Bayesian Modeling and Inference: An Introduction to STAN for the Social Sciences

Practical Tutorial: Bayesian Model Comparison with Stan

May 23, 2025

Contents

Objective

In this tutorial, we will:

- Understand the purpose of Bayesian model comparison.
- Learn about key metrics: PSIS-LOO and WAIC.
- Fit multiple competing models to the same dataset.
- Use the `loo` package in R to calculate and compare metrics.
- Interpret the results and discuss model selection strategies.

Prerequisites

- Completion of the "Bayesian Linear Regression with Stan" tutorial.
- R and RStudio with the following packages installed: `rstan`, `bayesplot`, `ggplot2`, `dplyr`, `loo`.
- Conceptual understanding of Bayesian inference, MCMC, and model selection principles.

Install packages if needed:

```r
install.packages(c("rstan", "bayesplot", "ggplot2", "dplyr", "loo"))
```

Section 1: Setting Up and Simulating Data

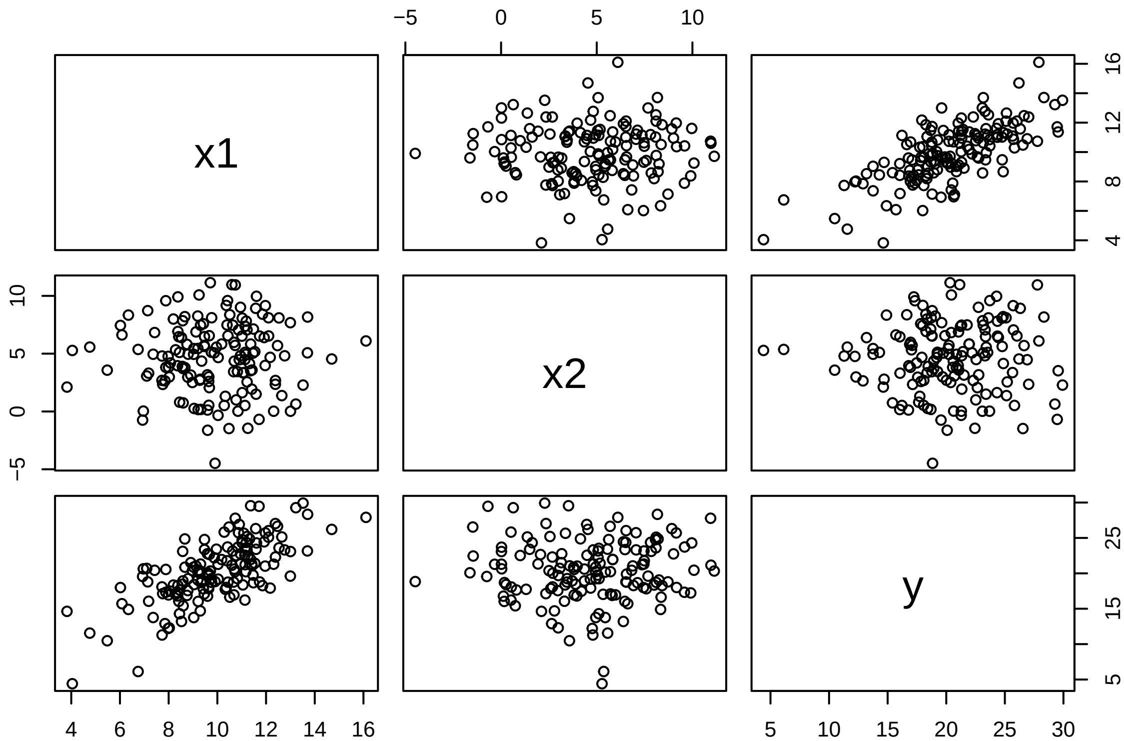
```r
library(rstan)
library(bayesplot)
library(ggplot2)
library(dplyr)
library(loo)

rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

