A Brief Introduction to RStanARM

Imad Ali

3/23/2017

Introduction

This is a brief example of how to use the rstanarm package in the context of logistic regression. See the rstanarm documentation (reference manual and vigenettes) for the most up to date material. We will also use the bayesplot package that provides additional plotting functionality that users can apply to rstan/rstanarm objects, and the loo package that provides a way to perform model selection.

Although we will focus on the `stan_glm` function to fit models it is useful to be aware of the other models that can be fit using rstanarm. Below we list the function and a very brief "story" about the data which identifies why you would use that particular function to model the data.

- `stan_lm`
    - $y \in \mathbb{R}$
- •
- Useful for modeling a continuous outcome that is linear in terms of parmeters.

- `stan_glm`
    - $y \in \{0, n\}$

where $n \in \mathbb{N}$

- •
- Useful for modeling bernoulli trials (`family=binomial(link = "logit")`).
- Useful for modeling count data (`family="poission"` or `family="neg_binomial_2`).

- `stan_glmer`

- y
  -
  - can be continuous or discrete.
  - Useful for modeling hierarchical structure.

- `stan_polr`
  - $y \in \{1, J\}$

where $J \in \mathbb{N}$

  -
  - .
  - Useful for mdoeling ordinal outcomes.

- `stan_betareg`
  - $y \in (0,1)$
    -
    - .
    - Useful for modeling rates/proportions/ratings.

Data Generation Process

Below we generate data to be used in logistic regression. The identifying feature of data that can be fit using logistic regression is that the outcome variable is binary.

```
# the inverse logit function
#' @param x A number fro the real line.
#' @return A probability.
inv_logit <- function(x) {
  return(exp(x)/(1+exp(x)))
}
# (not necessary) function to generate multinormal random number generation
for correlated predictors
#' @param n An integer value indicating the sample size
#' @param mu A vector of means of length K
#' @param sigma A K-by-K covariance matrix
#' @return A N-by-K matrix of multivariate normal random variables.
generate_multinorm_data <- function(n, mu, sigma) {
  if(any((eigen(sigma)$values) < 0))
    stop("\none or more eigenvalues of 'sigma' are negative")
  x <- matrix(rnorm(n * length(mu)), ncol = length(mu), nrow = n)
  sigma_decomp <- svd(sigma)
  u <- sigma_decomp$u
  d <- diag(sigma_decomp$d,length(sigma_decomp$d))
  y <- t(mu + u %*% sqrt(d) %*% t(x))
```

```
  return(y)
}
# function to generate logistic data
#' @param beta A vector of parameter values.
#' @param X A matrix of covariates.
#' @param const Indicate whether a constant should be included in the
#'              data generation process (defaults to TRUE).
#' @return A vector of binary data.
generate_logistic_data <- function(beta, X, cons = TRUE) {
  n <- nrow(X)
  X_orig <- X
  if (cons) {
    X <- cbind(rep(1,n),X)
  }
  prob <- inv_logit(X%*%beta)
  y <- rbinom(n, 1, prob)

  out <- data.frame(cbind(y,X_orig,prob))
  names <- paste0("x", 1:ncol(X_orig))
  colnames(out) <- c("y", names, "prob")

  return(out)
}
# generate data
N <- 300
prior_mu <- c(-0.5, 0.5, 1.5)
prior_sd <- rep(0.8,length(prior_mu))
beta <- c(rnorm(1, prior_mu[1], prior_sd[1]),
          rnorm(1, prior_mu[2], prior_sd[2]),
          rnorm(1, prior_mu[3], prior_sd[3]))
X <- generate_multinorm_data(N, beta[-1], matrix(c(1,0.5,0.5,1), ncol = 2))
dat_glm <- generate_logistic_data(beta, X, cons = TRUE)
# summarize dependent variable
table(dat_glm$y)
##
##   0   1
## 110 190
```

Our model is defined as follows,

$$y \sim Bin(1,\mathbf{p}) \text{logit}(\mathbf{p}) = \mathbf{X}\boldsymbol{\beta} \beta_0 \sim \mathcal{N}(-0.5,0.5) \beta_1 \sim \mathcal{N}(0.5,0.25) \beta_2 \sim \mathcal{N}(1.0,0.25)$$

Where $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$

. Note that the logit function $\log\left(\frac{p}{1-p}\right)$ maps from the closed unit interval $[0,1]$ to the real line $\mathbb{R}$. So we can use the inverse of the logit function to map our linear predictor $\mathbf{X}\boldsymbol{\beta}$ to the closed unit interval and then characterize these values as probabilities which can be passed into the binomial distribution with size $n = 1$

.

