Applied Analyses with BUILD

Carter Sevick

1 Introduction

This tutorial is designed for those tasked with the analysis of data derived from a secondary research study of a biobank or data repository with a binary longitudinal outcome. Participants were sampled from the repository based on an outcome dependent design (ODS) (Sevick 2025; Schildcrout 2018) that constituted a stratified random sample where the strata were formed from the longitudinal outcome pattern and dropout time. At the time of sampling, the participants were missing their primary exposure of interest, which was later assigned based on evaluation of stored bio-samples. Since the data were collected longitudinally, dropout is a concern and if the nature of the outcome is such that it is associated with study dropout then analytic adjustments may be necessary to avoid bias (Wilson 2019).

Here we introduce the BUILD package which implements our methods for **B**inary outcomes **U**sing **I**nformed sampling strategies for **L**ongitudinal studies with non-ignorable **D**ropout (BUILD) as an expansion to ODS to accommodate analytical methods adjusting for non-ignorable dropout. Familiarity with the background of this method (Sevick 2025) will be assumed as well as that the intent of the reader is to gain awareness of the methodology through an applied example.

In the following sections, we will introduce our function to simulate longitudinal data with non-ignorable dropout which uses methods that have been used previously in the evaluation of methods to adjust for this effect (Sevick 2025; Hogan 2004a; Forster et al. 2011; Camille M. Moore et al. 2017; Camille M. Moore, Carlson, and Kreidler 2019). Data generated will be used as grist for an example analysis using a mixture model to account for possible non-ignorable dropout adjusted for the ODS sampling design. Finally, marginal parameter effects will be computed and inference carried out using resampling.

1.1 Installation

The current BUILD package is available from GitHub. Before attempting installation be sure that you have the devtools package installed:

```
install.packages("devtools")
```

At this point, the BUILD package may be installed with:

```
devtools::install_github("csevick/build")
```

2 A Tour of BUILD Functions for Analysis

2.1 Data Simulation

If appropriate data is already available then this next section may be skipped and the later sections adapted (beginning with section 2.3). The function *simuDat()* is designed to simulate a data set, with a non-ignorable dropout effect, that can be used to demonstrate the capabilities of the BUILD package. Additional information for this function is available in the documentation for the BUILD package. Here we will present it as the data source, with minimal explanation.

```r
library(build)

set.seed(209587)

dat <- simuDat(
    B            = c(-3,2,5,2.5,0),
    G            = matrix(c(4), ncol = 1),
    random       = NULL,
    p_exposed    = 0.5,
    p_confounder = 0.25,
    conf         = 0,
    N            = 5000,
    M            = 5,
    D_B          = c(0, -2, -1, -0.5, 0),
    D_F          = c(1, 1, 1, 1, 1),
    D_parms      = list(c(1,1), c(1,1.5))
  )

# create a subset with non-random dropout
mdat <- subset(dat, time <= dtime)
```

The result is a dataframe of 5000 participants (with up-to 6 rows, for those that complete the study) where unexposed participants dropout uniformly (starting at the second follow up), but the exposed are less likely to dropout over time. The binary outcome will be generated from a model with a random intercept variance of 4 and the following structure:

$$f(y_{ij}|b_i) = -3 + 2t_{ij} + 5x_i + 2.5t_{if}x_i - 2d_it_{ij} - 1d_ix_i - 0.5d_it_{ij}x_i + b_i$$

```r
# create look up table for exposure status
## this will represent the assessment results for the selected random sample
exposure_status <- unique(mdat[c('id', 'exposure')])

# delete exposure status from the data source (we are not supposed to know!)
mdat <- subset(mdat, select = -exposure)
```

2.2 Identifying Outcome and Dropout Strata and Drawing a Sample

In preparation for drawing an ODS sample with possible non-ignorable dropout, the first step is to inspect the cell counts of the outcome by dropout time strata. Recall that the outcome strata are a grouping of the sum of the longitudinal binary outcomes for each participant. Let Y_{ij} and n_i be the j^{th} measurement and number of measurements for the i^{th} participant, respectively, then stratum assigned to participant i is:

$$v_i = \begin{cases} 1 & \sum_j y_{ij} = 0 \\ 2 & \sum_j y_{ij} < n_i \\ 3 & \sum_j y_{ij} = n_i \end{cases}$$