```r
set.seed(789)
N_mc <- 150
alpha_true_mc <- 5
beta1_true_mc <- 1.5
beta2_true_mc <- 0.0
sigma_true_mc <- 3

x1_mc <- rnorm(N_mc, 10, 2)
x2_mc <- rnorm(N_mc, 5, 3)
y_mc <- rnorm(N_mc, alpha_true_mc + beta1_true_mc * x1_mc, sigma_true_mc)

sim_data_mc <- data.frame(x1 = x1_mc, x2 = x2_mc, y = y_mc)
pairs(sim_data_mc)
```



Section 2: Stan Models

Save the following Stan code in your working directory as two separate files:

`linear_model_x1.stan`

```
data {
  int<lower=0> N;
  vector[N] x1;
  vector[N] y;
}
parameters {
  real alpha;
  real beta1;
  real<lower=0> sigma;
}
model {
  alpha ~ normal(0, 50);
  beta1 ~ normal(0, 10);
  sigma ~ cauchy(0, 10);
  y ~ normal(alpha + beta1 * x1, sigma);
}
```

```
generated quantities {
  vector[N] log_lik;
  vector[N] y_rep;
  for (n in 1:N) {
    real mu_n = alpha + beta1 * x1[n];
    log_lik[n] = normal_lpdf(y[n] | mu_n, sigma);
    y_rep[n] = normal_rng(mu_n, sigma);
  }
}
```

linear_model_x1_x2.stan

```
data {
  int<lower=0> N;
  vector[N] x1;
  vector[N] x2;
  vector[N] y;
}
parameters {
  real alpha;
  real beta1;
  real beta2;
  real<lower=0> sigma;
}
model {
  alpha ~ normal(0, 50);
  beta1 ~ normal(0, 10);
  beta2 ~ normal(0, 10);
  sigma ~ cauchy(0, 10);
  y ~ normal(alpha + beta1 * x1 + beta2 * x2, sigma);
}
generated quantities {
  vector[N] log_lik;
  vector[N] y_rep;
  for (n in 1:N) {
    real mu_n = alpha + beta1 * x1[n] + beta2 * x2[n];
    log_lik[n] = normal_lpdf(y[n] | mu_n, sigma);
    y_rep[n] = normal_rng(mu_n, sigma);
  }
}
```

Section 3: Fitting the Models

```
setwd("/Users/user/Desktop/Lectures 2024/Bayesian Course - UoM/Bayesian Linear Regression")
stan_data_m1 <- list(N = N_mc, x1 = sim_data_mc$x1, y = sim_data_mc$y)
model_m1_compiled <- stan_model(file = "linear_model_x1.stan")
fit_m1 <- sampling(model_m1_compiled, data = stan_data_m1, iter = 2000, warmup = 1000, chains = 4, seed

stan_data_m2 <- list(N = N_mc, x1 = sim_data_mc$x1, x2 = sim_data_mc$x2, y = sim_data_mc$y)
```

```r
model_m2_compiled <- stan_model(file = "linear_model_x1_x2.stan")
fit_m2 <- sampling(model_m2_compiled, data = stan_data_m2, iter = 2000, warmup = 1000, chains = 4, seed
```

```r
print(fit_m1, pars = c("alpha", "beta1", "sigma"), digits = 3)
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##        mean se_mean    sd  2.5%   25%   50%   75% 97.5% n_eff  Rhat
## alpha 4.599   0.037 1.302 2.044 3.734 4.624 5.481 7.038  1261 1.002
## beta1 1.585   0.004 0.128 1.345 1.496 1.581 1.670 1.836  1243 1.001
## sigma 3.067   0.004 0.183 2.727 2.940 3.059 3.185 3.438  1913 1.001
##
## Samples were drawn using NUTS(diag_e) at Mon May 19 15:33:56 2025.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```r
print(fit_m2, pars = c("alpha", "beta1", "beta2", "sigma"), digits = 3)
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##        mean se_mean    sd   2.5%    25%   50%   75% 97.5% n_eff  Rhat
## alpha 4.553   0.029 1.327  1.981  3.649 4.541 5.460 7.150  2061 1.000
## beta1 1.582   0.003 0.127  1.338  1.494 1.585 1.670 1.821  2142 1.000
## beta2 0.017   0.002 0.082 -0.146 -0.036 0.019 0.071 0.179  2769 1.000
## sigma 3.090   0.004 0.183  2.762  2.962 3.079 3.208 3.472  2683 1.001
##
## Samples were drawn using NUTS(diag_e) at Mon May 19 15:33:59 2025.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Section 4: Model Comparison

