Chapter 7 Applied CV, loo, WAIC

2024-10-28

Chapter 7

Examples of CV, PSIS-loo, and WAIC, using brms, a go-to R package for fitting a wide range of Bayesian models using Stan, using the "familiar" formula syntax in many R-packages.

What follows is a selection of content from the Chapter 7 section of Solomon Kurtz's rethinking with brms and tidyverse tutorials. I've added a few bits (e.g., example of how to fit a quadratic approximation in brms)

We can walk through some of this during class, if useful.

NOTE I've had some strange conlicts between brms (or maybe it's the dependency rstan) and rethinking (some conflict with the definitions of the stanfit class that differ between the packages). To get around this, I've had to uninstall rethinking and install the *slim* version of the package. How cool is it that the package has a *slim* version that allows for this!

```
library(tidyverse)
library(tidybayes)
library(rstan)
## installing the slim version of rethinking
## there's a possible conflict between the "stanfit" class in rethinking and in rstan
#devtools::install_github("rmcelreath/rethinking@slim")
library(rethinking)
library(brms)
library(rcartocolor)
options("brms.backend" = "cmdstanr")
options("brms.file_refit" = "on_change") # only refits models if the data or model differ from previous
```

To start I'll run some models from chapter 6.

Here's a section from Chapter 6 in Solomon Kurtz's brms version of rethinking.

Post-treatment bias - plants, treatments and fungus

Simulate the plants and fungus data

```
# how many plants would you like?
n <- 100

set.seed(71)
d <-
  tibble(h0        = rnorm(n, mean = 10, sd = 2),
         treatment = rep(0:1, each = n / 2),
         fungus    = rbinom(n, size = 1, prob = .5 - treatment * 0.4),
         h1        = h0 + rnorm(n, mean = 5 - 3 * fungus, sd = 1))
```

Model b6.6

$$
\begin{aligned}
h_{1i} &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= h_{0i} \times p \\
p &\sim \text{Log-Normal}(0, 0.25) \\
\sigma &\sim \text{Exponential}(1).
\end{aligned}
$$

brms model description

brms models are defined similarly to other R packages, with the **formula** syntax.

```
b6.6 <-
  brm(data = d,
      family = gaussian,
      h1 ~ 0 + h0, # model formula
      prior = c(prior(lognormal(0, 0.25), class = b, lb = 0),
                prior(exponential(1), class = sigma)),
      iter = 2000, warmup = 1000, chains = 4, cores = 4,
      seed = 6,
      file = "fits/b06.06",
      refresh = 1000)
```

The model formula syntax is generally familiar, but also note the `prior` argument. This argmument allows you control of the priors in the model `?brms::set_prior`. Syntax for setting priors is complicated (brms can fit a wide range of model types), but the package documentation is great and there is an active community of users that are open and constructive Stan Forums

brms model summaries also look familiar (Regression coefficients, sigma, estimates, standard errors, and 90% credible limits). And with a few extra bits of information, including the convergence diagnostics (rhat, Bulk_ess, Tail_ess), which we can return to after the sections on MCMC.

```
print(b6.6)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: h1 ~ 0 + h0
##    Data: d (Number of observations: 100)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Regression Coefficients:
##    Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## h0     1.43      0.02     1.39     1.46 1.00     3641     2814
##
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     1.82      0.13     1.59     2.10 1.00     2913     2424
##
## Draws were sampled using sample(hmc). 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).
```

brms models also allow you to export and explore the raw Stan code `brms::stancode(b6.6)`, however the syntax of brms models is quite different than most human-written Stan models, so not a great place to try to understand Stan.

So then, the expectation is an increase of about 43 percent relative to h_0. But this isn't the best model. We're leaving important predictors on the table.

Model b6.7: predicting with fungus and treatment

```
b6.7 <-
  brm(data = d,
      family = gaussian,
      bf(h1 ~ h0 * (a + t * treatment + f * fungus),
         a + t + f ~ 1,
         nl = TRUE),
      prior = c(prior(lognormal(0, 0.2), nlpar = a, lb = 0),
                prior(normal(0, 0.5), nlpar = t),
                prior(normal(0, 0.5), nlpar = f),
                prior(exponential(1), class = sigma)),
      iter = 2000, warmup = 1000, chains = 4, cores = 4,
      seed = 6,
      file = "fits/b06.07",
      refresh = 1000)
```

Summary

```
print(b6.7)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: h1 ~ h0 * (a + t * treatment + f * fungus)
##          a ~ 1
##          t ~ 1
##          f ~ 1
##    Data: d (Number of observations: 100)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Regression Coefficients:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## a_Intercept     1.48      0.03     1.43     1.53 1.00     1850     2291
## t_Intercept     0.00      0.03    -0.06     0.06 1.00     1953     2479
## f_Intercept    -0.27      0.04    -0.34    -0.19 1.00     2373     2434
##
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     1.45      0.10     1.27     1.67 1.00     3280     2665
##
## Draws were sampled using sample(hmc). 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).
```

