Exercises week 8

prof. Richard Torkar

Oct. 17, 2021

> Ich habe meine Ergebnisse schon lange gehabt: aber ich weiß noch nicht, wie ich zu ihnen kommen soll.
>
> Carl Friedrich Gauss

Here we'll introduce another tool, `brms`, which many, if not all of you, will use instead of `rethinking`, when you do your theses. This tool allows you to write very complex model specifications in a simple way. However, with power comes great responsibility. It's important that we understand what the models imply, what our priors mean, and that we conduct prior and posterior predictive checks.

The course book has been translated to `brms`: `https://bookdown.org/content/4857/`

We will create a varying intercepts *and* varying slopes model and we'll compare it with a varying intercepts model. We will use the same data as previous week, but now we'll use a new predictor `devs` as a slope (`devs` stands for the total number of developers committing code to the project).

First, download and clean data (you might have already done that, so reuse the data from last week in that case),

```
library(tidyverse) # much data wrangling so install this
# the below should be on one line!
url <- "https://raw.githubusercontent.com/torkar/BDA-PL/main/
    docs/utils.R"
destFile <- "~/Downloads/utils.R"
download.file(url, destFile)
# I have no idea where you put it so change below if needed
source("~/Downloads/utils.R")
setup.data()
d <- load.FSE(cleanup=TRUE)
d <- by.project.language(d)
fse.data <- d # save orig data
d$devs_log <- log(d$devs) # log the var since magnitude matters
d$lang_id <- as.integer(d$language) # make int
d <- d[ , -c(1:5,7:9)] # rm columns not needed
```

Next, install some new packages and load them.

```
# first run the below line if you haven't installed these
# install.packages(c("bayesplot", "brms", "ggplot2"))
library(ggplot2) # plotting
library(bayesplot) # for plotting Bayesian stuff
library(brms) # lme4 syntax when specifying models
```

Now let's specify a varying intercepts model. In math notation, according to McElreath, it would look something like this,

$$
\begin{aligned}
\text{n\_bugs}_i &\sim \text{Negative-Binomial}(\lambda_i, \phi) \\
\log(\lambda_i) &= \alpha_{\text{LANG\_ID}[i]} && \text{varying intercept} \\
\alpha_j &\sim \text{Normal}(\bar{\alpha}, \sigma) && \text{adaptive prior} \\
\bar{\alpha} &\sim \text{Normal}(0, 1.5) && \text{hyperprior} \\
\sigma &\sim \text{Weibull}(2, 1) && \text{hyperprior} \\
\phi &\sim \text{Gamma}(0.01, 0.01) && \text{shape parameter}
\end{aligned}
$$

In `brms`'s `lme4` syntax, the second line is written as,

```
n_bugs ~ 1 + (1 | lang_id)
```

Let's now set priors also, and do prior predictive checks,

```
# First check which priors should be set
p <- get_prior(n_bugs ~ 1 + (1 | lang_id),
               family = negbinomial,
               data = d)
p
```

So, these are the priors we must set (and we see some default priors already set),

```
                  prior     class      coef   group  ...
 student_t(3, 4.2, 2.5) Intercept                    ...
   student_t(3, 0, 2.5)        sd                    ...
   student_t(3, 0, 2.5)        sd           lang_id  ...
   student_t(3, 0, 2.5)        sd Intercept lang_id  ...
      gamma(0.01, 0.01)     shape                    ...
```

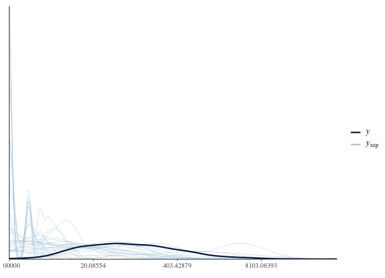
We need to set a prior on the Intercept, $\bar{\alpha}$, and the standard deviation σ. The shape prior we can keep since it's a default prior.

```
p$prior[1] <- "normal(0,1.5)"
p$prior[2] <- "weibull(2,1)" # same for all sd params
```

Now we sample *only* from the priors to do prior predictive checks,

```
m0 <- brm(n_bugs ~ 1 + (1 | lang_id),
          family = negbinomial,
          data = d,
          sample_prior = "only", # note this!
          prior = p,  # and here we use our priors!
          iter = 500, # enough for now
          chains = 1  # no need to run more
          )

pp_check(m0, nsamples = 250) + scale_x_continuous(trans="log")
```

