been working on updates for the simstudy package. In the past few weeks, a couple of folks independently reached out to me about generating correlated binary data. I’ve added an option to use an algorithm developed by [Emrich and Piedmonte](https://www.tandfonline.com/doi/abs/10.1080/00031305.1991.10475828) in 1991, and will be incorporating that option soon in the functions genCorGen and addCorGen. I’ll write about that change some point soon.

A second researcher was trying to generate data using parameters that could be recovered using GEE model estimation. I’ve always done this by using an underlying mixed effects model, but of course, the marginal model parameter estimates might be quite different from the conditional parameters. As a result, the model and the data generation process don’t match, which may not be such a big deal, but is not so helpful when trying to illuminate the models.

Conditional Vs Marginal Model For clustered Data

The marginal model is estimated using a generalized estimating equation (GEE) model (here using function geeglm in package geepack). If the intervention is binary, the intervention effect (log-odds ratio) is interpreted as the average effect across all individuals regardless of the group or cluster they might belong to. (This estimate is sensitive to the relative sizes of the clusters.)

The conditional model is estimated using a random mixed effect generalized linear model (using function glmer in package lme4), and provides the log-odds ratio conditional on the cluster. (The estimate is not as sensitive to the relative sizes of the clusters since it is essentially providing a within-cluster effect.)

As the variation across clusters increases, so does the discrepancy between the conditional and marginal models. Using a generalized linear model that ignores clustering altogether will provide the correct (marginal) point estimate, but will underestimate the underlying variance (and standard errors) as long as there is between cluster variation. If there is no between cluster variation, the GLM model should be fine.

### Simulation

To start, here is a function that uses simstudy to define and generate a data set of individuals that are clustered in groups. A key argument passed to this function is the across cluster variation.

**library**(lme4)

**library**(geepack)

**library**(broom)

genFunc <- **function**(nClusters, effVar) {

# define the cluster

def1 <- defData(varname = "clustEff", formula = 0,

variance = effVar, id = "cID")

def1 <- defData(def1, varname = "nInd", formula = 40,

dist = "noZeroPoisson")

# define individual level data

def2 <- defDataAdd(varname = "Y", formula = "-2 + 2\*grp + clustEff",

dist = "binary", link = "logit")

# generate cluster level data

dtC <- genData(nClusters, def1)

dtC <- trtAssign(dtC, grpName = "grp")

# generate individual level data

dt <- genCluster(dtClust = dtC, cLevelVar = "cID", numIndsVar = "nInd",

level1ID = "id")

dt <- addColumns(def2, dt)

**return**(dt)

}

A plot of the average site level outcome from data generated with across site variance of 1 (on the log-odds scale) shows the treatment effect:

set.seed(123)

dt <- genFunc(100, 1)

dt

## cID grp clustEff nInd id Y

## 1: 1 0 -0.5604756 35 1 1

## 2: 1 0 -0.5604756 35 2 0

## 3: 1 0 -0.5604756 35 3 0

## 4: 1 0 -0.5604756 35 4 0

## 5: 1 0 -0.5604756 35 5 0

## ---

## 3968: 100 1 -1.0264209 45 3968 0

## 3969: 100 1 -1.0264209 45 3969 0

## 3970: 100 1 -1.0264209 45 3970 1

## 3971: 100 1 -1.0264209 45 3971 0

## 3972: 100 1 -1.0264209 45 3972 0

dplot <- dt[, mean(Y), keyby = .(grp, cID)]

davg <- dt[, mean(Y)]