Model Fitting

```
library(rstanarm)
# fit model using mcmc
fit1 <- stan_glm(y ~ x1 + x2, data = dat_glm, family = binomial(link =
"logit"), cores = 4,
                 prior_intercept = normal(prior_mu[1], prior_sd[1]), prior =
normal(prior_mu[-1], 0.5*prior_sd[-1]))
cbind("stan_glm" = coef(fit1),  # model fit with mcmc
      "beta" = beta,            # the parameter realized from the prior
distributions
      "beta_mu" = prior_mu)     # the prior distribution location parameters
##                stan_glm       beta beta_mu
## (Intercept) -0.6613041 -0.9392810    -0.5
## x1          -0.6930683 -0.8349270     0.5
## x2           0.7392915  0.8497792     1.5
```

We can take a look at the priors actually used in the model with the `prior_summary()` function.

```
prior_summary(fit1)
## Priors for model 'fit1'
## ------
## Intercept (after predictors centered)
##  ~ normal(location = -0.5, scale = 0.8)
##
## Coefficients
##  ~ normal(location = [0.5,1.5], scale = [0.4,0.4])
##      **adjusted scale = [0.38,0.38]
## ------
## See help('prior_summary.stanreg') for more details
```

Diagnostics

Diagnostics refers to ensuring that the sampler is performing appropriately. First lets look at the summary output available from the stanreg object.

```
summary(fit1, digits = 2)
##
## Model Info:
##
##  function:  stan_glm
##  family:    binomial [logit]
##  formula:   y ~ x1 + x2
##  algorithm: sampling
##  priors:    see help('prior_summary')
##  sample:    4000 (posterior sample size)
##  num obs:   300
##
## Estimates:
##                mean    sd      2.5%    25%     50%     75%     97.5%
## (Intercept)    -0.67    0.24   -1.14   -0.82   -0.66   -0.51   -0.21
## x1             -0.69    0.14   -0.98   -0.79   -0.69   -0.60   -0.41
## x2              0.74    0.14    0.47    0.64    0.74    0.83    1.03
```

```
## mean_PPD         0.63    0.04    0.55    0.60    0.63    0.65    0.70
## log-posterior -192.55    1.26 -195.83 -193.10 -192.22 -191.64 -191.16
##
## Diagnostics:
##               mcse Rhat n_eff
## (Intercept)   0.00 1.00 2673
## x1            0.00 1.00 2684
## x2            0.00 1.00 2573
## mean_PPD      0.00 1.00 3240
## log-posterior 0.03 1.00 1700
##
## For each parameter, mcse is Monte Carlo standard error, 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).
```

Model Info provides some high-level information about the type of model you have fit, algorithm used, size of the posterior sample (which equals `chains` * (`iter` - `warmup`)), etc.

Estimates provides statistics on the posterior samples for the parameters. Here we have information on,

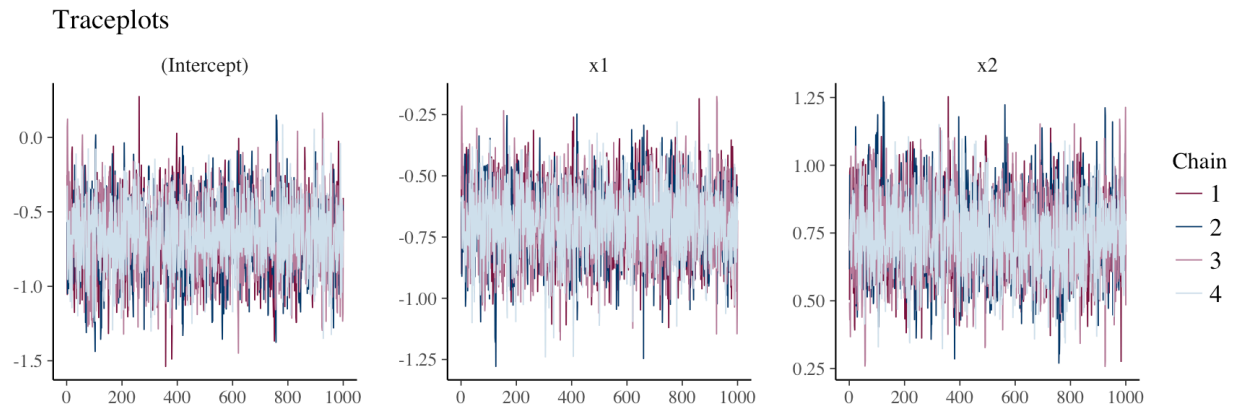
- The **mean** and **standard deviation** of the marginal posterior distribution of each parameter.
- A set of **quantiles**. For example, 50% of the distribution for the (Intercept) parameter lies in the interval [-0.72,-0.37].
- The **mean of the posterior predictive distribution** (`mean_PPD`), which is the mean of the predictions made using the estimated parameters.
- The **log-posterior** is the logarithm of the likelihood times the prior up to some normalizing constant.