The function *build_sample_frame*() takes a dataframe, destined for an ODS sample, and produces a 2 element list. The first element is a dataframe containing the unique individuals and their outcome and dropout time strata. The outcome strata assignment is called *oc_strata*. The second element is a simple cross-tabulation of the strata. Both elements are needed for future steps in this process. The arguments of the function are:

- **data**: the name of a dataframe to select rows from

- **study_id**: character, variable in the dataframe that identifies the subject
- **outcome**: character, variable in the dataframe that specifies the study outcome
- **drop_strata**: character - variable in the dataframe that identifies the dropout strata

```
# create the sample frame
s_frame <- build_sample_frame(data        = mdat
                            , study_id    = 'id'
                            , outcome     = 'y'
                            , drop_strata = 'dtime')

# the strata cell counts are:
s_frame$strata_counts
```

```
##          dtime
## oc_strata 0.2 0.4 0.6 0.8   1
##         1 460 377 375 351 311
##         2 222 319 397 380 306
##         3 505 376 293 203 125
```

```
# the marginal outcome strata counts are:
rowSums(s_frame$strata_counts)
```

```
##    1    2    3
## 1874 1624 1502
```

```
# the marginal dropout time strata counts are:
colSums(s_frame$strata_counts)
```

```
##  0.2  0.4  0.6  0.8    1
## 1187 1072 1065  934  742
```

```
# the first 6 rows of the sample frame
head(s_frame$sample_frame)
```

```
##      id dtime oc_strata
## 1.1   1   0.6         3
## 2.7   2   0.8         2
## 3.13  3   0.4         2
## 4.19  4   1.0         1
## 5.25  5   0.2         1
## 6.31  6   0.6         1
```

Next, we specify the sampling plan by the desired counts selected from the marginal outcome strata and then how to distribute that count across the dropout strata. The function for this task is *build_sample_plan*() and has the following arguments:

- **n_samp**: a scalar (if only specifying total desired sample size), or three element vector (if specifying marginal outcome strata sampling goal)
- **N_bystrata**: matrix of outcome by dropout counts. dimensions must be named
- **oc_sample_adj**: numeric vector of dim 3. Specified only if a scalar sample size is specified. Amount to adjust the observed outcome strata proportions expressed as an odds increase in the sampling proportion of the stratum. All three may be specified and the result is standardized to the sample size requested in **n_samp**
- **drop_grad**: numeric scalar. a value of one will indicate proportional sampling across the dropout strata. a non-zero value is a slope (expressed as an odds ratio) to adjust the sampling gradient across dropout strata.

The function outputs are a list with:

1. **sample_plan**: a data frame specifying the sample sizes and probabilities for selection from each strata
2. **P_matrix**: A matrix of sampling probabilities from the strata
3. **N_sample**: A matrix of sampling sample sizes from the strata

If the sample sizes from the marginal outcome strata are known and proportionate sampling across the dropout strata is desired then the following will create the sampling plan for selection of the sample data. A sampling plan selecting 75, 450 and 75 participants from strata 1, 2, and 3, respectively, is referred to as [75, 450, 75].

```
# create the sampling plan: marginal outcome samples at [75, 450, 75]
s_plan <- build_sample_plan(n_samp     = c(75, 450, 75)
                          , N_bystrata = s_frame$strata_counts
                          , drop_grad  = 1
                            )

# the [75, 450, 75] participants are distributed as:
s_plan$N_sample
```

```
##          dtime
## oc_strata 0.2 0.4 0.6 0.8   1
##         1  18  15  15  14  12
##         2  62  88 110 105  85
##         3  25  19  15  10   6
```

More detail on how to use the *drop_grad* argument can be found in the documentation and there is an additional function in the package to assist in selecting values for the *drop_grad* and *oc_sample_adj* arguments, however that is reserved for future vignettes.

