Notes on Project 02

BJ

11/15/2021

I just want to include some notes here that are related to problems students are having completing a rough draft technical appendix for HW10 / Project 02. I'll address three separate issues:

1. Calculating ICC's as a measure of rater agreement.
2. Constructing tables of counts cross-classifying the ratings that each pair of raters gives.
3. Dealing with the fact that `lmer()` is a bit fragile, giving off errors and warnings when fitting certain models.

1. Calculating ICC's as a measure of rater agreement.

To calculate an ICC (intra-class correlation, which is an older phrase for "within-group correlation"), we need to fit the random-intercept model

$$\begin{aligned} y_i &= \alpha_{j[i]} + \epsilon_i, \epsilon_i \sim N(0, \sigma^2) \\ \alpha_j &= \beta_0 + \eta_j, \eta_j \sim N(0, \tau^2) \end{aligned}$$

or in R's model formula language

```
y ~ 1 + (1|group)
```

The ICC is then

$$ICC = \frac{\tau^2}{\tau^2 + \sigma^2}$$

using estimates of σ^2 and τ^2 from the fitted model. As we know from homework

$$ICC = \text{Corr}\left(y_i, y_{i'}\right)$$

where i and i' come from the same group. See also Figure 1, which shows J groups: the ICC is the correlation between any two y's from the same group.

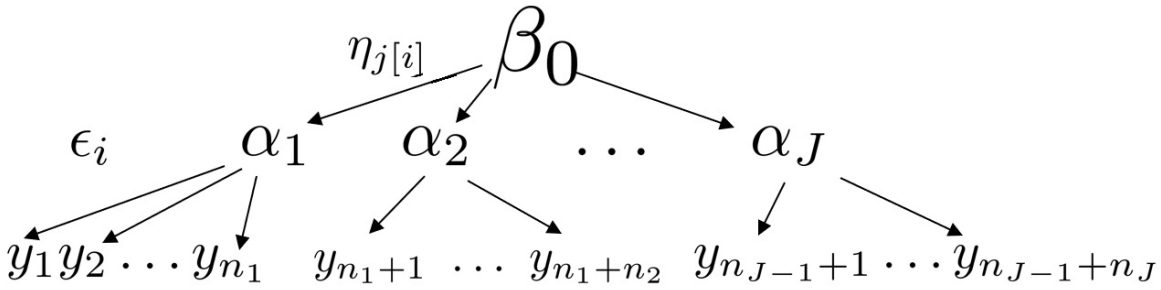

Figure 1: The ICC is the correlation between any two y's in the same group.

The key here is to figure out how to group the ratings to get what we want.

For the rating data in Project 02, the relevant data is the part of the tall data for the 13 artifacts that all three raters saw:

```
tall <- read.csv("tall.csv", header=T)
names(tall)
```

```
## [1] "X"        "Rater"    "Artifact" "Repeated" "Semester" "Sex"      "Rubric"
## [8] "Rating"
```

```
common <- tall[grep("O",tall$Artifact),]
head(common)
```

```
##     X Rater Artifact Repeated Semester Sex Rubric Rating
## 1   1     3       O5        1      F19   M RsrchQ      3
## 2   2     3       O7        1      F19   F RsrchQ      3
## 3   3     3       O9        1      S19   F RsrchQ      2
## 4   4     3       O8        1      S19   M RsrchQ      2
## 10 10     3      O10        1      F19   F RsrchQ      2
## 11 11     3      O13        1      F19   M RsrchQ      2
```

```
dim(common)
```

```
## [1] 273   8
```

Suppose we extract from this all data from one rubric

```
RsrchQ.ratings <- common[common$Rubric=="RsrchQ",]
```

and then fit the model

```
lmer(Rating ~ 1 + (1|Rater), data=RsrchQ.ratings)
```

Then, in Figure 1, there would be 3 groups, one for each rater, and the y's in each group are the 13 artifact ratings that that rater made (you can check this with a command like `table(RsrchQ.ratings$Rater)`.). In that case, the ICC is going to be the correlation between ratings on any two different artifacts by the same rater. *We would expect this correlation to be low*, since knowing the rating on one student's artifact shouldn't be a good predictor of the rating on another student's artifact.

