An economist contacted me about the ability of simstudy to generate correlated ordinal categorical outcomes. He is trying to generate data as an aide to teaching cost-effectiveness analysis, and is hoping to simulate responses to a quality-of-life survey instrument, the EQ-5D. The particular instrument has five questions related to mobility, self-care, activities, pain, and anxiety. Each item has three possible responses: (1) no problems, (2) some problems, and (3) a lot of problems. Although the instrument has been designed so that each item is orthogonal (independent) from the others, it is impossible to avoid correlation. So, in generating (and analyzing) these kinds of data, it is important to take this into consideration.

I had recently added functions to generate correlated data from non-normal distributions, and I had also created a function that generates ordinal categorical outcomes, but there was nothing to address the data generation problem he had in mind. After a little back forth, I came up with some code that will hopefully address his needs. And I hope the new function genCorOrdCat is general enough to support other data generation needs as well.

Hidden Categorical Responses

**Categorical outcomes, generally**

Certainly, group membership is not necessarily intrinsically ordered. In a general categorical or multinomial outcome, a group does not necessarily have any quantitative relationship vis a vis the other groups. For example, if we were interested in primary type of meat consumption, individuals might be grouped into those favoring (1) chicken, (2) beef, (3) pork, or (4) no meat. We might be interested in estimating the different distributions across the four groups for males and females. However, since there is no natural ranking or ordering of these meat groups (though maybe I am just not creative enough), we are limited to comparing the odds of being in one group relative to another for two exposure groups A and B, such as

\small{\frac{P(Beef|Group = A)}{P(Chicken|Group = A)} \ vs. \frac{P(Beef|Group = B)}{P(Chicken|Group = B)}}*P*(*Chicken*∣*Group*=*A*)*P*(*Beef*∣*Group*=*A*)​ *vs*.*P*(*Chicken*∣*Group*=*B*)*P*(*Beef*∣*Group*=*B*)​.

**Ordinal outcomes**

Order becomes relevant when the categories take on meanings related strength of opinion or agreement (as in a Likert-type response) or frequency. In the motivating example I described in the initial post, the response of interest was the frequency meat consumption in a month, so the response categories could be (1) none, (2) 1-3 times per month, (3) once per week, (4) 2-6 times per week, (5) 1 or more times per day. Individuals in group 2 consume meat more frequently than group 1, individuals in group 3 consume meat more frequently than those both groups 1 & 2, and so on. There is a natural quantitative relationship between the groups.

Once we have thrown ordering into the mix, we can expand our possible interpretations of the data. In particular it is quite common to summarize the data by looking at *cumulative* probabilities, odds, or log-odds. Comparisons of different exposures or individual characteristics typically look at how these cumulative measures vary across the different exposures or characteristics. So, if we were interested in cumulative odds, we would compare\small{\frac{P(Response = 1|Group = A)}{P(Response &gt; 1|Group = A)} \ \ vs. \ \frac{P(Response = 1|Group = B)}{P(Response &gt; 1|Group = B)}},*P*(*Response*>1∣*Group*=*A*)*P*(*Response*=1∣*Group*=*A*)​  *vs*. *P*(*Response*>1∣*Group*=*B*)*P*(*Response*=1∣*Group*=*B*)​,

\small{\frac{P(Response \leq 2|Group = A)}{P(Response &gt; 2|Group = A)} \ \ vs. \ \frac{P(Response \leq 2|Group = B)}{P(Response &gt; 2|Group = B)}},*P*(*Response*>2∣*Group*=*A*)*P*(*Response*≤2∣*Group*=*A*)​  *vs*. *P*(*Response*>2∣*Group*=*B*)*P*(*Response*≤2∣*Group*=*B*)​,

and continue until the last (in this case, fourth) comparison

\small{\frac{P(Response \leq 4|Group = A)}{P(Response = 5|Group = A)} \ \ vs. \ \frac{P(Response \leq 4|Group = B)}{P(Response = 5|Group = B)}}.*P*(*Response*=5∣*Group*=*A*)*P*(*Response*≤4∣*Group*=*A*)​  *vs*. *P*(*Response*=5∣*Group*=*B*)*P*(*Response*≤4∣*Group*=*B*)​.

**Multiple responses, multiple thresholds**

The latent process that was described for the binary outcome is extended to the multinomial outcome by the addition of more thresholds. These thresholds define the portions of the density that define the probability of each possible response. If there are k*k* possible responses (in the meat example, we have 5), then there will be k-1*k*−1 thresholds. The area under the logistic density curve of each of the regions defined by those thresholds (there will be k*k* distinct regions) represents the probability of each possible response tied to that region. In the example here, we define five regions of a logistic density by setting the four thresholds. We can say that this underlying continuous distribution represents the probability distribution of categorical responses for a specific population, which we are calling *Group A*.

# preliminary libraries and plotting defaults

**library**(ggplot2)

**library**(data.table)

my\_theme <- **function**() {

theme(panel.background = element\_rect(fill = "grey90"),

panel.grid = element\_blank(),

axis.ticks = element\_line(colour = "black"),

panel.spacing = unit(0.25, "lines"),

plot.title = element\_text(size = 12, vjust = 0.5, hjust = 0),

panel.border = element\_rect(fill = NA, colour = "gray90"))

}

# create data points density curve

x <- seq(-6, 6, length = 1000)

pdf <- dlogis(x, location = 0, scale = 1)

dt <- data.table(x, pdf)

# set thresholds for Group A

thresholdA <- c(-2.1, -0.3, 1.4, 3.6)

pdf <- dlogis(thresholdA)

grpA <- data.table(threshold = thresholdA, pdf)

aBreaks <- c(-6, grpA$threshold, 6)