The sampling plan is now defined and we may move on to selecting the sample for which the participant exposure status will be determined. The function *strat_samp*() is designed to select stratified random samples in two different ways. First, selection can be set to randomly select a set number from each strata, this yields a predictable sample size. Alternatively, participants can be selected with a set probability, leading to variable sample sizes. The arguments are:

- **df**: the name of a dataframe to select rows from
- **oc_strata**: character, variable in the dataframe that specifies the outcome strata membership
- **drop_strata**: character, variable in the dataframe that specifies the dropout strata membership
- **sample_plan**: dataframe - contains the sampling plan from **build_sample_plan**
- **sample_by**: (p/n) specify whether to sample by counts or probability (default is 'n')

The output is a dataframe selected from *df* with an additional column specifying the sampling weight (*sampWT*) of the selected participant.

```
# draw the stratified sample
set.seed(4580978)
sample_subj <- strat_samp(df = s_frame$sample_frame
                          , oc_strata = 'oc_strata'
                          , drop_strata = 'dtime'
                          , sample_plan = s_plan$sample_plan
                          , sample_by = 'n')

# The first 6 rows from the data frame holding the selected participants
head(sample_subj)
```

```
##     dtime oc_strata   id drop_strata   sampWT
## 145   0.2         1 2095           1 25.55556
## 232   0.2         1 2586           1 25.55556
## 169   0.2         1  582           1 25.55556
## 459   0.2         1 3906           1 25.55556
## 366   0.2         1 2720           1 25.55556
## 325   0.2         1 1222           1 25.55556
```

Now that the sample of participants has been selected, the stored bio-samples are submitted for additional analysis to determine exposure. To simulate this the selected subjects are merged with the exposure dataset, created earlier.

```
# get exposure status
sample_subj <- merge(sample_subj, exposure_status, by = 'id')
```

```
sample <- merge(mdat
              , sample_subj[, c('id'
                                , 'oc_strata'
                                , 'drop_strata'
                                , 'sampWT'
                                , 'exposure')]
              , by = 'id')

# sort for good form
sample <- sample[order(sample$id, sample$time),]
```

2.3 Analysis of the Sample Data

The *lin_mix()* function is the main analytic tool of the BUILD package. Adjustment for the sampling design can be done with ascertainment adjusted maximum likelihood (ACML) or by inverse probability of selection weighted maximum likelihood (WL). It is also integral to the multiple imputation algorithms supplied by the package.

This function estimates a generalized linear mixed model with a logit link and binomial distribution. The R optim function (R Core Team 2023) is used to optimize the likelihood with the BFGS method. Integration of the likelihood function is by Gauss-Hermite quadrature (McCulloch, Searle, and Neuhaus 2008). Robust variances are available for the fixed effects, if needed (Rabe-Hesketh and Skrondal 2006). At present, clustering is limited to one level but any number of random effects may be included, up to computational capacity, and available correlation structures are diagonal and unstructured.

The following function call fits a random intercept model with fixed effects for time, exposure and their interaction. The additional fixed effects for dropout (dtime and dtime interactions with the other fixed effects) are the components necessary to fit a conditional linear model (CLM) (Wu and Bailey 1989), which assumes that the relationship between dropout and the fixed effects assumed to be affected by dropout is linear. This function has many arguments so we will introduce them as needed. The documentation has a thorough presentation of all arguments.

- **df**: a data frame
- **fixed**: formula for the fixed effect portion of the model
- **random**: formula for the random effect portion of the model
- **cluster**: character - name of the variable in df that defines a cluster
- **sweights**: character - name of the variable in df that holds sampling weights (cluster level only)
- **robust**: boolean - if true then compute robust variances for the fixed effects
- **qpoints**: numeric scalar - number of quadrature points to use when integrating the likelihood

```
# fit the model
WL_fit <- logit_mix(df = sample
                    , fixed = y ~ time + exposure + time:exposure +
                                  dtime + time:dtime + exposure:dtime +
                                  time:exposure:dtime
                    , random = ~ 1
                    , cluster = 'id'
                    , sweights = 'sampWT'
                    , robust = T
                    , qpoints = 20
                    )

# fixed effect estimates
knitr::kable(WL_fit$fixed, digits = 2)
```

	estimate	std_err	wald_stat
(Intercept)	-3.04	0.67	-4.57
time	2.71	1.16	2.33
exposure	5.06	0.91	5.55
dtime	0.16	0.93	0.17
time:exposure	0.42	1.66	0.25
time:dtime	-2.99	1.39	-2.16
exposure:dtime	-1.10	1.34	-0.82
time:exposure:dtime	1.78	2.06	0.87

In this example, dropout time is discrete thus we can compute marginal (expected) parameters (with respect to dropout) as a the mean of the dropout specific slopes taken over the empirical distribution of dropout time. The complexity of this computation is proportional to the dependencies we assume for the empirical distribution of dropout time, and for this example we will assume dependency on exposure status.