ggplot(data = dplot, aes(x=factor(grp), y = V1)) +

geom\_jitter(aes(color=factor(grp)), width = .10) +

theme\_ksg("grey95") +

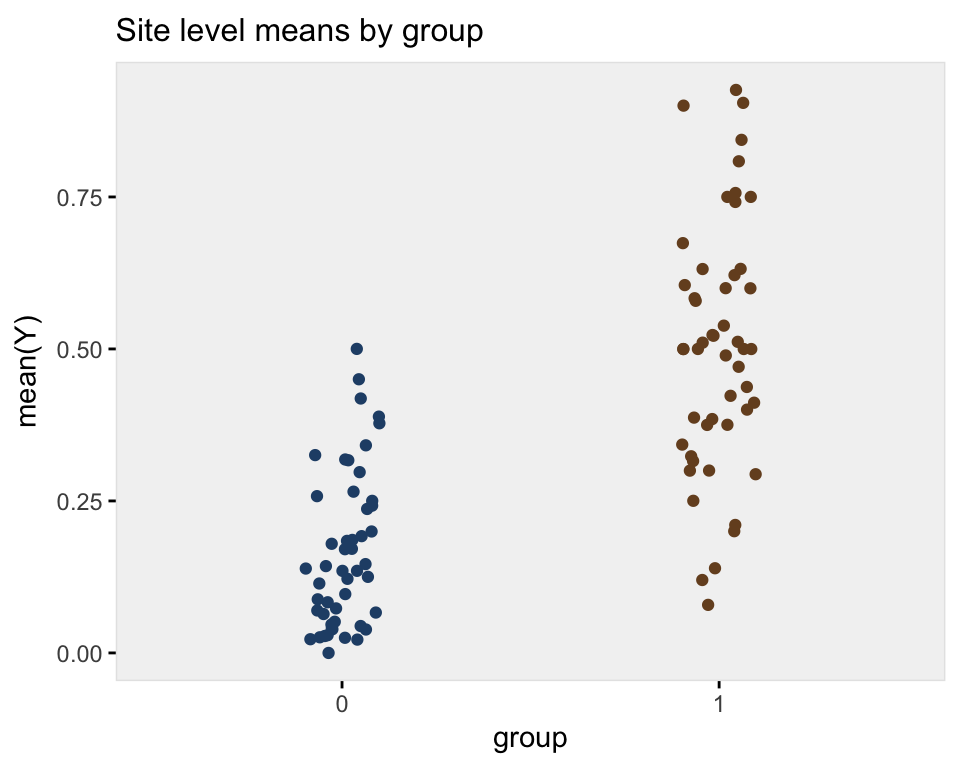
xlab("group") +

ylab("mean(Y)") +

theme(legend.position = "none") +

ggtitle("Site level means by group") +

scale\_color\_manual(values = c("#264e76", "#764e26"))



### Model fits

First, the conditional model estimates a log-odds ratio of 1.89, close to the actual log-odds ratio of 2.0.

glmerFit <- glmer(Y ~ grp + (1 | cID), data = dt, family="binomial")

tidy(glmerFit)

## term estimate std.error statistic p.value group

## 1 (Intercept) -1.8764913 0.1468104 -12.781729 2.074076e-37 fixed

## 2 grp 1.8936999 0.2010359 9.419711 4.523292e-21 fixed

## 3 sd\_(Intercept).cID 0.9038166 NA NA NA cID

The marginal model that takes into account clustering yields an estimate of 1.63. This model is not wrong, just estimating a different quantity:

geeFit <- geeglm(Y ~ grp, family = binomial, data = dt,

corstr = "exchangeable", id = dt$cID)

tidy(geeFit)

## term estimate std.error statistic p.value

## 1 (Intercept) -1.620073 0.1303681 154.42809 0

## 2 grp 1.628075 0.1740666 87.48182 0

The marginal model that ignores clustering also estimates a log-odds ratio, 1.67, but the standard error estimate is much smaller than in the previous model (0.076 vs. 0.174). We could say that this model is not appropriate given the clustering of individuals:

glmFit <- glm(Y ~ grp, data = dt, family="binomial")

tidy(glmFit)

## term estimate std.error statistic p.value

## 1 (Intercept) -1.639743 0.0606130 -27.05267 3.553136e-161

## 2 grp 1.668143 0.0755165 22.08978 3.963373e-108

### Multiple replications

With multiple replications (in this case 100), we can see how each model performs under different across cluster variance assumptions. I have written two functions (that are shown at the end in the appendix) to generate multiple datasets and create a plot. The plot shows (1) the average point estimate across all the replications in black, (2) the true standard deviation of all the point estimates across all replications in blue, (3) the average estimate of the standard errors in orange.