Model b6.8 predicting with just treatment

This is the correct causal model.

$$
\begin{aligned}
h_{1i} &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= h_{0i} \times (\alpha + \beta_1 \text{treatment}_i) \\
\alpha &\sim \text{Log-Normal}(0, 0.25) \\
\beta_1 &\sim \text{Normal}(0, 0.5) \\
\sigma &\sim \text{Exponential}(1).
\end{aligned}
$$

```
b6.8 <-
  brm(data = d,
      family = gaussian,
      bf(h1 ~ h0 * (a + t * treatment),
         a + t ~ 1,
         nl = TRUE),
      prior = c(prior(lognormal(0, 0.2), nlpar = a, lb = 0),
                prior(normal(0, 0.5), nlpar = t),
                prior(exponential(1), class = sigma)),
      iter = 2000, warmup = 1000, chains = 4, cores = 4,
      seed = 6,
      file = "fits/b06.08",
      refresh = 1000)
```

Summary

```
print(b6.8)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: h1 ~ h0 * (a + t * treatment)
##          a ~ 1
##          t ~ 1
##    Data: d (Number of observations: 100)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Regression Coefficients:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## a_Intercept     1.38      0.03     1.33     1.43 1.00     2298     2091
## t_Intercept     0.09      0.03     0.02     0.15 1.00     2272     2240
##
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     1.78      0.13     1.55     2.04 1.00     2516     2077
##
## Draws were sampled using sample(hmc). 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).
```

Model comparison

In the sections to follow, we'll practice the model comparison approach, as opposed to the widely-used model selection approach.

Model mis-selection.

> We must keep in mind the lessons of the previous chapters: Inferring cause and making predictions are different tasks. Cross-validation and WAIC aim to find models that make good predictions. They don't solve any causal inference problem. If you select a model based only on expected predictive accuracy, you could easily be confounded. The reason is that backdoor paths do give us valid information about statistical associations in the data. So they can improve prediction, as long as we don't intervene in the system and the future is like the past. But recall that our working definition of knowing a cause is that we can predict the consequences of an intervention. So a good PSIS or WAIC score does not in general indicate a good causal model. (p. 226)

The `file` argument in the `brm()` function saves the fitted model object as external `.rds` files so that they can be retrieved without refitting using `readRDS()`.

```
b6.6 <- readRDS("fits/b06.06.rds")
b6.7 <- readRDS("fits/b06.07.rds")
b6.8 <- readRDS("fits/b06.08.rds")
```

With our **brms** paradigm, we also use the `waic()` function. Both the **rethinking** and **brms** packages get their functionality for the `waic()` and related functions from the **loo** package [@R-loo; @vehtariPractical-BayesianModel2017; @yaoUsingStackingAverage2018]. Since the `brms::brm()` function fits the models with HMC, we don't need to set a seed before calling `waic()` the way McElreath did with his `rethinking::quap()` model. We're already drawing from the posterior.

```
waic(b6.7)
```

```
##
## Computed from 4000 by 100 log-likelihood matrix.
##
##           Estimate   SE
## elpd_waic   -180.6  6.7
## p_waic         3.3  0.5
## waic         361.1 13.4
```

The WAIC estimate and its standard error are on the bottom row. The p_{WAIC}–what McElreath's output called the `penalty`–and its SE are stacked atop that. And look there on the top row. Remember how we pointed out, above, that we get the WAIC by multiplying `(lppd - pwaic)` by -2? Well, if you just do the subtraction without multiplying the result by -2, you get the `elpd_waic`. File that away. It'll become important in a bit. In McElreath's output, that was called the `lppd`.

Following the version 2.8.0 update to brms, part of the suggested workflow for using information criteria with **brms** (i.e., execute `?loo.brmsfit`) is to add the estimates to the `brm()` fit object itself. You do that with the `add_criterion()` function. Here's how we'd do so with `b6.7`.

```
b6.7 <- add_criterion(b6.7, criterion = "waic")
```

With that in place, here's how you'd extract the WAIC information from the fit object.

```
b6.7$criteria$waic
```

```
##
## Computed from 4000 by 100 log-likelihood matrix.
##
##           Estimate   SE
## elpd_waic   -180.6  6.7
## p_waic         3.3  0.5
## waic         361.1 13.4
```

Why would I go through all that trouble?, you might ask. Well, two reasons. First, now your WAIC information is saved with all the rest of your fit output, which can be convenient. But second, it sets you up to use the `loo_compare()` function to compare models by their informati on criteria. To get a sense of that workflow, here we use `add_criterion()` for the next three models. Then we'll use `loo_compare()`.

