Robust Model Selection Using Stan

Georg-August-Universität Göttingen

Joaquin Cavieres

1. Introduction

Sometimes, transforming the response variable allows us to achieve a better fit of a statistical model compared to using it directly. But how can we know if such a transformation leads to a better fit? Well, there are various methods to compare models, for example; If we consider y as our response variable, we can propose a model with a certain likelihood function for fitting the data, and another model that considers $\sqrt{y}$ but uses a different likelihood function (I won't name all the methods available in the literature because this comparison also depends on the methodology we are working under. However, for example, in frequentist statistics, we can find the Akaike Information Criterion (AIC, Akaike, H., 1978) or the Schwarz Information Criterion (SIC)). Thus, through a numerical metric, we can choose which model fits our data better. However, this is not done correctly most of the time.

In this little document, I want to outline the proper way of comparing models when transforming the response variable using Bayesian inference based on their predictive predictive distribution. To accomplish this, we will use Stan (via the 'rstan' package (Stan Development Team, 2024)) and the 'loo' package (Vehtari, A et al., 2024).

2. Data simulation

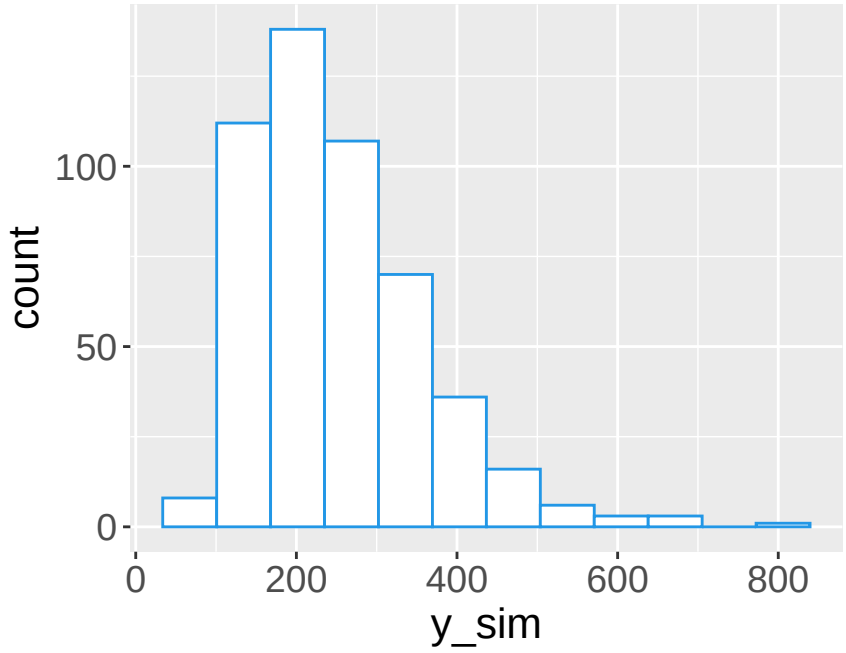
First, let's simulate a dataset, meaning we will simulate a response variable that follows a certain probability distribution (with a clear asymmetry towards the left side of the distribution). The response variable, denoted as y_{sim}, is completely determined by a mean (μ), and this mean will depend on certain parameters such that: $y_{\text{sim}} = \mu = \beta_0 + \beta_1 * x$, where x is a numeric variable simulated from a uniform distribution ($\mathcal{U} \sim (0, 1)$).

The next step is; fit the data (y_{sim}) using a statistical model that incorporates a specific log-likelihood in order to compare its fit with another model that considers the transformation of the simulated variable but uses a different loglikelihood.

To obtain y_{sim}, we will simulate from a Gamma distribution with parameters α (shape) and β (rate), that is:

```
# Data simulated
library(ggplot2)
options(scipen=999) # Scientific notation
set.seed(1)         # Fix the seed
n = 500             # N observations (data)
```

```r
# 'Real' paramaters
beta0 <- 5.0
beta1 <- 1.0
x <- runif(n, 0, 1)
mu <- exp(beta0 + beta1*x) # link function = log
shape <- 10
rate <- shape / mu
y_sim <- rgamma(n, shape = shape, rate = rate)

# Dataset
data <- data.frame(y_sim, x)

# Histogram of the data
ggplot(data, aes(x=y_sim)) +
       geom_histogram(colour = 4, fill = "white", bins = 12) +
       theme(axis.title.x = element_text(size = 16),
       axis.title.y = element_text(size = 16),
       legend.text = element_text(size = 14),
       axis.text = element_text(size = 14))
```