# plot density with cutpoints

dt[, grpA := cut(x, breaks = aBreaks, labels = F, include.lowest = TRUE)]

p1 <- ggplot(data = dt, aes(x = x, y = pdf)) +

geom\_line() +

geom\_area(aes(x = x, y = pdf, group = grpA, fill = factor(grpA))) +

geom\_hline(yintercept = 0, color = "grey50") +

annotate("text", x = -5, y = .28, label = "Group A", size = 5) +

scale\_fill\_manual(values = c("#d0d7d1", "#bbc5bc", "#a6b3a7", "#91a192", "#7c8f7d"),

labels = c("None", "1-3/month", "1/week", "2-6/week", "1+/day"),

name = "Frequency") +

scale\_x\_continuous(breaks = thresholdA) +

scale\_y\_continuous(limits = c(0, 0.3), name = "Density") +

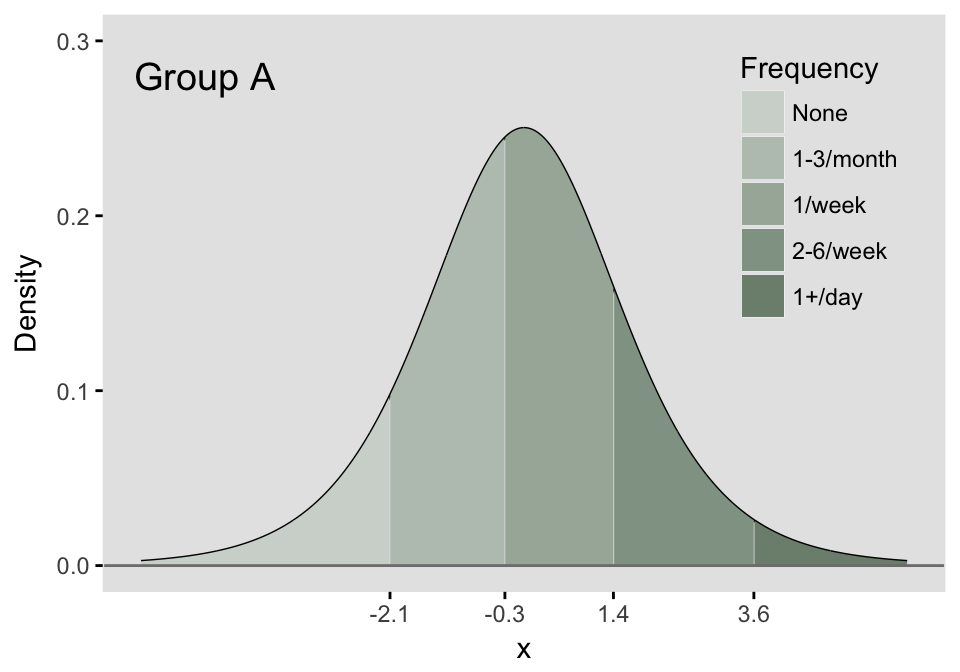
my\_theme() +

theme(legend.position = c(.85, .7),

legend.background = element\_rect(fill = "grey90"),

legend.key = element\_rect(color = "grey90"))

p1



The area for each of the five regions can easily be calculated, where each area represents the probability of each response:

pA= plogis(c(thresholdA, Inf)) - plogis(c(-Inf, thresholdA))

probs <- data.frame(pA)

rownames(probs) <- c("P(Resp = 1)", "P(Resp = 2)",

"P(Resp = 3)", "P(Resp = 4)", "P(Resp = 5)")

probs

## pA

## P(Resp = 1) 0.109

## P(Resp = 2) 0.316

## P(Resp = 3) 0.377

## P(Resp = 4) 0.171

## P(Resp = 5) 0.027

As I’ve already mentioned, when we characterize a multinomial response, we typically do so in terms of cumulative probabilities. I’ve calculated several quantities below, and we can see that the logs of the cumulative odds for this particular group are indeed the threshold values that we used to define the sub-regions.

# cumulative probabilities defined by the threshold

probA <- data.frame(

cprob = plogis(thresholdA),

codds = plogis(thresholdA)/(1-plogis(thresholdA)),

lcodds = log(plogis(thresholdA)/(1-plogis(thresholdA)))

)

rownames(probA) <- c("P(Grp < 2)", "P(Grp < 3)", "P(Grp < 4)", "P(Grp < 5)")

probA

## cprob codds lcodds

## P(Grp < 2) 0.11 0.12 -2.1

## P(Grp < 3) 0.43 0.74 -0.3

## P(Grp < 4) 0.80 4.06 1.4

## P(Grp < 5) 0.97 36.60 3.6

The last column of the table below matches the thresholds defined in vector thresholdA.

thresholdA

## [1] -2.1 -0.3 1.4 3.6

**Comparing response distributions of different populations**

In the cumulative logit model, the underlying assumption is that the odds ratio of one population relative to another is constant across all the possible responses. This means that all of the cumulative odds ratios are equal:

\small{\frac{codds(P(Resp = 1 | A))}{codds(P(Resp = 1 | B))} = \frac{codds(P(Resp \leq 2 | A))}{codds(P(Resp \leq 2 | B))} = \ ... \ = \frac{codds(P(Resp \leq 4 | A))}{codds(P(Resp \leq 4 | B))}}*codds*(*P*(*Resp*=1∣*B*))*codds*(*P*(*Resp*=1∣*A*))​=*codds*(*P*(*Resp*≤2∣*B*))*codds*(*P*(*Resp*≤2∣*A*))​= ... =*codds*(*P*(*Resp*≤4∣*B*))*codds*(*P*(*Resp*≤4∣*A*))​

In terms of the underlying process, this means that each of the thresholds shifts the same amount, as shown below, where we add 1.1 units to each threshold that was set Group A:

# Group B threshold is an additive shift to the right

thresholdB <- thresholdA + 1.1

pdf <- dlogis(thresholdB)

grpB <- data.table(threshold = thresholdB, pdf)

bBreaks <- c(-6, grpB$threshold, 6)

Based on this shift, we can see that the probability distribution for Group B is quite different:

pB = plogis(c(thresholdB, Inf)) - plogis(c(-Inf, thresholdB))

probs <- data.frame(pA, pB)

rownames(probs) <- c("P(Resp = 1)", "P(Resp = 2)",

"P(Resp = 3)", "P(Resp = 4)", "P(Resp = 5)")

probs

## pA pB

## P(Resp = 1) 0.109 0.269

## P(Resp = 2) 0.316 0.421

## P(Resp = 3) 0.377 0.234

## P(Resp = 4) 0.171 0.067

## P(Resp = 5) 0.027 0.009

Plotting Group B along with Group A, we can see visually how that shift affects the sizes of the five regions (I’ve left the thresholds of Group A in the Group B plot so you can see clearly the shift).

# Plot density for group B

dt[, grpB := cut(x, breaks = bBreaks, labels = F, include.lowest = TRUE)]

p2 <- ggplot(data = dt, aes(x = x, y = pdf)) +

geom\_line() +

geom\_area(aes(x = x, y = pdf, group = grpB, fill = factor(grpB))) +

geom\_hline(yintercept = 0, color = "grey5") +

geom\_segment(data=grpA,

aes(x=threshold, xend = threshold, y=0, yend=pdf),

size = 0.3, lty = 2, color = "#857284") +

annotate("text", x = -5, y = .28, label = "Group B", size = 5) +

scale\_fill\_manual(values = c("#d0d7d1", "#bbc5bc", "#a6b3a7", "#91a192", "#7c8f7d"),

labels = c("None", "1-3/month", "1/week", "2-6/week", "1+/day"),

name = "Frequency") +

scale\_x\_continuous(breaks = thresholdB) +

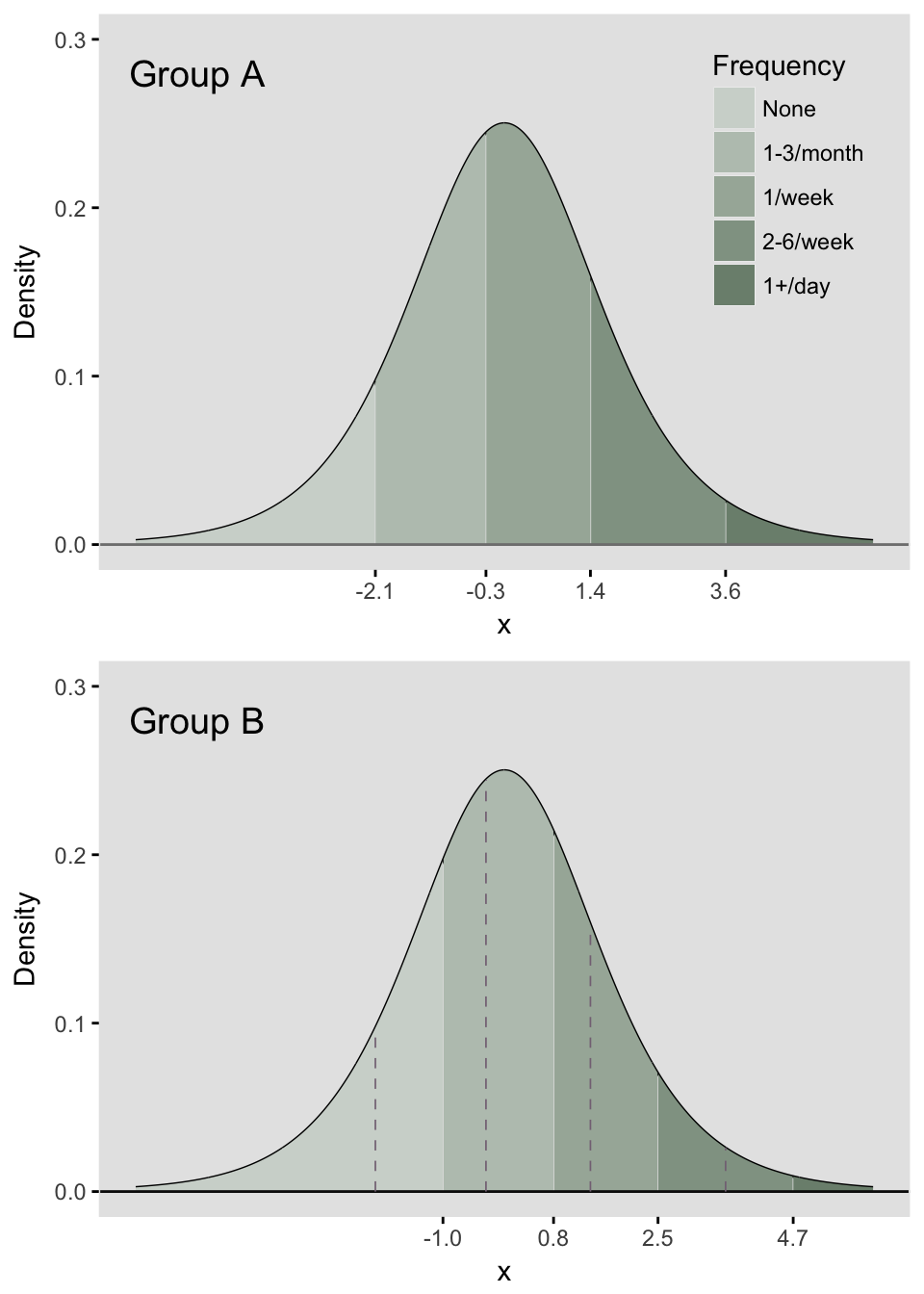
scale\_y\_continuous(limits = c(0.0, 0.3), name = "Density") +

my\_theme() +

theme(legend.position = "none")

**library**(gridExtra)

grid.arrange(p1, p2, nrow = 2 )



When we look at the cumulative odds ratio comparing the odds of Group B to Group A for each response category, we see a constant ratio. And, of course, a constant log odds ratio, which also reflects the size of the shift from Group A to Group B.