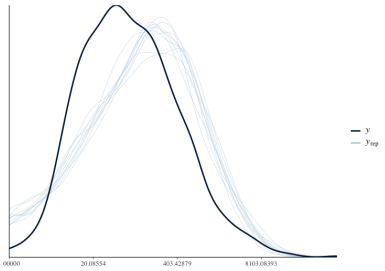


Clearly we allow some absurd values (look to the right). Let's go with these priors for now and sample with data.

```
m0 <- brm(n_bugs ~ 1 + (1 | lang_id),
          family = negbinomial,
          data = d,
          iter = 5e3, # up the samples
          prior = p)
```

```
max(rhat(m0), na.rm = TRUE) # check max Rhat value
min(neff_ratio(m0), na.rm = TRUE) # check min ESS

# posterior predictive check
pp_check(m0) + scale_x_continuous(trans="log")
```



The last line plots our empirical data against what the model predicts. We clearly see that our model skews to the right.

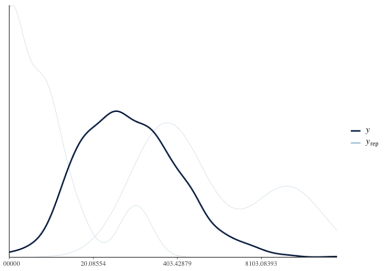
VARYING INTERCEPTS AND SLOPES could perhaps capture the characteristics of the data better? Let's use the same procedure as above.

```
p <- get_prior(n_bugs ~ 1 + devs_log + (1 + devs_log | lang_id),
               family = negbinomial,
               data = d)
p$prior[1] <- "normal(0,1)"
p$prior[3] <- "lkj(2)" # see pp. 442!
p$prior[5] <- "normal(0,2.5)"
p$prior[6] <- "weibull(2,1)"
```

Note the formula specification above, `1 + devs_log + (1 + devs_log | lang_id)`. Page 441 in the book lists such a model. What's important here is that `devs_log` is a β parameter first, and then we use it as a varying slopes component `(1 + devs_log |...` We also need to add a special prior when dealing with covariance matrices, i.e., the Lewandowski-Kurowicka-Joe (LKJ) prior. A value of 2 simply states that we are somewhat skeptical towards extreme correlations.

```
m1 <- brm(n_bugs ~ 1 + devs_log + (1 + devs_log | lang_id),
          family = negbinomial,
          data = d,
          sample_prior = "only",
          iter = 500,
          chains = 1,
          prior = p)

pp_check(m1) + scale_x_continuous(trans="log")
```

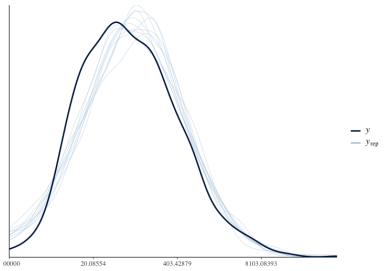


Once again we see that the priors allow a range of values so let's stick with this for now.

```
m1 <- brm(n_bugs ~ 1 + devs_log + (1 + devs_log | lang_id),
          family = negbinomial,
          data = d,
          iter = 5e3,
          prior = p)

max(rhat(m1), na.rm = TRUE)
min(neff_ratio(m1), na.rm = TRUE)

pp_check(m1) + scale_x_continuous(trans="log")
```



There's still some skew, but it is better (see to the right and compare with the top plot). Let's use LOO and see how the models compare.

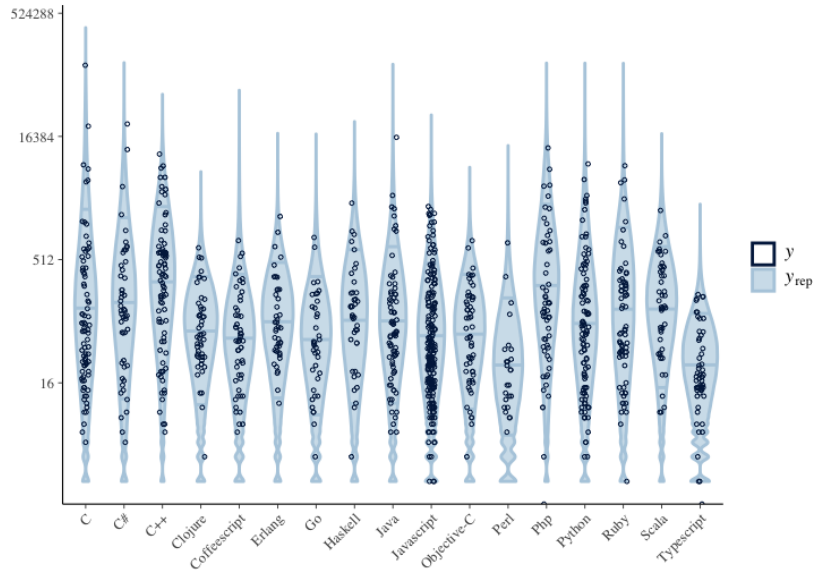
```
m0 <- add_criterion(m0, "loo")
m1 <- add_criterion(m1, "loo")
loo_compare(m0, m1) # you'll get some warnings but it's ok
```

