Simple Notes on Bayesian Estimation using Package 'brms'

Contents

Introduction

This is a summary on how to do Bayesian Estimation using brms packages. We are already familiar with the key idea with Bayesian Estimation.

$$p(\theta|x) \propto p(x|\theta)p(\theta)$$

In reality, the right hand side is usually very complicated. Therefore we need to use MCMC samplying to generate the posterior distribution of θ. This summary skips the decriptions of sampling procedure, but pay big attention on the diagnostic analysis on posterior distribution.

Diagnostic Analysis

Check for convergence

One rule of thumb is that R-hat values for all less than 1.1 source. Note that all parameters must show convergence. This is a necessary but not sufficient condition for convergence

Remedies

- Put parameters on the same scale. The samplers work best when all parameters are roughly on the same scale. Try to avoid situations where parameters are orders of magnitude different, e.g. 1e-5 and 1e+10.
- Increase the informativeness of priors. If parameters are too uninformative, the posterior distribution may have wide tails that hamper sampling. One way of thinking about it is that the model is only "weakly identified" and requires either more data or more informative priors to estimate. But how to choose? When no explicit prior beliefs are available, you can (reference):
  - derive (or simply use if already available, a great ressource is http://www.stats.org.uk/priors/noninformative/YangBerger1998.pdf) a Jeffreys (e.g. uniform for a location parameter) or a reference prior (especially in case of multivariate parameters).
  - sometimes such choices are impossible or quite difficult to derive and in this case, you can try to choose among one of the many "generic" weakly informative prior (e.g. uniform shrinkage distribution for scale parameters of hierarchical model or g-prior for gaussian regression).
- Increase the number of iterations.

Autocorrelation

Autocorrelation is a measure of how much the value of a signal correlates to other values of that signal at different points in time. In the context of MCMC, autocorrelation is a measure of how independent different samples from your posterior distribution are – lower autocorrelation indicating more independent results.

Consequence of high autocorrelation

When you have high autocorrelation the samples you've drawn don't accurately represent the posterior distribution and therefore don't provide as meaningful information for the solution to the problem. In other words, lower autocorrelation means higher efficiency in your chains and better estimates. A general rule would be that the lower your autocorrelation, the less samples you need for the method to be effective (but that might be oversimplifying).

If autocorrelation is high, then N samples are not giving you N pieces of information about your distribution but fewer than that. To be more specific, autocorrelation gives you unrepresentative samples 'in the short run'. Moreover, the more autocorrelation there is, the longer that 'short run' is. For very strong autocorrelation, the short run might be a good fraction of your total samples.

In summary, a strongly autocorrelated chain takes longer to get from its starting conditions to the target distribution you want, while being less informative and taking longer to explore that distribution when it gets there. In other words, highly correlated MCMC samplers requires more samples to produce the same level of Monte Carlo error for an estimate.

Check

-The Effective Sample Size (ESS) is one measure of how much information you're really getting (and is a function of the autocorrelation parameter). In general it is safe for ESS to be above 100.

-Also plot the autocorrelation for your sampled parameters. Parameters with lower ESS tends to have strong auto-corrleation

Remedies

- The usual direct remedies are re-parameterisation or sampling parameters that you expect to be intercorrelated in blocks rather than separately since they will otherwise generate autocorrelation in the chain.
- If you have decided on the sampler(model) already, and do not have the option of playing around with other samplers, then

- People often also 'thin'. That is, during the sampling, we save one for every $x(x > 1)$ samples. This would reduce the correlation between samples. Thinning trades off sample size for memory, and due to autocorrelation in samples, loss in effective sample size is less than the loss in sample size.Thinning has become less of an issue as memory has become less of a computational constraint, and samplers have become more efficient.
- find good starting values. The autocorrelation is also affected by starting values of the Markov chain. There is generally an (unknown) optimum starting value that leads to the comparatively less autocorrelation.
- run the sampler for a long time, so that you Effective Sample Size (ESS) is large.
- Check the correlations between explanatory variables. If highly correlated, consider the extract their orthogonal parts.

Inaccuracy of Estimation

We can use Monte Carlo standard error to measure the inaccuaracy of the paramter sample.

Fitness of the model to data

We also wants to know how the model fits the data. Suppose we already know $p(\hat{\theta}|y)$, and for each $\hat{\theta}$ we can generate simulated series of $\hat{y}|\hat{\theta}$. Intuitively, we can compare how the distribution of y and $\hat{y}|\hat{\theta}$ differ. We can also average such differences over different sample of $\hat{\theta}$.

We already know that there are many methods(statistics) for checking whether two groups have same distribution or not. This means, that for each $\hat{\theta}$,based on the method we choose we can get a p value, which is a function of $\hat{\theta}$ and $\hat{\theta}$. we then average the p across various $\hat{\theta}$

Residual Analysis

In the case where outcome variable is generated from a normal distribution, make sure to check where the residual (y - yhat) is normally distributed, homoskadascity and independent.