# cumulative probabilities defined by the threshold

probB <- data.frame(

cprob = plogis(thresholdB),

codds = plogis(thresholdB)/(1-plogis(thresholdB)),

lcodds = log(plogis(thresholdB)/(1-plogis(thresholdB)))

)

oddsratio <- data.frame(coddsA = probA$codds,

coddsB = probB$codds,

cOR = probB$codds / probA$codds,

logcOR = log(probB$codds / probA$codds)

)

rownames(oddsratio) <- c("P(Grp < 2)", "P(Grp < 3)", "P(Grp < 4)", "P(Grp < 5)")

oddsratio

## coddsA coddsB cOR logcOR

## P(Grp < 2) 0.12 0.37 3 1.1

## P(Grp < 3) 0.74 2.23 3 1.1

## P(Grp < 4) 4.06 12.18 3 1.1

## P(Grp < 5) 36.60 109.95 3 1.1

**The cumulative proportional odds model**

In the R package ordinal, the model is fit using function clm. The model that is being estimated has the form

log \left( \frac{P(Resp \leq i)}{P(Resp &gt; i)} | Group \right) = \alpha\_i - \beta\*I(Group=B) \ \ , \ i \in \{1, 2, 3, 4\}*log*(*P*(*Resp*>*i*)*P*(*Resp*≤*i*)​∣*Group*)=*αi*​−*β*∗*I*(*Group*=*B*)  , *i*∈{1,2,3,4}

The model specifies that the cumulative log-odds for a particular category is a function of two parameters, \alpha\_i*αi*​ and \beta*β*. (Note that in this parameterization and the model fit, -\beta−*β* is used.) \alpha\_i*αi*​ represents the cumulative log odds of being in category i*i* or lower for those in the reference exposure group, which in our example is Group A. *\alpha\_iαi​ also represents the threshold of the latent continuous (logistic) data generating process.* \beta*β* is the cumulative log-odds ratio for the category i*i* comparing Group B to reference Group A. *\betaβ also represents the shift of the threshold on the latent continuous process for Group B relative to Group A*. The proportionality assumption implies that the shift of the threshold for each of the categories is identical. This is what I illustrated above.

**Simulation and model fit**

To show how this process might actually work, I am simulating data from the standardized logistic distribution and applying the thresholds defined above based on the group status. In practice, each individual could have her own set of thresholds, depending on her characteristics (gender, age, etc.). In this case, group membership is the only characteristic I am using, so all individuals in a particular group share the same set of thresholds. (We could even have random effects, where subgroups have random shifts that are subgroup specific. In the addendum, following the main part of the post, I provide code to generate data from a mixed effects model with group level random effects plus fixed effects for exposure, gender, and a continuous outcome.)

set.seed(123)

n = 1000

x.A <- rlogis(n)

acuts <- c(-Inf, thresholdA, Inf)

catA <- cut(x.A, breaks = acuts, label = F)

dtA <- data.table(id = 1:n, grp = "A", cat = catA)

Not surprisingly (since we are using a generous sample size of 1000), the simulated proportions are quite close to the hypothetical proportions established by the thresholds:

cumsum(prop.table(table(catA)))

## 1 2 3 4 5

## 0.11 0.44 0.81 0.97 1.00

probA$cprob

## [1] 0.11 0.43 0.80 0.97

Now we generate a sample from Group B and combine them into a single data set:

x.B <- rlogis(n)

bcuts <- c(-Inf, thresholdA + 1.1, Inf)

catB <- cut(x.B, breaks = bcuts, label = F)

dtB <- data.table(id = (n+1):(2\*n), grp = "B", cat=catB)

dt <- rbind(dtA, dtB)

dt[, cat := factor(cat, labels = c("None", "1-3/month", "1/week", "2-6/week", "1+/day"))]

dt

## id grp cat

## 1: 1 A 1-3/month

## 2: 2 A 1/week

## 3: 3 A 1-3/month

## 4: 4 A 2-6/week

## 5: 5 A 2-6/week

## ---

## 1996: 1996 B 1/week

## 1997: 1997 B 1-3/month

## 1998: 1998 B 1/week

## 1999: 1999 B 1-3/month

## 2000: 2000 B 1-3/month

Finally, we estimate the parameters of the model using function clm and we see that we recover the original parameters quite well.

**library**(ordinal)

clmFit <- clm(cat ~ grp, data = dt)

summary(clmFit)

## formula: cat ~ grp

## data: dt

##

## link threshold nobs logLik AIC niter max.grad cond.H

## logit flexible 2000 -2655.03 5320.05 6(0) 1.19e-11 2.3e+01

##

## Coefficients:

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

## grpB -1.0745 0.0848 -12.7 <2e-16 \*\*\*

## ---

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

##

## Threshold coefficients:

## Estimate Std. Error z value

## None|1-3/month -2.0912 0.0784 -26.68

## 1-3/month|1/week -0.2465 0.0612 -4.02

## 1/week|2-6/week 1.4212 0.0728 19.51

## 2-6/week|1+/day 3.5150 0.1643 21.39

In the model output, the grpB coefficient of -1.07 is the estimate of -\beta−*β*, which was set to 1.1 in the simulation. The threshold coefficients are the estimates of the \alpha\_i*αi*​’s in the model, and we can see the estimates are not too bad by looking at the initial thresholds:

coeffit <- coef(clmFit)[1:4]

names(coeffit) <- c(1:4)

rbind( thresholdA, coeffit)

## 1 2 3 4

## thresholdA -2.1 -0.30 1.4 3.6

## coeffit -2.1 -0.25 1.4 3.5

This was a relatively simple simulation. However it highlights how it would be possible to generate more complex scenarios of multinomial response data to more fully explore other types of models. These more flexible models might be able to handle situations where the possibly restrictive assumptions of this model (particularly the proportional odds assumption) do not hold.