4.1 Extract Log-Likelihood

```r
log_lik_m1 <- extract_log_lik(fit_m1, parameter_name = "log_lik", merge_chains = TRUE)
log_lik_m2 <- extract_log_lik(fit_m2, parameter_name = "log_lik", merge_chains = TRUE)
dim(log_lik_m1)
```

```
## [1] 4000  150
```

```r
dim(log_lik_m2)
```

```
## [1] 4000  150
```

4.2 PSIS-LOO

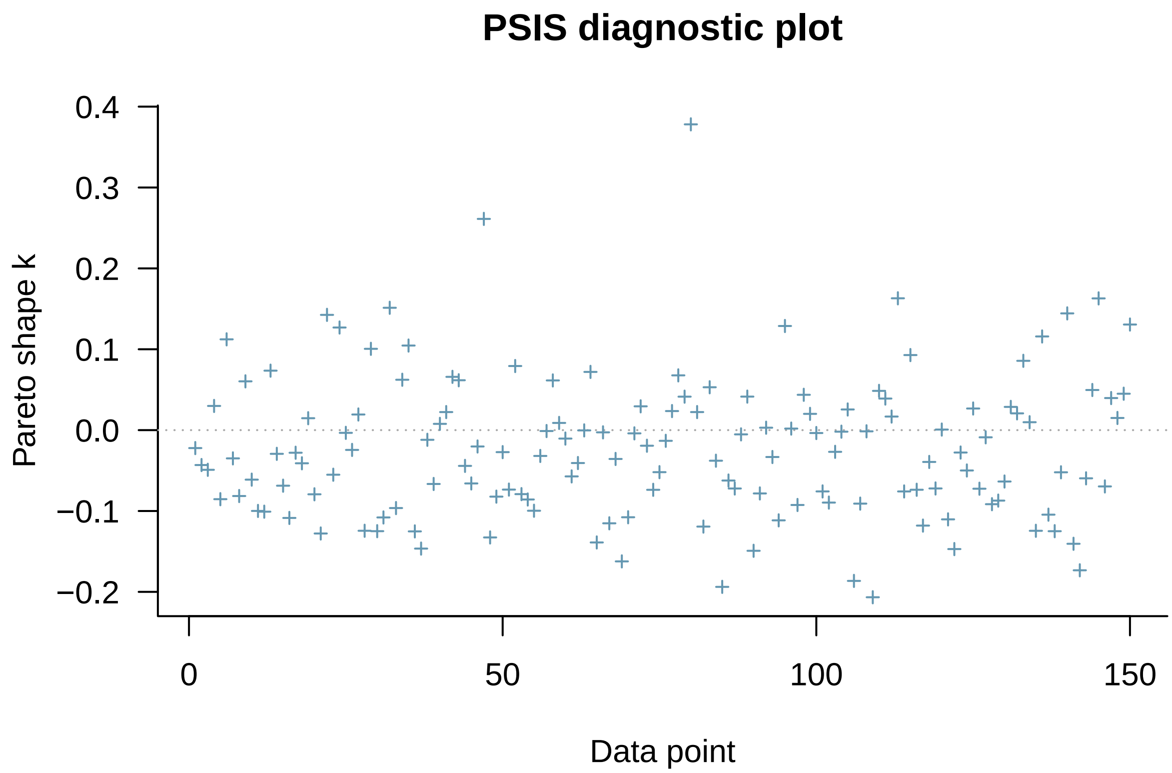
```r
loo_m1 <- loo(log_lik_m1)
loo_m2 <- loo(log_lik_m2)
print(loo_m1)
```

```
##
## Computed from 4000 by 150 log-likelihood matrix.
##
##          Estimate   SE
## elpd_loo   -382.6  8.4
## p_loo         3.3  0.6
## looic       765.2 16.8
## ------
## 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.
```