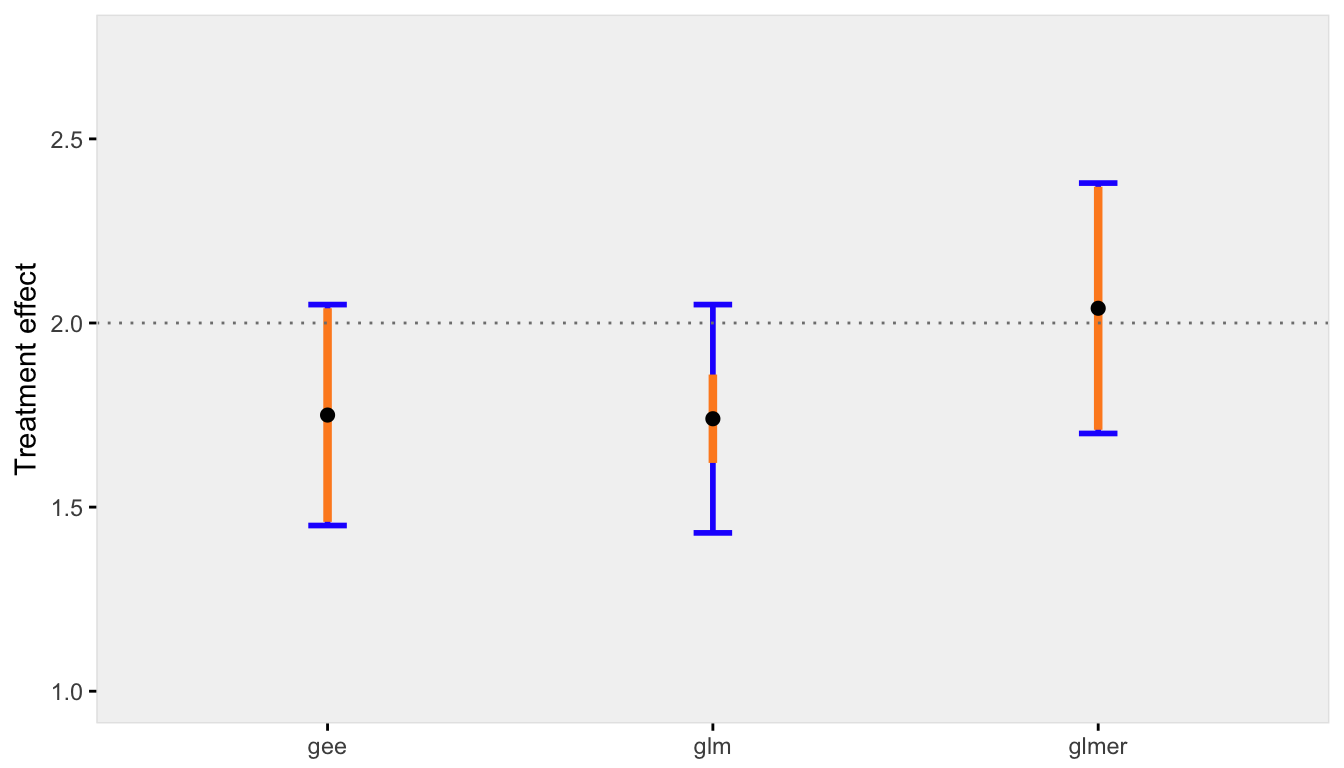
In the first case, the variability across sites is highest. The discrepancy between the marginal and conditional models is relatively large, but both the GEE and mixed effects models estimate the standard errors correctly (the orange line overlaps perfectly with blue line). The generalized linear model, however, provides a biased estimate of the standard error - the orange line does not cover the blue line:

set.seed(235)

res1.00 <- iterFunc(40, 1.00, 100)

s1 <- sumFunc(res1.00)

