Mandallaz' model-assisted small area estimators

Andreas Dominik Cullmann

July 2, 2015

1 Introduction

Model-*based* small area estimators (for example [1], chapters 5ff.) depend on model assumptions to hold. This dependency doesn't make them very attractive for official statistics.

Model-*assisted* small area estimators do not depend on the model assumptions to hold, albeit their variances will be higher if the model is inappropiate (see [2], chapter 6.7). The synthetic-regression estimator (SRE) (for example [1], chapter 4.2.2) is biased, and the variance of its biased-corrected version, the generalized regression estimator (for example [1], chapter 2.5), "depends crucially on the [...] residuals in the small area" ([3], p. 444), which basically means that its variance will be unacceptably high in many applications.

Daniel Mandallaz and others ([4], [3], [5], [6], [7] and [8]) propose an unbiased extension of the SRE for two- and three-phase sampling designs with or without clustering. The variance of the extended SRE is, like the that of the SRE, based on all residuals in the second (or third) phase and asymtotically equivalent to the SRE's variance (see [3], p. 444).

2 non-exhaustive auxiliary information

2.1 two-phase sampling

Let us suppose we have a two-phase sampling design, and the sample data can be loaded via

```
> data('s2')
> data('s1')
```

We now add sampling phase indicators to the data and join (from a database point of view we do a union, but I'll keep calling it join) it into a single data.frame `s12`:

```
> s2$phase1 <- s2$phase2 <- TRUE
> s1$phase1 <- TRUE
> s1$phase2 <- FALSE
> eval(parse(text=(paste('s1$'
+                        , setdiff(names(s2), names(s1))
+                        , ' <- NA' , sep = ''))))
> s12 <- rbind(s1, s2)
```

We build a small area Object

```
> saeO <- saObj(data = s12, f = y ~x1 + x2 + x3 | g
+               , s2 = 'phase2')
```

and get the small area estimations for the non-exhaustive case as

```
> predict(saeO)
```

```
  smallArea prediction variance
1         a   378.8590 487.3680
2         b   391.8262 417.3442
```

2.2 three-phase sampling

Suddenly we stumble across data from a third sampling phase, the null phase (`s0` has all the predictors of `s2`, but we keep only one – if we kept all of them, we'ld be back to two-phase sampling with more observations), and we join all three phases into `s012`:

```
> data('s0')
> s0$x1 <- s0$x3 <- NULL
> s0$phase1 <- s0$phase2 <- FALSE
> eval(parse(text=(paste('s0$'
+                        , setdiff(names(s12), names(s0))
+                        , ' <- NA' , sep = ''))))
> s012 <- rbind(s0, s12)
```

from which again we do a predict:

```
> predict(saObj(data = s012,  f = y ~x1 + x2 + x3 | g
+               , s2 = 'phase2', s1 = 'phase1'))
```

```
  smallArea prediction variance
1         a   397.1866 311.8078
2         b   404.2197 271.9036
```

Note the drop in variance induced by the extensive null phase sampling.

3 partially exhaustive auxiliary information

Let us suppose we *knew* the estimated small area means of the fixed effect sampled in all three phases to be the true small area means:

```
> tm1 <- as.data.frame(tapply(s012$x2 , s012$g, mean))
>  names(tm1)[1] <- c('x2'); tm1$g <- row.names(tm1)
> predict(saObj(data = s12, f = y ~x1 + x2 + x3 | g
+               , s2 = 'phase2', smallAreaMeans = tm1))
```

```
  smallArea prediction variance
1         a   397.1866 295.1857
2         b   404.2197 254.2177
```

Again, the variance estimation is reduced, but not as markedly as before: due to the extensive null phase the mean estimations had very small variances (which added to the small area estimation variances).

4 exhaustive auxiliary information

Of course we could also take our estimated small area means of all fixed effects from the first and second phase to be the true means:

```
> preds <- paste('x',1:3, sep='')
> tm <- as.data.frame(
+     rbind(
+         colMeans(subset(s12, g =='a')[, preds])
+         , colMeans(subset(s12, g =='b')[, preds])
+         )
+     ); tm$g=c('a', 'b')
```

That would give us the smallest variance estimates:

```
> predict(saObj(data = s12, f = y ~x1 + x2 + x3 | g
+               , s2 = 'phase2', smallAreaMeans = tm))

  smallArea prediction variance
1         a   378.8590 223.1295
2         b   391.8262 142.0318
```

5 model-based small area estimation

Wait! If I use the code in Appendix A to calculate the EBLUP for the basic unit level model given by [1], chapter 7.2, I get at least a much smaller mse1:

```
> source('Rao.R')
> library(nlme)
> dat <- subset(s2, ! is.na(s2$g))
> dat <- dat[with(dat, order(g)), ]
> aLmeObj <- lme(y ~x1 + x2 + x3
+                , data = dat
+                , random =  ~1|g)
> foo <- new(Class = "sae", lmeObj = aLmeObj, domain.df = tm)
> sae(foo)

     eblup      mse1     mse2
a 376.6563  98.99138 258.9288
b 393.4619 101.48354 300.4883
```

Of course you do, but you rely on the model's assumptions to hold. Have you checked them?

