Using 'buildmer' to automatically find & compare maximal (mixed) models

Cesko C. Voeten

27 Augustus 2019

Introduction

Barr, Levy, Scheepers, & Tily (2013) suggest that for valid statistical inference, a regression model must control for all possible confounding factors, specifically those coming from random effects such as subjects and items. Bates, Kliegl, Vasishth, & Baayen (2015) suggest that this proposed strategy leads to overfitting and that an appropriately-parsimonious model must be chosen, preferably based on theory but possibly also using stepwise elimination (Matuschek, Kliegl, Vasishth, Baayen, & Bates, 2017). Both strategies require a maximal model to be identified (for Barr et al. (2013), this is the final model; for Matuschek et al. (2017), this is the basis for backward stepwise elimination), but for many psycholinguistic experiments, the *truly* maximal model will fail to converge and a reasonable subset model needs to be chosen.

The `buildmer` package aims to automate the procedures identifying the maximal model that can still converge & performing backward stepwise elimination based on a variety of criteria (change in log-likelihood, AIC, BIC). The package does not contain any model-fitting code, but functions as an administrative loop around other packages by simply building up a maximal formula object and passing it along. Currently, the package supports models that can be fitted by `(g)lm`, `(g)lmer` (package `lme4`), `gls`, `lme` (package `nlme`), `gam`, `bam` (package `mgcv`), `gamm4` (package `gamm4`), `glmmTMB` (package `glmmTMB`), `multinom` (package `nnet`), `glmertree` (package `glmertree`), and it can use `JuliaCall` to drive Douglas Bates's `MixedModels` package for Julia.

A vowel study

To illustrate what `buildmer` can do for you, the package comes with a particularly pathological dataset called `vowels`. It looks like this:

```
library(buildmer)
head(vowels)
```

```
##   participant   word vowel neighborhood  timepoint       f1       f2
## 1           1 Keulen   oey      36.1961 0.05118788 407.8202 1838.611
## 2           1 Keulen   oey      36.1961 0.11941466 416.2701 1635.436
## 3           1 Keulen   oey      36.1961 0.18764143 450.6488 1654.561
## 4           1 Keulen   oey      36.1961 0.25586821 449.7890 1645.075
## 5           1 Keulen   oey      36.1961 0.32409498 444.6863 1606.261
## 6           1 Keulen   oey      36.1961 0.39232176 438.5311 1607.581
##   following information stress
## 1      lOns    4.511475   TRUE
## 2      lOns    4.511475   TRUE
## 3      lOns    4.511475   TRUE
## 4      lOns    4.511475   TRUE
## 5      lOns    4.511475   TRUE
## 6      lOns    4.511475   TRUE
```

This is a pilot study that I conducted when I was just starting my PhD, and attempted to analyze in probably the worst way possible. The research question was whether vowel diphthongization in the Dutch vowels

/eː,øː,oː,ɛi,œy/ was affected by syllable structure, such that an /l/ within the same syllable would block diphthongization but an /l/ in the onset of the next syllable would permit it. In plain English, the question was whether these five vowels in Dutch were pronounced like the vowel in English 'fear', with the tongue held constant for the duration of the vowel, or like the vowel in English 'fade', which has an upward tongue movement towards the position of the vowel in English 'fit'. The position of the tongue can be measured in a simple word-list reading experiment by measuring the speech signal's so-called 'first formant', labeled `f1` in this dataset, where lower F1 = higher tongue. Thus, the research question is if the F1 either changes or remains stable for the duration of each vowel depending on whether the following consonant is an 'l' in the same syllable (coded as `lCda` in column `following`) or in the next syllable (coded as `lOns`). Additionally, I wanted to control for the factors `neighborhood` (a measure of entropy: 'if only one sound is changed anywhere in this word, how many new words could be generated?'), `information` (another measure of entropy derived from the famous Shannon information measure), and `stress` (a dummy encoding whether the vowel was stressed or unstressed).