s1$p

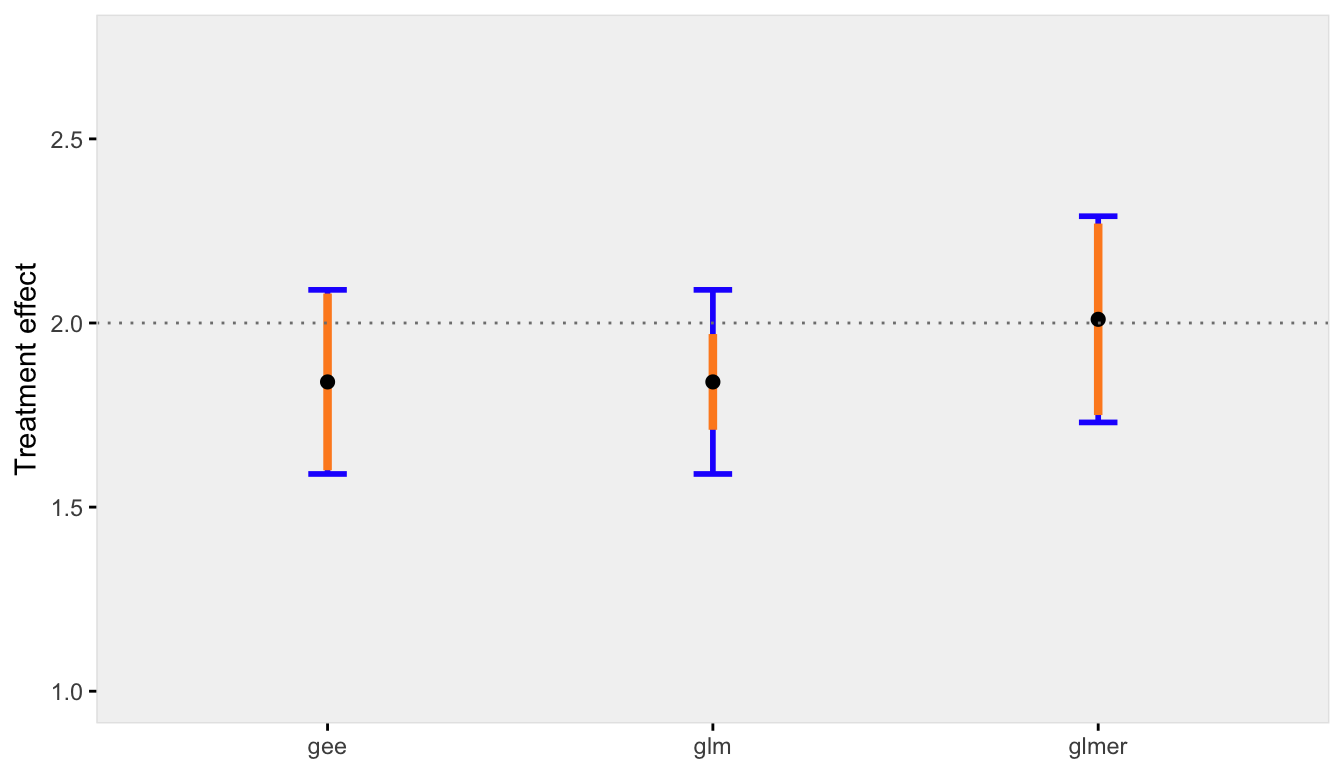


When the across cluster variation is reduced, the discrepancy between the marginal and conditional models is reduced, as is the bias of standard error estimate for the GLM model:

res0.50 <- iterFunc(40, 0.50, 100)

s2 <- sumFunc(res0.50)