Diagnostics provides information about the sampler's performance.

- **Rhat** is a measure of convergence among chains. It is the ratio of the average variance of the draws within each chain to the variance of the pooled draws across chains. If Rhat is 1 then the chains are in equilibirium (i.e. the chains have converged). You should be concerned if Rhat is greater than 1. Sometimes this can be rectified by increasing the number of iterations. Other times it might be an issue with the way your model is identified.
- **n_eff** is a rough measure of effective sample size.
- **mcse** is the "Monte Carlo Standard Error" a measure of inaccuracy of Monte Carlo samples.

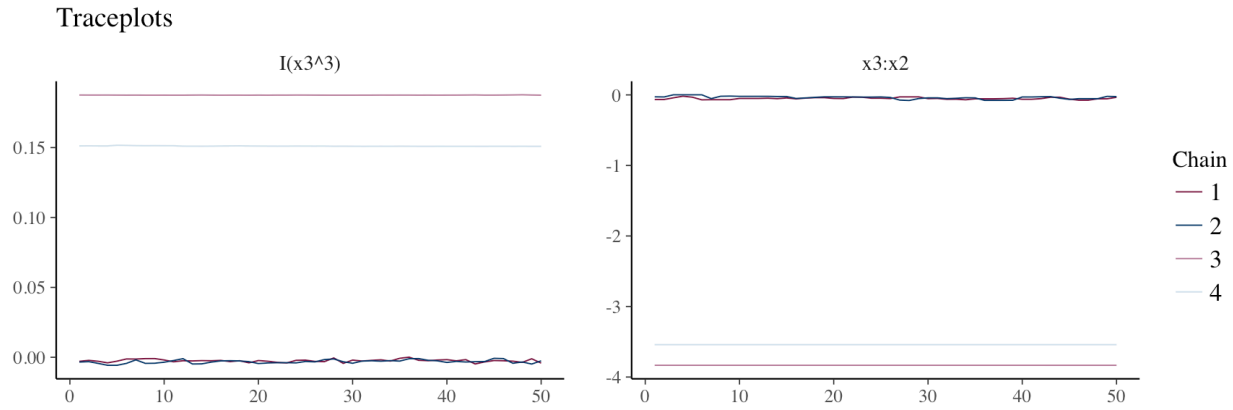
Below is one of the more important diagnostic plots. It is a **traceplot** of each parameter. Notice how, for each parameter, the chains mix well. This is also reflected in the Rhat values equalling one as mentioned above.

```
library(bayesplot)
library(ggplot2)
color_scheme_set("mix-blue-pink")
plot(fit1, plotfun = "trace") + ggtitle("Traceplots")
```



For illustrative purposes we fit an inappropriately specified model below and present the traceplot.

```
dat_glm$x3 <- rnorm(nrow(dat_glm), -5, 0.01)
fit2 <- stan_glm(y ~ I(x3^3) + x3:x2 - 1, data = dat_glm, family =
binomial(link = "logit"), cores = 4, prior = normal(0, 10), iter = 100)
## Warning: There were 94 divergent transitions after warmup. Increasing
adapt_delta above 0.95 may help. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## Warning: Examine the pairs() plot to diagnose sampling problems
## Warning: Markov chains did not converge! Do not analyze results!
plot(fit2, plotfun = "trace") + ggtitle("Traceplots")
```



Notice how there is pretty much no convergence among the chains. A lot of time, especially for complicated models, convergence can be achived by increasing the number of iterations. However, in some cases it may signify that you have a poorly identified model.

