Logistic regression and parametric bootstrapping on a moderately large simulated data set

Ben Bolker

February 18, 2012

Set up parameters:

```
ncomm <- 20
nschool <- 100
nclass <- 300
ntot <- 9000
```

Sample random effects:

```
set.seed(101)
u.comm <- rnorm(ncomm)
u.school <- rnorm(nschool)
u.class <- rnorm(nclass)
```

Construct simulated data:

```
dt <- expand.grid(commune=factor(seq(ncomm)),
                  school=factor(seq(nschool/ncomm)),
                  class=factor(seq(nclass/nschool)),
                  pupil=factor(seq(ntot/nclass)))
eta <- with(dt,u.comm[commune]+u.school[commune:school]+
            u.class[commune:school:class])
dt$loss <- rbinom(ntot,plogis(eta),size=1)
```

Basic model fit:

```
library(lme4)
t1 <- system.time(g1 <-
glmer(loss~(1|commune/school/class),data=dt,
                        family=binomial))
summary(g1)
```

```
## Generalized linear mixed model fit by the Laplace approximation
## Formula: loss ~ (1 | commune/school/class)
##    Data: dt
##    AIC   BIC logLik deviance
##  10031 10059  -5011    10023
## Random effects:
##  Groups                 Name        Variance Std.Dev.
##  class:(school:commune) (Intercept) 0.937    0.968
##  school:commune         (Intercept) 0.906    0.952
##  commune                (Intercept) 0.513    0.716
## Number of obs: 9000, groups: class:(school:commune), 300; school:commune, 100; commune, 2
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.159      0.196   -0.81     0.42
```

Parametric bootstrap:

```
pbootfun <- function(object) {
  robj <- suppressWarnings(refit(object,simulate(object)))
  unlist(VarCorr(robj))
}
```

On my machine the next chunk of code took about 200*2.5 seconds = 8 minutes. `raply` is a nice wrapper, but there's nothing here you couldn't do with a `for` loop, or with `replicate` from base R.

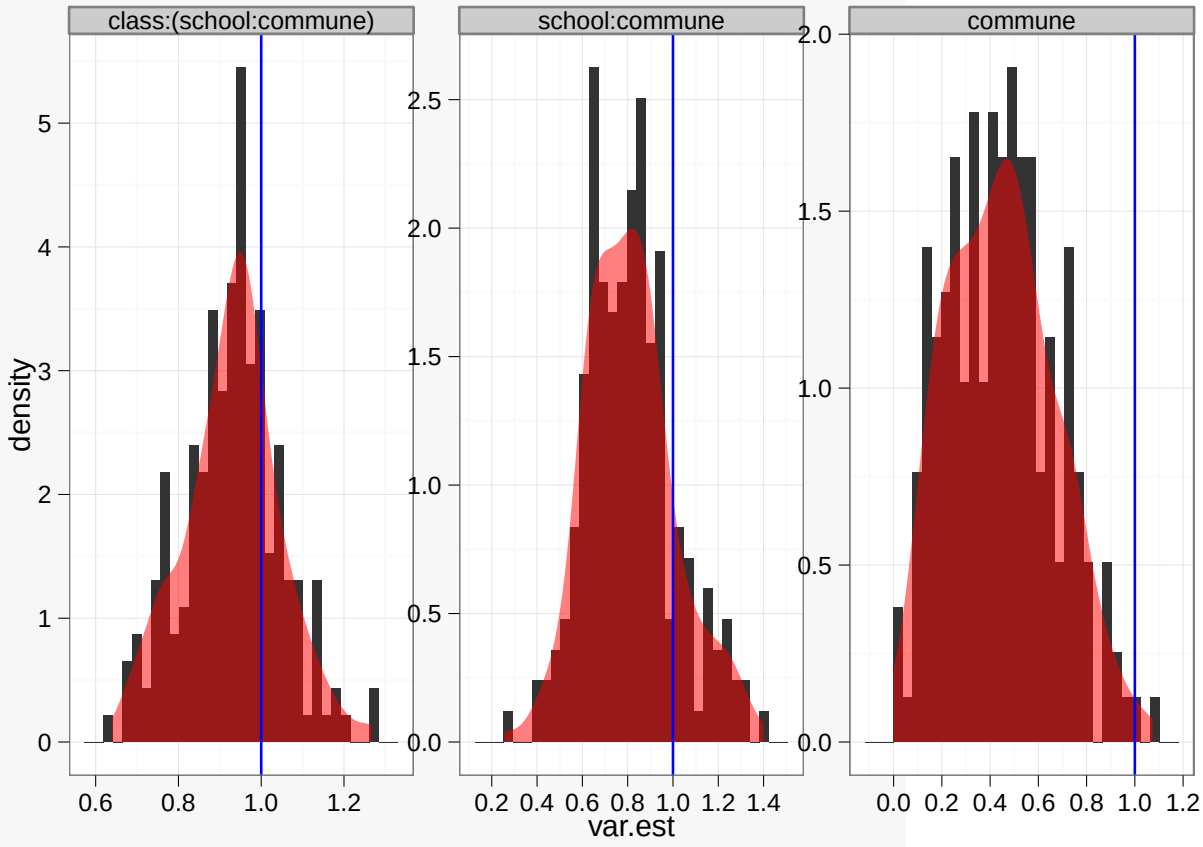
```
library(plyr)
set.seed(101)
fn <- "boot_results.RData"
if (!file.exists(fn)) {
  bootdist <- raply(200,pbootfun(g1),.progress="text")
  save("bootdist",file=fn)
} else load(fn)
```

