How to extract different information from fitted lmer models, a reference

Miratrix

August 16, 2016

Introduction

This document walks through various R code to pull information out of a multilevel model (and OLS models as well, since the methods generally work on everything). For illustration, we will use a random-slope model on the HS&B dataset with some level 1 and level 2 fixed effects.

Libraries

We use the following libraries in this file:

```
library( lme4 )
library( foreign ) # to load data
library( arm )
library( tidyverse )
```

Loading the data

Loading the data is simple. We read student and school level data and merge:

```
dat = read.spss( "hsb1.sav", to.data.frame=TRUE )
sdat = read.spss( "hsb2.sav", to.data.frame=TRUE )
```

```
re-encoding from CP1252
```

```
dat = merge( dat, sdat, by="id", all.x=TRUE )
head( dat, 3 )
```

```
    id minority female    ses mathach size sector pracad disclim himinty
1 1224        0      1 -1.528   5.876  842      0   0.35   1.597       0
2 1224        0      1 -0.588  19.708  842      0   0.35   1.597       0
3 1224        0      0 -0.528  20.349  842      0   0.35   1.597       0
  meanses
1  -0.428
2  -0.428
3  -0.428
```

Fitting and viewing the model

Now we fit the random slope model with the level-2 covariates:

```
M1 = lmer( mathach ~ 1 + ses + meanses + (1 + ses|id), data=dat )
```

To get an overview of what our fitted model is, use `arm`'s `display()` method:

```r
display( M1 )
```

```
lmer(formula = mathach ~ 1 + ses + meanses + (1 + ses | id),
    data = dat)
            coef.est coef.se
(Intercept) 12.65     0.15
ses          2.19     0.12
meanses      3.78     0.38

Error terms:
 Groups   Name        Std.Dev. Corr
 id       (Intercept) 1.64
          ses         0.67     -0.21
 Residual             6.07
---
number of obs: 7185, groups: id, 160
AIC = 46575.4, DIC = 46552.4
deviance = 46556.9
```

The `summary()` method

We can also look at the messier default `summary()` command, which gives you more output. The real win is if we use the `lmerTest` library and fit our model with that package loaded, our `summary()` is more exciting and has p-values:

```r
library( lmerTest )
M1 = lmer( mathach ~ 1 + ses + meanses + (1 + ses|id), data=dat )
summary( M1 )
```

```
Linear mixed model fit by REML t-tests use Satterthwaite approximations
  to degrees of freedom [lmerMod]
Formula: mathach ~ 1 + ses + meanses + (1 + ses | id)
   Data: dat

REML criterion at convergence: 46561.4

Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.1671 -0.7270  0.0163  0.7547  2.9646

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 id       (Intercept)  2.6953  1.6417
          ses          0.4531  0.6731   -0.21
 Residual             36.7956  6.0659
Number of obs: 7185, groups:  id, 160

Fixed effects:
            Estimate Std. Error       df t value Pr(>|t|)
(Intercept)  12.6513     0.1506 152.9600  84.000   <2e-16 ***
ses           2.1903     0.1218 178.2100  17.976   <2e-16 ***
meanses       3.7812     0.3826 181.7700   9.883   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Correlation of Fixed Effects:
        (Intr) ses
ses     -0.080
meanses -0.028 -0.256
```

If we just print the object, e.g., by typing the name of the model on the console, we get minimal information:

```r
M1
```

```
Linear mixed model fit by REML ['merModLmerTest']
Formula: mathach ~ 1 + ses + meanses + (1 + ses | id)
   Data: dat
REML criterion at convergence: 46561.42
Random effects:
 Groups   Name        Std.Dev. Corr
 id       (Intercept) 1.6417
          ses         0.6731   -0.21
 Residual             6.0659
Number of obs: 7185, groups:  id, 160
Fixed Effects:
(Intercept)          ses      meanses
     12.651        2.190        3.781
```

Obtaining Fixed Effects

R thinks of models in reduced form. Thus when we get the fixed effects we get both the level-1 and level-2 fixed effects

```r
fixef( M1 )
```

```
(Intercept)         ses     meanses
  12.651300    2.190350    3.781221
```

The above is a vector of numbers. Each element is named, but we can index them as so:

```r
fixef( M1 )[2]
```

```
    ses
2.19035
```

We can also use the `[[]]` which means "give me that element not as a list but as just the element!" When in doubt, if you want one thing out of a list or vector, use `[[]]` instead of `[]`:

```r
fixef( M1 )[[2]]
```

```
[1] 2.19035
```

See how it gives you the number without the name here?