On the other hand, if we try to fit the model

```
lmer(Rating ~ 1 + (1|Artifact), data=RsrchQ.ratings)
```

now there will be 13 groups, one for each artifact, and three y's per group: one for each rater's rating on that artifact. Now the ICC is the correlation between any two rater's ratings on the same artifact. If the raters are consistent with one another in how they rate, we would expect this correlation to be higher. Moreover, *this between-raters correlation does tell us something useful about rater agreement*: raters agree more when their correlations are higher.

You'll want to do something like this for each of the seven rubrics.

2. Cross-classifying the ratings that each pair of raters gives.

To compute exact agreement rates between raters (and also to have an idea of how severe disagreements can be) it is useful to create tables of counts cross-classifying the rating that each pair of raters gives. It's easiest to start with the original, wide data, but again we want to subset to (a) just the 13 artifacts seen by everyone, and (b) just the data related to a single rubric at a time:

```
ratings <- read.csv("ratings.csv",header=TRUE)
repeated <- ratings[ratings$Repeated==1,]

raters_1_and_2_on_RsrchQ <- data.frame(r1=repeated$RsrchQ[repeated$Rater==1],
```

```
                                        r2=repeated$RsrchQ[repeated$Rater==2],
                                        a1=repeated$Artifact[repeated$Rater==1],
                                        a2=repeated$Artifact[repeated$Rater==2]
                                       )
```

It is useful to check to make sure that the artifacts are in the same order for both raters; you could check this with `View(raters_1_and_2_on_RsrchQ)`.

If we just naively use the `table()` command, we don't get the full table, and we can't easily assess rater agreement:

```
with(raters_1_and_2_on_RsrchQ,table(r1,r2))
```

```
##    r2
## r1  1 2 3
##   2 1 4 3
##   3 1 3 1
```

But we can supply the missing rating levels as follows:

```
r1 <- factor(raters_1_and_2_on_RsrchQ$r1,levels=1:4)
r2 <- factor(raters_1_and_2_on_RsrchQ$r2,levels=1:4)

(t12 <- table(r1,r2))
```

```
##    r2
## r1  1 2 3 4
##   1 0 0 0 0
##   2 1 4 3 0
##   3 1 3 1 0
##   4 0 0 0 0
```

and then compute exact agreement from the table `t12`.

You can create similar tables `t23` and `t13` for the other rater pairs for this rubric, and then repeat for the other rubrics.

3. Dealing with fragile `lmer()` fits

The likelihoods for mixed effects linear models quickly become complex and difficult to maximize; the same can be true of the REML objective function.

If you fit something like

```
m4 <- lmer(...)
```

and you get a lot of warnings (and possibly errors) about singular matrices, nonconvergence, or other complaints, you can change the way R does the maximization. For example to use the `bobyqa` maximizer, you would say something like

```
m4 <- lmer(...)

ss <- getME(m4,c("theta","fixef"))

m4u<- update(m4,start=ss,
             control=lmerControl(optimizer="bobyqa",
                                 optCtrl=list(maxfun=2e5)))
```

Note that I've also specified the maximum number of iterations before R gives up, `maxfun`. The default is `maxfun=10000` (=1e4); I've increased `maxfun` by a factor of 20 above.

Other maximizers you can try include

```
bobyqa
nloptwrap       <-- this is the one lmer uses by default
Nelder_Mead
optimx.nlminb
optimx.L-BFGS-B
nloptwrap.NLOPT_LN_NELDERMEAD
nloptwrap.NLOPT_LN_BOBYQA
```

You can learn something about these maximizers (and other ways to improve `lmer()` performance) from `https://cran.r-project.org/web/packages/lme4/vignettes/lmerperf.html`, but the main thing to do is try different things until (hopefully) `lmer()` behaves better (no more complaints; sensible answers) on your model.

Some additional advice can be found here:

A. lme4's advice on interpreting and fixing convergence errors in lmer & glmer:

```
library(lme4)
help("convergence")
```

B. Practical advice from a longtime lmer user:

`http://svmiller.com/blog/2018/06/mixed-effects-models-optimizer-checks/`

C. Some additional advice about "false positive" warning messages from lmer:

`https://stackoverflow.com/questions/33670628/solution-to-the-warning-message-using-glmer`

I hope this helps,

-BJ