The empirical distribution of dropout time, conditional on exposure is:

For the unexposed:

```
edist0 <- with(subset(sample_subj, exposure == 0), table(dtime)/length(dtime))
edist0
```

```
## dtime
##       0.2       0.4       0.6       0.8         1
## 0.1449275 0.1775362 0.2644928 0.1992754 0.2137681
```

For the exposed:

```
edist1 <- with(subset(sample_subj, exposure == 1), table(dtime)/length(dtime))
edist1
```

```
## dtime
##       0.2       0.4       0.6       0.8         1
## 0.2012384 0.2260062 0.2074303 0.2291022 0.1362229
```

Our marginal parameters of interest are the slope in the unexposed and exposed groups as well as the slope difference. Had no difference, by exposure status, in the empirical distribution of dropout time been assumed we could simply have computed the marginal time by exposure interaction term. However, assuming the distribution differs, the difference in slopes must be computed explicitly since the two will be different and carry different interpretations. For code reuse, we will code the computations into a function:

```r
slope_n_diff <- function(fit_obj) {

  # compute empirical dropout time distributions
  edist0 <- with(subset(fit_obj$data, exposure == 0 & time == 0)
                 , table(dtime)/length(dtime))
  edist1 <- with(subset(fit_obj$data, exposure == 1 & time == 0)
                 , table(dtime)/length(dtime))

  # extract fixed effects
  feff <- fit_obj$fixed$estimate
  names(feff) <- row.names(fit_obj$fixed)

  # unique ordered dropout times
  dtimes <- unique(fit_obj$data$dtime)
  dtimes <- sort(dtimes)

  # compute marginal effects
  results <- data.frame(
    slope_unexposed = feff['time'] + edist0 %*% dtimes * feff['time:dtime']
  , slope_exposed = feff['time'] + feff['time:exposure'] +
                    edist1 %*% dtimes * feff['time:dtime'] +
                    edist1 %*% dtimes * feff['time:exposure:dtime']
  )

  results$slope_diff <- with(results, slope_exposed - slope_unexposed)

  return(results)

}

# compute the observed marginal slopes and slope difference
obs_splope_n_diff <- slope_n_diff(WL_fit)
obs_splope_n_diff
```

```
##   slope_unexposed slope_exposed slope_diff
## 1       0.8161838      2.430881   1.614697
```

Variances can be computed using the delta method, however, in an applied setting there are two issues to consider. First, the delta method approximation is only good to the degree that our model assumptions have been satisfied. Second, if the sample is sparsely distributed over many dropout time levels the delta method approximation will be poor (Hogan 2004a, 2004b). Due to these considerations we recommend resampling methods, such as the bootstrap (Davison and Hinkley 1997; Shao and Tu 1995), for general use. A simple bootstrap procedure, where rows are selected independently, would be inappropriate here and lead to variance estimates that are far too small. The correct bootstrap procedure will reflect the sampling procedure and, for ODS, this is a stratified cluster sample. The BUILDER function *strata_clus_boot()* was designed to facilitate this for a BUILD analysis. It accepts a data frame and returns a single bootstrap sample, based on specified strata and cluster identifiers. In addition, the new dataframe will contain a pseudo cluster identifier (*.ClusID.*) that will need to be used in cluster based analysis in place of the original identifier. The arguments of the function are:

- **df**: a dataframe to select a bootstrap sample from
- **strata**: character, name of a variable that defines stratum membership
- **cluster**: character, name of a variable that defines cluster membership

