Multinomial Model

General Principles

To model the relationship between a vector outcome variable in which each element of the vector is a frequency from a set of more than two categories and one or more independent variables, we can use a *Multinomial* model.

Considerations

Note

- We have the same considerations as for the Categorical model.

Example

Below is an example code snippet demonstrating a Bayesian multinomial model using the Bayesian Inference (BI) package. This example is based on McElreath (2018).

Python

```
from BI import bi, jnp
import jax
# Setup device ------------------------------------------------
m = bi('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=',')
```

```python
# Define model -------------------------------------------------
def model(income, career):
    # Parameter prior distributions
    alpha = m.dist.normal(0, 1, shape=(2,), name='a')
    beta = m.dist.half_normal(0.5, shape=(1,), name='b')
    s_1 = alpha[0] + beta * income[0]
    s_2 = alpha[1] + beta * income[1]
    s_3 = [0]
    p = jnp.exp(jnp.stack([s_1[0], s_2[0], s_3[0]]))
    # Likelihood
    m.dist.multinomial(probs = p[career], obs=career)

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

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

```
jax.local_device_count 16

  0%|          | 0/1000 [00:00<?, ?it/s]warmup:   0%|          | 1/1000 [00:01<18:20,  1.10s/
arviz - WARNING - Shape validation failed: input_shape: (1, 500), minimum_shape: (chains=2, d
```

	mean	sd	hdi_5.5%	hdi_94.5%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
a[0]	0.00	0.97	-1.60	1.51	0.05	0.04	428.89	395.09	NaN
a[1]	82.06	1.02	80.40	83.58	0.05	0.05	472.75	340.24	NaN
b[0]	40.96	0.50	40.12	41.65	0.02	0.02	616.30	368.44	NaN

R

```r
library(BayesianInference)
jax = reticulate::import('jax')
# Setup platform------------------------------------------------
m=importBI(platform='cpu')

# import data ------------------------------------------------
m$data(m$load$sim_multinomial(only_path=T), sep=',')
keys <- c("income", "career")
```

```r
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 = bi.dist.normal(0, 1, name='alpha', shape = c(2))
  beta = bi.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$dist$multinomial(probs=p[career], obs=career)
}


# Run sampler ------------------------------------------------
m$fit(model)

# Summary ------------------------------------------------
m$summary()
```

Julia

```julia
using BayesianInference

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

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

# Define model ------------------------------------------------
```

```
@BI function model(income, career)
    # Parameter prior distributions
    alpha = m.dist.normal(0, 1, shape=(2,), name='a')
    beta = m.dist.half_normal(0.5, shape=(1,), name='b')
    s_1 = alpha[0] + beta * income[0]
    s_2 = alpha[1] + beta * income[1]
    #   Use jnp.array to create a Python object, so [0] indexing works
    s_3 = jnp.array([0.0])
    p = jnp.exp(jnp.stack([s_1[0], s_2[0], s_3[0]]))
    # Likelihood
    m.dist.multinomial(probs = p[career], 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 vector of frequencies using a Dirichlet distribution. For an outcome variable Y_i with K categories, the *Dirichlet* likelihood function is:

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

Where:

- Y_i is the outcome (i.e. the vector of frequencies for each k categories) for observation i.
- 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 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.