No, I don't even know what they are. But even if the tests would not reject the hypotheses of the assumptions being valid on, say, a .95 confidence level, I could never be sure.

That's why I wrote `maSAE`.

I see, but wait again, what is . . .

6 cluster sampling

```
>  grep('.*clust', capture.output(str(s1)), value = TRUE)

[1] " $ clustid  : int  288 288 349 349 349 349 427 427 434 505 ..."
```

. . . this "clustid" in the data.frames?
Dear, I forgot that sampling for my completely made up example data was done with a cluster design! So all the above variances estimates are too optimistic. My fault. For the sake of CPU time on CRAN, I'll leave the following to you:

```
> predict(saObj(data = s12, f = y ~x1 + x2 + x3 | g
+               , s2 = 'phase2'
+               , cluster = 'clustid'))

> predict(saObj(data = s012,  f = y ~x1 + x2 + x3 | g
+               , s2 = 'phase2', s1 = 'phase1'
+               , cluster = 'clustid'))

> predict(saObj(data = s12, f = y ~x1 + x2 + x3 | g
+               , s2 = 'phase2', smallAreaMeans = tm1
+               , cluster = 'clustid'))

> predict(saObj(data = s12, f = y ~x1 + x2 + x3 | g
+               , s2 = 'phase2', smallAreaMeans = tm
+               , cluster = 'clustid'))
```

References

[1] J.N.K. Rao. *Small Area Estimation*. Wiley Series in Survey Methodology. Wiley, 2003.

[2] Carl-Erik Särndal, Bengt Swensson, and Jan Wretman. *Model Assisted Survey Sampling*. Springer Series in Statistics. Springer, 1992.

[3] Daniel Mandallaz. Design-based properties of some small-area estimators in forest inventory with two-phase sampling. *Canadian Journal of Forest Research*, 43(5):441–449, 2013.

[4] Daniel Mandallaz. Design-based properties of some small-area estimators in forest inventory with two-phase sampling. Technical report, Eidgenössische Technische Hochschule Zürich, Departement Umweltsystemwissenschaften, 2012.

[5] Daniel Mandallaz, Jochen Breschan, and Andreas Hill. New regression estimators in forest inventories with two-phase sampling and partially exhaustive information: a design-based monte carlo approach with applications to small-area estimation. *Canadian Journal of Forest Research*, 43(11):1023–1031, 2013.

[6] Daniel Mandallaz. Regression estimators in forest inventories with two-phase sampling and partially exhaustive information with applications to small-area estimation. Technical report, Eidgenössische Technische Hochschule Zürich, Departement Umweltsystemwissenschaften, 2013.

[7] Daniel Mandallaz. A three-phase sampling extension of the generalized regression estimator with partially exhaustive information. *Canadian Journal of Forest Research*, 44(4):383–388, 2014.

[8] Daniel Mandallaz. Regression estimators in forest inventories with three-phase sampling and two multivariate components of auxiliary information. Technical report, Eidgenössische Technische Hochschule Zürich, Departement Umweltsystemwissenschaften, 2013.