2. Gamma model

Looking the histogram, and assuming we don't know how the data were generated, we could propose a Gamma model to obtain a posterior distribution of the parameters of interest (in this case, β_0, β_1, α). Therefore, we will formulate the following model in Stan considering the following steps:

1) We define the cores of the computer that we want to use to execute the model:

```
#==============================================================================
#                                Stan model
#------------------------------------------------------------------------------
library(rstan)
rstan_options(auto_write = TRUE)
no_cores <- parallelly::availableCores()  - 1
```

2) We propose the model with Gamma loglikelihood:

```
data {
  int<lower=1> N;  // N observations
  vector[N] y;     // Response variable
  vector[N] x;     // Numeric covariate
}

parameters {
  real beta0;
  real beta1;
  real<lower=0> shape;
}

transformed parameters {
    vector[N] mu = rep_vector(0.0, N);
    mu = beta0 + beta1*x;
    mu = exp(mu);
}

model {
    beta0 ~ cauchy(0, 10);
    beta1 ~ cauchy(0, 10);
    shape ~ exponential(2);
    y ~ gamma(shape, shape ./ mu);
}

generated quantities {
  vector[N] y_rep;
  for(n in 1:N){
  y_rep[n] = gamma_rng(shape, shape ./ mu[n]);
 }

  vector[N] log_lik;
  for (n in 1:N) {
    log_lik[n] = gamma_lpdf(y[n] | shape, shape ./ mu[n]);
```

```
  }
}
```

3) We create a data list to be read it by Stan (a c++ template) and execute the model:

```
#==============================================================================
#  List of the simulated data
#------------------------------------------------------------------------------
data_gamma = list(N = length(x),
                  y = data$y_sim,
                  x = data$x)

#==============================================================================
#  Gamma model
#------------------------------------------------------------------------------
startTime <- Sys.time()
fit_gamma <- stan(file='stan_gamma.stan', data = data_gamma,
                  iter = 3000, chains = 3, warmup = 700,
                  control = list(max_treedepth = 12, adapt_delta = 0.9),
                  cores = no_cores, open_progress=FALSE, seed=1234)
endTime <- Sys.time()
timeUsed = endTime - startTime
print(timeUsed)
```

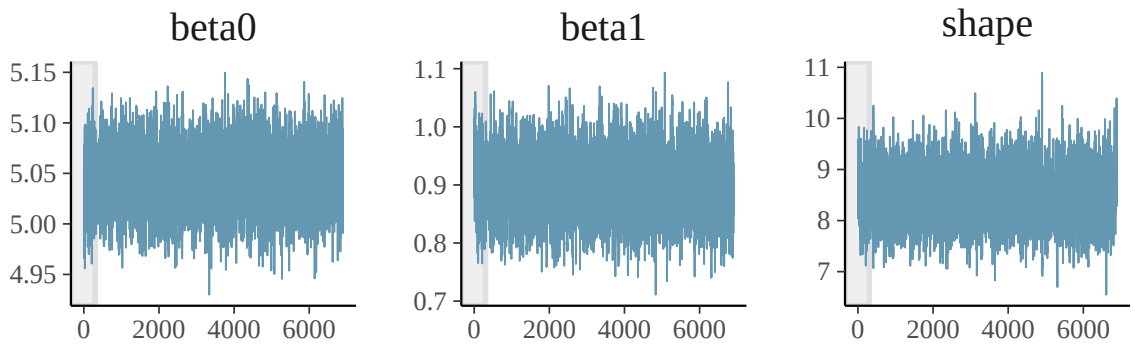
```
## Time difference of 43.99815 secs
```

The `rstan()` function has certain important arguments when using Stan's Hamiltonian Monte Carlo algorithm, such as control parameters like `max_treedepth` o `adapt_delta`, but I won't delve into explaining them here as they are not the main focus of this document. For more information about these control parameters, you can refer to the Stan User's Guide (https://mc-stan.org/docs/stan-users-guide/index.html).