NOTE the `elpd_waic` criterion. This stands for *expected* log predictive density (it's still log pointwise predictive density but I guess the acronym got too long). WAIC relies on estimates of the expected log predictive density. You can access the pointwise estimates.

```
head(b6.7$criteria$waic$pointwise)
```

```
##      elpd_waic      p_waic     waic
## [1,] -1.465944 0.011126894 2.931887
## [2,] -2.434843 0.061245568 4.869686
## [3,] -1.545287 0.013839551 3.090575
## [4,] -1.319338 0.006084585 2.638675
## [5,] -1.338698 0.007709372 2.677396
## [6,] -1.307765 0.005272726 2.615530
```

```
# compute and save the WAIC information for the next three models
b6.6 <- add_criterion(b6.6, criterion = "waic")
b6.8 <- add_criterion(b6.8, criterion = "waic")

# compare the WAIC estimates
w <- loo_compare(b6.6, b6.7, b6.8, criterion = "waic")

print(w)
```

```
##      elpd_diff se_diff
## b6.7   0.0       0.0
## b6.8 -20.7       4.9
## b6.6 -22.4       5.8
```

```
print(w, simplify = F)
```

```
##      elpd_diff se_diff elpd_waic se_elpd_waic p_waic se_p_waic waic   se_waic
## b6.7    0.0       0.0  -180.6       6.7          3.3    0.5     361.1   13.4
## b6.8  -20.7       4.9  -201.3       5.4          2.5    0.3     402.5   10.8
## b6.6  -22.4       5.8  -203.0       5.7          1.6    0.2     405.9   11.4
```

You don't have to save those results as an object like we just did with `w`. But that'll serve some pedagogical purposes in just a bit. With respect to the output, notice the `elpd_diff` column and the adjacent `se_diff` column. Those are our WAIC differences in the elpd metric. The models have been rank ordered from the highest (i.e., `b6.7`) to the highest (i.e., `b6.6`). The scores listed are the differences of `b6.7` minus the comparison model. Since `b6.7` is the comparison model in the top row, the values are naturally 0 (i.e., $x - x = 0$). But now here's another critical thing to understand: Since the **brms** version 2.8.0 update, WAIC and LOO differences are no longer reported in the $-2 \times x$ metric. Remember how multiplying `(lppd - pwaic)` by -2 is a historic artifact associated with the frequentist χ^2 test? We'll, the makers of the **loo** package aren't fans and they no longer support the conversion.

So here's the deal. The substantive interpretations of the differences presented in an `elpd_diff` metric will be the same as if presented in a WAIC metric. But if we want to compare our `elpd_diff` results to those in the text, we will have to multiply them by -2. And also, if we want the associated standard error in the same metric, we'll need to multiply the `se_diff` column by 2. You wouldn't multiply by -2 because that would return a negative standard error, which would be silly. Here's a quick way to do those conversions.

```
cbind(waic_diff = w[, 1] * -2,
      se        = w[, 2] * 2)
```

```
##      waic_diff        se
## b6.7   0.00000  0.000000
## b6.8  41.37370  9.795014
## b6.6  44.80276 11.535483
```

Now those match up reasonably well with the values in McElreath's `dWAIC` and `dSE` columns.

One more thing. On page 227, and on many other pages to follow in the text, McElreath used the `rethinking::compare()` function to return a rich table of information about the WAIC information for several models. If we're tricky, we can do something similar with `loo_compare`. To learn how, let's peer further into the structure of our `w` object.

```
str(w)
```

```
##  'compare.loo' num [1:3, 1:8] 0 -20.7 -22.4 0 4.9 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:3] "b6.7" "b6.8" "b6.6"
##   ..$ : chr [1:8] "elpd_diff" "se_diff" "elpd_waic" "se_elpd_waic" ...
```

When we used `print(w)`, a few code blocks earlier, it only returned two columns. It appears we actually have eight. We can see the full output with the `simplify = F` argument.

```
print(w, simplify = F)
```

```
##      elpd_diff se_diff elpd_waic se_elpd_waic p_waic se_p_waic waic   se_waic
## b6.7    0.0       0.0  -180.6       6.7          3.3    0.5     361.1   13.4
## b6.8  -20.7       4.9  -201.3       5.4          2.5    0.3     402.5   10.8
## b6.6  -22.4       5.8  -203.0       5.7          1.6    0.2     405.9   11.4
```

The results are quite analogous to those from `rethinking::compare()`. Again, the difference estimates are in the metric of the elpd. But the interpretation is the same and we can convert them to the traditional information criteria metric with simple multiplication. As we'll see later, this basic workflow also applies to the PSIS-LOO.