An entirely reasonable way to analyze these data, and the approach I ultimately pursued later in my PhD, would be to take samples from each vowel at 75% realization and at 25% realization, subtract these two, and use this 'delta score' as dependent variable: if this score is non-zero, the vowel changes over time, if it is approximately zero, the vowel was stable. In this dataset, however, I instead took as many samples as were present in the part of the wave file corresponding to these vowels, and wanted to fit a linear regression line through all of these samples as a function of the sample number. This number, scaled from 0 to 1 per token, is listed in column `timepoint`. To make the model even more challenging to fit, only six participants were tested in this pilot study, making it very difficult to find an optimum when including a full random-slope structure.

In `lme4` syntax, the fully maximal model would be given by the following formula:

```
f <- f1 ~ vowel*timepoint*following * neighborhood*information*stress +
     (vowel*timepoint*following * neighborhood+information+stress | participant) +
     (timepoint | word)
```

It should go without saying that this is a completely unreasonable model that will never converge. A first step towards reducing the model structure could be to reason that effects of neighborhood, information, and stress, which are all properties of the individual words in this data set, could be subsumed into the random effects by words. This reduces the maximal model to:

```
f <- f1 ~ vowel*timepoint*following +
     (vowel*timepoint*following | participant) +
     (timepoint | word)
```

This model is still somewhat on the large side, so we will now use `buildmer` to check: - if this model is capable of converging at all; - if all of these terms are really necessary.

Finding the maximal *feasible* model & doing stepwise elimination from it

To illustrate `buildmer`'s modular capabilities, we'll fit this model in two steps. We start by identifying the maximal model that is still capable of converging. We do this by running `buildmer`, with the `direction` argument set to `'order'`. We also set `lme4`s optimizer to `bobyqa`, as this manages to get much further than the default `nloptwrap`. The output of the command is edited for brevity.

```
library(lme4)
m <- buildmer(f,data=vowels,direction='order',control=lmerControl(optimizer='bobyqa'))
```

```
## Determining predictor order
## Currently evaluating LRT for: vowel, timepoint, following
```

```
## Fitting as (g)lm: f1 ~ vowel
## Fitting as (g)lm: f1 ~ timepoint
## Fitting as (g)lm: f1 ~ following
## Updating formula: f1 ~ following
## Currently evaluating LRT for: vowel, timepoint
## Fitting as (g)lm: f1 ~ following + vowel
## Fitting as (g)lm: f1 ~ following + timepoint
## Updating formula: f1 ~ following + vowel
## Currently evaluating LRT for: timepoint, vowel:following
## [...]
## Currently evaluating LRT for: vowel | participant
## Fitting via lme4, with REML: f1 ~ following + vowel + timepoint + vowel:timepoint + following:timepoi
## boundary (singular) fit: see ?isSingular
## None of the models converged - giving up ordering attempt.
```

The `order` step is useful if the maximal model includes random effects: `buildmer` will start out with an empty model and keeps adding terms to this model until convergence can no longer be achieved. The `order` step adds terms in order of their contribution to a certain criterion, such that the most important random slopes will be included first; this criterion is controlled by the `crit` argument. The default criterion is the significance of the change in log-likelihood (`LRT`: terms which provide lower chi-square p values are considered more important), but other options are also supported. These are the raw log-likelihood (`LL`: terms which provide the largest increase in the log-likelihood; this measure will favor categorical predictors with many levels), AIC (`AIC`), and BIC (`BIC`); you can select among them by passing e.g. `crit='LRT'`. The default `direction` is `c('order','backward')`, i.e. proceeding directly to backward stepwise elimination, but for illustration purposes we separate those steps here. (The `crit` argument also accepts vectors, such that e.g. `direction=c('order','backward'),crit=c('LL','LRT')` is allowed.)