The model converges without problems and the computation time to achieve chain convergence is low, indicating that the model is well parameterized and the fit is appropriate. Now, let's examine the parameter trajectories within the Hamiltonian Monte Carlo and observe how they behave during the fitting process. For this purpose, we will use the 'bayesplot' library (Gabry J and Mahr, T., 2024), which is optimized for models of class `stanfit`.

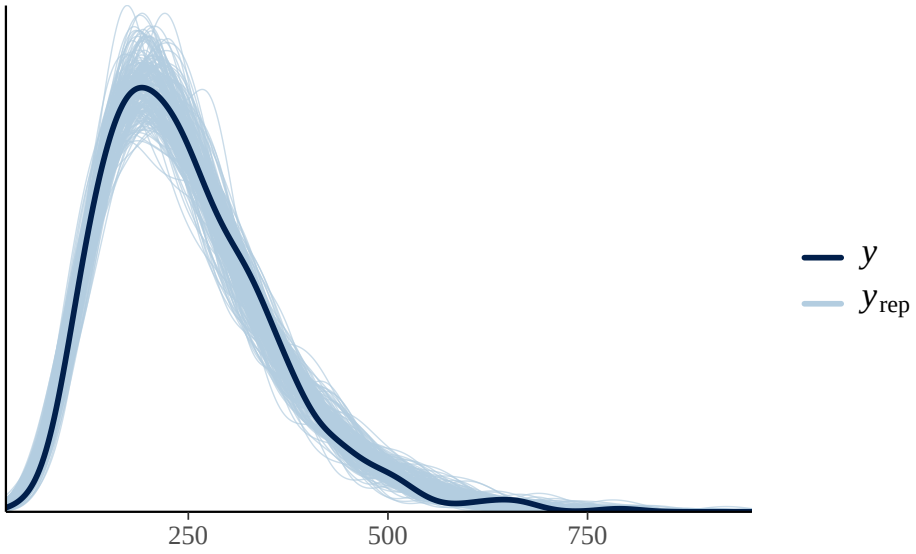
```
library(bayesplot)
color_scheme_set("blue")

posterior_gamma <- as.matrix(fit_gamma)

p <- mcmc_trace(posterior_gamma, pars = c("beta0", "beta1", "shape"),
                n_warmup = 300,
```

```
                facet_args = list(nrow = 1, labeller = label_parsed))
p + facet_text(size = 15)
```



Now, we are not going to compare the estimated parameters of the model, specifically, the posterior means of β_0, β_1, and α, with the initial parameters used to simulate y_{sim}. We will assume that these are well-determined, allowing us to obtain a posterior predictive distribution of the data. To obtain this posterior predictive distribution, we should focus on the `generated quantities {}` block in Stan, which uses the posterior means of the parameters to simulate the response variable. This approach allows us to visualize whether our parameters effectively can represents the response variable based in simulations.

```
y_rep <- as.matrix(fit_gamma, pars = "y_rep")
ppc_dens_overlay(data_gamma$y, y_rep[1:200, ])
```



Looking the plot, we can appropriately assume that the parameters were estimated correctly by the model and that the Gamma loglikelihood assumed within the Bayesian model accurately represents

our data. This is what we would have expected initially since y_{sim} was generated by simulating a random variable with a Gamma distribution.

Note: I'm unable to provide detailed information on how to generate this posterior predictive distribution in the Stan, so I will assume that you can infer it from the link I provided earlier. Again, the purpose of this document is to demonstrate how to properly compare models based on the posterior distributions of the parameters.