s2$p

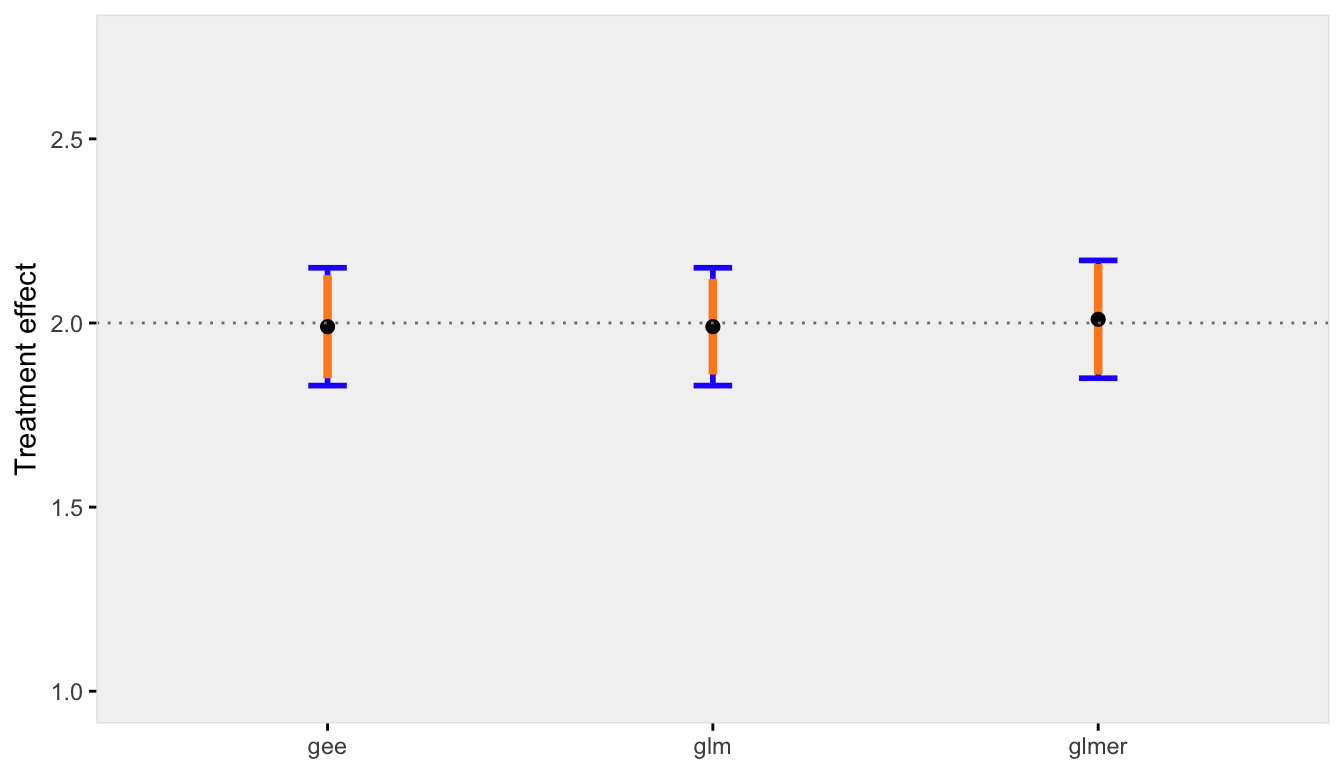


Finally, when there is negligible variation across sites, the conditional and marginal models are pretty much one and the same. And even the GLM model that ignores clustering is unbiased (which makes sense, since there really is no clustering):

res0.05 <- iterFunc(40, 0.05, 100)

s3 <- sumFunc(res0.05)

s3$p



### Appendix

Here are the two functions that generated the the replications and created the plots shown above.

iterFunc <- **function**(nClusters, effVar, iters = 250) {

results <- list()

**for** (i **in** 1:iters) {

dt <- genFunc(nClusters, effVar)

glmerFit <- glmer(Y ~ grp + (1 | cID), data = dt, family="binomial")

glmFit <- glm(Y ~ grp, data = dt, family="binomial")

geeFit <- geeglm(Y ~ grp, family = binomial, data = dt,

corstr = "exchangeable", id = dt$cID)

res <- unlist(c(coef(summary(glmerFit))[2,1:2],

coef(summary(glmFit))[2,1:2],

as.vector(coef(summary(geeFit))[2,1:2])))

results[[i]] <- data.table(t(res))

}

**return**(rbindlist(results))

}

sumFunc <- **function**(dtRes, precision = 2) {

setnames(dtRes, c("estGlmer", "sdGlmer",

"estGlm","sdGlm",

"estGEE", "sdGEE"))

meanEst <- round(apply(dtRes[, c(1, 3, 5)], 2, mean), precision)

estSd <- round(sqrt(apply(dtRes[, c(2, 4, 5)]^2, 2, mean)), precision)

sdEst <- round(apply(dtRes[, c(1, 3, 5)], 2, sd), precision)

x <- data.table(rbind(c(meanEst[1], estSd[1], sdEst[1]),

c(meanEst[2], estSd[2], sdEst[2]),

c(meanEst[3], estSd[3], sdEst[3])

))

setnames(x, c("estMean","estSD","sd"))

x[, method := c("glmer","glm","gee")]

p <- ggplot(data = x, aes(x = method, y = estMean)) +

geom\_errorbar(aes(ymin = estMean - sd, ymax = estMean + sd),

width = 0.1, color = "#2329fe", size = 1) +

geom\_errorbar(aes(ymin = estMean - estSD, ymax = estMean + estSD),

width = 0.0, color = "#fe8b23", size = 1.5) +

geom\_point(size = 2) +

ylim(1,2.75) +

theme\_ksg("grey95") +

geom\_hline(yintercept = 2, lty = 3, color = "grey50") +

theme(axis.title.x = element\_blank()) +

ylab("Treatment effect")

**return**(list(mean=meanEst, sd=sdEst, p=p))

}

One simple solution is using a *beta-binomial* mixture data generating process. The [*beta* distribution](https://en.wikipedia.org/wiki/Beta_distribution) is a continuous probability distribution that is defined on the interval from 0 to 1, so it is not too unreasonable as model for probabilities. If we assume that cluster-level probabilities have a beta distribution, and that within each cluster the individual outcomes have a *binomial* distribution defined by the cluster-specific probability, we will get the data generation process we are looking for.