**Addendum 1**

Here is code to generate cluster-randomized data with an ordinal outcome that is a function of treatment assignment, gender, and a continuous status measure at the individual level. There is also a group level random effect. Once the data are generated, I fit a mixed cumulative logit model.

**library**(simstudy)

# define data

defSchool <- defData(varname = "reS", formula = 0,

variance = 0.10, id = "idS")

defSchool <- defData(defSchool, varname = "n",

formula = 250, dist = "noZeroPoisson")

defInd <- defDataAdd(varname = "male", formula = 0.45, dist = "binary")

defInd <- defDataAdd(defInd, varname = "status",

formula = 0, variance = 1, dist = "normal")

defInd <- defDataAdd(defInd,

varname = "z",

formula = "0.8 \* grp + 0.3 \* male - 0.2 \* status + reS",

dist = "nonrandom")

# generate data

dtS <- genData(100, defSchool)

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

dt <- genCluster(dtS, "idS", "n", "id")

dt <- addColumns(defInd, dt)

# set reference probabilities for 4-category outcome

probs <- c(0.35, 0.30, 0.25, 0.10)

cprop <- cumsum(probs)

# map cumulative probs to thresholds for reference group

gamma.c <- qlogis(cprop)

matlp <- matrix(rep(gamma.c, nrow(dt)),

ncol = length(cprop),

byrow = TRUE

)

head(matlp)

## [,1] [,2] [,3] [,4]

## [1,] -0.62 0.62 2.2 Inf

## [2,] -0.62 0.62 2.2 Inf

## [3,] -0.62 0.62 2.2 Inf

## [4,] -0.62 0.62 2.2 Inf

## [5,] -0.62 0.62 2.2 Inf

## [6,] -0.62 0.62 2.2 Inf

# set individual thresholds based on covariates,

# which is an additive shift from the reference group

# based on z

matlpInd <- matlp - dt[, z]

head(matlpInd)

## [,1] [,2] [,3] [,4]

## [1,] -1.52 -0.28 1.3 Inf

## [2,] -1.58 -0.34 1.2 Inf

## [3,] -0.95 0.29 1.9 Inf

## [4,] -1.53 -0.29 1.3 Inf

## [5,] -1.49 -0.25 1.3 Inf

## [6,] -1.13 0.11 1.7 Inf

# convert log odds to cumulative probabability

matcump <- 1 / (1 + exp(-matlpInd))

matcump <- cbind(0, matcump)

head(matcump)

## [,1] [,2] [,3] [,4] [,5]

## [1,] 0 0.18 0.43 0.78 1

## [2,] 0 0.17 0.42 0.78 1

## [3,] 0 0.28 0.57 0.87 1

## [4,] 0 0.18 0.43 0.78 1

## [5,] 0 0.18 0.44 0.79 1

## [6,] 0 0.24 0.53 0.84 1

# convert cumulative probs to category probs:

# originally, I used a loop to do this, but

# thought it would be better to vectorize.

# see 2nd addendum for time comparison - not

# much difference

p <- t(t(matcump)[-1,] - t(matcump)[-5,])

# show some indvidual level probabilities

head(p)

## [,1] [,2] [,3] [,4]

## [1,] 0.18 0.25 0.36 0.22

## [2,] 0.17 0.24 0.36 0.22

## [3,] 0.28 0.29 0.29 0.13

## [4,] 0.18 0.25 0.36 0.22

## [5,] 0.18 0.25 0.35 0.21

## [6,] 0.24 0.28 0.32 0.16

apply(head(p), 1, sum)

## [1] 1 1 1 1 1 1

# generate indvidual level category outcomes based on p

cat <- simstudy:::matMultinom(p)

catF <- ordered(cat)

dt[, cat := catF]

When we fit the mixed effects model, it is not surprising that we recover the parameters used to generate the data, which were based on the model. The fixed effects were specified as “0.8 \* grp + 0.3 \* male - 0.2 \* status”, the variance of the random group effect was 0.10, and the latent thresholds based on the category probabilities were {-0.62, 0.62, 2.20}:

fmm <- clmm(cat ~ grp + male + status + (1|idS), data=dt)

summary(fmm)

## Cumulative Link Mixed Model fitted with the Laplace approximation

##

## formula: cat ~ grp + male + status + (1 | idS)

## data: dt

##

## link threshold nobs logLik AIC niter max.grad cond.H

## logit flexible 24990 -33096.42 66206.85 705(2118) 2.37e-02 1.3e+02

##

## Random effects:

## Groups Name Variance Std.Dev.

## idS (Intercept) 0.109 0.331

## Number of groups: idS 100

##

## Coefficients:

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

## grp 0.8117 0.0702 11.6 <2e-16 \*\*\*

## male 0.3163 0.0232 13.7 <2e-16 \*\*\*

## status -0.1959 0.0116 -16.9 <2e-16 \*\*\*

## ---

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

##

## Threshold coefficients:

## Estimate Std. Error z value

## 1|2 -0.6478 0.0511 -12.7

## 2|3 0.6135 0.0511 12.0

## 3|4 2.1789 0.0529 41.2

**Addendum 2 - vector vs loop**

In case any one is obsessed with vectorization in R, here is a comparison of two different functions that convert cumulative probabilities into probabilities. One method uses a loop, the other uses matrix operations. In this case, it actually appears that my non-loop approach is slower - maybe there is a faster way? Maybe not, since the loop is actually quite short - determined by the number of possible responses in the categorical measure…