3. Gaussian model

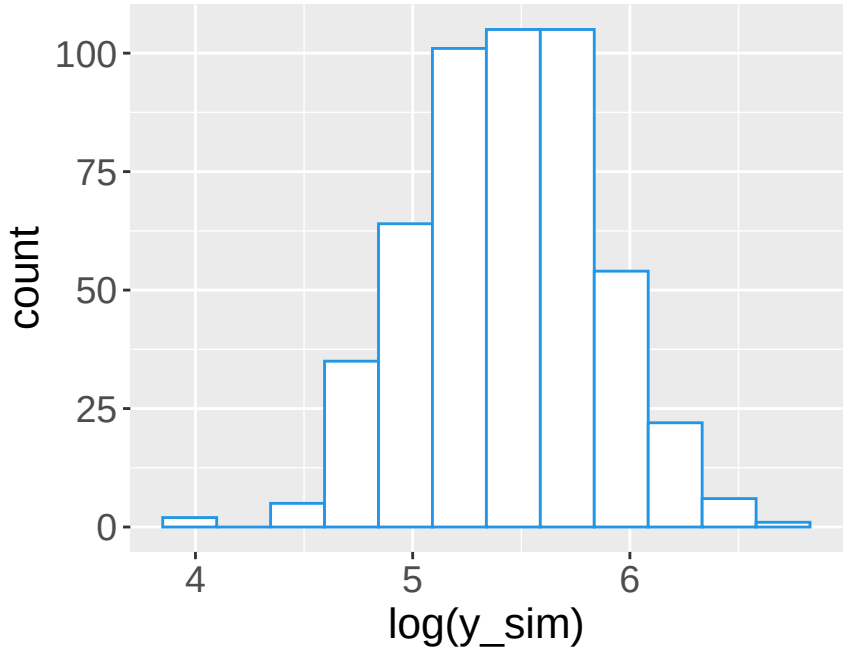
Alright, we now have a model that allows us to perform Bayesian inference using the MCMC method (via the Hamiltonian Monte Carlo algorithm). As an additional note, I'm including here some further readings that will help you better understand how to use the MCMC method for Bayesian inference and the Stan software. These two books are comprehensive and cover both theory and practical aspects, allowing you to deepen more about the topics seen (very briefly) in this document.

- Bayesian data analysis: http://www.stat.columbia.edu/~gelman/book/
- Statistical Rethinking: https://xcelab.net/rm/statistical-rethinking/

Let's continue with the analysis. To do this, we must ask ourselves the following question: What would happen if we transform the response variable? Could my model fit the data better?

Transform the response variable is a commonly used technique by different users, and the most prominent transformation is the logarithmic transformation (log()). This transformation allows us to correct the asymmetry of a variable, making it very useful for obtaining a new variable that follows a Gaussian distribution. Great, let's see how our new transformed variable looks, that is: $\log(y_{\text{sim}})$

```
ggplot(data, aes(x=log(y_sim))) +
  geom_histogram(colour = 4, fill = "white", bins = 12) +
  theme(axis.title.x = element_text(size = 16),
        axis.title.y = element_text(size = 16),
        legend.text = element_text(size = 14),
        axis.text = element_text(size = 14))
```



The distribution of this new variable seems to follow a Gaussian distribution, doesn't it? So let's give it a try! The interesting aspect of modelling a variable with a Gaussian distribution is that the interpretation of the model becomes much simpler since its structure is governed by the parameters μ and σ (it's hard to find someone unfamiliar with the concepts of mean and standard deviation in the world of data analysis). Additionally, it is straightforward to explain, and the computational time for a Bayesian model is low. Given these reasons, many users transform the response variable (and, well, there are also many other advantages such as the diagnostics associated with the normal model, etc.). So, let's see how well a model with a Gaussian loglikelihood fits $\log(y_{\text{sim}})$ in Stan:

```
data {
      int<lower=0> N;
      vector[N] y;
      vector[N] x;
}

parameters {
  real beta0;
  real beta1;
  real<lower=0> sigma;
}

transformed parameters {
  vector[N] mu;
  mu = beta0 + beta1*x;
}

model {
```

```
// Priors
  beta0 ~ normal(0,10);
  beta1 ~ normal(0,10);
  sigma  ~ cauchy(0, 5);

//Likelihood
y ~ normal(mu, sigma);
}

generated quantities {
  vector[N] y_rep;
  for(n in 1:N){
  y_rep[n] = normal_rng(mu[n], sigma);
 }

  vector[N] log_lik;
  for (n in 1:N) {
    log_lik[n] = normal_lpdf(y[n] | mu[n], sigma);
  }
}
```