**Generating the clustered data**

In these examples, I am using 500 clusters, each with cluster size of 40 individuals. There is a cluster-level covariate x that takes on integer values between 1 and 3. The beta distribution is typically defined using two shape parameters usually referenced as \(\alpha\) and \(\beta\), where \(E(Y) = \alpha / (\alpha + \beta)\), and \(Var(Y) = (\alpha\beta)/[(\alpha + \beta)^2(\alpha + \beta + 1)]\). In simstudy, the distribution is specified using the mean probability (\(p\_m\)) and a *precision* parameter (\(\phi\_\beta > 0\)) (that is specified using the variance argument). Under this specification, \(Var(Y) = p\_m(1 – p\_m)/(1 + \phi\_\beta)\). Precision is inversely related to variability: lower precision is higher variability.

In this simple simulation, the cluster probabilities are a function of the cluster-level covariate and precision parameter \(\phi\_\beta\). Specifically

\[logodds(p\_{clust}) = -2.0 + 0.65x.\]  
The binomial variable of interest \(b\) is a function of \(p\_{clust}\) only, and represents a count of individuals in the cluster with a “success”:

library(simstudy)

set.seed(87387)

phi.beta <- 3 # precision

n <- 40 # cluster size

def <- defData(varname = "n", formula = n,

dist = 'nonrandom', id = "cID")

def <- defData(def, varname = "x", formula = "1;3",

dist = 'uniformInt')

def <- defData(def, varname = "p", formula = "-2.0 + 0.65 \* x",

variance = phi.beta, dist = "beta", link = "logit")

def <- defData(def, varname = "b", formula = "p", variance = n,

dist = "binomial")

dc <- genData(500, def)

dc

## cID n x p b

## 1: 1 40 2 0.101696930 4

## 2: 2 40 2 0.713156596 32

## 3: 3 40 1 0.020676443 2

## 4: 4 40 2 0.091444678 4

## 5: 5 40 2 0.139946091 6

## ---

## 496: 496 40 1 0.062513419 4

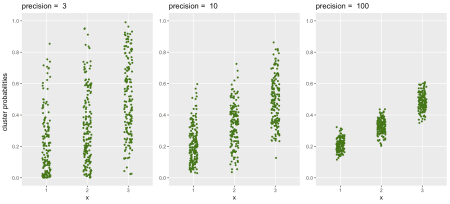
## 497: 497 40 1 0.223149651 5

## 498: 498 40 3 0.452904009 14

## 499: 499 40 2 0.005143594 1

## 500: 500 40 2 0.481283809 16

The generated data with \(\phi\_\beta = 3\) is shown on the left below. Data sets with increasing precision (less variability) are shown to the right:



The relationship of \(\phi\_\beta\) and variance is made clear by evaluating the variance of the cluster probabilities at each level of \(x\) and comparing these variance estimates with the theoretical values suggested by parameters specified in the data generation process:

p.clust = 1/(1 + exp(2 - 0.65\*(1:3)))

cbind(dc[, .(obs = round(var(p), 3)), keyby = x],

theory = round( (p.clust\*(1 - p.clust))/(1 + phi.beta), 3))

## x obs theory

## 1: 1 0.041 0.041

## 2: 2 0.054 0.055

## 3: 3 0.061 0.062

**Beta and beta-binomial regression**

Before getting to the GEE estimation, here are two less frequently used regression models: beta and beta-binomial regression. Beta regression may not be super-useful, because we would need to observe (and measure) the probabilities directly. In this case, we randomly generated the probabilities, so it is fair to estimate a regression model to recover the same parameters we used to generate the data! But, back in the real world, we might only observe \(\hat{p}\), which results from generating data based on the underlying true \(p\). This is where we will need the beta-binomial regression (and later, the GEE model).