Bootstrap analyses are computationally intensive and BUILD models can be complex. Together, CPU time may be prohibitive. Parallel processing will be effective in reducing computation time and is simple to implement using th doParallel package (Corporation and Weston 2022). In addition, the doRNG package (Gaujoux 2025) allows the seamless use of seed values for reproducibility when parallel processing is used. Here is an example bootstrap analysis analysis.

```
# function to carryout a single analysis
boot_fun <- function(data) {

  #draw a bootstrap sample
  B <- strata_clus_boot(df = data
                        , strata = c('oc_strata','dtime')
                        , cluster = 'id')
  # fit the model
  ## note: robust standard errors are intensive and not needed in a bootstrap analysis
  B_WL_fit <- logit_mix(df = B
                        , fixed = y ~ time + exposure + time:exposure +
                                      dtime + time:dtime + exposure:dtime +
                                      time:exposure:dtime
                        , random = ~ 1
                        , cluster = '.ClusID.'
                        , sweights = 'sampWT'
                        , robust = F
                        , qpoints = 20
                        )

  return(slope_n_diff(B_WL_fit))
}

boot_fun(sample)
```

```
##   slope_unexposed slope_exposed slope_diff
## 1       0.8183421      2.083679   1.265337
```

Next the process is iterated in parallel. Note that the analysis is set for only 100 bootstrap iterations for brevity in run time, but in an actual analysis this number would be based on an evaluation of the bootstrap distribution and desired precision.

```
library(doParallel)
library(doRNG)

# set up a cluster
n_cores <- detectCores() - 1
registerDoParallel(cores = n_cores)

# set the number of bootstrap iterations
n_boots <- 100
system.time({
B_result <- foreach(1:n_boots
                    , .packages = c('build')
                    , .options.RNG = 4879549
                    , .combine = 'rbind'
) %dorng% {
  boot_fun(sample)
}
})
```

```
##    user  system elapsed
##    0.14    0.03   95.04
```

```
#stop the cluster
stopImplicitCluster()

# compute bootstrap SE's
bse <- sqrt(diag(var(B_result)))

# compute test statistics
Z <- obs_splope_n_diff / bse

# compute p-values
p_val <- sapply(Z, function(Z) {pchisq(q = Z^2, df = 1, lower.tail = F)})

# compute 95% confidence limits
lcl <- obs_splope_n_diff - qnorm(0.975)*bse
ucl <- obs_splope_n_diff + qnorm(0.975)*bse

# summary table
boot_tab <- data.frame(
  estimate = t(obs_splope_n_diff)
 ,lcl_95 = t(lcl)
 ,ucl_95 = t(ucl)
 ,pvalue = format.pval(p_val, eps=0.001, digits = 2)
)
row.names(boot_tab) <- names(obs_splope_n_diff)
knitr::kable(boot_tab
             , digits = 2
             , caption = 'Results of bootstap analysis of marginal parameters')
```

Table 2: Results of bootstap analysis of marginal parameters

	estimate	lcl_95	ucl_95	pvalue
slope_unexposed	0.82	0.23	1.40	0.0062
slope_exposed	2.43	1.63	3.23	<0.001
slope_diff	1.61	0.55	2.68	0.0029

This example assumes that the bootstrap distribution is normal. This may be a safe assumption when the data source is large, but the bootstrap distributions should always be inspected to ensure that they meet this assumption. Statistics computed for small categories, in otherwise large data sets, may also exhibit non-normality. When normality is in question percentile bootstrap intervals or other advances methods, such as bias-corrected and Accelerated (BCa), may be advisable (Davison and Hinkley 1997).

3 Discussion

Nonignorable dropout is a potential concern for longitudinal studies. There are well developed methods for exposure based sampling and randomized designs to obtain adjusted analyses, but until now, not for informed sampling strategies. This tutorial has introduces the BUILD package implementing our expansion to outcome dependent sampling to accommodate dropout effects. The use of mixture models allows a solution that does not require distributional assumptions on the dropout distribution (Verbeke and Molenberghs 2000) and is readily extensible to cover a wide variety of dropout associations and even multiple dropout reasons (C. M. Moore et al. 2019; Camille M. Moore et al. 2017, 2020; Camille M. Moore, Carlson, and Kreidler 2019).

The increased rate of storage of biosamples with longitudinal studies is of great importance to biomedical research but the lack of tools to leverage these data sources has been a significant barrier to increased utilization. BUILD is a flexible and powerful solution which will aid investigators to develop informative studies with these valuable resources.