**library**(microbenchmark)

loopdif <- **function**(mat) {

ncols <- ncol(mat)

p <- matrix(0, nrow = nrow(mat), ncol = ( ncols - 1 ))

**for** (i **in** 1 : ( ncol(mat) - 1 )) {

p[,i] <- mat[, i+1] - mat[, i]

}

**return**(p)

}

vecdif <- **function**(mat) {

ncols <- ncol(mat)

p <- t(t(mat)[-1,] - t(mat)[-ncols,])

**return**(p)

}

head(loopdif(matcump))

## [,1] [,2] [,3] [,4]

## [1,] 0.18 0.25 0.36 0.22

## [2,] 0.17 0.24 0.36 0.22

## [3,] 0.28 0.29 0.29 0.13

## [4,] 0.18 0.25 0.36 0.22

## [5,] 0.18 0.25 0.35 0.21

## [6,] 0.24 0.28 0.32 0.16

head(vecdif(matcump))

## [,1] [,2] [,3] [,4]

## [1,] 0.18 0.25 0.36 0.22

## [2,] 0.17 0.24 0.36 0.22

## [3,] 0.28 0.29 0.29 0.13

## [4,] 0.18 0.25 0.36 0.22

## [5,] 0.18 0.25 0.35 0.21

## [6,] 0.24 0.28 0.32 0.16

microbenchmark(loopdif(matcump), vecdif(matcump),

times = 1000L, unit = "ms")

## Unit: milliseconds

## expr min lq mean median uq max neval

## loopdif(matcump) 0.96 1.4 1.9 1.7 1.9 112 1000

## vecdif(matcump) 0.92 1.7 3.1 2.3 2.7 115 1000

**Implementing the copula algorithm in R**

While this hasn’t been implemented just yet in simstudy, this is along the lines of what I am thinking:

**library**(simstudy)

**library**(data.table)

set.seed(555)

# Generate 1000 observations of 4 RVs from a multivariate normal

# dist - each N(0,1) - with a correlation matrix where rho = 0.4

dt <- genCorData(1000, mu = c(0, 0, 0, 0), sigma = 1,

rho = 0.4, corstr = "cs" )

dt

## id V1 V2 V3 V4

## 1: 1 -1.1667574 -0.05296536 0.2995360 -0.5232691

## 2: 2 0.4505618 0.57499589 -0.9629426 1.5495697

## 3: 3 -0.1294505 1.68372035 1.1309223 0.4205397

## 4: 4 0.0858846 1.27479473 0.4247491 0.1054230

## 5: 5 0.4654873 3.05566796 0.5846449 1.0906072

## ---

## 996: 996 0.3420099 -0.35783480 -0.8363306 0.2656964

## 997: 997 -1.0928169 0.50081091 -0.8915582 -0.7428976

## 998: 998 0.7490765 -0.09559294 -0.2351121 0.6632157

## 999: 999 0.8143565 -1.00978384 0.2266132 -1.2345192

## 1000: 1000 -1.9795559 -0.16668454 -0.5883966 -1.7424941

round(cor(dt[,-1]), 2)

## V1 V2 V3 V4

## V1 1.00 0.41 0.36 0.44

## V2 0.41 1.00 0.33 0.42

## V3 0.36 0.33 1.00 0.35

## V4 0.44 0.42 0.35 1.00

### create a long version of the data set

dtM <- melt(dt, id.vars = "id", variable.factor = TRUE,

value.name = "X", variable.name = "seq")

setkey(dtM, "id") # sort data by id

dtM[, seqid := .I] # add index for each record

### apply CDF to X to get uniform distribution

dtM[, U := pnorm(X)]

### Generate correlated Poisson data with mean and variance 8

### apply inverse CDF to U

dtM[, Y\_pois := qpois(U, 8), keyby = seqid]

dtM

## id seq X seqid U Y\_pois

## 1: 1 V1 -1.16675744 1 0.12165417 5

## 2: 1 V2 -0.05296536 2 0.47887975 8

## 3: 1 V3 0.29953603 3 0.61773446 9

## 4: 1 V4 -0.52326909 4 0.30039350 6

## 5: 2 V1 0.45056179 5 0.67384729 9

## ---

## 3996: 999 V4 -1.23451924 3996 0.10850474 5

## 3997: 1000 V1 -1.97955591 3997 0.02387673 3

## 3998: 1000 V2 -0.16668454 3998 0.43380913 7

## 3999: 1000 V3 -0.58839655 3999 0.27813308 6

## 4000: 1000 V4 -1.74249414 4000 0.04071101 3

### Check mean and variance of Y\_pois

dtM[, .(mean = round(mean(Y\_pois), 1),

var = round(var(Y\_pois), 1)), keyby = seq]

## seq mean var

## 1: V1 8.0 8.2

## 2: V2 8.1 8.5

## 3: V3 8.1 7.6

## 4: V4 8.0 7.9

### Check correlation matrix of Y\_pois's - I know this code is a bit ugly

### but I just wanted to get the correlation matrix quickly.

round(cor(as.matrix(dcast(data = dtM, id~seq,

value.var = "Y\_pois")[,-1])), 2)

## V1 V2 V3 V4

## V1 1.00 0.40 0.37 0.43

## V2 0.40 1.00 0.33 0.40

## V3 0.37 0.33 1.00 0.35

## V4 0.43 0.40 0.35 1.00

The correlation matrices for \mathbf{X}**X** and \mathbf{Y\_{Pois}}**YPois**​ aren’t too far off.

