z)* is the function that relates X and Z to Y. - $\epsilon is the error term, typically assumed to be normally distributed with mean 0 and variance$

*In Bayesian regression with latent variables, we consider the uncertainty in both the observed and latent variables. We declare prior distributions for the latent variables, in addition to the usual priors for regression coefficients and intercepts. These latent variables are often modeled using Gaussian distributions (*Normal *priors) or more flexible distributions such as* Multivariate Normal *for correlations among the latent variables.*
The goal is to infer the posterior distribution over both the parameters and the latent variables, given the observed data.

20.2 Example

Below is an example code snippet demonstrating Bayesian regression with latent variables using TensorFlow Probability:

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

# Data Simulation ----------------------
NY = 4  # Number of dependent variables or outcomes (e.g., dimensions for latent variables,
NV = 8  # Number of observations or individual-level data points (e.g., subjects)

# Initialize the matrix Y2 with shape (NV, NY) filled with NaN values, to be filled later
Y2 = np.full((NV, NY), np.nan)

# Generate the means and offsets for the data
# means: Generate random normal means for each of the NY outcomes
# offsets: Generate random normal offsets for each of the NV observations
means = bi.distribution.normal(0, 1, shape=(NY,), sample = True, seed = 10)
offsets = bi.distribution.normal(0, 1, shape=(NV,1), sample = True, seed = 20)

# Fill the matrix Y2 with simulated data based on the generated means and offsets
# Each observation (i) is the sum of an individual-specific offset and an outcome-specific
for i in range(NV):
    for k in range(NY):
        Y2[i, k] = means[k] + offsets[i]

# Simulate individual-level random effects (e.g., random slopes or intercepts)
# b_individual: A matrix of size (N, K) where N is the number of individuals and K is the n
b_individual = bi.distribution.normal(0, 1, shape=N, K), sample = True, seed = 0)

# mu: Add an additional effect 'a' to the individual-level random effects 'b_individual'
# 'a' could represent a population-level effect or a baseline
mu = b_individual + a

# Convert Y2 to a JAX array for further computation in a JAX-based framework
Y2 = jnp.array(Y2)


# Set data -------------------------------------------------
dat = dict(
    NY = NY,
    NV = NV,
    Y2 = Y2
)

# Define model ---------------------------------------------
def model(NY, NV, Y2):
    means = bi.dist.normal(0, 1, shape=(NY,), name = 'means')
    offset = bi.dist.normal(0, 1, shape=(NY,1), name = 'offset')
    sigma = bi.dist.exponential(1, shape=(NY,), name = 'sigma')
    sigma =  numpyro.sample('sigma', numpyro.distributions.Exponential(1).expand([NY]))
    tmp = jnp.tile(means, (NV, 1)).reshape(NV,NY)
    mu_l = tmp + offset
    numpyro.sample('Y2', Normal(mu_l, jnp.tile(sigma, [NV, 1])), obs=Y2)

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

20.3 Mathematical Details

We can express the Bayesian latent variable model using probability distributions as follows:

$$p(Y|X, Z, W, \sigma) = Normal(X * W + Z, \sigma^2)$$
$$p(Z) = Normal(0, \tau^2)$$
$$p(W) = Normal(0, \alpha^2)$$

Where: - p(Y | X, Z, W, σ)$istheliklihoodfunctionfortheobservedoutcomevariable, whichdependsonboththeob$
$p(Z)isthepriordistributionforthelatentvariableZ, oftenmodeledasNormalwithameanof0andvariance$

- **Latent Variable (Z)**: Represents hidden factors not captured by the observed variables, allowing the model to explain more of the variance in the outcome. For instance, in a psychological model, Z might represent a latent trait such as intelligence or anxiety that influences the outcome.
- **Posterior Inference**: The posterior distribution of the latent variable Z can give insights into how much the unobserved factors contribute to the outcome.

20.4 Use Cases

- **Latent Factors in Psychometrics**: In psychometric models, latent variables represent traits or abilities that are not directly observed, such as cognitive ability or personality traits.
- **Time-Varying Effects**: Latent variables can represent unobserved time trends or individual-specific effects in time-series or longitudinal models.
- **Mixed Models**: In hierarchical or mixed models, latent variables can represent group-specific intercepts or slopes.

21 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.

21.1 Considerations

Caution

- The particularity here is that varying intercepts and slopes are generated for both nodal effects and dyadic effects . 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.

21.2 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_individu
    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], cho
    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
    focal_individual_predictors = data['individual_predictors'],
    target_individual_predictors = data['individual_predictors']
)

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

21.3 Mathematical Details

21.3.1 *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 .
-

21.3.2 *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 :

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

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

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

$$L \sim LKJ(2)$$

21.4 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, 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:

```
def model2(idx, result_outcomes, dyad_effects, focal_individual_predictors, target_indivi
    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_predict

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

    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], c
    dr = dr_terms + rf

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

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 multipl
    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.

22 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.

22.1 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(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.

22.2 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_pr
    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', na
    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_

    ## SR ---------------------------------------
    sr_terms, focal_effects, target_effects = bi.net.nodes_terms(focal_individual_predictor
    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_densit
    dr = dr_terms + rf

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

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]']]
```

22.3 Mathematical Details

22.3.1 *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 and j.
- B_{ji} is the link probability between category j to i.