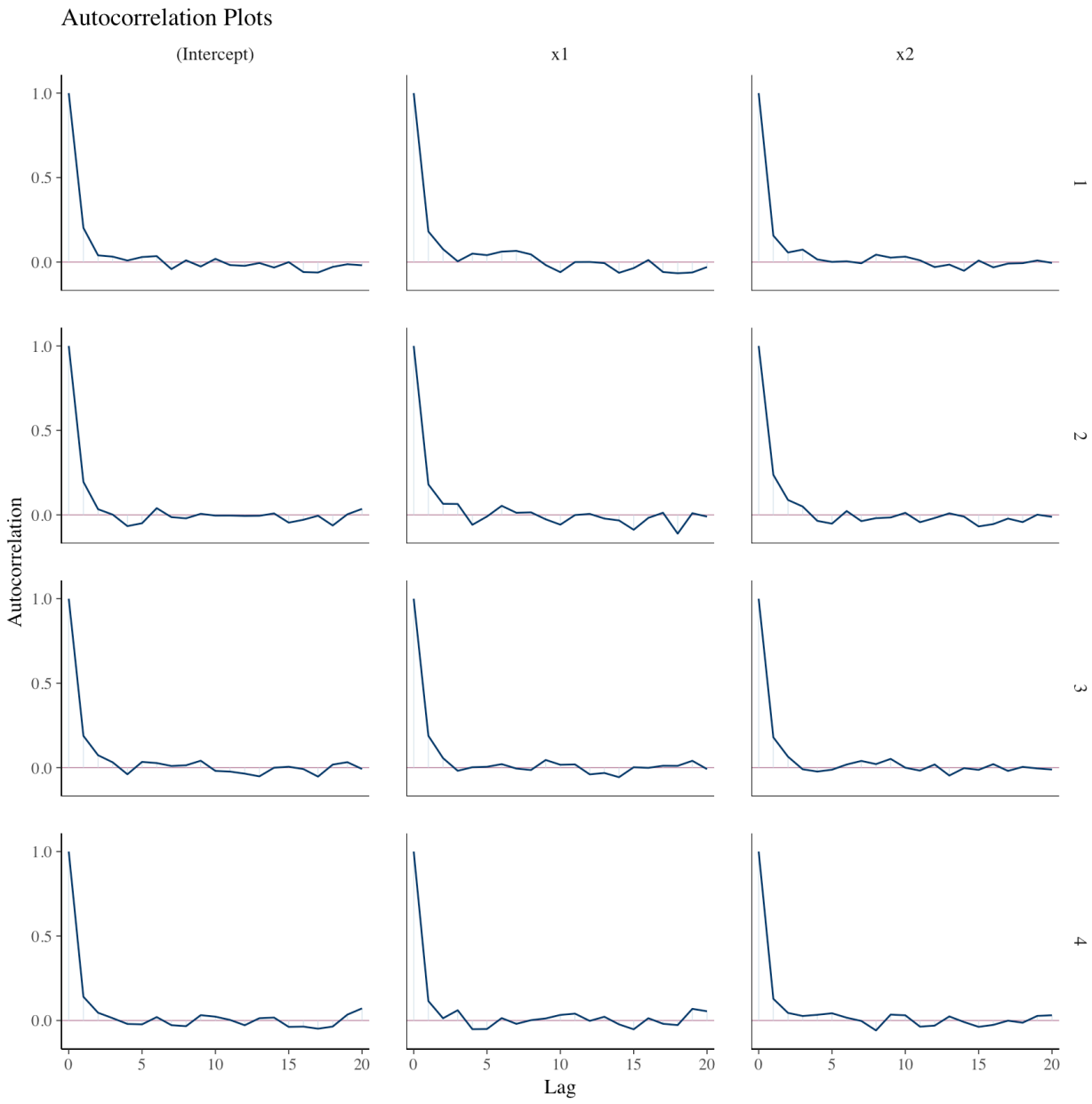
```
summary(fit2)
##
## Model Info:
##
##  function:  stan_glm
##  family:    binomial [logit]
##  formula:   y ~ I(x3^3) + x3:x2 - 1
##  algorithm: sampling
```

```
##  priors:    see help('prior_summary')
##  sample:    200 (posterior sample size)
##  num obs:   300
##
## Estimates:
##                 mean    sd      2.5%    25%     50%     75%     97.5%
## I(x3^3)           0.1     0.1     0.0     0.0     0.1     0.2     0.2
## x3:x2            -1.9     1.8    -3.8    -3.6    -1.8     0.0     0.0
## mean_PPD          0.5     0.1     0.4     0.4     0.5     0.6     0.7
## log-posterior -1206.8  1019.7 -2394.1 -2130.5 -1119.7  -197.1  -196.6
##
## Diagnostics:
##               mcse   Rhat   n_eff
## I(x3^3)          0.1  121.8 2
## x3:x2            1.3  180.5 2
## mean_PPD         0.1    4.3 2
## log-posterior  717.4 1353.7 2
##
## For each parameter, mcse is Monte Carlo standard error, 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).
```

Looking at the summary output notice that we also have high Rhat values and extremely low effective sample sizes for each parameter.