Here are the results for an auto-regressive (AR-1) correlation structure. (I am omitting some of the code for brevity’s sake):

# Generate 1000 observations of 4 RVs from a multivariate normal

# dist - each N(0,1) - with a correlation matrix where rho = 0.4

dt <- genCorData(1000, mu = c(0, 0, 0, 0), sigma = 1,

rho = 0.4, corstr = "ar1" )

round(cor(dt[,-1]), 2)

## V1 V2 V3 V4

## V1 1.00 0.43 0.18 0.12

## V2 0.43 1.00 0.39 0.13

## V3 0.18 0.39 1.00 0.38

## V4 0.12 0.13 0.38 1.00

### Check mean and variance of Y\_pois

dtM[, .(mean = round(mean(Y\_pois), 1),

var = round(var(Y\_pois), 1)), keyby = seq]

## seq mean var

## 1: V1 8.1 8.3

## 2: V2 7.9 7.8

## 3: V3 8.0 8.4

## 4: V4 8.0 7.5

### Check correlation matrix of Y\_pois's

round(cor(as.matrix(dcast(data = dtM, id~seq,

value.var = "Y\_pois")[,-1])), 2)

## V1 V2 V3 V4

## V1 1.00 0.41 0.18 0.13

## V2 0.41 1.00 0.39 0.14

## V3 0.18 0.39 1.00 0.36

## V4 0.13 0.14 0.36 1.00

Again - comparing the two correlation matrices - the original normal data, and the derivative Poisson data - suggests that this can work pretty well.

Using the last data set, I fit a GEE model to see how well the data generating process is recovered:

**library**(geepack)

geefit <- geepack::geeglm(Y\_pois ~ 1, data = dtM, family = poisson,

id = id, corstr = "ar1")

summary(geefit)

##

## Call:

## geepack::geeglm(formula = Y\_pois ~ 1, family = poisson, data = dtM,

## id = id, corstr = "ar1")

##

## Coefficients:

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

## (Intercept) 2.080597 0.007447 78060 <2e-16 \*\*\*

## ---

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

##

## Estimated Scale Parameters:

## Estimate Std.err

## (Intercept) 0.9984 0.02679

##

## Correlation: Structure = ar1 Link = identity

##

## Estimated Correlation Parameters:

## Estimate Std.err

## alpha 0.3987 0.02008

## Number of clusters: 1000 Maximum cluster size: 4

In the GEE output, alpha is an estimate of \rho*ρ*. The estimated alpha is 0.399, quite close to 0.40, the original value used to generate the normally distributed data.

**Binary outcomes**

We can also generate binary data:

### Generate binary data with p=0.5 (var = 0.25)

dtM[, Y\_bin := qbinom(U, 1, .5), keyby = seqid]

dtM

## id seq X seqid U Y\_pois Y\_bin

## 1: 1 V1 1.7425 1 0.959288 13 1

## 2: 1 V2 1.4915 2 0.932086 12 1

## 3: 1 V3 0.7379 3 0.769722 10 1

## 4: 1 V4 -1.6581 4 0.048644 4 0

## 5: 2 V1 2.3262 5 0.989997 15 1

## ---

## 3996: 999 V4 -0.3805 3996 0.351772 7 0

## 3997: 1000 V1 -0.8724 3997 0.191505 6 0

## 3998: 1000 V2 -1.0085 3998 0.156600 5 0

## 3999: 1000 V3 -2.0451 3999 0.020420 3 0

## 4000: 1000 V4 -2.7668 4000 0.002831 1 0

### Check mean and variance of Y\_bin

dtM[, .(mean = round(mean(Y\_bin), 2),

var = round(var(Y\_bin), 2)), keyby = seq]

## seq mean var

## 1: V1 0.52 0.25

## 2: V2 0.50 0.25

## 3: V3 0.48 0.25

## 4: V4 0.49 0.25

### Check correlation matrix of Y\_bin's

round(cor(as.matrix(dcast(data = dtM, id~seq,

value.var = "Y\_bin")[,-1])), 2)