After a lot of model fits, the model converges onto the following maximal model:

```
(f <- formula(m@model))
```

```
## f1 ~ following + vowel + timepoint + vowel:timepoint + following:timepoint +
##     following:vowel + following:vowel:timepoint + (1 + timepoint +
##     following + timepoint:following | participant) + (1 + timepoint |
##     word)
```

The maximal *feasible* model, i.e. the maximal model that is actually capable of converging, is one excluding random slopes for vowels by participants. This is not optimal for inference purposes, but for now it will do; we will see below that taking out the correlation parameters in the random effects makes it possible to include random slopes for vowels as well. We now proceed to the next step: stepwise elimination. This could also be done using e.g. `lmerTest`, but since the machinery was needed for `direction='order'` anyway it came at very little cost to also implement stepwise elimination in `buildmer` (both forward and backward are supported). This uses the same elimination criterion as could be specified previously; if left unspecified, it defaults to `crit='LRT'`, for the likelihood-ratio test. This is the preferred test for mixed models in Matuschek et al. (2017).

```
m <- buildmer(f,data=vowels,direction='backward',control=lmerControl(optimizer='bobyqa'))
```

```
## Fitting ML and REML reference models
## Fitting with REML: f1 ~ following + vowel + timepoint + vowel:timepoint + following:timepoint + follo
## Fitting with ML: f1 ~ following + vowel + timepoint + vowel:timepoint + following:timepoint + follow
## Testing terms
## Fitting with ML: f1 ~ following + vowel + timepoint + vowel:timepoint + following:timepoint + follow
## Fitting with REML: f1 ~ following + vowel + timepoint + vowel:timepoint + following:timepoint + follo
## Fitting with REML: f1 ~ following + vowel + timepoint + vowel:timepoint + following:timepoint + follo
##       grouping                       term
## 1         <NA>                          1
```

```
## 2         <NA>                 following
## 3         <NA>                     vowel
## 4         <NA>                 timepoint
## 5         <NA>           vowel:timepoint
## 6         <NA>       following:timepoint
## 7         <NA>           following:vowel
## 8         <NA> following:vowel:timepoint
## 9  participant                         1
## 10 participant                 timepoint
## 11 participant                 following
## 12 participant       timepoint:following
## 13        word                         1
## 14        word                 timepoint
##                                  block           LRT Iteration
##  1                            NA NA 1            NA         1
##  2                     NA NA following            NA         1
##  3                         NA NA vowel            NA         1
##  4                     NA NA timepoint            NA         1
##  5               NA NA vowel:timepoint            NA         1
##  6           NA NA following:timepoint            NA         1
##  7               NA NA following:vowel            NA         1
##  8    NA NA following:vowel:timepoint  4.967287e-01         1
##  9                     NA participant 1            NA         1
## 10            NA participant timepoint            NA         1
## 11            NA participant following            NA         1
## 12 NA participant timepoint:following  8.319280e-11         1
## 13                           NA word 1            NA         1
## 14                   NA word timepoint 2.134598e-153         1
## Updating formula: f1 ~ following + vowel + timepoint + vowel:timepoint + following:timepoint + follou
## Fitting ML and REML reference models
## [...]
## All terms are significant
## Finalizing by converting the model to lmerTest
```