Below is a graph of the **autocorrelation function** for the chain of each parameter. Notice how, as per the definition of Markov Chains, the past parameter values have no influence on future parameter values. Autocorrelation values close to zero ensure that each value obtained along the chain is random.

```
plot(fit1, plotfun = "acf") + ggtitle("Autocorrelation Plots")
```



Posterior Predictive Checks

Our outcome variable is binary so we can look at a contingency table to determine how well our model predicts the outcome. Ideally we want the off diagonals of the 2-by-2 table to be as close to zero as possible (i.e. predicting few false positives and false negatives).

```
# posterior predictive check
yrep_stan <- round(colMeans(posterior_predict(fit1)))
table("stan_glm" = yrep_stan, "y_true" = dat_glm$y)
##          y_true
## stan_glm   0   1
##        0  37  24
```

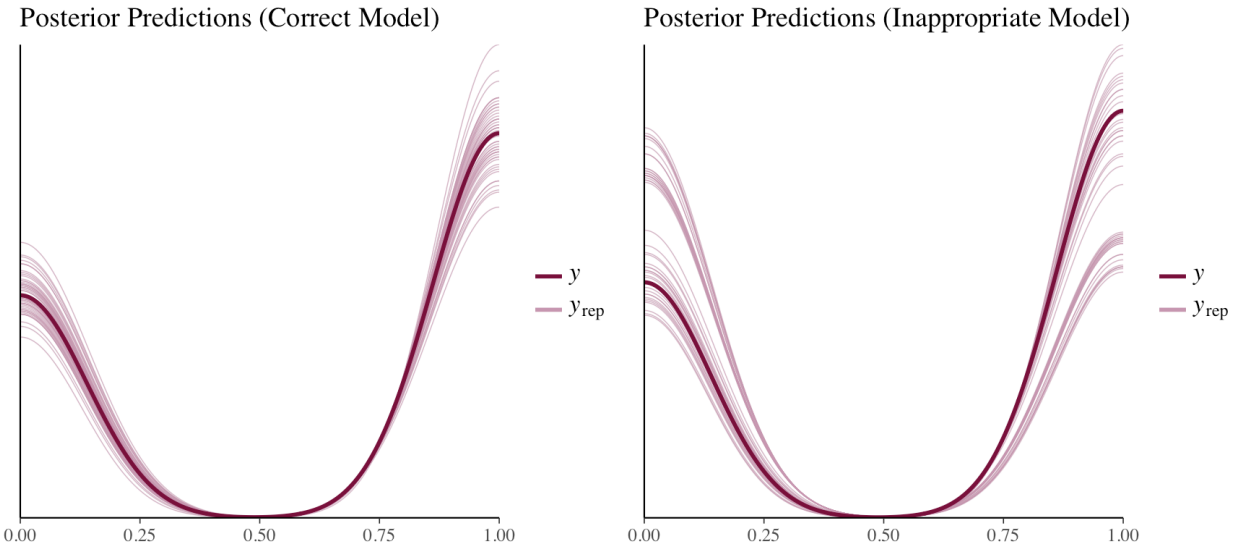
```
##         1  73 166
```

The tables generated above compare posterior predictions of the data used to fit the model with the true outcome. We can also see how the predictions fair when using new data. This is presented in the table below.

```
# how do the predictions compare with new data realized from the prior
predicitive distribution
new_beta <- c(rnorm(1, prior_mu[1], prior_sd[1]),
              rnorm(1, prior_mu[2], prior_sd[2]),
              rnorm(1, prior_mu[3], prior_sd[3]))
new_dat <- generate_logistic_data(new_beta, X, cons = TRUE)
yrep_new_stan <- round(colMeans(posterior_predict(fit1, newdata = new_dat)))
table("stan_glm" = yrep_new_stan, "y_true" = new_dat$y)
##          y_true
## stan_glm   0   1
##        0  29  33
##        1 148  90
```

With the posterior predictive check function `pp_check()` we can look at how well predictions from the posterior distribution match the true outcome variable that was used to fit the data. Notice that the inappropriate model produces poor posterior predictions. (Here we also use the `bayesplot_grid()` function to combine several plots.)