- If not normally distributed: consider transforming the outcome variable (e.g., logarithm transformation) or respecify the error term.
- If not homoskadasticity: consider model with variable sigmas. May need to write down the rstan code by yourself. (easy!)

Example

We next walk through a specific example using brms package. For univariate models, the brms package is an excellent tool to do the bayesian estimation. For state space models, we may want to write our own stan models.

The Package and Data Introduction

```
library(readstata13)
library(dplyr)
library(rstan)
library(brms)
library(purrr)
library(ggplot2)
library(bayesplot)
library(datasets)
```

```
rstan_options(auto_write = TRUE)
options (mc.cores = parallel :: detectCores())
df_raw =read.csv("./fish.txt", sep=",")%>%
  mutate(weather = ifelse (weather == 'sunny', 1, 0))

head(df_raw)
```

```
##   fish_num weather temperature id
## 1        0       0         5.0  1
## 2        1       0        24.2  2
## 3        6       0        11.5  3
## 4        0       0         9.8  4
## 5        1       0        18.1  5
## 6        1       0        18.1  6
```

Model Estimation

Suppose that we want to estimate a simple model:

$$fish\_num = \beta_0 + \beta_1 * temperature + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

We want to estimate β_0, β_1, σ.

```
bayesian_res = brm (fish_num ~ 1 + temperature , # specify your model
                    data   = df_raw,
                    family = poisson(),# default is gaussian
                    prior   = NULL,
                    seed = 1,
                    chains = 4,
                    iter = 4000,
                    warmup = 2000)
```

Diagnostic:

See Rhat and ESS

Make sure that Rhat is smaller than 1.1

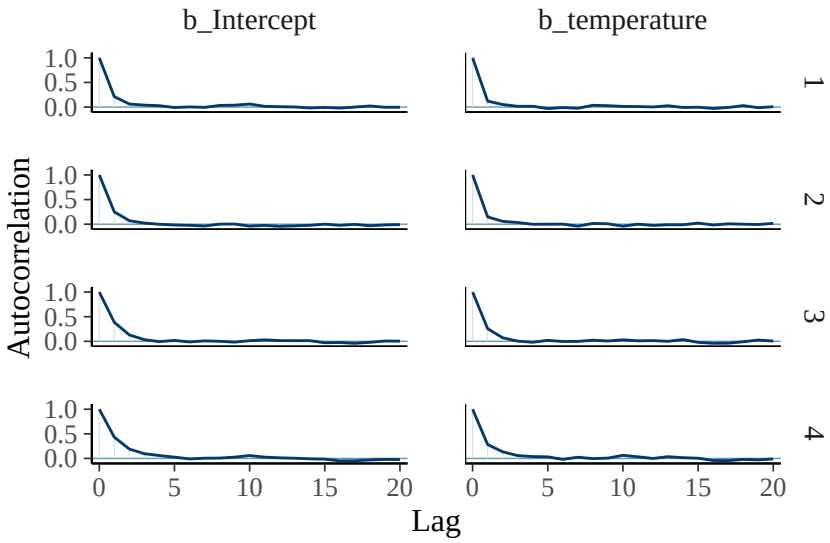
```
bayesian_res
```

```
##  Family: poisson
##   Links: mu = log
## Formula: fish_num ~ 1 + temperature
##    Data: df_raw (Number of observations: 100)
## Samples: 4 chains, each with iter = 4000; warmup = 2000; thin = 1;
##          total post-warmup samples = 8000
##
## Population-Level Effects:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept       0.13      0.15    -0.17     0.42 1.00     4022     3905
## temperature     0.05      0.01     0.04     0.07 1.00     4887     4397
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

The ACF plot

If parameter shows high autocorrelation, the result may not be reliable.

```
stanplot(bayesian_res, type = 'acf')
```



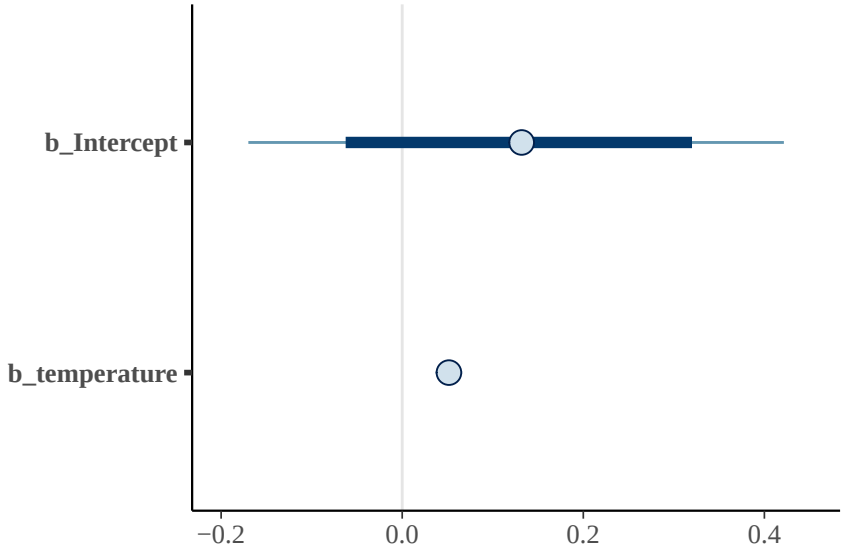
Posterior distribution

We can use plot() to get the graph of posterior distribution.