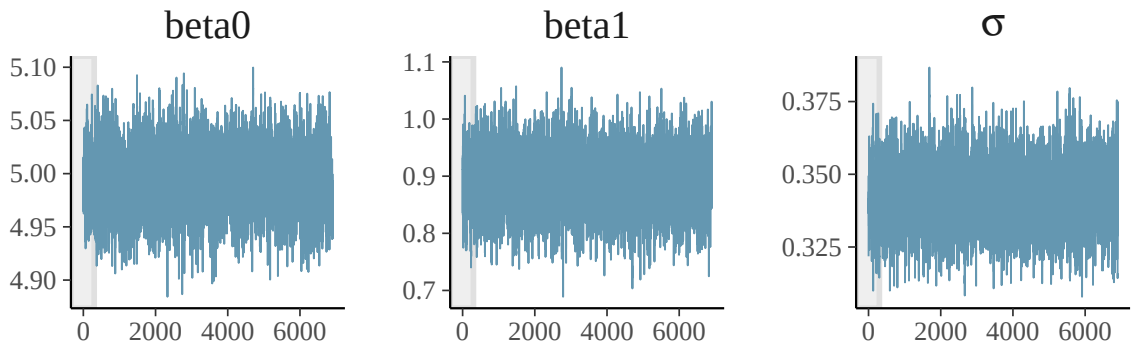
We have to construct a data list again, but this time considering the transformed response variable ($\log(y_{\text{sim}})$) for the new Stan model, that is:

```
#=============================================================================
#  List of the simulated data
#-----------------------------------------------------------------------------
data_gaussian = list(N = length(data$x),
                     y = log(data$y_sim), # Transformed variable
                     x = data$x)


#=============================================================================
# Gaussian model
#-----------------------------------------------------------------------------
startTime <- Sys.time()
fit_gaussian <- stan(file='stan_gaussian.stan', data = data_gaussian,
                 iter = 3000, chains = 3, warmup = 700,
                 control = list(max_treedepth = 12, adapt_delta = 0.9),
                 cores = no_cores, open_progress=FALSE, seed=483892929)
endTime <- Sys.time()
timeUsed = endTime - startTime
print(timeUsed)
```

```
## Time difference of 33.59732 secs
```

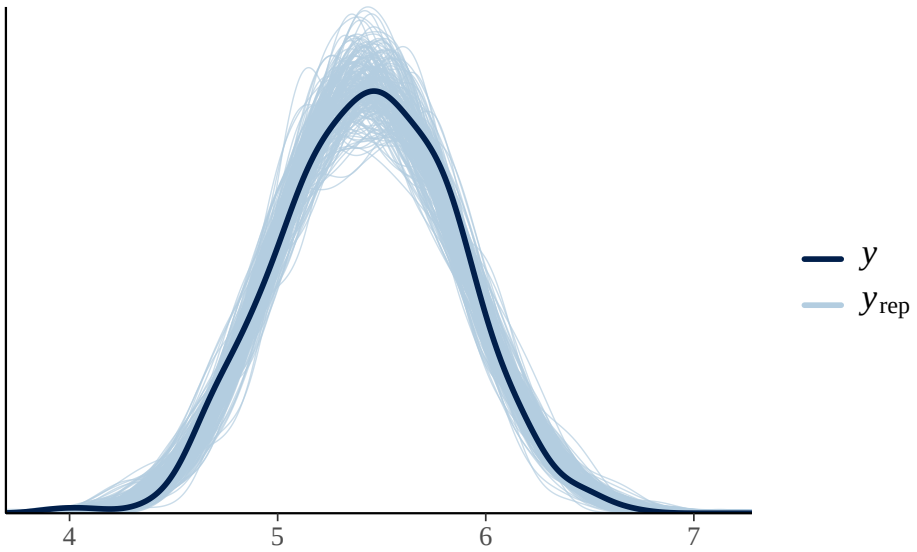
This model also converges, and its computation speed (in time) is faster than the Gamma model (the difference is not much better for this particular case, but believe me, when fitting models with complex structures such as incorporating more predictors, spatial or temporal effects into the model, computation becomes very demanding). Now, let's see how the parameter trajectories behave within the MCMC:

```
posterior_gaussian <- as.matrix(fit_gaussian)

p <- mcmc_trace(posterior_gaussian, pars = c("beta0", "beta1", "sigma"),
                n_warmup = 300,
                facet_args = list(nrow = 1, labeller = label_parsed))
p + facet_text(size = 15)
```



The chains are well mixed and remain stable throughout the iterative process, similar to the Gamma model. Finally, let's assess its predictive performance:

```
y_rep <- as.matrix(fit_gaussian, pars = "y_rep")
ppc_dens_overlay(data_gaussian$y, y_rep[1:200, ])
```



The previous figure indicates that the Gaussian model, based on the posterior distributions of the estimated parameters, represents the structure of the response variable across all simulations and meets all requirements to be considered a good model. So the question that now arises is, which of the two models fits the data better and has better predictive performance?

