Poisson Model with an Offset

General Principles

When we want to model count data, where the counts are observed over different periods or areas of exposure, we use a *Poisson model with an offset*. This is a type of generalized linear model used for modeling count data and contingency tables.

An *offset* is a predictor variable with a coefficient that is fixed at 1. It is used to account for the "exposure" variable, which represents the opportunity for an event to occur. For instance, if we are counting the number of sick individuals in different cities, the population of each city would be the exposure variable. A city with a larger population is expected to have more sick individuals. The offset accounts for this by essentially modeling the rate of events per unit of exposure.

Considerations

> **Note**
>
> - The dependent variable in a Poisson regression must be a non-negative count.
> - The exposure variable used as an offset cannot contain zeros.
> - A key assumption of the Poisson distribution is that the mean and variance of the count variable are equal. If the variance is greater than the mean, a condition known as overdispersion, a Negative Binomial regression might be more appropriate.
> - The logarithm of the exposure variable is typically used as the offset. This is because Poisson regression models the logarithm of the expected count. By including the log of the exposure as an offset, we are effectively modeling the rate.

Example

Below is an example of code that demonstrates a Bayesian Poisson regression with an offset. The data consists of the number of elephant aggressions (`agressions`), the age of the elephants (`age`), and the number of years they have been observed (`years_obs`). The goal is to model the rate of aggressions per year, accounting for the age of the elephants.

Python

```python
from BayesForge import bf
import jax.numpy as jnp
import numpy as np

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

# Simulated data --------------------------------------------
# Using numpy for simulation to easily set the true values
np.random.seed(42)
N = 100
population = np.random.normal(0, 1, size=N)
cid = np.random.binomial(1, 0.5, size=N)
hours = np.random.uniform(1, 10, size=N)

true_a = np.array([2.5, 3.5])
true_b = np.array([-0.1, 0.2])

l = hours * np.exp(true_a[cid] + true_b[cid] * population)
total_tools = np.random.poisson(l)

print(f"True a: {true_a}")
print(f"True b: {true_b}")

# Model data ------------------------------------------------
def model_offset(cid, population, hours, total_tools):
    a = m.dist.normal(3, 0.5, shape=(2,), name='a')
    b = m.dist.normal(0, 0.2, shape=(2,), name='b')
    l = hours * jnp.exp(a[cid] + b[cid]*population)
    m.dist.poisson(l, obs=total_tools)

m.data_on_model = dict(cid=cid, population=population, hours=hours, total_tools=total_tools)
```

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

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

```
bf v 0.0.48 package loaded

E0526 16:15:52.580885 1059979 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source
E0526 16:15:52.582631 1059979 cuda_dnn.cc:523] Loaded runtime CuDNN library: 9.1.0 but source


jax.local_device_count 32
True a: [2.5 3.5]
True b: [-0.1  0.2]

Summary results:
      mean
a[0]  2.49
a[1]  3.49
b[0] -0.10
b[1]  0.19
```

R

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

# Simulated data -------------------------------------------
set.seed(42)
N = 100
population = rnorm(N, 0, 1)
cid = rbinom(N, 1, 0.5) + 1 # R indices are 1-based
hours = runif(N, 1, 10)

true_a = c(2.5, 3.5)
true_b = c(-0.1, 0.2)
```

```r
l = hours * exp(true_a[cid] + true_b[cid] * population)
total_tools = rpois(N, l)

print(paste("True a:", paste(true_a, collapse=" ")))
print(paste("True b:", paste(true_b, collapse=" ")))

# Define model ------------------------------------------------
model <- function(cid, population, hours, total_tools){
  # Parameter prior distributions
  a = bf.dist.normal(3, 0.5, name='a', shape = c(2))
  b = bf.dist.normal(0, 0.2, name='b', shape = c(2))
  l = hours * jnp$exp(a[cid] + b[cid]*population)
  # Likelihood
  bf.dist.poisson(l, obs=total_tools)
}

m$data_on_model = list(
  cid = as.integer(cid - 1), # Back to 0-based for JAX/BF if needed, check BF convention
  population = population,
  hours = hours,
  total_tools = total_tools
)

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

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

Julia

```julia
using BayesForge
using Random
using Distributions

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

# Simulated data ----------------------------------------------
Random.seed!(42)
```

```julia
N = 100
population = rand(Normal(0, 1), N)
cid = rand(Binomial(1, 0.5), N) .+ 1 # Julia indices are 1-based
hours = rand(Uniform(1, 10), N)

true_a = [2.5, 3.5]
true_b = [-0.1, 0.2]

l = hours .* exp.(true_a[cid] .+ true_b[cid] .* population)
total_tools = [rand(Poisson( )) for   in l]

println("True a: ", true_a)
println("True b: ", true_b)

# Define model ------------------------------------------------
@BF function model(cid, population, hours, total_tools)
    a = m.dist.normal(3, 0.5, shape= (2,), name="a")
    b = m.dist.normal(0, 0.2, shape=(2,), name="b")
    l = hours .* exp.(a[cid] .+ b[cid] .* population)
    m.dist.poisson(l, obs=total_tools)
end

# Pass data to model
m.data_on_model = Dict(
    "cid" => cid .- 1, # Back to 0-based for JAX/BF if needed
    "population" => population,
    "hours" => hours,
    "total_tools" => total_tools
)

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

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

Mathematical Details

Frequentist formulation

We model the relationship between the independent variables (X) and the expected count ( ) using the following equation:

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

Where:

- λ_i is the expected count for observation i.
- α is the intercept term.
- β is the regression coefficient for the independent variable.
- X_i is the value of the independent variable for observation i.
- $\log(\text{exposure}_i)$ is the offset, which is the natural logarithm of the exposure for observation i.

The number of observed counts Y_i is assumed to follow a Poisson distribution with mean λ_i:

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

Bayesian formulation

In the Bayesian framework, we assign prior distributions to the model parameters. The model can be expressed as:

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

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

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

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

Where:

- Y_i is the observed count for observation i.

- λ_i is the expected count.
- α is the intercept with a unit-normal prior.
- β is the slope coefficient with a unit-normal prior.
- X_i is the independent variable.
- $\log(\text{exposure}_i)$ is the offset.

Notes

> **Note**
>
> - The use of an offset is crucial when the goal is to compare rates of events rather than absolute counts.
> - It is a common practice to use the natural logarithm of the exposure variable as the offset.

Reference(s)