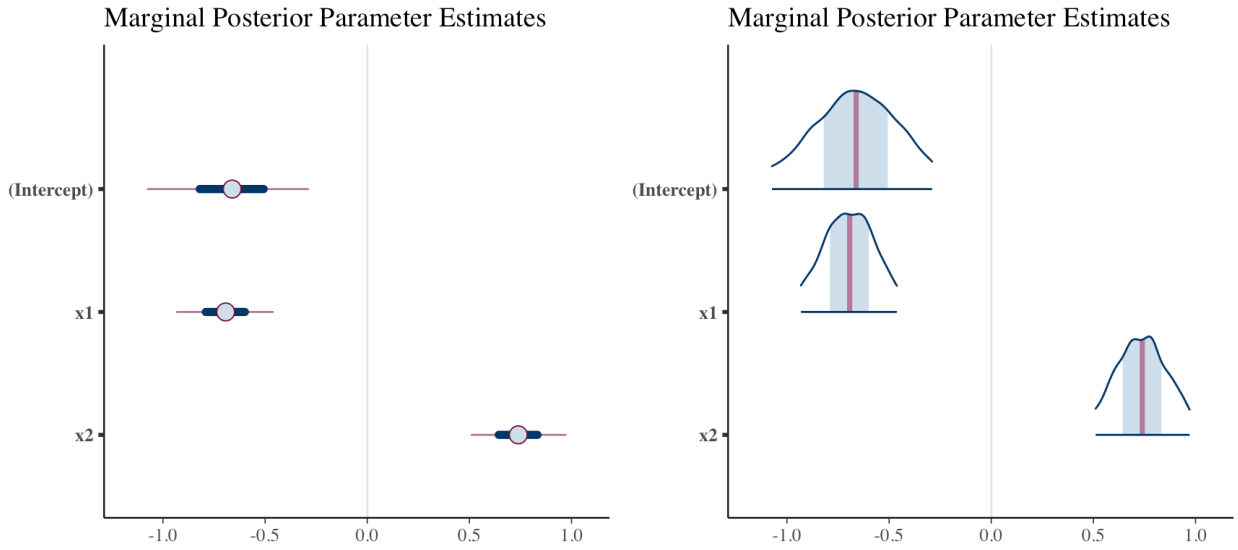
```
bayesplot_grid(
  pp_check(fit1) + ggtitle("Posterior Predictions (Correct Model)"),
  pp_check(fit2) + ggtitle("Posterior Predictions (Inappropriate Model)"),
  grid_args = list(ncol = 2)
)
```



We can use the (generic) `plot()` function with various assignments to the `plotfun` argument to create various posterior plots.