4. Leave-one-out cross-validation (loo-cv)

We are in the final phase of comparing our models, and for this purpose, we will use the 'loo' library. This library allows us to perform cross-validation for predictive purposes by removing one observation from the dataset, then re-fitting the model without that particular observation. Subsequently, the removed observation is used to test whether the model predicts that value well or not based on the posterior distribution of the model parameters.

In order to compare different models fitted with Stan (`rstan()`) based on their predictive performance, it is necessary to incorporate within the `generated guantities{}` block a vector named `log_lik`, which can represent a log predictive density function for continuous functions, or a log predictive mass function for discrete probability functions. These quantities are then extracted using the `extract_log_lik()` function for subsequent analysis and simulations.

Taking the above into consideration, now we can compare our models using the `loo_compare()` function like this:

```
library(loo)
loglik_gamma    <- extract_log_lik(fit_gamma,    merge_chains = FALSE)
loglik_gaussian <- extract_log_lik(fit_gaussian, merge_chains = FALSE)

loo_gamma <- loo(loglik_gamma, cores = 2)
print(loo_gamma)
```

```
##
## Computed from 6900 by 500 log-likelihood matrix.
##
##          Estimate   SE
## elpd_loo  -2887.5 16.4
## p_loo         2.7  0.3
## looic      5775.1 32.7
## ------
## MCSE of elpd_loo is 0.0.
## MCSE and ESS estimates assume independent draws (r_eff=1).
##
## All Pareto k estimates are good (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
loo_gaussian <- loo(loglik_gaussian, cores = 2)
print(loo_gaussian)
```

```
##
```

```
## Computed from 6900 by 500 log-likelihood matrix.
##
##          Estimate   SE
## elpd_loo   -174.2 16.0
## p_loo         2.9  0.3
## looic       348.3 32.0
## ------
## MCSE of elpd_loo is 0.0.
## MCSE and ESS estimates assume independent draws (r_eff=1).
##
## All Pareto k estimates are good (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
# Comparando los modelos
comp <- loo_compare(loo_gamma, loo_gaussian)
comp
```

```
##        elpd_diff se_diff
## model2     0.0       0.0
## model1 -2713.4      10.1
```

Here, the `comp` object contains the difference in expected log predictive densities between the two models considering their predictive performance. In simple terms, the model appearing in the first row is the 'better model' based on its predictive power.

The results indicate that the Gaussian model fits our data better and has greater predictive capacity than the Gamma model, as the difference in expected log-predictive distribution (`el_pdf`) is 0 when comparing the Gaussian model with itself, and -2713.5 when comparing it with the Gamma model. With this result, we would be confident in selecting the Gaussian model as our 'best model,' and the transformation helped us obtain a posterior distribution of parameters that allows us to predict our response variable. However, the comparison is not complete.

5. Jacobian correction

Finally, we have reached the point where we want to demonstrate how to correctly compare the models. For example, if we denote our response variable as X and assume it comes from a certain probability distribution function $f_X(x)$ for $x \in \mathbb{R}X$, and w is a one-to-one function for $x \in \mathbb{R}X$, then a random variable $Y = w(X)$ has a probability distribution function:

$$f_Y(y) = f_X(x) \left| \frac{dx}{dy} \right|, \tag{1}$$

where $f_X(x) \left| \frac{dx}{dy} \right|$ is expressed in terms of y. From the above equation, the expression $\left| \frac{dx}{dy} \right|$ is called the 'Jacobian transformation' or 'Jacobian adjustment'. Therefore, when comparing two (or more) models, we must consider this correction when transforming our response variable

Given the above, let's do the calculations to strengthen our comparison. The response variable is y_{sim}, which is modeled using a Gamma distribution. On the other hand, $\log(y_{\text{sim}})$ is modeled

using a Gaussian distribution. Following the notation of the Jacobian, let's denote $Y = y_{\text{sim}}$ and $X = \log(y_{\text{sim}})$.