Variance and Covariance estimates of Random Effects

We can get the Variance-Covariance matrix of the random effects with `VarCorr`.

```r
VarCorr( M1 )
```

```
 Groups   Name        Std.Dev. Corr
 id       (Intercept) 1.6417
          ses         0.6731   -0.212
 Residual             6.0659
```

It displays nicely if you just print it out, but inside it are covariance matrices for each random effect group. (In our model we only have one group, `id`.) These matrices also have correlation matrices for reference. Here is how to get these pieces:

```
vc = VarCorr( M1 )$id
vc
```

```
            (Intercept)        ses
(Intercept)    2.695317 -0.2339210
ses           -0.233921  0.4530689
attr(,"stddev")
(Intercept)         ses
   1.641742    0.673104
attr(,"correlation")
            (Intercept)        ses
(Intercept)   1.0000000 -0.2116811
ses          -0.2116811  1.0000000
```

You might be wondering what all the `attr` stuff is. R can "tack on" extra information to a variable via "attributes". Attributes are not part of the variable exactly, but they follows their variable around. The `attr` (for attribute) method is a way to get these extra bits of information. In the above, R is tacking the correlation matrix on to the variance-covariance matrix to save you the trouble of calculating it yourself. Get it as follows:

```
attr( vc, "correlation" )
```

```
            (Intercept)        ses
(Intercept)   1.0000000 -0.2116811
ses          -0.2116811  1.0000000
```

You can also just use the `vc` object as a matrix. Here we take the diagonal of it

```
diag( vc )
```

```
(Intercept)         ses
  2.6953168   0.4530689
```

If you want an element from a matrix use row-column indexing like so:

```
vc[1,2]
```

```
[1] -0.233921
```

for row 1 and column 2.

The `sigma.hat()` and `sigma()` methods

If you just want the variances and standard deviations of your random effects, use `sigma.hat()`. This also gives you the residual standard deviation as well. The output is a weird object, with a list of things that are themselves lists in it. Let's examine it. First we look at what the whole thing is:

```
sigma.hat( M1 )
```

```
$sigma
$sigma$data
```

```
[1] 6.065939

$sigma$id
(Intercept)         ses
   1.641742    0.673104


$cors
$cors$data
[1] NA

$cors$id
            (Intercept)        ses
(Intercept)   1.0000000 -0.2116811
ses          -0.2116811  1.0000000
```

```
names( sigma.hat( M1 ) )
```

```
[1] "sigma" "cors"
```

```
sigma.hat( M1 )$sigma
```

```
$data
[1] 6.065939

$id
(Intercept)         ses
   1.641742    0.673104
```

Our standard deviations of the random effects are

```
sigma.hat( M1 )$sigma$id
```

```
(Intercept)         ses
   1.641742    0.673104
```

We can get our residual variance by this weird thing (we are getting `data` from the `sigma` inside of `sigma.hat( M1 )`):

```
sigma.hat( M1 )$sigma$data
```

```
[1] 6.065939
```

But here is an easier way using the `sigma()` utility function:

```
sigma( M1 )
```

```
[1] 6.065939
```

Obtaining Emperical Bayes Estimates of the Random Effects

Random effects come out of the `ranef()` method. Each random effect is its own object inside the returned object. You refer to these sets of effects by name. Here our random effect is called `id`.

```
ests = ranef( M1 )$id
head( ests )
```

```
     (Intercept)         ses
```

```
1224 -0.26204176  0.08765692
1288  0.03805001  0.11842355
1296 -1.91525421  0.03572786
1308  0.30485857 -0.10501005
1317 -1.15834629 -0.10815425
1358 -0.98212769  0.44614647
```