And here's the output,

```
   elpd_diff se_diff
m1     0.0       0.0
m0  -521.9      50.7
```

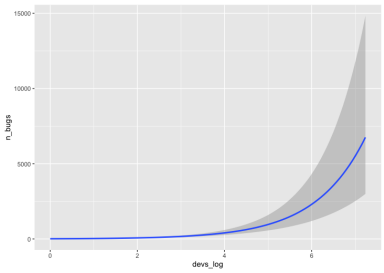
where we see that we are > 10 ΔSE away. In short, $\mathcal{M}_1$ is superior. Let's check what the results imply,

```
pp_check(m1, type = "violin_grouped", group = "lang_id", y_draw
    ="points") +
  scale_y_continuous(trans = "log2") +
  scale_x_discrete(labels=levels(fse.data$language)) +
  theme(axis.text.x = element_text(angle=45, hjust=1))
```



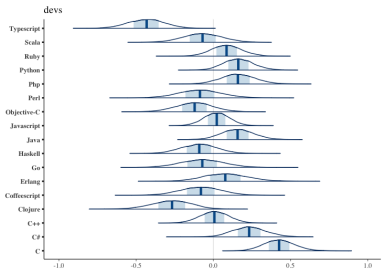
Clearly some languages are 'better', i.e., having lower values. Let's plot the conditional effect of `devs` to see how it varies,

```
conditional_effects(m1)
```



To the right we see that the more `devs` the more bugs, but also the more uncertainty. Finally, let's plot another version of the violin plot where we simply use the densities of each estimate.

```
mcmc_areas(m1, regex_pars = "devs_log]") +
  scale_y_discrete(labels=levels(fse.data$language)) +
  ggtitle("devs")
```



According to out model, there's only one significant language, i.e., `C`, when accounting for `devs`. It has significantly more bugs than any other language (it doesn't cross zero).

CAN WE IMPROVE THE MODEL? Well, yes. We have a number of variables we could add (we removed all variables we didn't need in the beginning). They could be added as varying slopes...

GAUSSIAN PROCESSES ($\mathcal{GP}$) can be quite hard to get right. Additionally, they are often computationally heavy so it would be nice if we could benefit from some optimizations. Thankfully Paul Bürkner's `brms` makes using $\mathcal{GP}$ a bit easier. Additionally, he has optimized `brms` to make it faster.

Here we will go through a simple case (it's in a paper which I conducted the analysis for).[1] The case consists of data from software engineers filling out a survey (instrument) on a daily and weekly basis. As you will see in the analysis, modeling a $\mathcal{GP}$ is relatively straightforward in `brms`. I would recommend you to look at Sect. 3.1–3.2 that cover the weekly and daily analysis, respectively.

[1] `https://rpubs.com/torkar/713862`

In the two sections you will see how I apply exactly what I have taught you in this course,

1. Prior predictive checks.
2. Posterior predictive checks.
3. Check diagnostics.
4. Check parameter estimates.
5. Conduct inference if needed.

This case, together with other case, could serve as examples on how to do these types of analyses, but also on how to report them.

- Analysis of telecom data and feature selection concerning requirements prioritization: `https://github.com/torkar/feature-selection-RBS`
- Bayesian analysis of programming language data: `https://github.com/torkar/BDA-PL`
- Bayesian analysis from an experiment (part of a MSc thesis): `https://github.com/torkar/affective_states`
- How to connect Bayesian data analysis to utility theory, i.e., practical significance: `https://github.com/torkar/docker-b3`
- Replication package for a book chapter on handling missing data in Bayesian data analysis: `https://github.com/torkar/BDA_in_ESE`

In all of the above cases you will find replication packages published online.