Data on successes and failures can be summarized and analyzed as counted proportions via the binomial distribution or as long format 0/1 binary data. I most often see summarized data when there are multiple trials done within a study unit; for example, when tallying up the number of dead trees out of the total number of trees in a plot.

If these within-plot trials are all independent, analyzing data in a binary format instead of summarized binomial counts doesn’t change the statistical results. If trials are not independent, though, neither approach works correctly and we would see overdispersion/underdispersion in a binomial model.

In a future post I’ll talk more about simulating data to explore binomial overdispersion and how lack of fit can be diagnosed in binomial vs binary datasets. Today, however, my goal is show how to take binomial count data and expand it into binary data.

In the past We’ve done the data expansion with rowwise() and do() from package **dplyr**, but these days I’m using purrr::pmap\_dfr(). We’ll demonstrate the pmap\_dfr() approach as well as a nest()/unnest() approach using functions from **tidyr**.

**Load R packages**

I’m using **purrr** for looping through rows with pmap\_dfr(). I also load **dplyr** and **tidyr** for a nest()/unnest() approach.

library(purrr) # 0.3.2

library(tidyr) # 1.0.0

library(dplyr) # 0.8.3

**The dataset**

I created a dataset with a total of 8 plots, 4 plots in each of two groups. The data has been summarized up to the plot level. The number of trials (total) per plot varied. The number of successes observed is in num\_dead.

dat = structure(list(plot = structure(1:8, .Label = c("plot1", "plot2",

"plot3", "plot4", "plot5", "plot6", "plot7", "plot8"), class = "factor"),

group = structure(c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L), .Label = c("g1",

"g2"), class = "factor"), num\_dead = c(4L, 6L, 6L, 5L, 1L, 4L,

3L, 2L), total = c(5L, 7L, 9L, 7L, 8L, 10L, 10L, 7L)), class = "data.frame", row.names = c(NA,

-8L))

dat

# plot group num\_dead total

# 1 plot1 g1 4 5

# 2 plot2 g1 6 7

# 3 plot3 g1 6 9

# 4 plot4 g1 5 7

# 5 plot5 g2 1 8

# 6 plot6 g2 4 10

# 7 plot7 g2 3 10

# 8 plot8 g2 2 7

**Expanding binomial to binary with pmap\_dfr()**

To make the binomial data into binary data, I need to make a vector with a \(1\) for every “success” listed in num\_dead and a \(0\) for every “failure” (the total number of trials minus the number of successes). Since I want to do a *rowwise* operation I’ll use one of the pmap functions. I want my output to be a data.frame so I use pmap\_dfr().

I use an anonymous function within pmap\_dfr() for creating the output I want from each row. I purposely make the names of the function arguments match the column names. You can either match on position or on names in pmap functions, and I tend to go for name matching. You can use formula coding with the tilde in pmap variants, but I find the code more difficult to understand when I have more than three or so columns.

Within the function I make a column for the response variable, repeating \(1\) num\_dead times and \(0\) total - num\_dead times for each row of the original data. I’m taking advantage of recycling in data.frame() to keep the plot and group columns in the output, as well.

binary\_dat = pmap\_dfr(dat,

function(group, plot, num\_dead, total) {

data.frame(plot = plot,

group = group,

dead = c( rep(1, num\_dead),

rep(0, total - num\_dead) ) )

}

)

Here are the first 6 rows of this new dataset. You can see for the first plot, plot1, there are five rows of \(1\) and one row of \(0\).

head(binary\_dat)

# plot group dead

# 1 plot1 g1 1

# 2 plot1 g1 1

# 3 plot1 g1 1

# 4 plot1 g1 1

# 5 plot1 g1 0

# 6 plot2 g1 1

This matches the information in the first row of the original dataset.

dat[1, ]

# plot group num\_dead total

# 1 plot1 g1 4 5

**Aside: pmap functions with more columns**

My anonymous function in pmap\_dfr() works fine in its current form as long as every column is included as a function argument. If I had extra columns that I didn’t want to remove and wasn’t using in the function, however, I would get an error.

To bypass this problem you can add dots, ..., to the anonymous function to refer to all other columns not being used.

function(group, plot, num\_dead, total, ...)

**Comparing analysis results**

While I definitely learned that binomial data can be analyzed in binary format and returns identical results in a GLM class, for some reason I often have to re-convince myself this is true. This is clear when I do an analysis with each dataset and compare results.

**Binomial model**

Here’s results from comparing the two groups for the binomial model.

fit = glm( cbind(num\_dead, total - num\_dead) ~ group,

data = dat,

family = binomial)

summary(fit)$coefficients

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

# (Intercept) 1.098612 0.4364358 2.517237 0.0118279240

# groupg2 -2.014903 0.5748706 -3.504968 0.0004566621

**Binary model**

The binary model gives identical results for estimates and statistical tests.

fit\_binary = glm( dead ~ group,

data = binary\_dat,

family = binomial)

summary(fit\_binary)$coefficients

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

# (Intercept) 1.098612 0.4364354 2.517239 0.0118278514

# groupg2 -2.014903 0.5748701 -3.504971 0.0004566575

**Expanding binomial to binary via nesting**

Doing the expansion with nesting plus purrr::map() inside mutate() is another option, although this seems less straightforward to me for this particular case. It does keep the other variables in the dataset, though, without having to manually include them in the output data.frame like I did above.

binary\_dat2 = dat %>%

nest(data = c(num\_dead, total) ) %>%

mutate(dead = map(data, ~c( rep(1, .x$num\_dead),

rep(0, .x$total - .x$num\_dead) ) ) ) %>%

select(-data) %>%

unnest(dead)

head(binary\_dat2)

# # A tibble: 6 x 3

# plot group dead

#

# 1 plot1 g1 1

# 2 plot1 g1 1

# 3 plot1 g1 1

# 4 plot1 g1 1

# 5 plot1 g1 0

# 6 plot2 g1 1

**Just the code, please**

Here’s the code without all the discussion.

library(purrr) # 0.3.2

library(tidyr) # 1.0.0

library(dplyr) # 0.8.3

dat = structure(list(plot = structure(1:8, .Label = c("plot1", "plot2",

"plot3", "plot4", "plot5", "plot6", "plot7", "plot8"), class = "factor"),

group = structure(c(1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L), .Label = c("g1",

"g2"), class = "factor"), num\_dead = c(4L, 6L, 6L, 5L, 1L, 4L,

3L, 2L), total = c(5L, 7L, 9L, 7L, 8L, 10L, 10L, 7L)), class = "data.frame", row.names = c(NA,

-8L))

dat

binary\_dat = pmap\_dfr(dat,

function(group, plot, num\_dead, total) {

data.frame(plot = plot,

group = group,

dead = c( rep(1, num\_dead),

rep(0, total - num\_dead) ) )

}

)

head(binary\_dat)

dat[1, ]

function(group, plot, num\_dead, total, ...)

fit = glm( cbind(num\_dead, total - num\_dead) ~ group,

data = dat,

family = binomial)

summary(fit)$coefficients

fit\_binary = glm( dead ~ group,

data = binary\_dat,

family = binomial)

summary(fit\_binary)$coefficients

binary\_dat2 = dat %>%

nest(data = c(num\_dead, total) ) %>%

mutate(dead = map(data, ~c( rep(1, .x$num\_dead),

rep(0, .x$total - .x$num\_dead) ) ) ) %>%

select(-data) %>%

unnest(dead)

head(binary\_dat2)