First, here is the beta regression using package betareg, which provides quite good estimates of the two coefficients and the precision parameter \(\phi\_\beta\), which is not so surprising given the large number of clusters in our sample:

library(betareg)

model.beta <- betareg(p ~ x, data = dc, link = "logit")

summary(model.beta)

##

## Call:

## betareg(formula = p ~ x, data = dc, link = "logit")

##

## Standardized weighted residuals 2:

## Min 1Q Median 3Q Max

## -3.7420 -0.6070 0.0306 0.6699 3.4952

##

## Coefficients (mean model with logit link):

## Estimate Std. Error z value Pr(>|z|)

## (Intercept) -2.09663 0.12643 -16.58 <2e-16 \*\*\*

## x 0.70080 0.05646 12.41 <2e-16 \*\*\*

##

## Phi coefficients (precision model with identity link):

## Estimate Std. Error z value Pr(>|z|)

## (phi) 3.0805 0.1795 17.16 <2e-16 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Type of estimator: ML (maximum likelihood)

## Log-likelihood: 155.2 on 3 Df

## Pseudo R-squared: 0.2388

## Number of iterations: 13 (BFGS) + 1 (Fisher scoring)

The beta-binomial regression model, which is estimated using package aod, is a reasonable model to fit in this case where we have observed binomial outcomes and unobserved underlying probabilities:

library(aod)

model.betabinom <- betabin(cbind(b, n - b) ~ x, ~ 1, data = dc)

model.betabinom

## Beta-binomial model

## -------------------

## betabin(formula = cbind(b, n - b) ~ x, random = ~1, data = dc)

##

## Convergence was obtained after 100 iterations.

##

## Fixed-effect coefficients:

## Estimate Std. Error z value Pr(> |z|)

## (Intercept) -2.103e+00 1.361e-01 -1.546e+01 0e+00

## x 6.897e-01 6.024e-02 1.145e+01 0e+00

##

## Overdispersion coefficients:

## Estimate Std. Error z value Pr(> z)

## phi.(Intercept) 2.412e-01 1.236e-02 1.951e+01 0e+00

##

## Log-likelihood statistics

## Log-lik nbpar df res. Deviance AIC AICc

## -1.711e+03 3 497 1.752e+03 3.428e+03 3.428e+03

A couple of interesting things to note here. First is that the coefficient estimates are pretty similar to the beta regression model. However, the standard errors are slightly higher, as they should be, since we are using only observed probabilities and not the true (albeit randomly selected or generated) probabilities. So, there is another level of uncertainty beyond sampling error.

Second, there is a new parameter: \(\phi\_{overdisp}\). What is that, and how does that relate to \(\phi\_\beta\)? The variance of a binomial random variable \(Y\) with a single underlying probability is \(Var(Y) = np(1-p)\). However, when the underlying probability varies across different subgroups (or clusters), the variance is augmented by \(\phi\_{overdisp}\): \(Var(Y) = np(1-p)[1 + (n-1)\phi\_{overdisp}]\). It turns out to be the case that \(\phi\_{overdisp} = 1/(1+\phi\_\beta)\):

round(model.betabinom@random.param, 3) # from the beta - binomial model

## phi.(Intercept)

## 0.241

round(1/(1 + coef(model.beta)["(phi)"]), 3) # from the beta model

## (phi)

## 0.245

The observed variances of the binomial outcome \(b\) at each level of \(x\) come quite close to the theoretical variances based on \(\phi\_\beta\):

phi.overdisp <- 1/(1+phi.beta)

cbind(dc[, .(obs = round(var(b),1)), keyby = x],

theory = round( n\*p.clust\*(1-p.clust)\*(1 + (n-1)\*phi.overdisp), 1))