Pictures:

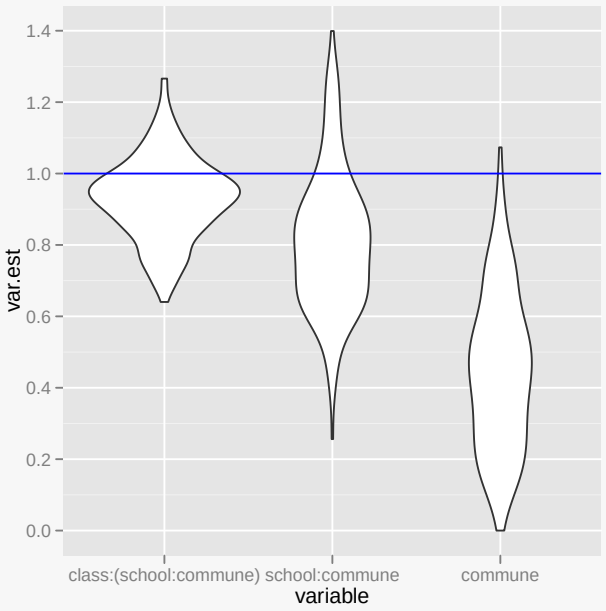
```
library(ggplot2)
library(reshape)
```

```
mm <- rename(melt(bootdist),c(X2="variable",value="var.est"))
## rearrange order back to original, from alphabetical
mm$variable <- factor(mm$variable,levels=colnames(bootdist))
```

```
ggplot(mm,
       aes(x=var.est))+geom_histogram(aes(y=..density..))+
  theme_bw()+
  geom_density(fill="red",colour=NA,alpha=0.5)+
  facet_wrap(~variable,scale="free")+
  geom_vline(xintercept=1.0,colour="blue")
```

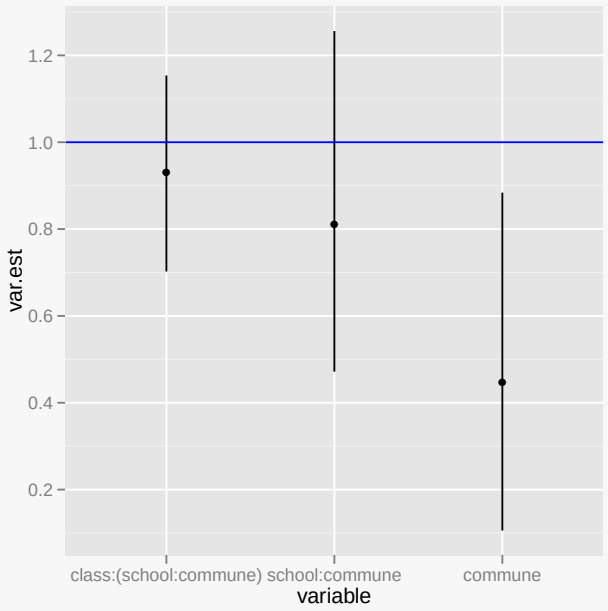



```
## requires ggplot 0.9.0
ggplot(mm,aes(x=variable,y=var.est))+
  geom_violin()+geom_hline(yintercept=1.0,colour="blue")
```




```
ggplot(mm,aes(x=variable,y=var.est))+
  stat_summary(fun.data=function(x) {

data.frame(y=mean(x),ymin=quantile(x,0.025),ymax=quantile(x,0.975))
  })+
  geom_hline(yintercept=1,colour="blue")
```



Naive (quantile) confidence limits on variance component estimates (same as those pictured above):

```
t(apply(bootdist,2,quantile,c(0.025,0.975)))
```

```
##                            2.5% 97.5%
## class:(school:commune) 0.7023 1.154
## school:commune         0.4716 1.256
## commune                0.1056 0.884
```

Other approaches to confidence intervals etc. are available through the `boot` package. The following is a small example of constructing parametric bootstrap intervals via `boot`:

```
library(boot)
```

Because of its generality, the procedure for constructing bootstrap intervals via `boot` is actually (alas) slightly clunkier than just doing it directly, as above.

```
bootres <- boot(dt,sim="parametric",R=100,
                statistic=function(dvec) {
                  robj <- refit(g1,dvec$loss)
                  unlist(VarCorr(robj))
                },
```

```
                ran.gen=function(dvec,p) {
                  data.frame(loss=simulate(g1)[[1]])
                })
```

```
bootres

##
## PARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = dt, statistic = function(dvec) {
##     robj <- refit(g1, dvec$loss)
##     unlist(VarCorr(robj))
## }, R = 100, sim = "parametric", ran.gen = function(dvec, p) {
##     data.frame(loss = simulate(g1)[[1]])
## })
##
##
## Bootstrap Statistics :
##     original    bias    std. error
## t1*   0.9371 -0.005552      0.1177
## t2*   0.9062 -0.016643      0.2115
## t3*   0.5133 -0.053215      0.2142

## calculate bootstrap CI for variable 3 (commune)
boot.ci(bootres,index=3,type=c("norm","basic","perc"))

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 100 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bootres, type = c("norm", "basic", "perc"),
##     index = 3)
##
## Intervals :
## Level      Normal              Basic              Percentile
## 95%   ( 0.1466,  0.9863 )   ( 0.1693,  0.9274 )   ( 0.0991,  0.8572 )
## Calculations and Intervals on Original Scale
## Some basic intervals may be unstable
## Some percentile intervals may be unstable
```

See this question on Stack Overflow for some discussion about what it takes to get adjusted bootstrap intervals for a parametric bootstrap.