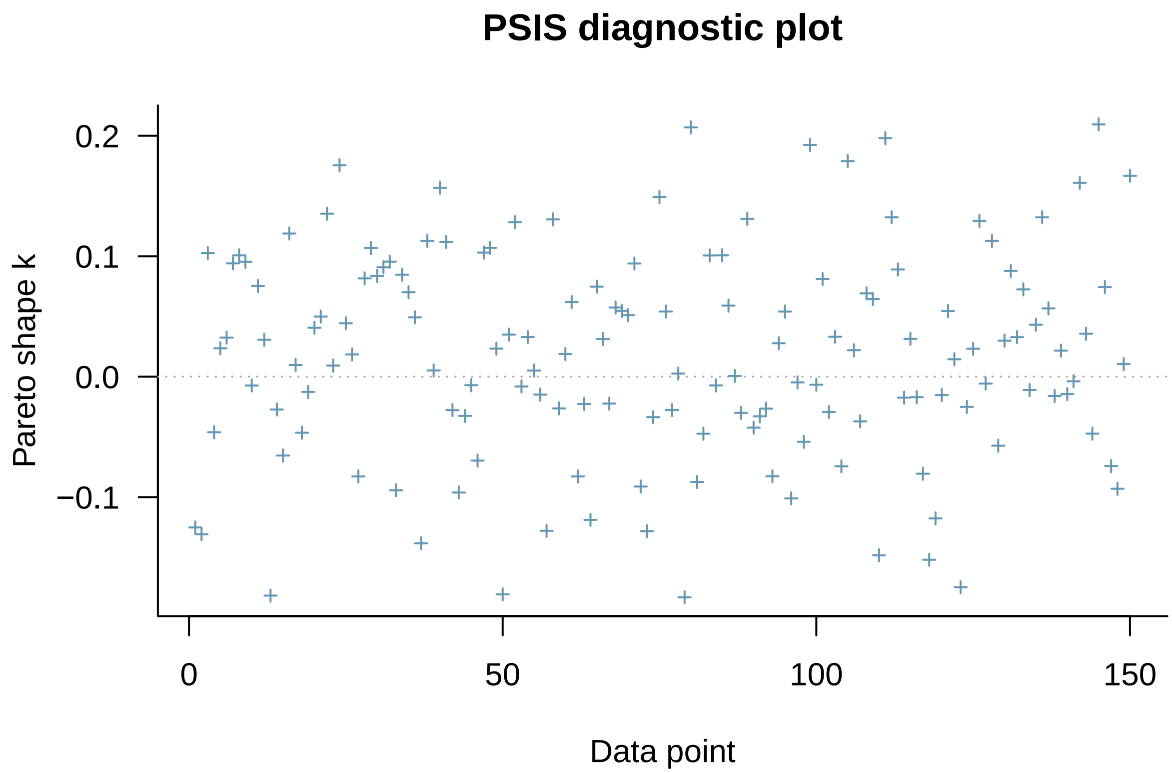
```r
print(loo_m2)
```

```
##
## Computed from 4000 by 150 log-likelihood matrix.
##
##          Estimate   SE
## elpd_loo   -383.4  8.4
## p_loo         4.1  0.7
## looic       766.9 16.7
## ------
## 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.
```

```r
plot(loo_m1, label_points = TRUE)
```




```
plot(loo_m2, label_points = TRUE)
```