Corporation, Microsoft, and Steve Weston. 2022. *doParallel: Foreach Parallel Adaptor for the 'Parallel' Package*. https://doi.org/10.32614/CRAN.package.doParallel.

Davison, A. C., and A. V. Hinkley. 1997. *Bootstrap Methods and Their Applications*. Cambridge University Press.

Forster, Jeri E., Samantha MaWhinney, Erika L. Ball, and Diane Fairclough. 2011. "A Varying-Coefficient Method for Analyzing Longitudinal Clinical Trials Data with Nonignorable Dropout." Journal Article. *Contemporary Clinical Trials* 33 (2): 378–85. https://doi.org/10.1016/j.cct.2011.11.009.

Gaujoux, Renaud. 2025. *doRNG: Generic Reproducible Parallel Backend for 'Foreach' Loops*. https://doi.org/10.32614/CRAN.package.doRNG.

Hogan, J. W. 2004a. "Mixtures of Varying-Coefficient Models for Longitudinal Data with Discrete or Continuous Nonignorable Dropout." Journal Article. *Biometrics* 60: 854–64.

———. 2004b. "Tutorial in Biostatistics, Handling Drop-Out in Longitudinal Studies." Journal Article. *Statistics in Medicine* 23: 1455–97.

McCulloch, C. E., S. R. Searle, and J. M. Neuhaus. 2008. *Generalized, Linear, and Mixed Models, 2nd Edition*. John Wiley; Sons, Inc.

Moore, C. M., S. MaWhinney, N. E. Carlson, and Kreidler. 2019. "InformativeDropout Package." GitHub.

Moore, Camille M., Samantha Carlson Nichole E. MaWhinney, and Sarah Kreidler. 2019. "A Dirichlet Process Mixture Model for Non-Ignorable Dropout." Advance Publication, https://projecteuclid.org/download/pdfview_1/euclid.ba *Bayesian Analysis*.

Moore, Camille M., Samantha MaWhinney, Nichole E. Carlson, and Sarah Kreidler. 2020. "A Bayesian Natural Cubic b-Spline Varying Coefficient Method for Non-Ignorable Dropout." Journal Article. *BMC Med Res Methodol* 20 (1). https://doi.org/0.1186/s12874-020-01135-3.

Moore, Camille M., Samantha MaWhinney, Jeri E. Forster, Nichole E. Carlson, Amanda Allshouse, Xinshuo Wang, Jean-Pierre Routy, Brian Conway, and Elizabeth Connick. 2017. "Accounting for Dropout Reason in Longitudinal Studies with Nonignorable Dropout." Journal Article. *Statistical Methods in Medical Research* 26 (4): 1854–66. https://doi.org/10.1177/0962280215590432.

R Core Team. 2023. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rabe-Hesketh, Sophia, and Anders Skrondal. 2006. "Multilevel Modelling of Complex Survey Data." *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 169 (4): 805–27. https://doi.org/https://doi.org/10.1111/j.1467-985X.2006.00426.x.

Schildcrout, J. S. 2018. "Extending the Case-Control Design to Longitudinal Data: Stratified Sampling Based on Repeated Binary Outcomes." Journal Article. *Epidemiology* 29 (1): 67–75. https://doi.org/10.1097/EDE.0000000000000764.

Sevick, C. J. 2025. "Statistical Methods for Binary Outcomes Adjusting for Outcome Dependent Sampling in Longitudinal Studies with Nonignorable Dropout." PhD thesis, University of Colorado (MaWhinney Mentor).

Shao, J., and D. Tu. 1995. *The Jackknife and Bootstrap*. Springer.

Verbeke, G., and G. Molenberghs. 2000. *Linear Mixed Models for Longitudinal Data*. Springer Publishiing.

Wilson, M. P. 2019. "Effects of Missing Data on Outcome-Dependent Sampling Methods for Longitudinal Cohorts with Binary Data." Master's thesis, University of Colorado (MaWhinney Mentor).

Wu, M. C., and K. Bailey. 1989. "Estimation and Comparison of Changes in the Presence of Informative Right Censoring; Conditional Linear Model." Journal Article. *Biometrics* 45: 939–55.