Generally, what you get back from these calls is a new data frame with a row for each group. The rows are named with the original id codes for the groups, but if you want to connect it back to your group-level information you are going to want to merge stuff. To do this, and to keep things organized, I recommend adding the id as a column to your dataframe:

```
names(ests) = c( "u0", "u1" )
ests$id = rownames( ests )
head( ests )
```

```
              u0          u1   id
1224 -0.26204176  0.08765692 1224
1288  0.03805001  0.11842355 1288
1296 -1.91525421  0.03572786 1296
1308  0.30485857 -0.10501005 1308
1317 -1.15834629 -0.10815425 1317
1358 -0.98212769  0.44614647 1358
```

We also renamed our columns of our dataframe to give them names nicer than `(Intercept)`. You can use these names if you wish, however. You just need to quote them with back ticks (this code is not run):

```
head( ests$`(Intercept)` )
```

The `coef()` method

We can also get a slighly different (but generally easier to use) version these things through `coef()`. What `coef()` does is give you the estimated regression lines for each group in your data by combining the random effect for each group with the corresponding fixed effects. Note how in the following the `meanses` coefficient is the same, but the others vary due to the random slope and random intercept.

```
coefs = coef( M1 )$id
head( coefs )
```

```
     (Intercept)      ses  meanses
1224    12.38926 2.278007 3.781221
1288    12.68935 2.308773 3.781221
1296    10.73605 2.226078 3.781221
1308    12.95616 2.085340 3.781221
1317    11.49295 2.082196 3.781221
1358    11.66917 2.636496 3.781221
```

Note that if we have level 2 covariates in our model, they are not incorperated in the intercept and slope via `coef()`. We have to do that by hand:

```
names( coefs ) = c( "beta0.adj", "beta.ses", "beta.meanses" )
coefs$id = rownames( coefs )
coefs = merge( coefs, sdat, by="id" )
coefs = mutate( coefs, beta0 = beta0.adj + beta.meanses * meanses )
coefs$beta.meanses = NULL
```

Here we added in the impact of mean ses to the intercept (as specified by our model). Now if we look at the intercepts (the beta0 variables) they will incorperate the level 2 covariate effects. If we then plotted a line using beta0 and beta.ses for each school, we would get the estimated lines for each school including the school-level covariate impacts.

Standard errors

We can get an object with all the standard errors of the coefficients, including the individual Emperical Bayes estimates for the individual random effects. This is a lot of information. We first look at the Standard Errors for the fixed effects, and then for the random effects. Standard errors for the variance terms are not given (this is tricker to calculate).

Fixed effect standard errors

```
ses = se.coef( M1 )
names( ses )
```

```
[1] "fixef" "id"
```

Our fixed effect standard errors:

```
ses$fixef
```

```
[1] 0.1506106 0.1218479 0.3826084
```

You can also get the uncertainty estimates of your fixed effects as a variance-covariance matrix:

```
vcov( M1 )
```

```
3 x 3 Matrix of class "dpoMatrix"
            (Intercept)         ses     meanses
(Intercept)  0.02268355 -0.00146548 -0.00161948
ses         -0.00146548  0.01484692 -0.01195418
meanses     -0.00161948 -0.01195418  0.14638920
```

The standard errors are the diagonal of this matrix, square-rooted. See how they line up?:

```
sqrt( diag( vcov( M1 ) ) )
```

```
[1] 0.1506106 0.1218479 0.3826084
```

Random effect standard errors

Our random effect standard errors for our EB estimates:

```
head( ses$id )
```

```
     (Intercept)       ses
1224   0.7845852 0.5804270
1288   0.9819216 0.6277229
1296   0.7779956 0.5766401
1308   1.0911711 0.6556742
1317   0.8045709 0.6188646
1358   0.9163541 0.6174061
```

