Mean and Standard Deviation in rstan

Carolyn J. Anderson

11/1/2019

I will create a series of documents that run various analysis on the nels 23 schools data. I could put them all in one document, but this would take a really, really long time. The documents in the set will do the following

* Estimate mean and sd of math scores
* Fit math ~ homework
* Fit math ~ homework + ses
* Fit HLM: math ~ homework + (1 |school)
* Fit HLM: math ~ homework + (1 + homework|school)

## Packages

So, let’s load up the rstan package at the start and see the messages that we’re given:

library(rstan)

## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.19.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan\_options(auto\_write = TRUE)

## For improved execution time, we recommend calling
## Sys.setenv(LOCAL\_CPPFLAGS = '-march=native')
## although this causes Stan to throw an error on a few processors.

Note that there three recommendations here about what to run before we use rstan. The first one, “For execution on a local, multicore CPU with excess RAM we recommend calling”, only needs to be issue once per session. This make the default number of core to use equal to the number of cores available

options(mc.cores = parallel::detectCores())

The second recommendation, “To avoid recompilation of unchanged Stan programs, we recommend calling”, save the compile model so that it can be re-used. Models are compiled in C++ and this can take some time.

rstan\_options(auto\_write = TRUE)

The third recommendation, “For improved execution time, we recommend calling”, I’m not exactly sure what this does, but we’ll use it.

This package computes some useful statistics for model evaluation and comparsion. One statistic is efficient approximate “leave one out” (hence it’s called “loo”) cross-validation for Bayesian models. The loo package will also compute waic (“widely applicable information criteria”) and compare waic for multiple models using the loo\_comare function.

library(loo)

## This is loo version 2.1.0.
## \*\*NOTE: As of version 2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' argument or set options(mc.cores = NUM\_CORES) for an entire session. Visit mc-stan.org/loo/news for details on other changes.

## \*\*NOTE for Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see https://github.com/stan-dev/loo/issues/94).

##
## Attaching package: 'loo'

## The following object is masked from 'package:rstan':
##
## loo

We get the recommendation to set mc.cores; however, we have already done this.

## Directory and Data Set Up

My working directory where the data live is

setwd("D:\\Dropbox\\edps 590BAY\\Lectures\\9 Ham\_Stan\_brms")

And read in data

nels <- read.table("school23\_data.txt",header=TRUE)
nels <- nels[order(nels$school,nels$student), ]
# Get information on numbers of school and students
school <- unique(nels$school)
N <- length(school)
# total number of students
n <- length(nels$math)
names(nels)

## [1] "school" "student" "sex" "race" "homew" "schtype"
## [7] "ses" "pared" "math" "classtr" "schsize" "urban"
## [13] "geo" "minority" "ratio"

I like computing sample statistics

#############################################
# Sample statistics
#############################################
ybar = mean(nels$math)
s2 = var(nels$math)
stderr <- sqrt(s2/n)
sample.stat <- c(n,ybar,s2,stderr)
names(sample.stat) <- c("n","ybar","s2","stderr")
sample.stat

## n ybar s2 stderr
## 519.0000000 51.7225434 114.6873480 0.4700825

## Estimate Mean and Standard Deviation

Our first stan model! We’ll start simple and work up to more complex ones.

We could either definte the model as below and run in rstan like we did with rjags OR we could put this in a file with extension .stan

#############################################
# stan model definition
#############################################
model1 <- "
data { // We need to define the data
 int<lower=0> n; // number of students
 real y[n]; // vector of data
}
parameters {
 real<lower=0> sigma;
 real mu;
}
model {
 sigma ~ cauchy(0,1); // other way to do this
 mu ~ normal(0, 10);
 y ~ normal(mu,sigma);
}
generated quantities{
 real log\_lik[n];
 for (i in 1:n) {
 log\_lik[i] = normal\_lpdf(y[i] | mu, sigma);
 }
}
"

Alternatively the model part could be written as

 target += cauchy\_lpdf(sigma | 0, 1);
 target += normal\_lpdf(mu |0, 30);
 target += normal\_lpdf(y | mu, sigma);
}

where “lpdf” stands for log proability density function.