A Rao.R

```
setOldClass(c('lme'))
setClass(Class = "sae"
         , representation = representation(
             lmeObj = "lme"
             , domain.df = "data.frame"
             )
         , validity = function(object){
           if(object@lmeObj$method != 'REML')
             return("lme method wasn't REML, can't handle this")
           if(! all(c(attr((getGroups(object@lmeObj)), 'label')
                      , dimnames(attr(terms(formula(object@lmeObj))
                                      , "factors"))[[2]])
                    %in% names(object@domain.df)
                    )
              ) return('Names in domain.df do not match names in lmeObj')
           unitName <- attributes(getGroups(object@lmeObj))$label
           if(! all(eval(parse(text = paste(sep = ''
                                            , 'object@domain.df$'
                                            , unitName)))
                    %in% unique(getGroups(object@lmeObj))
                    )
              ) return('Unknown unit in domain.df')
         }
         )
setGeneric(name = "sae"
           , def = function(object, ...){standardGeneric("sae")}
           )


setMethod(f = "sae"
          , signature(object = "sae")
          , function(object){
            lmeObj  <- object@lmeObj
            domain.df <- object@domain.df

            eval(parse(text=paste(sep = ''
                                  , 'domain.df$'
                                  , setdiff(dimnames(attr(terms(formula(lmeObj))
                                                          , "factors"))[[1]]
                                            , dimnames(attr(terms(formula(lmeObj))
                                                            , "factors"))[[2]])
                                  , '<- 1'))
                 )

            unitName <- attributes(getGroups(lmeObj))$label
            units <- unique(getGroups(lmeObj))
            unitseq <- seq(along = units); names(unitseq) <- units
            n <- tapply(getGroups(lmeObj), getGroups(lmeObj), length)
```

```
## the estimated Variances, see p.137, last paragraph
varV <- as.numeric(VarCorr(lmeObj)[1,1])
varE <- as.numeric(VarCorr(lmeObj)[2,1])
## Vector of the mean Residual per Unit
mUnitRes <- tapply(resid(lmeObj, level = 0)
                   , names(resid(lmeObj, level = 0))
                   , mean)[levels(units)]
## list of design matrices, multipurpose
lX <- lapply(X = unitseq
             , FUN = function(i, lmeObj, unitName){
               unit <- units[i]
               X <- model.matrix(formula(lmeObj)
                                 , subset(lmeObj$data
                                          , eval(parse(text
                                                       = unitName))
                                          == unit))
               return(X)
             }
             , lmeObj
             , unitName
             )
## list of R7.2.4, multipurpose
lR7.2.4 <- lapply(X = unitseq
                  , FUN = function(i, n, varV, varE){
                    unit <- units[i]
                    n.i <- n[unit]
                    R7.2.4 <- varV / (varV + varE / n.i)
                    return(R7.2.4)
                  }
                  , n
                  , varV
                  , varE
                  )
## list of R7.2.7 to build the sum in R7.2.12
lR7.2.7 <- lapply(X = unitseq
                  , FUN = function(i, n, lX, lR7.2.4, varV, varE){
                    unit <- units[i]
                    X <- lX[[unit]]
                    n.i <- n[unit]
                    R7.2.4 <- lR7.2.4[[unit]]
                    R7.2.2 <- 1 / varE *
                      (diag(n.i) - R7.2.4 / n.i * rep(1, n.i)
                       %*% t(rep(1, n.i)))
                    R7.2.7 <- t(X) %*% R7.2.2 %*% X
                    return(R7.2.7)
                  }
                  , n
                  , lX
                  , lR7.2.4
                  , varV
                  , varE
                  )
sumAi <- Reduce('+',lR7.2.7)
lR7.2.30 <- lapply(X = unitseq
                   , FUN = function(i, n, varV, varE){
                     unit <- units[i]
                     n.i <- n[unit]
                     R7.2.30 <- varE + varV * n.i
                     return(R7.2.30)
                   }
                   , n
                   , varV
```

```
                    , varE
                    )
  R7.2.27 <- 0.5 * sum(n^2 * unlist(lR7.2.30)^-2)
  R7.2.28 <- 0.5 * sum((n - 1) * sqrt(varE)^-4 + unlist(lR7.2.30)^-2 )
  R7.2.29 <- 0.5 * sum(n * unlist(lR7.2.30)^-2)
  informationMatrix <- matrix(data = c(R7.2.27, R7.2.29, R7.2.29, R7.2.28)
                              , ncol = 2)
  asymtoticCovarianceMatrix <- solve(informationMatrix)
  R7.2.23 <- varE^2 * asymtoticCovarianceMatrix[1, 1] +
    varV^2 * asymtoticCovarianceMatrix[2, 2] -
      2 * varE * varV * asymtoticCovarianceMatrix[1, 2]
  units <- eval(parse(text = paste(sep = ''
                                   , 'domain.df$'
                                   , unitName)))
  unitseq <- seq(along = units)
  names(unitseq) <- units
  foo <- lapply(##  over observations in domain.df
                X = unitseq
                , FUN = function(i, n, lX, lR.7.2.4, varV
                    , varE, sumAi, R7.2.23){
                  unit <- units[i]
                  X <- lX[[unit]]
                  n.i <- n[unit]
                  R7.2.4 <- lR7.2.4[[unit]]
                  R7.2.11 <- R7.2.4 * varE / n.i
                  Xbar <- model.matrix(formula(lmeObj)
                                       , subset(domain.df
                                                , eval(parse(text =
                                                             unitName)) == unit
                                                )
                                       )
                  xbar <- colMeans(X)
                  tmp <- as.numeric(Xbar) - R7.2.4 * xbar
                  R7.2.12 <- t(tmp) %*% solve(sumAi) %*% tmp
                  R7.2.30 <- lR7.2.30[units[unit]]
                  R7.2.22 <- n.i^-2 * (varV + varE / n.i)^-3 * R7.2.23
                  R7.2.32 <- n.i^-2 * (varV + varE
                                       / n.i)^-4 * R7.2.23 * mUnitRes[unit]^2
                  R7.2.33 <- R7.2.11 + R7.2.12 + 2 * R7.2.32
                  R7.2.34 <- R7.2.11 + R7.2.12 + R7.2.22 + R7.2.32
                  return(c(mse1 = R7.2.33
                           , mse2 = R7.2.34
                           ))
                }
                , n
                , lX
                , lR7.2.4
                , varV
                , varE
                , sumAi
                , R7.2.23
                )
  ## transform list to transposed data.frame
  foo <- as.data.frame(t(as.data.frame(foo)))
  return(cbind(eblup = as.numeric(predict(lmeObj
                                          , newdata = domain.df))
  , foo))
}
)
```