Warning: these come as a matrix, not data frame. It is probably best to do this:

```
SEs = as.data.frame( se.coef( M1 )$id )
head( SEs )
```

```
     (Intercept)       ses
1224   0.7845852 0.5804270
1288   0.9819216 0.6277229
1296   0.7779956 0.5766401
1308   1.0911711 0.6556742
1317   0.8045709 0.6188646
1358   0.9163541 0.6174061
```

Confidence intervals and uncertainty

We can compute profile confidence intervals (warnings have been suppressed)

```
confint( M1 )
```

```
                 2.5 %     97.5 %
.sig01       1.4012799  1.8897549
.sig02      -0.8762352  0.1946822
.sig03       0.2044720  0.9849958
.sigma       5.9659922  6.1689341
(Intercept) 12.3559620 12.9462385
ses          1.9512025  2.4296954
meanses      3.0278219  4.5329237
```

Fitted values

Fitted values are the predicted value for each individual given the model.

```
yhat = fitted( M1 )
head( yhat )
```

```
        1         2         3         4         5         6
 7.290101  9.431427  9.568108  9.249187 10.410970 10.821012
```

Residuals are the difference between predicted and observed:

```
resids = resid( M1 )
head( resids )
```

```
         1          2          3          4          5          6
-1.4141011 10.2765725 10.7808921 -0.4681869  7.4870296 -6.2380116
```

We can also predict for hypothetical new data. Here we predict the outcome for a random student with ses of -1, 0, and 1 in a school with mean ses of 0:

```
ndat = data.frame( ses = c( -1, 0, 1 ), meanses=c(0,0,0), id = -1 )
predict( M1, newdata=ndat, allow.new.levels=TRUE )
```

```
       1        2        3
10.46095 12.65130 14.84165
```

The `allow.new.levels=TRUE` bit says to predict for a new school (our fake school id of `-1` in `ndat` above). In this case it assumes the new school is typical, with 0s for the random effect residuals.

If we predict for a current school, the random effect estimates are incorporated:

```
ndat$id = 1296
predict( M1, newdata=ndat )
```

```
        1         2         3
 8.509968 10.736046 12.962124
```

Appendix: the guts of the object

When we fit our model and store it in a variable, R stores *a lot* of stuff. The following lists some other functions that pull out bits and pieces of that stuff.

First, to get the model matrix (otherwise called the design matrix)

```
mm = model.matrix( M1 )
head( mm )
```

```
  (Intercept)    ses meanses
1           1 -1.528  -0.428
2           1 -0.588  -0.428
3           1 -0.528  -0.428
4           1 -0.668  -0.428
5           1 -0.158  -0.428
6           1  0.022  -0.428
```

This can be useful for predicting individual group mean outcomes, for example.

We can also ask questions such as number of groups, number of individuals:

```
ngrps( M1 )
```

```
 id
160
```

```
nobs( M1 )
```

```
[1] 7185
```

We can list all methods for the object (`merMod` is a more generic version of `lmerMod` and has a lot of methods we can use)

```
class( M1 )
```

```
[1] "merModLmerTest"
attr(,"package")
[1] "lmerTest"
```

```
methods(class = "lmerMod")
```

```
[1] coerce      coerce<-    display     getL        mcsamp      se.coef
[7] show        sim         standardize
see '?methods' for accessing help and source code
```

```
methods(class = "merMod")
```

```
 [1] anova        as.function  coef         confint      deviance
 [6] df.residual  display      drop1        extractAIC   extractDIC
[11] family       fitted       fixef        formula      fortify
[16] getL         getME        hatvalues    isGLMM       isLMM
```

```
[21] isNLMM       isREML       logLik       mcsamp       model.frame
[26] model.matrix ngrps        nobs         plot         predict
[31] print        profile      ranef        refit        refitML
[36] residuals    se.coef      show         sigma.hat    sigma
[41] sim          simulate     standardize  summary      terms
[46] update       VarCorr      vcov         weights
see '?methods' for accessing help and source code
```