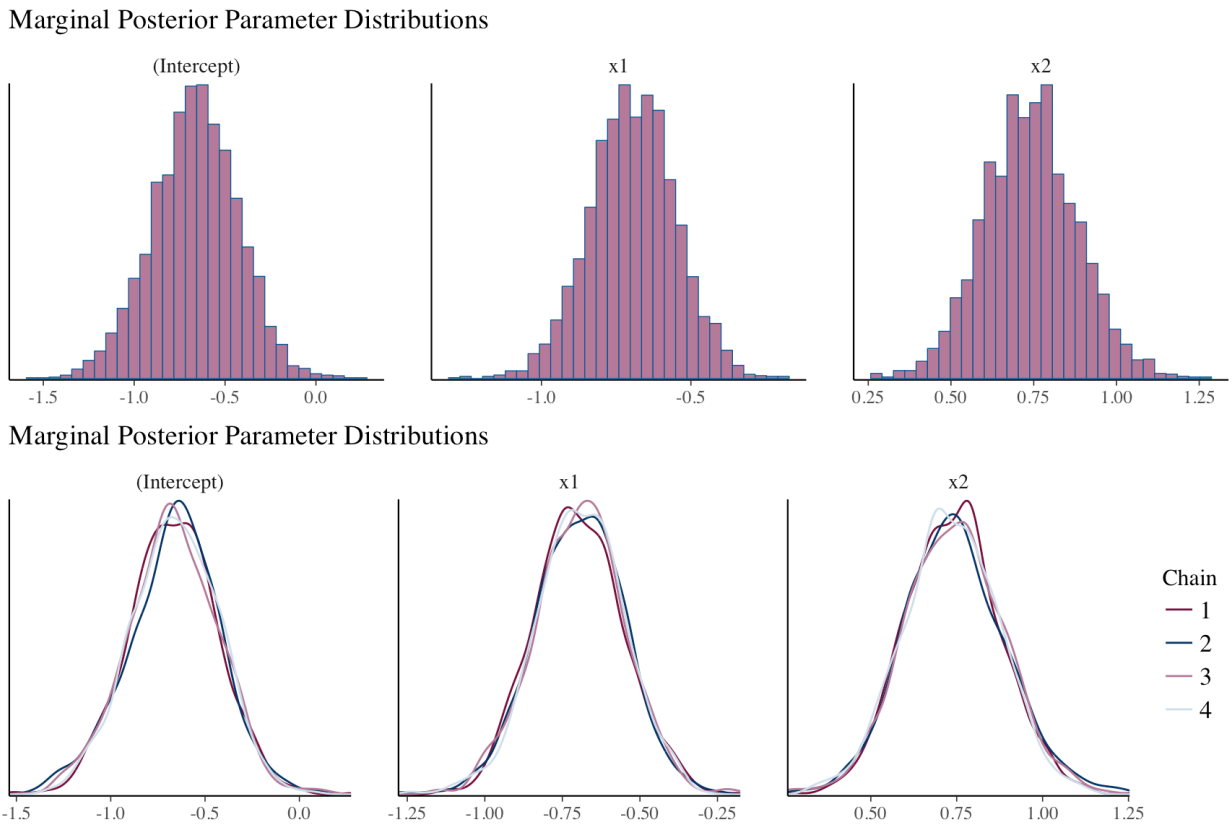
The default argument (`plotfun = "intervals"`) plots point estimates for each parameter along with credibility intervals. In this case we are plotting the 50% uncertainty interval (thick horizontal lines) and the 90% uncertainty interval (thin horizontal lines). In terms of interpretation, the 50% uncertainty interval identifies where 50% of the marginal distribution lies for each parameter. The `plotfun="areas"` argument plots the posterior distribution of each parameter with the uncertainty intervals shaded as areas below the cuve.

```
bayesplot_grid(
  plot(fit1, plotfun = "intervals", prob = 0.5, prob_outer = 0.9, point_est =
"median") + ggtitle("Marginal Posterior Parameter Estimates"),
  plot(fit1, plotfun = "areas", prob = 0.5, prob_outer = 0.9, point_est =
"median") + ggtitle("Marginal Posterior Parameter Estimates"),
  grid_args = list(ncol = 2)
)
```