Okay, we've deviated a bit from the text. Let's reign things back in and note that right after McElreath's **R** code 7.26, he wrote: "PSIS will give you almost identical values. You can add `func=PSIS` to the `compare` call to check" (p. 227). Our `brms::loo_compare()` function has a similar argument, but it's called `criterion`. We set it to `criterion = "waic"` to compare the models by the WAIC. What McElreath is calling `func=PSIS`, we'd call `criterion = "loo"`. Either way, we're asking the software the compare the models using leave-one-out cross-validation with Pareto-smoothed importance sampling.

```
b6.6 <- add_criterion(b6.6, criterion = "loo")
b6.7 <- add_criterion(b6.7, criterion = "loo")
b6.8 <- add_criterion(b6.8, criterion = "loo")

# compare the WAIC estimates
loo_compare(b6.6, b6.7, b6.8, criterion = "loo") %>%
  print(simplify = F)
```

```
##      elpd_diff se_diff elpd_loo se_elpd_loo p_loo  se_p_loo looic  se_looic
## b6.7    0.0       0.0  -180.6      6.7         3.3    0.5    361.2   13.4
## b6.8  -20.7       4.9  -201.3      5.4         2.5    0.3    402.5   10.8
## b6.6  -22.4       5.8  -203.0      5.7         1.6    0.2    406.0   11.4
```

Yep, the LOO values are very similar to those from the WAIC. Anyway, at the bottom of page 227, McElreath showed how to compute the standard error of the WAIC difference for models `m6.7` and `m6.8`. Here's the procedure for us.

Outliers and diagnostics for WAIC and PSIS-loo

Time to bring back the `WaffleDivorce` data.

```
data(WaffleDivorce, package = "rethinking")

d <-
  WaffleDivorce %>%
  mutate(d = rethinking::standardize(Divorce),
         m = rethinking::standardize(Marriage),
         a = rethinking::standardize(MedianAgeMarriage))

rm(WaffleDivorce)
```

Refit the divorce models from Section 5.1.

```
b5.1 <-
  brm(data = d,
      family = gaussian,
      d ~ 1 + a,
      prior = c(prior(normal(0, 0.2), class = Intercept),
                prior(normal(0, 0.5), class = b),
                prior(exponential(1), class = sigma)),
      iter = 2000, warmup = 1000, chains = 4, cores = 4,
      seed = 5,
      sample_prior = T,
      file = "fits/b05.01",
      refresh = 1000)
```

```
b5.2 <-
  brm(data = d,
      family = gaussian,
      d ~ 1 + m,
      prior = c(prior(normal(0, 0.2), class = Intercept),
                prior(normal(0, 0.5), class = b),
                prior(exponential(1), class = sigma)),
      iter = 2000, warmup = 1000, chains = 4, cores = 4,
      seed = 5,
      file = "fits/b05.02",
      refresh = 1000)

b5.3 <-
  brm(data = d,
      family = gaussian,
      d ~ 1 + m + a,
      prior = c(prior(normal(0, 0.2), class = Intercept),
                prior(normal(0, 0.5), class = b),
                prior(exponential(1), class = sigma)),
      iter = 2000, warmup = 1000, chains = 4, cores = 4,
      seed = 5,
      file = "fits/b05.03",
      refresh = 1000)
```

Compute and save the LOO estimates for each.

```
b5.1 <- add_criterion(b5.1, criterion = "loo")
b5.2 <- add_criterion(b5.2, criterion = "loo")
b5.3 <- add_criterion(b5.3, criterion = "loo")
```

Now compare the models by the PSIS-LOO-CV.

```
loo_compare(b5.1, b5.2, b5.3, criterion = "loo") %>%
  print(simplify = F)
```

```
##      elpd_diff se_diff elpd_loo se_elpd_loo p_loo se_p_loo looic se_looic
## b5.1   0.0       0.0   -63.0      6.5         3.8   1.8    126.0  12.9
## b5.3  -0.8       0.4   -63.8      6.4         4.7   1.9    127.7  12.8
## b5.2  -6.8       4.7   -69.8      5.0         3.1   1.0    139.6  10.0
```

Like in the text, our `b5.1` has the best LOO estimate, but only by a little bit when compared to `b5.3`. Unlike McElreath reported in the text, we did not get a warning message from `loo_compare()`. Let's investigate more carefully with the `loo()` function.

```
loo(b5.3)
```

```
##
## Computed from 4000 by 50 log-likelihood matrix.
##
##          Estimate   SE
## elpd_loo    -63.8  6.4
```

```
## p_loo         4.7  1.9
## looic       127.7 12.8
## ------
## MCSE of elpd_loo is 0.1.
## MCSE and ESS estimates assume MCMC draws (r_eff in [0.6, 1.3]).
##
## All Pareto k estimates are good (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

Using brms to fit a quadratic approximation

brms has a diverse range of estimation options under the hood. By default, it uses the HMC sampling algorithm that is the default MCMC sampling algorithm used by Stan `brm(..., algorithm = "sampling")`. It also allows some posterior approximations that are often faster than full HMC sampling. For example, we can re-run model 5.3 using a laplace approximation of the posterior, which is essentially the same as teh quadratic approximation used in `quap()`.

