Binomial Model

General Principles

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

Considerations

> **Note**
>
> - We have the same considerations as for Linear Regression for continuous variable.
> - This is the first model for which we need a link function: e.g., the *logit* function. The *logit* link function converts the linear combination of predictor variables into probabilities, making it suitable for modeling the probability of binary outcomes. It helps to estimate the relationship between predictors and the probability of success, ensuring that model predictions fall within the bounds of the binomial distribution's success parameter $\in (0, 1)$.

Example

Below is an example code snippet that demonstrates Bayesian binomial regression using the BayesForge (BF) package. The data consist of one binary dependent variable (*pulled_left*), which represents which lever each chimpanzee pulled in an experimental setup. The goal is to evaluate the probability of pulling the left side. This example is based on McElreath (2018).

Python

```python
from BayesForge import bf

# setup platform-------------------------------------------------
m = bf(platform='cpu')
# import data ---------------------------------------------------
# Import
from importlib.resources import files
data_path = m.load.chimpanzees(only_path = True)
m.data(data_path, sep=';')
m.data_to_model(['pulled_left'])

# Define model --------------------------------------------------
def model(pulled_left):
    alpha = m.dist.normal( 0, 10,  name = 'alpha')
    m.dist.binomial(total_count = 1, logits=alpha, obs=pulled_left)

# Run sampler ---------------------------------------------------
m.fit(model, progress_bar=False)

# Diagnostic ----------------------------------------------------
m.summary()
```

```
/home/sosa/work/3.12venv/lib/python3.10/site-packages/tqdm/auto.py:21: TqdmWarning:

IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs

bf v 0.0.47 package loaded
jax.local_device_count 32
```

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha	0.32	0.09	0.18	0.47	0.0	0.0	1758.15	2257.24	1.0

R

```r
library(BayesForge)

# setup platform-------------------------------------------------
m=importBF(platform='cpu')
```

```r
# import data ------------------------------------------------
m$data(m$load$chimpanzees(only_path = T), sep=';')
m$data_to_model(list('pulled_left')) # Send to model (convert to jax array)

# Define model ------------------------------------------------
model <- function(pulled_left){
  # Parameters priors distributions
  alpha = bf.dist.normal( 0, 10, name = 'alpha')
  # Likelihood
  bf.dist.binomial(total_count = 1, logits = alpha, obs=pulled_left)
}


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

# Summary ------------------------------------------------
m$summary() # Get posterior distribution
```

Julia

```julia
using BayesForge

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

# Import Data & Data Manipulation ------------------------------------------------
# Import
data_path = m.load.chimpanzees(only_path = true)
m.data(data_path, sep=';')

# Define model ------------------------------------------------
@BF function model(pulled_left)
    a = m.dist.normal( 0, 10, shape=(1,), name = "a")
    m.dist.binomial(total_count = 1, logits=a[0], obs=pulled_left)
end

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

```
# Summary ----------------------------------------------
m.summary() # Get posterior distributions
```

Mathematical Details

Bayesian formulation

We can express the Bayesian Binomial regression model including prior distributions as follows:

$$Y_i \sim \text{Binomial}(N_i, p_i)$$

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

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

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

Where:

- Y_i is the count of successes for observation i (often a binary 0 or 1).
- N_i is the count of trials for observation i (1 in the case of binary outcomes, as in the example for `total_count` above).
- p_i is the probability of success ($0 < p_i < 1$) for observation i, the probability of a success.
- $logit(p_i)$ 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.
- β and α are the regression coefficient and intercept, respectively.

Notes

Note

- We can apply multiple variables similarly to chapter 2.
- We can apply interaction terms similarly to chapter 3.
- We can apply categorical variables similarly to chapter 4.
- Below is an example code snippet demonstrating a Bayesian binomial model for multiple categorical variables using the BayesForge (BF) package. The data consist of one binary dependent variable (*pulled_left*), which represents which 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 for the different conditions.

```
from BayesForge import bf
m = bf(platform='cpu')
m.data('../resources/data/chimpanzees.csv', sep=';')
m.df['treatment'] =  m.df.prosoc_left + 2 * m.df.condition
m.df['actor'] = m.df['actor'] - 1

m.data_to_model(['actor', 'treatment', 'pulled_left'])

def model(actor, treatment, pulled_left):
    a = m.dist.normal(0, 1.5, shape = (7,), name='a')
    b = m.dist.normal(0, 0.5, shape = (4,), name='b')
    p = a[actor] + b[treatment]
    m.dist.binomial(total_count = 1, logits=p, obs=pulled_left)

# Run sampler ------------------------------------------------
m.fit(model)
# Diagnostic -------------------------------------------------
m.summary()
```

```r
library(BayesForge)

# setup platform-----------------------------------------------
m=importBF(platform='cpu')

# import data -------------------------------------------------
m$data(paste(system.file(package = "BayesForge"),"/data/chimpanzees.csv", sep = ''), sep=';
m$data_to_model(list('pulled_left')) # Send to model (convert to jax array)

# Define model ------------------------------------------------
model <- function(pulled_left){
  # Parameters priors distributions
  a = bf.dist.normal( 0, 1.5, name = 'a')
  b = bf.dist.normal( 0, 0.5, name = 'b')
  p = a[actor] + b[treatment]
  # Likelihood
  m$binomial(total_count = 1, logits = alpha, obs=pulled_left)
}

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

# Summary -----------------------------------------------------
m$summary() # Get posterior distribution
```

Reference(s)

McElreath, Richard. 2018. *Statistical Rethinking: A Bayesian course with examples in R and Stan.* Chapman; Hall/CRC.