Before we run rstan, we need to create a data list, as follows:

########################################
# Make data list
########################################
y <- as.numeric(nels$math) # otherwise it's class integer
n <- nrow(nels)
dataList <- list(n=n,y=y)

Now for the part that takes a bit of time

fit.1 <- stan(
 model\_code = model1,
 model\_name = "Nels23: mean & sd",
 data = dataList,
 iter = 2000,
 chains = 4,
 warmup = floor(2000/2),
 verbose = FALSE)

And some output but I don’t want all the loglikelihoods here, so I’ll only ask for mean and standard deviation:

print(fit.1,pars = c("mu","sigma"), probs = c(0.025, 0.975),
 digits = 4)

## Inference for Stan model: Nels23: mean & sd.
## 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% 97.5% n\_eff Rhat
## mu 51.6027 0.0089 0.4771 50.6711 52.5137 2879 1.0005
## sigma 10.7118 0.0063 0.3319 10.0935 11.3883 2739 0.9997
##
## Samples were drawn using NUTS(diag\_e) at Sun Nov 03 13:21:23 2019.
## 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).

and various graphics

plot(fit.1)
stan\_trace(fit.1)
stan\_dens(fit.1)
stan\_hist(fit.1)
stan\_ac(fit.1)
stan\_scat(fit.1,pars=c("mu","sigma"))
pairs(fit1, pars = c("mu", "sigma", "lp\_\_"), las = 1)

There are some more that we could look at but I’m not sure what to do with these

stan\_diag(fit.2,information=c("stepsize"))
stan\_diag(fit.2,information=c("divergence"))
stan\_diag(fit.2,information=c("treedepth"))

And to use the loo package,

log\_lik.1 <- extract\_log\_lik(fit.1, merge\_chains = FALSE)
r\_eff <- relative\_eff(exp(log\_lik.1))

loo.1 <- loo(log\_lik.1, r\_eff = r\_eff, cores = 4)
print(loo.1)

##
## Computed from 4000 by 519 log-likelihood matrix
##
## Estimate SE
## elpd\_loo -1968.3 10.5
## p\_loo 1.4 0.1
## looic 3936.6 21.1
## ------
## Monte Carlo SE of elpd\_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.

waic.1 <- waic(log\_lik.1)
print(waic.1)

##
## Computed from 4000 by 519 log-likelihood matrix
##
## Estimate SE
## elpd\_waic -1968.3 10.5
## p\_waic 1.4 0.1
## waic 3936.6 21.1

If we had two models we might want to look at

compare(loo.1,loo.2)

## Linear Regression with 1 predictor

This requires a new model

#############################################
# Simple Linear Regression in stan model
#############################################
lr.1 <- "
data{
 int<lower=0> n;
 vector[n] x; // added x and changed from real y[n]
 vector[n] y;
}
parameters {
 real beta0;
 real beta1;
 real<lower=0> sigma;
}
model {
 beta0 ~ normal(0,10);
 beta1 ~ normal(0,10);
 sigma ~ cauchy(0,1);
 y ~ normal(beta0+beta1\*x,sigma);
}
generated quantities{
 real log\_lik[n];
 for (i in 1:n) {
 log\_lik[i] = normal\_lpdf(y[i] | beta0 + beta1\*x, sigma);
 }
}
"

Now the dataList:

#################################
# Data List #
#################################
y <- as.numeric(nels$math)
hmwk <- as.numeric(nels$homew)
n <- nrow(nels)
dataList <- list(n=n,y=y,x=hmwk)

If I want to change the predictor, all I have to do is put in another variable for x.

Running stan is same as before (change model and name of model)

#################################
# Run rstan #
#################################
fit.lr1 <- stan(
 model\_code = lr.1,
 model\_name = "Nels23: math ~ 1 + homework",
 data = dataList,
 iter = 2000,
 chains = 4,
 cores = 4,
 verbose = FALSE)

Take a look at the output. Since I didn’t want to look at the values of log\_lik, I specifid what parameter estimates I want to see. The results should look fine.

print(fit.lr1,pars=c("beta0","beta1","sigma"),probs=c(.025,.50,.095), digits=4)

And if you want graphics,

plot(fit.lr1)
stan\_trace(fit.lr1)
stan\_dens(fit.lr1)
stan\_hist(fit.lr1)
stan\_ac(fit.lr1)
stan\_scat(fit.lr1,pars=c("beta0","beta1"))
pairs(fit.lr11, pars = c("beta0","beta1", "sigma", "lp\_\_"), las = 1)