By default, `buildmer` automatically calculates summary and ANOVA statistics based on Wald z-scores (summary) or Wald χ^2 tests (ANOVA). For answering our research question, we look at the summary:

```
summary(m)
```

```
## Linear mixed model fit by REML (p-values based on Wald z-scores) [lmerMod]
## Formula:
## f1 ~ following + vowel + timepoint + vowel:timepoint + following:timepoint +
##     (1 + timepoint | word) + (1 + timepoint + following + timepoint:following |
##     participant)
##    Data: vowels
## Control: control
##
## REML criterion at convergence: 149448.9
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -6.2347 -0.4168  0.0102  0.3877 21.6815
##
## Random effects:
##  Groups      Name                    Variance Std.Dev. Corr
```

```
##  word        (Intercept)              2539.3   50.39
##              timepoint               11089.5  105.31   -0.78
##  participant (Intercept)              2199.7   46.90
##              timepoint                3423.5   58.51   -0.50
##              followinglOns             747.3   27.34    0.11  0.65
##              timepoint:followinglOns  3053.2   55.26    0.64 -0.88 -0.67
##  Residual                            10018.1  100.09
## Number of obs: 12351, groups:  word, 148; participant, 6
##
## Fixed effects:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               551.733     20.075  27.484  < 2e-16 ***
## followinglOns              49.106     14.504   3.386  0.00071 ***
## vowel1                    114.254      8.951  12.765  < 2e-16 ***
## vowel2                    142.083      8.910  15.946  < 2e-16 ***
## vowel3                   -125.955      8.626 -14.601  < 2e-16 ***
## vowel4                    -79.211      9.970  -7.945 1.94e-15 ***
## timepoint                 -15.445     26.856  -0.575  0.56523
## vowel1:timepoint          -39.663     18.239  -2.175  0.02966 *
## vowel2:timepoint          -91.032     18.171  -5.010 5.45e-07 ***
## vowel3:timepoint          105.886     17.554   6.032 1.62e-09 ***
## vowel4:timepoint           38.924     20.271   1.920  0.05483 .
## followinglOns:timepoint -136.888     29.411  -4.654 3.25e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) fllwnO vowel1 vowel2 vowel3 vowel4 timpnt vwl1:t vwl2:t
## follwnglOns -0.043
## vowel1      -0.014  0.020
## vowel2      -0.017  0.025 -0.225
## vowel3      -0.029  0.031 -0.210 -0.207
## vowel4       0.034 -0.031 -0.278 -0.276 -0.266
## timepoint   -0.533  0.596  0.017  0.020  0.034 -0.041
## vowl1:tmpnt  0.011 -0.016 -0.787  0.177  0.165  0.218 -0.021
## vowl2:tmpnt  0.013 -0.020  0.177 -0.787  0.163  0.217 -0.025 -0.226
## vowl3:tmpnt  0.023 -0.024  0.166  0.164 -0.788  0.210 -0.044 -0.210 -0.208
## vowl4:tmpnt -0.027  0.024  0.219  0.218  0.210 -0.788  0.051 -0.277 -0.276
## fllwnglOns:  0.567 -0.720 -0.016 -0.020 -0.024  0.024 -0.788  0.020  0.024
##             vwl3:t vwl4:t
## follwnglOns
## vowel1
## vowel2
## vowel3
## vowel4
## timepoint
## vowl1:tmpnt
## vowl2:tmpnt
## vowl3:tmpnt
## vowl4:tmpnt -0.265
## fllwnglOns:  0.031 -0.030
```

The highly significant effect for `followinglOns:timepoint` shows that if the following /l/ is in the onset of the next syllable, there is a much larger change in F1 compared to the reference condition of having the

following /l/ in the coda of the same syllable.

Diagonal random-effects covariances

One hidden feature that is present in `buildmer` but that has not yet been discussed is the ability to group terms together in blocks for ordering and stepwise-elimination purposes. While the first argument to `buildmer` functions is normally a formula, it is also possible to pass a `buildmer terms list'. This is a data frame as generated by`tabulate.formula‘:

```
tabulate.formula(f)
```

```
##    index    grouping                      term
## 1   <NA>        <NA>                         1
## 2   <NA>        <NA>                 following
## 3   <NA>        <NA>                     vowel
## 4   <NA>        <NA>                 timepoint
## 5   <NA>        <NA>           vowel:timepoint
## 6   <NA>        <NA>       following:timepoint
## 7   <NA>        <NA>           following:vowel
## 8   <NA>        <NA> following:vowel:timepoint
## 9    9 1 participant                         1
## 10   9 1 participant                 timepoint
## 11   9 1 participant                 following
## 12   9 1 participant       timepoint:following
## 13  10 1        word                         1
## 14  10 1        word                 timepoint
##                                  code                              block
## 1                                   1                           NA NA 1
## 2                             following                   NA NA following
## 3                                 vowel                       NA NA vowel
## 4                             timepoint                   NA NA timepoint
## 5                     vowel:timepoint             NA NA vowel:timepoint
## 6                 following:timepoint          NA NA following:timepoint
## 7                     following:vowel             NA NA following:vowel
## 8           following:vowel:timepoint    NA NA following:vowel:timepoint
## 9                   9 1 participant 1                   NA participant 1
## 10          9 1 participant timepoint           NA participant timepoint
## 11          9 1 participant following           NA participant following
## 12 9 1 participant timepoint:following NA participant timepoint:following
## 13                        10 1 word 1                           NA word 1
## 14                10 1 word timepoint                 NA word timepoint
```