```
b5.3laplace <-
  brm(data = d,
      family = gaussian,
      d ~ 1 + m + a,
      prior = c(prior(normal(0, 0.2), class = Intercept),
                prior(normal(0, 0.5), class = b),
                prior(exponential(1), class = sigma)),
      #iter = 2000, warmup = 1000, chains = 4, cores = 4,
      seed = 5,
      file = "fits/b05.03laplace",
      #refresh = 1000,
      algorithm = "laplace")


b5.3laplace <- add_criterion(b5.3laplace, criterion = "loo")

print(b5.3laplace$criteria$loo)
```

```
##
## Computed from 1000 by 50 log-likelihood matrix.
##
##          Estimate   SE
## elpd_loo    -64.9  7.5
## p_loo         6.0  2.6
## looic       129.7 15.0
## ------
## MCSE of elpd_loo is NA.
## MCSE and ESS estimates assume MCMC draws (r_eff in [0.9, 1.0]).
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. ESS
## (-Inf, 0.67]   (good)     49    98.0%   128
##    (0.67, 1]   (bad)       0     0.0%   <NA>
##     (1, Inf)   (very bad)  1     2.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

Sure enough, the difference between the HMC version of b5.3 and the laplace/quap b5.3 seems to be related to the fitting algorithm. Let's finish exploring the PSIS-loo diagnostics for the models fit by the HMC algorithm.

Diagnostics for PSIS-loo, "ok k".

```
loo(b5.3)
```

```
##
## Computed from 4000 by 50 log-likelihood matrix.
##
##          Estimate   SE
## elpd_loo    -63.8  6.4
## p_loo         4.7  1.9
## looic       127.7 12.8
## ------
## MCSE of elpd_loo is 0.1.
## MCSE and ESS estimates assume MCMC draws (r_eff in [0.6, 1.3]).
##
## All Pareto k estimates are good (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

One of the observations was in the `(ok)` range, but none were in the `(bad)` or `(very bad)` ranges. By opening the **loo** package directly, we gain access to the `pareto_k_ids()` function, which will help us identify which observation crossed out of the `(good)` range into the `(ok)`.

```
library(loo)

loo(b5.3) %>%
  pareto_k_ids(threshold = 0.5)
```

```
## [1] 13
```

The number `13` refers to the corresponding row in the data used to fit the model. We can access that row directly with the `dplyr::slice()` function.

```
d %>%
  slice(13) %>%
  select(Location:Loc)
```

```
##   Location Loc
## 1    Idaho  ID
```

Here we subset the 13^{th} cell in the `loo::pareto_k_values()` output to see what that value was.

```
pareto_k_values(loo(b5.3))[13]
```

```
## [1] 0.5844272
```

Alternatively, we could have extracted that value from our `b5.3` fit object like so.

```
b5.3$criteria$loo$diagnostics$pareto_k[13]
```

```
## [1] 0.5844272
```

0.58 is a little high, but not high enough to cause `loo()` to return a warning message. Once a Pareto k value crosses the 0.7 threshold, though, the **loo** package will bark. Before we make our version of Figure 7.10, we'll want to compute the WAIC for `b5.3`. That will give us access to the p_{WAIC}.

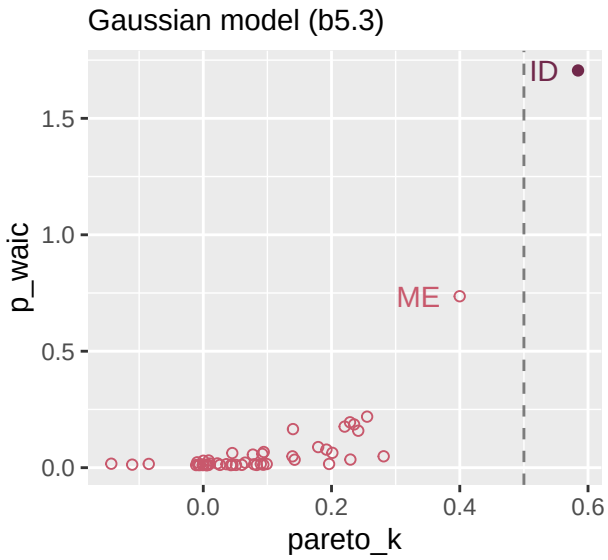
```
b5.3 <- add_criterion(b5.3, "waic", file = "fits/b05.03")
b5.3$criteria$waic
```

```
##
## Computed from 4000 by 50 log-likelihood matrix.
##
##           Estimate   SE
## elpd_waic    -63.7  6.3
## p_waic         4.6  1.8
## waic         127.5 12.7
##
## 2 (4.0%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