```
plot(bayesian_res, pars = ('b_temperature'))
```

We can also use stanplot to plot the interval or density

```
stanplot(bayesian_res, type = 'intervals', pars = '^b_', prob = 0.8, prob_outer = 0.95)
```



get_priors and set_priors

With brms packages, we can easily get the information of priors and set the priors for the parameters.

```
prior_summary(bayesian_res)
```

```
##                 prior     class        coef group resp dpar nlpar bound
##                (flat)         b
##                (flat)         b temperature
##  student_t(3, 0, 2.7) Intercept
##        source
##       default
##  (vectorized)
##       default
```

In which the 'class' indicates whether the parameter is coefficient, intercept, or sigma. If outcome is generated from normal distribution, then the sigma is the variance. In poisson generation process, of course, there is no need to estimate sigma, so actually we see no prior information about sigma. The 'coef' column further indicates the coefficient is of what variable.

We can specify a more informative prior:

```
bayesian_res = brm (fish_num ~ 1 + temperature , # specify your model
                    data   = df_raw,
                    family = poisson(),# default is gaussian
                    prior   = c(set_prior('normal(0,5)', class = 'b', coef =  'temperature'), set_prior
                    seed = 1,
                    chains = 4,
                    iter = 4000,
                    warmup = 2000)
```

Get Prediction

Directly use functions in brms

brms provides convenient functions to get the prediction result.

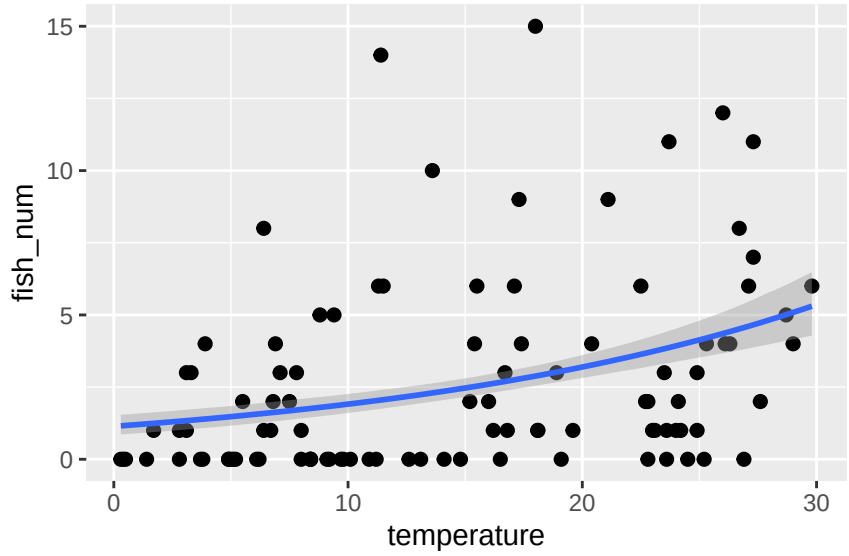
- We can use fitted function to produce the estimated outcome variable, given your specified explanatory variables.

```
new_data = data.frame (temperature = 20)
fitted(bayesian_res, new_data)
```

```
##      Estimate Est.Error     Q2.5    Q97.5
## [1,] 3.200067 0.2012542 2.815573 3.607352
```

- We can also use marginal_effects to directly get the fitted value .

```
 marginal_effects( bayesian_res,) %>%
plot(., points = TRUE)
```



Extract paramter samples

Like in Rstan, here we can also extract the samples that we want.

```
mcmc_sample = as.mcmc(bayesian_res, combine_chains = TRUE)
```

This is pretty much like the extract function in rstan. Check this mcmc_sample by

```
head(mcmc_sample)
```

References

- (An Excellent resource for studying Bayesian estimation)https://jrnold.github.io/bayesian_notes/
- https://mvuorre.github.io/posts/2017-01-02-how-to-compare-two-groups-with-robust-bayesian-estimation-using-r-stan-and-brms/
- https://mc-stan.org/docs/2_18/stan-users-guide/reparameterization-section.html
- https://discourse.mc-stan.org/t/aic-bic/11036
- https://fukamilab.github.io/BIO202/05-B-Bayesian-priors.html