22.3.2 *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.

22.4 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.

23 Modeling Network with control for data collection biases

Data collection biases are a persistent issue in studies of social networks. Two main types of biases can be considered: exposure biases and censoring biases .
To account for exposure biases, we can switch the network link probability model from a *Poisson* distribution to a *Binomial* distribution, as the binomial distribution allows us to account for the number of trials for each data estimation.
To address censoring biases, we need to add an additional equation to account for the probability of missing an interaction during observation when modeling interaction between individual i and j.

23.1 Considerations

Caution

23.2 Example 1

Below is an example code snippet demonstrating Bayesian network model with send-receiver effect, dyadic effect and block model effect while accounting for exposure biases:

23.3 Example2

Below is an example code snippet demonstrating Bayesian network model with send-receiver effect, dyadic effect and block model effect while accounting for exposure biases and cesnsoring biases:

23.4 Mathematical Details

23.4.1 *Main Formula*

$$Y_{[i,j]} \sim \text{Binomial}\Big(E_{[i,j]}, Q_{[i,j]}\Big)$$

$$Q_{[i,j]} \in \{0, 1\}$$

Where:

- $E_{[i,j]}$ is the number of trials for each observation (i.e., the sampling effort).
- $Q_{[i,j]}$ is the indicator of a true tie between i and j.

$$Q_{[i,j]} \sim \begin{cases} 0 & \text{when no interactions occurs or when i or j are not detectable} \\ 1 & \text{when i and j are both detectable} \end{cases}$$

$$Q_{[i,j]} = \phi_{[i,j]}\eta_{[i]}\eta_{[j]}$$

Where: - $\phi_{[i,j]}$ is the probability of a true tie between i and j. - $\eta_{[i]}$ is the probability of individual i being detectable. - $\eta_{[j]}$ is the probability of individual j being detectable.

23.4.2 *Defining formula sub-equations and prior distributions*

We can let $\eta_{[i]}$ depend on individual-specific covariates. To model the probability of censoring, we can model $1 - \eta_{[i]}$:

$$\text{logit}(1 - \eta_{[i]}) = \mu_\psi + \hat{\psi}_{[i]}\sigma_\psi + \dots$$

where - μ_ψ is an intercept - σ_ψ is a scalar for the variance of random effects - $\hat{\psi}_{[i]} \sim \text{Normal}(0, 1)$, and the ellipsis signifies any linear model of coefficients and individual-level covariates. For example, if C is an animal-specific measure, like a binary variable for cryptic coloration, then the ellipsis may be replaced with: $\kappa_{[5]}C_{[i]}$, to give the effects of coloration on censoring probability.

23.5 Note(s)

> **Note**
>
> - One major limitation of this model is the necessity of having an estimation of the censoring bias for each individual.

24 Network metrics

24.1 General Principles

24.2 Considerations

Caution

24.3 Example

Below is an example code snippet demonstrating Bayesian Network metrics model using the Bayesian Inference (BI) package:

24.4 Python

24.5 Mathematical Details

24.5.1 *Frequentist formulation*

24.5.2 *Bayesian formulation*

24.6 Notes

Note

24.7 Reference(s)

McElreath (2018)

25 Multiplex network model

25.1 General Principles

25.2 Considerations

Caution

25.3 Example

Below is an example code snippet demonstrating Bayesian Multiplex network model using the Bayesian Inference (BI) package:

25.4 Python

25.5 Mathematical Details

25.5.1 *Frequentist formulation*

25.5.2 *Bayesian formulation*

25.6 Notes

Note

25.7 Reference(s)

McElreath (2018)

26 Multiplex temporal network model

26.1 General Principles

26.2 Considerations

Caution

26.3 Example

Below is an example code snippet demonstrating Bayesian Multiplex temporal network model using the Bayesian Inference (BI) package:

26.4 Python

26.5 Mathematical Details

26.5.1 *Frequentist formulation*

26.5.2 *Bayesian formulation*

26.6 Notes

Note

26.7 Reference(s)

McElreath (2018)

27 Multilayer Networks

27.1 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.

27.2 Considerations

Caution

27.3 Example

Below is an example code snippet demonstrating Bayesian Multilayer Networks model using the Bayesian Inference (BI) package:

27.4 Python

27.5 Mathematical Details

27.5.1 *Frequentist formulation*

27.5.2 *Bayesian formulation*

27.6 Notes

Note

27.7 Reference(s)

McElreath (2018)

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