We're ready to make our version of Figure 7.10.

```
tibble(pareto_k = b5.3$criteria$loo$diagnostics$pareto_k,
       p_waic   = b5.3$criteria$waic$pointwise[, "p_waic"],
       Loc      = pull(d, Loc)) %>%

  ggplot(aes(x = pareto_k, y = p_waic, color = Loc == "ID")) +
  geom_vline(xintercept = .5, linetype = 2, color = "black", alpha = 1/2) +
  geom_point(aes(shape = Loc == "ID")) +
  geom_text(data = . %>% filter(p_waic > 0.5),
            aes(x = pareto_k - 0.03, label = Loc),
            hjust = 1) +
  scale_color_manual(values = carto_pal(7, "BurgYl")[c(5, 7)]) +
  scale_shape_manual(values = c(1, 19)) +
  labs(subtitle = "Gaussian model (b5.3)") +
  theme(legend.position = "none")
```



For both the Pareto k and the p_{WAIC}, our values are not as extreme as those McElreath reported in the text. I'm not sure if this is a consequence of us using HMC or due to recent algorithmic changes for the Stan/**loo** teams.

Note: yes, it is :-)

But at least the overall pattern is the same. Idaho is the most extreme/influential case.

In the text (p. 232), McElreath reported the effective number of parameters for `b5.3` was nearly 6. We can look at this with the `waic()` function.

```
waic(b5.3)
```

```
##
## Computed from 4000 by 50 log-likelihood matrix.
##
##           Estimate   SE
## elpd_waic    -63.7  6.3
## p_waic         4.6  1.8
## waic         127.5 12.7
##
## 2 (4.0%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

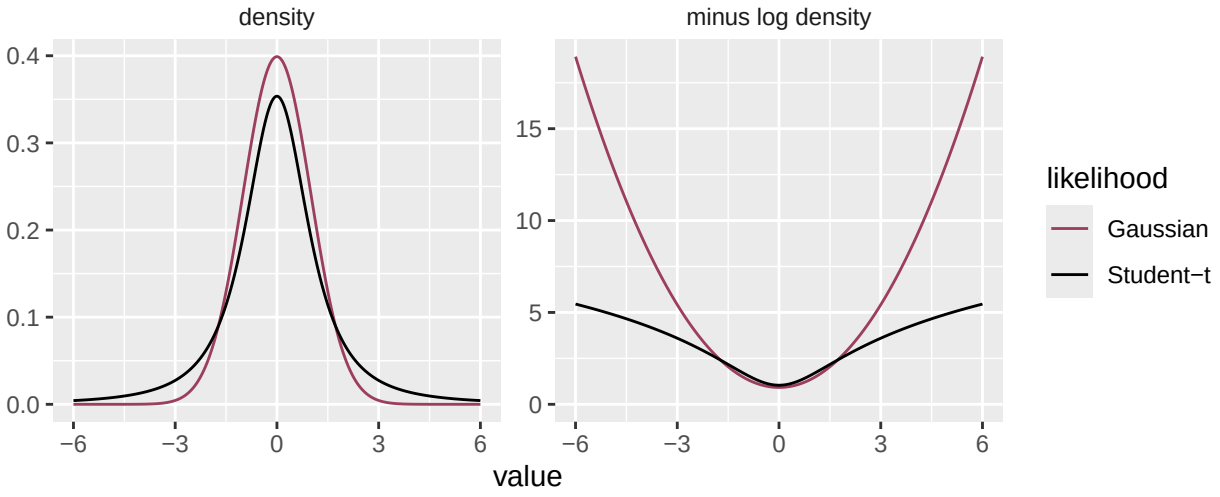
Our p_{WAIC} was about 4.6, which is still a little high due to Idaho but not quite as high as in the text. There is some concern that Idaho is putting us at risk of overfitting due to the large influence it has on the posterior.

> What can be done about this? There is a tradition of dropping outliers. People sometimes drop outliers even before a model is fit, based only on standard deviations from the mean outcome value. You should never do that–a point can only be unexpected and highly influential in light of a model. After you fit a model, the picture changes. If there are only a few outliers, and you are sure to report results both with and without them, dropping outliers might be okay. But if there are several outliers and we really need to model them, what then?
>
> A basic problem here is that the Gaussian error model is easily surprised. (p. 232)

This surprise has to do with the thinness of its tails relative to those of the Student's t-distribution. We can get a sense of this with Figure 7.11.