Now we have to solve $\left|\frac{dx}{dy}\right|$:

$$f_Y(y) = f_X(x) \cdot \left|\frac{\partial x(y)}{\partial y}\right| = f_X(x) \cdot \exp(y) \tag{2}$$

Since in Stan we have to declare the log-density, then we apply log to the equation (1) such that:

$$\log(f_Y(y)) = \log(f_X(exp(y))) + y, \tag{3}$$

but y here is $\log(y)$, hence the expression is:

$$\log(f_y) = \log(f_x) + \log(\text{y\_sim}) \tag{4}$$

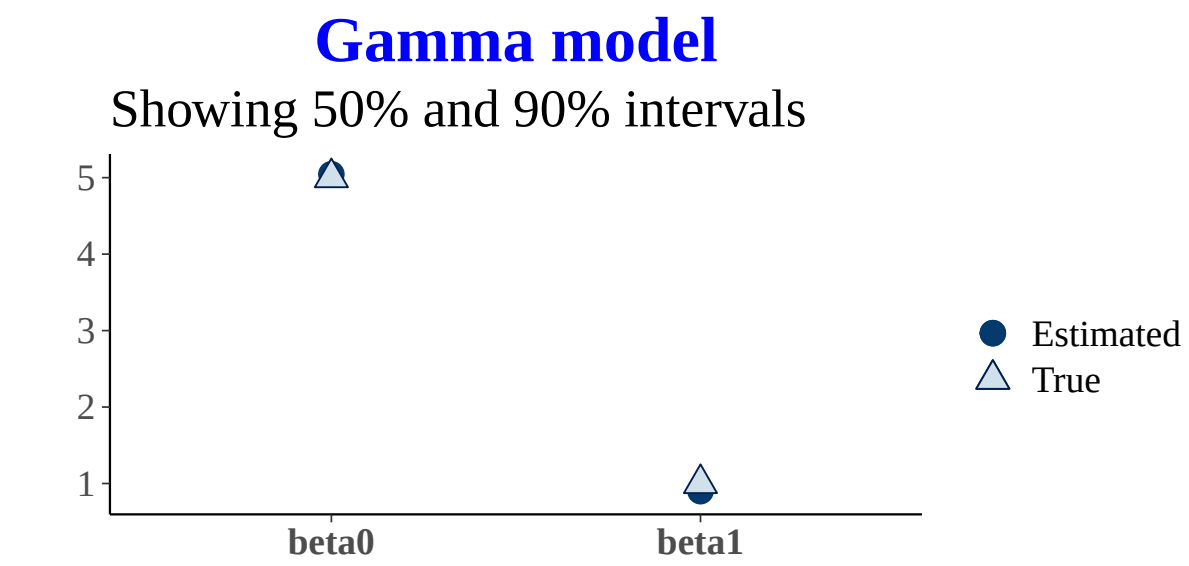
So, from the previous expression, we need to remove log(y_sim) from the right-hand side in order to directly compare $\log(f_y)$ and $\log(f_x)$. Finally, the correct way to perform the comparison is:

```
loo_gaussian_jacobian <- loo_gaussian
loo_gaussian_jacobian$pointwise[,1] <- loo_gaussian_jacobian$pointwise[,1] - log(data$y)
comp2 <- loo_compare(loo_gamma, loo_gaussian_jacobian)
print(comp2, digits = 2)
```

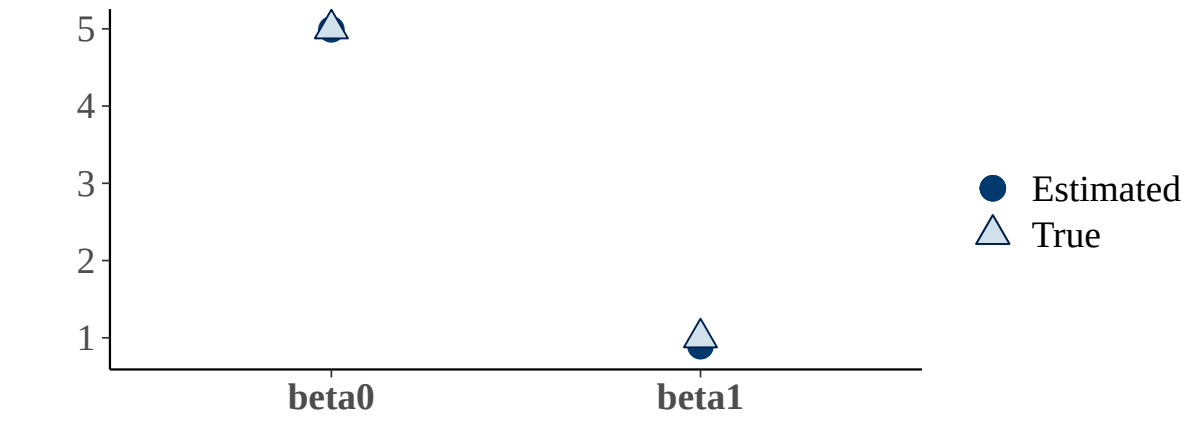
```
##        elpd_diff se_diff
## model2 0.00      0.00
## model1 3.75      3.17
```