We can also do some model comparisions now that we have 2 models:

log\_lik.lr1 <- extract\_log\_lik(fit.lr1, merge\_chains = FALSE)
r\_eff.lr1 <- relative\_eff(exp(log\_lik.lr1))

loo.lr1 <- loo(log\_lik.lr1, r\_eff = r\_eff.lr1, cores = 4)

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.

print(loo.lr1)

##
## Computed from 4000 by 519 log-likelihood matrix
##
## Estimate SE
## elpd\_loo -1113766.5 7983.5
## p\_loo 106381.3 2611.9
## looic 2227533.0 15967.0
## ------
## Monte Carlo SE of elpd\_loo is NA.
##
## Pareto k diagnostic values:
## Count Pct. Min. n\_eff
## (-Inf, 0.5] (good) 0 0.0% <NA>
## (0.5, 0.7] (ok) 0 0.0% <NA>
## (0.7, 1] (bad) 0 0.0% <NA>
## (1, Inf) (very bad) 519 100.0% 1
## See help('pareto-k-diagnostic') for details.

waic.lr1 <- waic(log\_lik.lr1)

## Warning: 519 (100.0%) p\_waic estimates greater than 0.4. We recommend
## trying loo instead.

print(waic.lr1)

##
## Computed from 4000 by 519 log-likelihood matrix
##
## Estimate SE
## elpd\_waic -1608934.4 33333.6
## p\_waic 601549.2 28176.1
## waic 3217868.8 66667.2

## Warning: 519 (100.0%) p\_waic estimates greater than 0.4. We recommend
## trying loo instead.

Comparing the two models:

compare(loo.1,loo.lr1)

## elpd\_diff se
## -1111798.2 7973.0

## Model defintion for Multiple Regression

Rather than list out all the regression parameters, we can make use vectors (regression coefficients) and matrices (design matrix which contains the explantory/predictor variables)

#############################################
# stan model definition #
#############################################
mlreg <- "
data {
 int<lower=0> N; // number of students
 int<lower=0> K; // number of predictors
 matrix[N,K] x; // predictor matrix
 vector[N] y; // outcome vector
}
parameters {
 real b0; // intercept
 vector[K] beta; // coefficients for predictors
 real<lower=0> sigma; // error scale
}
model {
 y ~ normal(b0 + x\*beta, sigma);
}
"

The model you need to make sure that matrices are conformable in the specification of the model for y.

We could have added the intercept to beta and a column of ones in x.

For the data list we need to create the design matrix, x, and include the number of columns of x (i.e., number of predictors and coefficients for the parameter vector beta). x should be of type matrix.

########################################
# Make data list
########################################
N <- nrow(nels)
y = nels$math
x <- matrix(cbind(nels$homework,nels$ses),nrow=N)
K <- ncol(x)
dat <- list(N = N, K=K, y = y, x = x);

And to run rstan, nothing really new here.

#########################################
# Get estimates #
#########################################

fitLR2 <- stan(
 model\_code = mlreg ,
 model\_name = "Multiple linear regression",
 data = dat,
 iter = 2000,
 chains = 4,
 cores = 4,
 warmup = floor(2000/2),
 verbose = FALSE,
 sample\_file = file.path(tempdir(), 'nels\_mlreg.txt'))

After the model runs, check for convergence and ask for what ever statistics (summaries) and graphics to help you here.

########################################
# Output and various graphics
########################################
print(fitLR2,probs=c(.025,.50,.975),digits=2)

## Inference for Stan model: Multiple linear regression.
## 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% 50% 97.5% n\_eff Rhat
## b0 51.74 0.01 0.41 50.95 51.74 52.54 4495 1
## beta[1] 5.97 0.01 0.46 5.09 5.97 6.86 4318 1
## sigma 9.36 0.00 0.29 8.82 9.36 9.96 3998 1
## lp\_\_ -1417.38 0.03 1.23 -1420.62 -1417.08 -1416.00 2136 1
##
## Samples were drawn using NUTS(diag\_e) at Sun Nov 03 13:23:37 2019.
## 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).

stan\_plot(fitLR2)
stan\_trace(fitLR2)
stan\_dens(fitLR2)
stan\_hist(fitLR2)
stan\_ac(fitLR2)
stan\_scat(fitLR2,pars=c("b0","b1"))

## Multilevel Model with Random Intercept

This will require haveing a vector that indentified the school that a student belongs to.