```
tibble(x = seq(from = -6, to = 6, by = 0.01)) %>%
  mutate(Gaussian    = dnorm(x),
         `Student-t` = dstudent(x)) %>%
  pivot_longer(-x,
               names_to = "likelihood",
               values_to = "density") %>%
  mutate(`minus log density` = -log(density)) %>%
  pivot_longer(contains("density")) %>%

  ggplot(aes(x = x, y = value, group = likelihood, color = likelihood)) +
  geom_line() +
  scale_color_manual(values = c(carto_pal(7, "BurgYl")[6], "black")) +
  ylim(0, NA) +
  labs(x = "value", y = NULL) +
  theme(strip.background = element_blank()) +
  facet_wrap(~ name, scales = "free_y")
```



I'm a little baffled as to why our curves don't quite match up with those in the text. The Gaussian curves were made using the `dnorm()` function with the default settings, which are Normal$(0, 1)$. The Student-t curves were made with McElreath's `rethinking::dstudent()`function with the default settings, which are Student-t$(2, 0, 1)$–you get the same results is you use `dt(x, df = 2)`. Discrepancies aside, the main point in the text still stands. The t distribution, especially as $\nu \to 1$, has thicker tails than the Gaussian. As a consequence, it is more robust to extreme values.

The **brms** package will allow us to fit a Student t model. Just set `family = student`. We are at liberty to estimate ν along with the other parameters in the model or to set it to a constant. We'll follow McElreath and fix `nu = 2`. Note that **brms** syntax requires we do so within a `bf()` statement.

```
b5.3t <-
  brm(data = d,
      family = student,
      bf(d ~ 1 + m + a, nu = 2),
      prior = c(prior(normal(0, 0.2), class = Intercept),
                prior(normal(0, 0.5), class = b),
```

```
                prior(exponential(1), class = sigma)),
      iter = 2000, warmup = 1000, chains = 4, cores = 4,
      seed = 5,
      refresh = 1000,
      file = "fits/b05.03t")
```

Check the summary.

```
print(b5.3t)
```

```
##  Family: student
##   Links: mu = identity; sigma = identity; nu = identity
## Formula: d ~ 1 + m + a
##          nu = 2
##    Data: d (Number of observations: 50)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Regression Coefficients:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     0.03      0.10    -0.17     0.21 1.00     4156     2655
## m             0.05      0.20    -0.31     0.47 1.00     3628     2714
## a            -0.70      0.15    -0.98    -0.41 1.00     3587     3116
##
## Further Distributional Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.58      0.09     0.43     0.77 1.00     4197     2735
## nu        2.00      0.00     2.00     2.00   NA       NA       NA
##
## Draws were sampled using sample(hmc). 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).
```

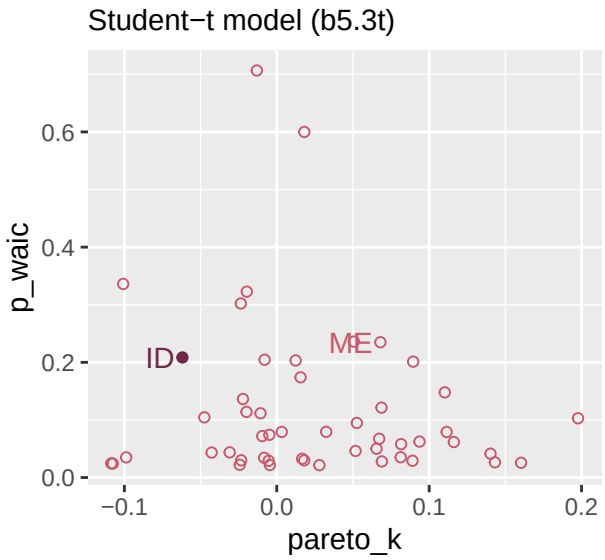
Here we use the `add_criterion()` function compute both the LOO-CV and the WAIC and add them to the model fit object.

```
b5.3t <- add_criterion(b5.3t, criterion = c("loo", "waic"))
```

Now we might remake the plot from Figure 7.10, this time based on the t model.

```
tibble(pareto_k = b5.3t$criteria$loo$diagnostics$pareto_k,
       p_waic   = b5.3t$criteria$waic$pointwise[, "p_waic"],
       Loc      = pull(d, Loc)) %>%

  ggplot(aes(x = pareto_k, y = p_waic, color = Loc == "ID")) +
  geom_point(aes(shape = Loc == "ID")) +
  geom_text(data = . %>% filter(Loc %in% c("ID", "ME")),
            aes(x = pareto_k - 0.005, label = Loc),
            hjust = 1) +
  scale_color_manual(values = carto_pal(7, "BurgYl")[c(5, 7)]) +
  scale_shape_manual(values = c(1, 19)) +
  labs(subtitle = "Student-t model (b5.3t)") +
  theme(legend.position = "none")
```



The high points in both the Pareto k and p_{WAIC} are much smaller and both ID and ME are now closer to the center of the bivariate distribution. We can formally compare the models with the WAIC and the LOO.