## x obs theory

## 1: 1 69.6 70.3

## 2: 2 90.4 95.3

## 3: 3 105.2 107.4

**GEE and individual level data**

With individual level binary outcomes (as opposed to count data we were working with before), GEE models are appropriate. The code below generates individual-level for each cluster level:

defI <- defDataAdd(varname = "y", formula = "p", dist = "binary")

di <- genCluster(dc, "cID", numIndsVar = "n", level1ID = "id")

di <- addColumns(defI, di)

di

## cID n x p b id y

## 1: 1 40 2 0.1016969 4 1 0

## 2: 1 40 2 0.1016969 4 2 0

## 3: 1 40 2 0.1016969 4 3 0

## 4: 1 40 2 0.1016969 4 4 0

## 5: 1 40 2 0.1016969 4 5 1

## ---

## 19996: 500 40 2 0.4812838 16 19996 0

## 19997: 500 40 2 0.4812838 16 19997 0

## 19998: 500 40 2 0.4812838 16 19998 1

## 19999: 500 40 2 0.4812838 16 19999 1

## 20000: 500 40 2 0.4812838 16 20000 0

The GEE model provides estimates of the coefficients as well as the working correlation. If we assume an “exchangeable” correlation matrix, in which each individual is correlated with all other individuals in the cluster but is not correlated with individuals in other clusters, we will get a single correlation estimate, which is labeled as *alpha* in the GEE output:

library(geepack)

geefit <- geeglm(y ~ x, family = "binomial", data = di,

id = cID, corstr = "exchangeable" )

summary(geefit)

##

## Call:

## geeglm(formula = y ~ x, family = "binomial", data = di, id = cID,

## corstr = "exchangeable")

##

## Coefficients:

## Estimate Std.err Wald Pr(>|W|)

## (Intercept) -2.07376 0.14980 191.6 <2e-16 \*\*\*

## x 0.68734 0.06566 109.6 <2e-16 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Estimated Scale Parameters:

## Estimate Std.err

## (Intercept) 1 0.03235

##

## Correlation: Structure = exchangeable Link = identity

##

## Estimated Correlation Parameters:

## Estimate Std.err

## alpha 0.256 0.01746

## Number of clusters: 500 Maximum cluster size: 40

In this case, *alpha* (\(\alpha\)) is estimated at 0.25, which is quite close to the previous estimate of \(\phi\_{overdisp}\), 0.24. So, it appears to be the case that if we have a target correlation \(\alpha\), we know the corresponding \(\phi\_\beta\) to use in the beta-binomial data generation process. That is, \(\phi\_\beta = (1 – \alpha)/\alpha\).

While this is certainly not a proof of anything, let’s give it a go with a target \(\alpha = 0.44\):

phi.beta.new <- (1-0.44)/0.44

def <- updateDef(def, "p", newvariance = phi.beta.new)

dc2 <- genData(500, def)

di2 <- genCluster(dc2, "cID", numIndsVar = "n", level1ID = "id")

di2 <- addColumns(defI, di2)

geefit <- geeglm(y ~ x, family = "binomial", data = di2,

id = cID, corstr = "exchangeable" )

summary(geefit)

##

## Call:

## geeglm(formula = y ~ x, family = "binomial", data = di2, id = cID,

## corstr = "exchangeable")

##

## Coefficients:

## Estimate Std.err Wald Pr(>|W|)

## (Intercept) -1.7101 0.1800 90.3 < 2e-16 \*\*\*

## x 0.5685 0.0806 49.8 1.7e-12 \*\*\*

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## Estimated Scale Parameters:

## Estimate Std.err

## (Intercept) 1 0.0307

##

## Correlation: Structure = exchangeable Link = identity

##

## Estimated Correlation Parameters:

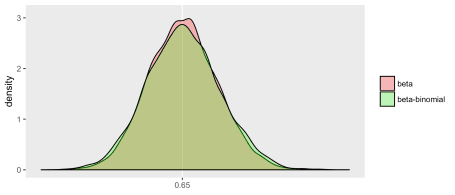
## Estimate Std.err

## alpha 0.444 0.0242

## Number of clusters: 500 Maximum cluster size: 40

**Addendum**

Above, I suggested that the estimator of the effect of x based on the beta model will have less variation than the estimator based on the beta-binomial model. I drew 5000 samples from the data generating process and estimated the models each time. Below is a density distribution of the estimates of each of the models from all 5000 iterations. As expected, the beta-binomial process has more variability, as do the related estimates; we can see this in the relative “peakedness”" of the beta density:



Also based on these 5000 iterations, the GEE model estimation appears to be less efficient than the beta-binomial model. This is not surprising since the beta-binomial model was the actual process that generated the data (so it is truly the correct model). The GEE model is robust to mis-specification of the correlation structure, but the price we pay for that robustness is a slightly less precise estimate (even if we happen to get the correlation structure right):