This is an internal `buildmer` data structure, but it is rather self-explanatory in how it is used. It is possible to modify the `block` column to force terms to be evaluated as a single group, rather than separately, by giving these terms the same `block` value. These values are not used in any other way than this purpose of selecting terms to be grouped together, which can be exploited to fit models with diagonal random-effects structures. The first step is to create explicit columns for the factor `vowel`; if this is not done, only random-effect correlations between vowels and *other* random slopes will be eliminated and those between the vowels themselves will remain.

```
vowels <- cbind(vowels,model.matrix(~vowel,vowels))
```

We next create a formula for this modified data set. To make it easier to type, we do not explicitly diagonalize the formula ourselves, but use `buildmer`'s `diag()` method for `formula` objects. We then call

`tabulate.formula()` on the new formula, providing a regular expression that matches terms belonging to the same vowel. Note that we *cannot* use the simple `vowel` factor in the fixed-effects part of the formula, as this will break `buildmer`'s marginality checks when considering which terms are eligible for inclusion or removal.

```
form <- diag(f1 ~ (vowel1+vowel2+vowel3+vowel4)*timepoint*following +
         ((vowel1+vowel2+vowel3+vowel4)*timepoint*following | participant) +
         (timepoint | word))
terms <- tabulate.formula(form,group='vowel[^:]')
```

Finally, we can instruct `buildmer` to use this specially-crafted `terms` object by simply passing it along instead of a regular formula. `buildmer` will recognize what is going on, and look for an additional `dep` argument for the name of the dependent variable in the data frame (provided as a character string).

```
m <- buildmer(terms,data=vowels,dep='f1',control=lmerControl(optimizer='bobyqa'))
```

This approach allows random slopes for `vowel` and for `vowel:timepoint` to make it in, both of which significantly improve model fit. This model seems much more adequate for statistical inference.

Other options

Because `buildmer` does not do any model fitting by itself but is only an administrative formula processor around pre-existing modeling fuctions, it was straightforward to extend it beyond its original purpose of mixed-effects models. The logical extension of `buildmer` to GAMMs is fully supported (with the exception of `gamm` models; see `buildgamm()` for options). Relevant functions are available as `buildgam`, `buildbam`, and `buildgamm4`. `glmmTMB` models are also supported via function `buildglmmTMB`, although their syntax for covariance structures (e.g. `diag(timepoint | participant)`) is not; these models are still useful for their ability to handle autocorrelation, zero-inflation, and to use REML for GLMMs. From package `nlme`, `gls` models are supported via `buildgls`, and the fixed-effects part of `lme` models can be eliminated via `buildlme`. At the request of Willemijn Heeren, `buildmer` was also extended to handle multinomial-logistic-regression models fitted by function `multinom` from package `nnet`; see function `buildmultinom`. It is also possible to use `buildjulia` to drive Douglas Bates's `MixedModels` package for Julia. Finally, `buildmertree` was recently added, which makes it possible to do term ordering and backward elimination of the random-effects part of `glmertree` models.

References

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, *68*(3), 255–278.

Bates, D., Kliegl, R., Vasishth, S., & Baayen, H. (2015). Parsimonious mixed models. *arXiv Preprint arXiv:1506.04967*.

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, H., & Bates, D. (2017). Balancing Type I error and power in linear mixed models. *Journal of Memory and Language*, *94*, 305–315.