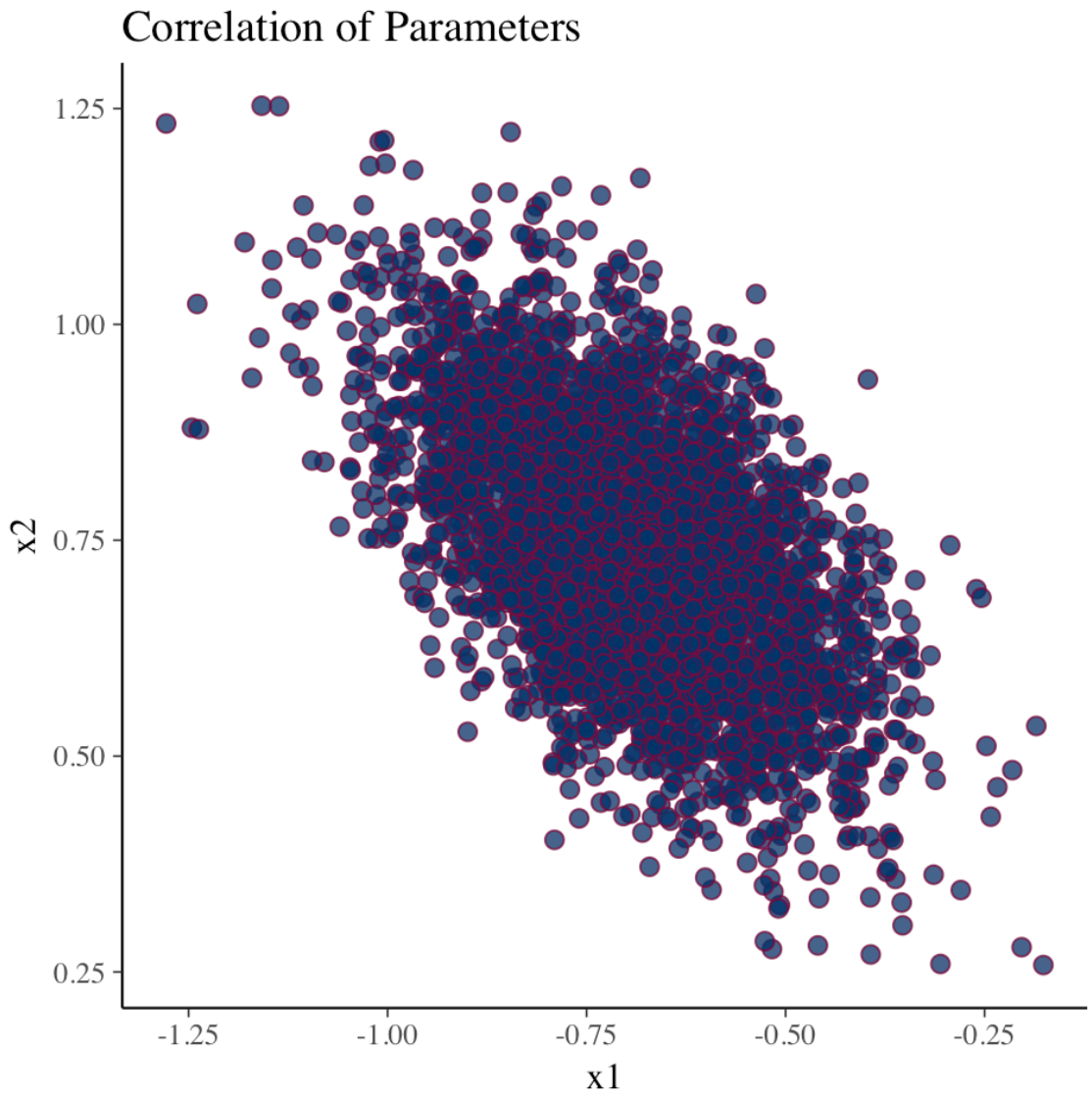
The `plotfun="hist"` and `plotfun="dens_overlay"` options plot the histograms for each parameter (pooled across chains) and the empirical density of each parameter, respectively.

```
bayesplot_grid(
  plot(fit1, plotfun = "hist") + ggtitle("Marginal Posterior Parameter
Distributions"),
  plot(fit1, plotfun = "dens_overlay") + ggtitle("Marginal Posterior
Parameter Distributions"),
  grid_args = list(nrow = 2)
)
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can use the `plotfun="scatter"` option to examine the correlation between the parameters.

```
plot(fit1, plotfun = "scatter", pars = c("x1","x2")) + ggtitle("Correlation
of Parameters")
```



Comparing Models

We can compare model performance using the loo package. Recall that we have fit two models. The appropriately specified model is `fit1` and the inappropriately specified model is `fit2`. Using the `loo()` function we can evaluate the **loo information criterion** (LOOIC).

```
library(loo)
loo1 <- loo(fit1)
loo2 <- loo(fit2)
## Warning: Found 31 observations with a pareto_k > 0.7. With this many
## problematic observations we recommend calling 'kfold' with argument 'K=10'
## to perform 10-fold cross-validation rather than LOO.
loo1
## Computed from 4000 by 300 log-likelihood matrix
##
##          Estimate   SE
```

```
## elpd_loo   -183.5  6.4
## p_loo         2.7  0.2
## looic       367.0 12.7
##
## All Pareto k estimates are good (k < 0.5)
## See help('pareto-k-diagnostic') for details.
loo2
## Computed from 200 by 300 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo  -2269.4 197.1
## p_loo      2045.4 190.0
## looic      4538.8 394.2
##
## Pareto k diagnostic values:
##                          Count  Pct
## (-Inf, 0.5]   (good)     268   89.3%
##  (0.5, 0.7]   (ok)         1    0.3%
##    (0.7, 1]   (bad)        1    0.3%
##    (1, Inf)   (very bad)  30   10.0%
## See help('pareto-k-diagnostic') for details.
```

Recall that information criterion (e.g. AIC) estimates the information lost when using the model to define the data generation process. Since we know that `fit1` is the appropriate model we expect that the LOOIC for `fit1` should be lower than the LOOIC for `fit2`. (Note that LOOIC and ELPD explain the same thing since $\text{LOOIC} = -2 \cdot \text{ELPD}$

.)

We can use the `compare()` function to create a table that compares various statisics generated by the loo package for several models.

```
compare(loo1, loo2)
## elpd_diff        se
##   -2085.9     195.7
```

The difference in the expected predictive accuracy (ELPD) is negative, indicating that the that the expected preditive accuracy for the first model is higher.