```
loo_compare(b5.3, b5.3t, criterion = "waic") %>% print(simplify = F)
```

```
##       elpd_diff se_diff elpd_waic se_elpd_waic p_waic se_p_waic waic  se_waic
## b5.3    0.0       0.0   -63.7       6.3          4.6    1.8     127.5  12.7
## b5.3t  -2.5       2.9   -66.3       5.8          6.0    1.0     132.5  11.6
```

```
loo_compare(b5.3, b5.3t, criterion = "loo") %>% print(simplify = F)
```

```
##       elpd_diff se_diff elpd_loo se_elpd_loo p_loo se_p_loo looic se_looic
## b5.3    0.0       0.0   -63.8      6.4         4.7   1.9    127.7  12.8
## b5.3t  -2.4       2.9   -66.2      5.8         5.9   1.0    132.5  11.5
```

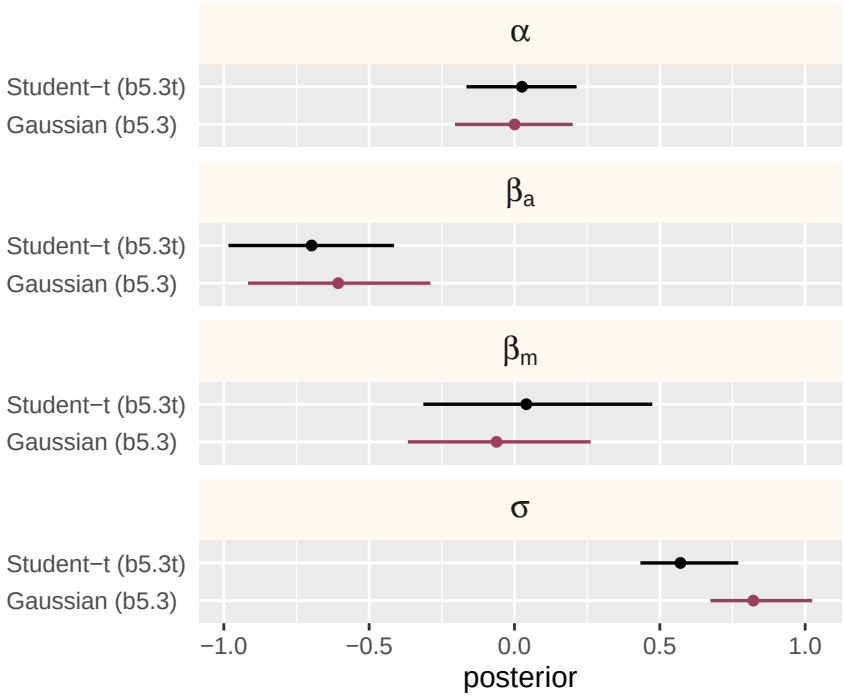
For both criteria, the standard error for the difference was about the same size as the difference itself. This suggests the models were close and hard to distinguish with respect to their fit to the data. This will not always be the case.

Just for kicks and giggles, let's compare the parameter summaries for the two models in a coefficient plot.

```
bind_rows(as_draws_df(b5.3),
          as_draws_df(b5.3t)) %>%
  mutate(fit = rep(c("Gaussian (b5.3)", "Student-t (b5.3t)"), each = n() / 2)) %>%
  pivot_longer(b_Intercept:sigma) %>%
  mutate(name = factor(name,
                       levels = c("b_Intercept", "b_a", "b_m", "sigma"),
                       labels = c("alpha", "beta[a]", "beta[m]", "sigma"))) %>%

  ggplot(aes(x = value, y = fit, color = fit)) +
  stat_pointinterval(.width = .95, size = 1) +
  scale_color_manual(values = c(carto_pal(7, "BurgYl")[6], "black")) +
  labs(x = "posterior", y = NULL) +
  theme(axis.text.y = element_text(hjust = 0),
```

```
        axis.ticks.y = element_blank(),
        legend.position = "none",
        strip.background = element_rect(fill = alpha(carto_pal(7, "BurgYl")[1], 1/4), color = "transpar
        strip.text = element_text(size = 12)) +
  facet_wrap(~ name, ncol = 1, labeller = label_parsed)
```



Overall, the coefficients are very similar between the two models. The most notable difference is in σ. This is, in part, because σ has a slightly different meaning for Student-t models than it does for the Gaussian. For both, we can refer to σ as the *scale* parameter, with is a measure of spread or variability around the mean. However, the scale is the same thing as the standard deviation only for the Gaussian. Kruschke walked this out nicely:

> It is important to understand that the scale parameter σ in the t distribution is not the standard deviation of the distribution. (Recall that the standard deviation is the square root of the variance, which is the expected value of the squared deviation from the mean, as defined back in Equation 4.8, p. 86.) The standard deviation is actually larger than σ because of the heavy tails. In fact, when ν drops below 2 (but is still ≥ 1), the standard deviation of the mathematical t distribution goes to infinity. [-@kruschkeDoingBayesianData2015, p. 159]