Now the comparison is done correctly, and the differences between models (based on their predictive performance) are much smaller. Finally, let's look at the parameter estimates β_0 and β_1 for both models.

```
library(gridExtra)
true_values <- c(beta0, beta1)
posterior_gamma <- as.matrix(fit_gamma)
mcmc_recover_intervals(posterior_gamma[, 1:2], true_values) +
                       ggtitle("Gamma model") +
        theme(plot.title = element_text(color = "blue", size = 24,
        face = "bold", hjust = 0.5),
        axis.title.x = element_text(size = 16),
        axis.title.y = element_text(size = 16),
        legend.text = element_text(size = 14),
        axis.text = element_text(size = 14),
        plot.subtitle = element_text(size = 20))
```




```
posterior_gaussian <- as.matrix(fit_gaussian)

mcmc_recover_intervals(posterior_gaussian[, 1:2], true_values) +
                       ggtitle("Gaussian model") +
        theme(plot.title = element_text(color = "blue", size = 24,
        face = "bold", hjust = 0.5),
        axis.title.x = element_text(size = 16),
        axis.title.y = element_text(size = 16),
        legend.text = element_text(size = 14),
        axis.text = element_text(size = 14),
        plot.subtitle = element_text(size = 20))
```



We can observe that the posterior means of the parameters governing the mean structure of the

model (β_0 and β_1) are estimated properly by both models, which is also demonstrated in the posterior predictive distribution for each of them presented earlier.

Update

To compare more than two models, please consider read the following link:

- https://discourse.mc-stan.org/t/is-it-possible-to-get-positive-and-negative-values-from-the-loo-compare-function/34690

6. Conclusion

Representing various phenomena using statistical models is a multifactorial problem. For example, it requires having a dataset that accurately approximates the behavior of the phenomenon and a set of variables that capture its variability. Once different models have been proposed to fit the observed data, the next task is to compare which one is the 'best' in terms of fit and predictive capacity.

In this document, we presented the correct way to compare two Bayesian models through their posterior predictive distribution. Using the 'loo' library and the `loo_compare` function, we determined that the Gaussian model exhibits better predictive power than the Gamma model. However, direct comparison between the two models is not appropriate because the normal model involves the transformation of y_{sim} to $\log(y_{\text{sim}})$. The correct comparison should be made by applying the Jacobian correction, where, in this particular case, $\log(y_{\text{sim}})$ should be removed from the loglikelihood of the Gaussian model. By making this adjustment, the differences between models decrease significantly, allowing both models to be used for predictive purposes

References

- Akaike, H. (1978). On the likelihood of a time series model. Journal of the Royal Statistical Society: Series D (The Statistician), 27(3-4), 217-235.
- Burnham, K. P., & Anderson, D. R. (2002). Model selection and multimodel inference: a practical information-theoretic approach. New York, NY: Springer New York.
- Stan Development Team (2024). "RStan: the R interface to Stan." R package version 2.32.6, https://mc-stan.org/.
- Vehtari, A., Gabry, J., Magnusson, M., Yao, Y., Bürkner, P. C., Paananen, T., Gelman, A. (2020). loo: Efficient leave-one-out cross-validation and WAIC for Bayesian models. R package version, 2(1), 12. https://mc-stan.org/loo/
- Gabry J, Mahr T (2024). "bayesplot: Plotting for Bayesian Models." R package version 1.11.1, https://mc-stan.org/bayesplot/