

Categorical Model

General Principles

To model the relationship between an outcome variable in which each observation is a single choice from a set of more than two categories and one or more independent variables, we can use a *Categorical* model.

Considerations

Caution

- We have the same considerations as for [Regression for continuous variable](#).
- One way to interpret a *Categorical* 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 “`jax.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.

Example

Below is an example code snippet demonstrating a Bayesian Categorical model using the BayesForge (BF) package. This example is based on McElreath (2018).

Python

```
from BayesForge import bf
import jax.numpy as jnp
import jax
# Setup device -----
m = bf('cpu')

# Import Data & Data Manipulation -----
# Import
from importlib.resources import files
data_path = m.load.sim_multinomial(only_path=True)
m.data(data_path, sep=',')

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

# Run sampler -----
m.fit(model, progress_bar=False) # Optimize model parameters through MCMC sampling

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

bf v 0.0.48 package loaded

E0526 16:11:56.527512 1054539 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
E0526 16:11:56.529276 1054539 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source

jax.local_device_count 32

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
a[0]	-0.19	0.59	-1.08	0.77	0.01	0.01	1931.39	1859.22	1.0
a[1]	-0.56	0.71	-1.67	0.59	0.02	0.01	1319.90	1366.07	1.0
b[0]	0.05	0.04	0.00	0.09	0.00	0.00	1133.49	1066.11	1.0

R

```
library(BayesForge)
jax = reticulate::import('jax')

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

# import data -----
m$data(m$load$sim_multinomial(only_path=T), sep=',')
keys <- c("income", "career")
income = unique(m$df$income)
income = income[order(income)]
values <- list(jnp$array(as.integer(income)),jnp$array( as.integer(m$df$career)))
m$data_on_model = py_dict(keys, values, convert = TRUE)

# Define model -----
model <- function(income, career){
  # Parameter prior distributions
  alpha = bf.dist.normal(0, 1, name='alpha', shape = c(2))
  beta = bf.dist.normal(0.5, name='beta')

  s_1 = alpha[0] + beta * income[0]
  s_2 = alpha[1] + beta * income[1]
  s_3 = 0 # reference category

  p = jax$nn$softmax(jnp$stack(list(s_1, s_2, s_3)))

  # Likelihood
  m$dist$categorical(probs=p[career], obs=career)
}

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

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

Julia

```
using BayesForge

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

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

# Define model -----
@BF function model(career, income)
    a = m.dist.normal(0, 1, shape=(2,), name = "a")
    b = m.dist.half_normal(0.5, shape=(1,), name = "b")

    # indexing works now because of the package update
    s_1 = a[0] + b * income[0]
    s_2 = a[1] + b * income[1]

    # Use jnp.array to create a Python object, so [0] indexing works
    s_3 = jnp.array([0.0])

    # Now s_3[0] is valid because it calls Python's __getitem__(0)
    p = jax.nn.softmax(jnp.stack([s_1[0], s_2[0], s_3[0]]))

    m.dist.categorical(probs=p, obs=career)
end

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

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

Mathematical Details

We can model a *Categorical* model using a *Categorical distribution*. The multinomial distribution models the counts of outcomes falling into different categories. For an outcome variable with K categories, the multinomial likelihood function is:

$$Y_i \sim \text{Categorical}(\theta_i) \theta_i = \text{Softmax}(\phi_i) \phi_{[i,1]} = \alpha_1 + \beta_1 X_i \phi_{[i,2]} = \alpha_2 + \beta_2 X_i \dots \phi_{[i,k]} = 0 \alpha_k \sim \text{Normal}(0, 1) \beta_k \sim \text{Normal}(0, 1)$$

Where:

- Y_i is the dependent categorical variable for observation i indicating the category of the observation.
- θ_i is a vector unique to each observation, i , which gives the probability of observing i in category k .
- ϕ_i give the linear model for each of the k categories. Note that we use the softmax function to ensure that the probabilities θ_i form a simplex.
- Each element of ϕ_i is obtained by applying a linear regression model with its own respective intercept α_k and slope coefficient β_k . To ensure the model is identifiable, one category, K , is arbitrarily chosen as a reference or baseline category. The linear predictor for this reference category is set to zero. The coefficients for the other categories then represent the change in the log-odds of being in that category versus the reference category.

Reference(s)

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