Dirichlet 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 *Dirichlet* model.

Considerations

> **Note**
>
> - We have the same considerations as for the Multinomial model.

Example

Python

```
from BayesForge import bf
import jax.numpy as jnp
import jax
import numpy as np
from jax.nn import softmax

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

# Simulated data --------------------------------------------
np.random.seed(42)
N = 200
X = np.random.normal(0, 1, size=N)
```

```
# True parameters
true_alpha = np.array([0.5, -0.5])
true_beta = np.array([1.0, 0.5])
true_kappa = 10.0

# Linear predictors
phi = np.zeros((N, 3))
phi[:, 0] = true_alpha[0] + true_beta[0] * X
phi[:, 1] = true_alpha[1] + true_beta[1] * X
phi[:, 2] = 0.0 # Reference category

# Probabilities (theta)
theta = softmax(phi, axis=1)

# Observations (Y)
Y = np.zeros((N, 3))
for i in range(N):
    Y[i, :] = np.random.dirichlet(theta[i, :] * true_kappa)

print(f"True alpha: {true_alpha}")
print(f"True beta: {true_beta}")
print(f"True kappa: {true_kappa}")

# Model data ----------------------------------------------
def model_dirichlet(X, Y):
    # Parameter prior distributions
    alpha = m.dist.normal(0, 5, shape=(2,), name="alpha")
    beta = m.dist.normal(0, 5, shape=(2,), name="beta")
    kappa = m.dist.exponential(1.0, name="kappa")

    # Linear predictors
    s1 = alpha[0] + beta[0] * X
    s2 = alpha[1] + beta[1] * X
    s3 = jnp.zeros_like(s1)

    phi = jnp.stack([s1, s2, s3], axis=1)
    theta = jax.nn.softmax(phi, axis=1)

    # Likelihood
    m.dist.dirichlet(theta * kappa, obs=Y)

m.data_on_model = dict(X=X, Y=Y)
```

```python
# Run sampler ------------------------------------------------
m.fit(model_dirichlet, num_samples=1000, num_warmup=500, progress_bar=False)

# Diagnostic -------------------------------------------------
summ = m.summary()
print("\nSummary results (Posterior Means):")
print(summ['mean']])
```

```
bf v 0.0.48 package loaded

E0526 16:18:00.732161 1065592 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
E0526 16:18:00.734019 1065592 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source


jax.local_device_count 32
True alpha: [ 0.5 -0.5]
True beta: [1.  0.5]
True kappa: 10.0

Summary results (Posterior Means):
          mean
alpha[0]  0.44
alpha[1] -0.50
beta[0]   1.08
beta[1]   0.63
kappa     9.82
```

R

```r
library(BayesForge)
m=importBF(platform='cpu')
jnp = reticulate::import('jax.numpy')

# Simulated data ---------------------------------------------
set.seed(42)
N = 200
X = rnorm(N, 0, 1)

# True parameters
true_alpha = c(0.5, -0.5, 0.0)
```

```
true_beta = c(1.0, 0.5, 0.0)
true_kappa = 10.0

# Linear predictors
phi = matrix(0, nrow=N, ncol=3)
phi[, 1] = true_alpha[1] + true_beta[1] * X
phi[, 2] = true_alpha[2] + true_beta[2] * X
phi[, 3] = 0.0

# Probabilities (theta) using softmax
exp_phi = exp(phi)
theta = exp_phi / rowSums(exp_phi)

# Observations (Y)
Y = matrix(0, nrow=N, ncol=3)
for(i in 1:N) {
  # Implementation of rdirichlet
  alpha_param = theta[i, ] * true_kappa
  gamma_samples = rgamma(3, shape = alpha_param, rate = 1)
  Y[i, ] = gamma_samples / sum(gamma_samples)
}

# Define model ----------------------------------------------
model <- function(X, Y){
  # Parameter prior distributions
  alpha = bf.dist.normal(0, 5, name='alpha', shape = c(2))
  beta = bf.dist.normal(0, 5, name='beta', shape = c(2))
  kappa = bf.dist.exponential(1.0, name='kappa')

  # Linear predictors
  s1 = alpha[1] + beta[1] * X
  s2 = alpha[2] + beta[2] * X
  s3 = jnp$zeros_like(s1)

  phi = jnp$stack(list(s1, s2, s3), axis=as.integer(1))
  theta = jnp$nn$softmax(phi, axis=as.integer(1))

  # Likelihood
  bf.dist.dirichlet(theta * kappa, obs=Y)
}

m$data_on_model = list(X=X, Y=Y)
```

```
# Run mcmc ------------------------------------------------
m$fit(model, progress_bar=FALSE)

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

Julia

```
using BayesForge
using Random
using Distributions

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

# Simulated data ------------------------------------------------
Random.seed!(42)
N = 200
X = rand(Normal(0, 1), N)

# True parameters
true_alpha = [0.5, -0.5, 0.0]
true_beta = [1.0, 0.5, 0.0]
true_kappa = 10.0

# Linear predictors and Observations
Y = zeros(N, 3)
for i in 1:N
    phi1 = true_alpha[1] + true_beta[1] * X[i]
    phi2 = true_alpha[2] + true_beta[2] * X[i]
    phi3 = 0.0

    # Softmax
    e = exp.([phi1, phi2, phi3])
      = e / sum(e)

    Y[i, :] = rand(Dirichlet(  * true_kappa))
end

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

```
@BF function model(X, Y)
    alpha = m.dist.normal(0, 5, shape=(2,), name="alpha")
    beta = m.dist.normal(0, 5, shape=(2,), name="beta")
    kappa = m.dist.exponential(1.0, name="kappa")

    # Linear predictors
    s1 = alpha[1] .+ beta[1] .* X
    s2 = alpha[2] .+ beta[2] .* X
    s3 = jnp.zeros_like(s1)

    phi = jnp.stack([s1, s2, s3], axis=1)
    theta = jax.nn.softmax(phi, axis=1)

    # Likelihood
    m.dist.dirichlet(theta * kappa, obs=Y)
end

m.data_on_model = Dict("X" => X, "Y" => Y)

# Run mcmc ------------------------------------------------
m.fit(model, progress_bar=false)

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

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:

$$Y_i \sim \text{Dirichlet}(\theta_i \kappa) \theta_i = \text{Softmax}(\phi_i) \phi_{[i,1]} = \alpha_1 + \beta_1 X_i \phi_{[i,2]} = \alpha_2 + \beta_2 X_i \ldots \phi_{[i,k]} = 0 \kappa \sim \text{Exponential}(1) \alpha_k \sim \text{Norma}$$

Where:

- Y_i is the outcome simplex for observation i.
- κ is the concentration parameter, it controls the prior weight on each category.
- 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)