4.3 WAIC

```r
waic_m1 <- waic(log_lik_m1)
waic_m2 <- waic(log_lik_m2)
print(waic_m1)
```

```
##
## Computed from 4000 by 150 log-likelihood matrix.
##
##           Estimate   SE
## elpd_waic   -382.6  8.4
## p_waic         3.3  0.6
## waic         765.1 16.8
##
## 1 (0.7%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```r
print(waic_m2)
```

```
##
## Computed from 4000 by 150 log-likelihood matrix.
##
##           Estimate   SE
## elpd_waic   -383.4  8.4
```

```
## p_waic         4.1  0.7
## waic         766.8 16.7
##
## 1 (0.7%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

4.4 Compare Models

```r
comp_loo <- loo_compare(loo_m1, loo_m2)
print(comp_loo)
```

```
##        elpd_diff se_diff
## model1  0.0       0.0
## model2 -0.8       0.2
```

Section 5: Decisions and Considerations

- Prefer simpler models when ELPDs are similar (lower p_loo).
- Combine metrics with PPCs, theory, and domain knowledge.
- Use model averaging if multiple models have similar predictive performance.

Section 6: Student Exploration Questions (Model Comparison)

Q1: Impact of a Truly Useful Predictor

- **Q1.1**: Modify the data simulation so that `beta2_true_mc <- 0.8`. Re-simulate the data.
- **Q1.2**: Re-fit `fit_m1` (using only x1) and `fit_m2` (using x1 and x2) with this new data.
- **Q1.3**: Perform model comparison using `loo_compare(loo_m1, loo_m2)`. Which model is preferred now? How does the `elpd_diff` and `se_diff` support your conclusion?

Q2: Overfitting Demonstration

- **Q2.1**: Create a new Stan model, `linear_model_complex.stan`, that includes x1, x2, and also terms like `x1^2`, `x2^2`, and an interaction `x1*x2`. Example:

```stan
data {
  int<lower=0> N;
  vector[N] x1;
  vector[N] x2;
  vector[N] y;
}
parameters {
  real alpha;
  real beta1;
  real beta2;
  real beta1_sq;
  real beta2_sq;
  real beta_int;
```

```
  real<lower=0> sigma;
}
model {
  alpha ~ normal(0, 50);
  beta1 ~ normal(0, 10);
  beta2 ~ normal(0, 10);
  beta1_sq ~ normal(0, 10);
  beta2_sq ~ normal(0, 10);
  beta_int ~ normal(0, 10);
  sigma ~ cauchy(0, 10);
  vector[N] mu;
  for (n in 1:N)
    mu[n] = alpha + beta1 * x1[n] + beta2 * x2[n] +
            beta1_sq * square(x1[n]) + beta2_sq * square(x2[n]) +
            beta_int * x1[n] * x2[n];
  y ~ normal(mu, sigma);
}
generated quantities {
  vector[N] log_lik;
  for (n in 1:N) {
    real mu_n = alpha + beta1 * x1[n] + beta2 * x2[n] +
                beta1_sq * square(x1[n]) + beta2_sq * square(x2[n]) +
                beta_int * x1[n] * x2[n];
    log_lik[n] = normal_lpdf(y[n] | mu_n, sigma);
  }
}
```

- **Q2.2**: Fit this complex model (`fit_m_complex`) using the original data.
- **Q2.3**: Compare `fit_m1`, `fit_m2`, and `fit_m_complex` using `loo_compare()`. How does `p_loo` for the complex model compare to the others? Does the complex model show better `elpd` despite many irrelevant terms?

Q3: Interpreting p_loo

- **Q3.1**: For `fit_m1` and `fit_m2` using the original data where x2 is irrelevant, inspect their `p_loo` values.
- How many parameters are in the Stan block for model 1? (alpha, beta1, sigma = 3)
- How many for model 2? (alpha, beta1, beta2, sigma = 4)
- Are the `p_loo` values roughly similar? When can `p_loo` deviate significantly from this count?

Q4: Model Comparison for Logistic Regression (Conceptual)

- **Q4.1**: How would you adapt this model comparison framework for logistic regression?
  - Modify the generated quantities block using `bernoulli_logit_lpmf`.
  - Use the same workflow: extract `log_lik`, apply `loo()` or `waic()`, and compare models with `loo_compare()`.