## V1 V2 V3 V4

## V1 1.00 0.29 0.10 0.05

## V2 0.29 1.00 0.27 0.03

## V3 0.10 0.27 1.00 0.23

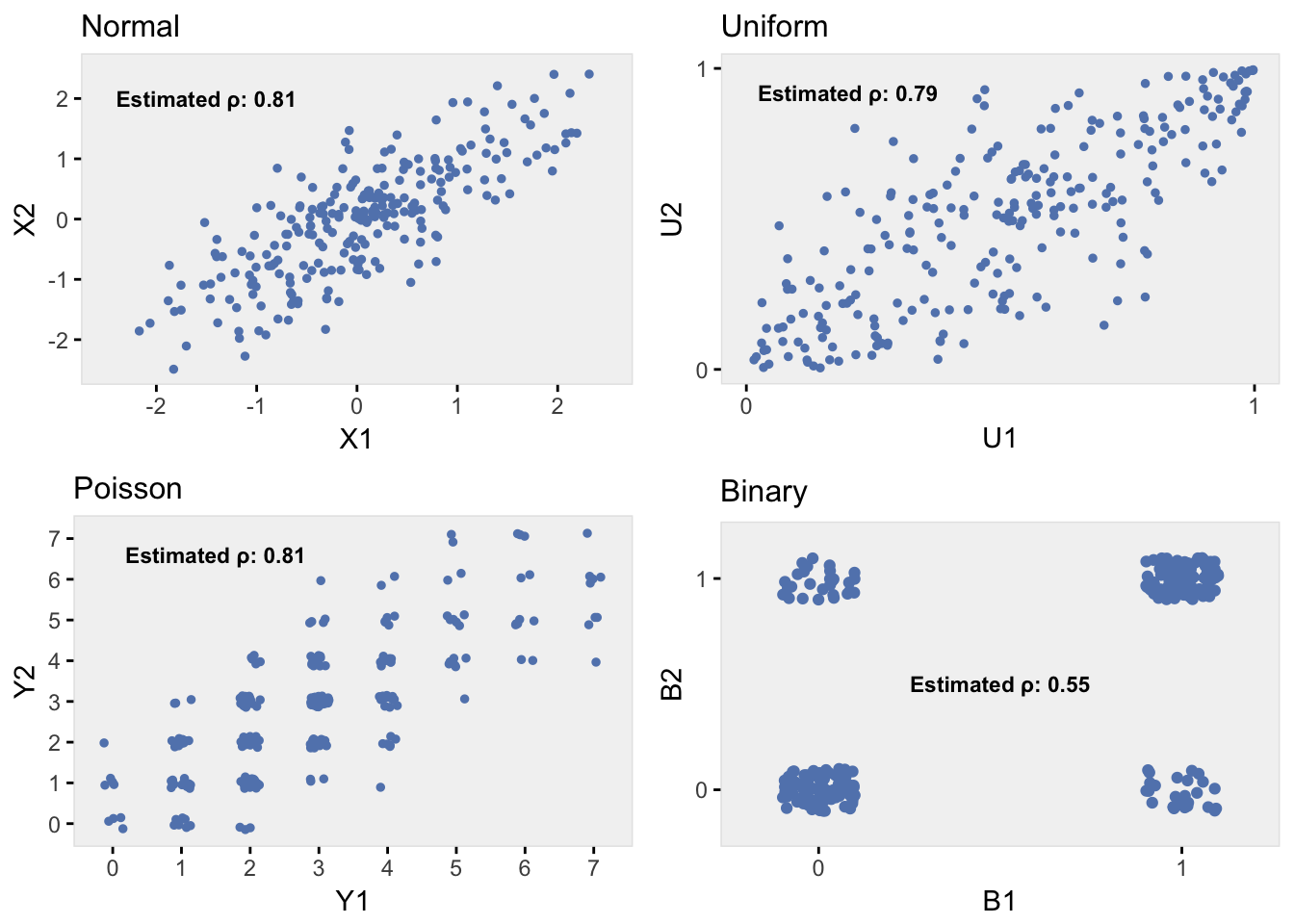
## V4 0.05 0.03 0.23 1.00

The binary data are correlated, but the correlation coefficient doesn’t replicate as well as the Poisson distribution. While both the Poisson and binary CDF’s are discontinuous, the extreme jump in the binary CDF leads to this discrepancy. Values that are relatively close to each other on the normal scale, and in particular on the uniform scale, can be ‘sent’ to opposite ends of the binary scale (that is to 0 and to 1) if they straddle the cutoff point p*p* (the probability of the outcome in the binary distribution); values similar in the original data are very different in the target data. This bias is partially attenuated by values far apart on the uniform scale yet falling on the same side of p*p* (both driven to 0 or both to 1); in this case values different in the original data are similar (actually identical) in the target data.

The series of plots below show bivariate data for the original multivariate normal data, and the corresponding uniform, Poisson, and binary data. We can see the effect of extreme discontinuity of the binary data.

R Code

|  |
| --- |
| set.seed(58) |
|  |  |
|  | dt <- genCorData(250, mu = c(0, 0), sigma = 1, |
|  | rho = 0.8, corstr = "cs", |
|  | cnames = c("X1","X2")) |
|  |  |
|  | dt[, U1 := pnorm(X1)] |
|  | dt[, U2 := pnorm(X2)] |
|  |  |
|  | dt[, Y1 := qpois(U1, 3)] |
|  | dt[, Y2 := qpois(U2, 3)] |
|  |  |
|  | dt[, B1 := qbinom(U1, 1, .5)] |
|  | dt[, B2 := qbinom(U2, 1, .5)] |
|  |  |
|  | dtCor <- dt[, .(X = cor(X1, X2), U = cor(U1, U2), |
|  | Y = cor(Y1, Y2), B = cor(B1, B2))] |
|  |  |
|  | dtS <- dtCor[,.(X = paste("Est. corr:", sprintf("%1.2f",X)), |
|  | U = paste("Est. corr:", sprintf("%1.2f",U)), |
|  | Y = paste("Est. corr:", sprintf("%1.2f",Y)), |
|  | B = paste("Est. corr:", sprintf("%1.2f",B)))] |
|  |  |
|  | p1 <- ggplot(data = dt, aes(x=X1, y=X2)) + |
|  | geom\_point(color="#6285BA", size = 1) + |
|  | theme\_ksg("grey95") + |
|  | scale\_x\_continuous(limits = c(-2.5,2.5)) + |
|  | scale\_y\_continuous(limits = c(-2.5,2.5)) + |
|  | ggtitle("Normal") + |
|  | annotate(geom = "text", label = dtS$X, x = -2, y = 2, fontface = 2) |
|  |  |
|  | p2 <- ggplot(data = dt, aes(x=U1, y=U2)) + |
|  | geom\_point(color="#6285BA", size = 1) + |
|  | theme\_ksg("grey95") + |
|  | scale\_x\_continuous(limits = c(0,1), breaks = c(0,1)) + |
|  | scale\_y\_continuous(limits = c(0,1), breaks = c(0,1)) + |
|  | theme(legend.position = "none") + |
|  | ggtitle("Uniform") + |
|  | annotate(geom = "text", label = dtS$U, x = .10, y = 0.92, fontface = 2) |
|  |  |
|  |  |
|  | p3 <- ggplot(data = dt, aes(x=Y1, y=Y2)) + |
|  | geom\_jitter(color="#6285BA", height = .15, width = .15, size = 1) + |
|  | theme\_ksg("grey95") + |
|  | scale\_x\_continuous(limits = c(-.2, 7.2), breaks = c(0:7)) + |
|  | scale\_y\_continuous(limits = c(-.2, 7.2), breaks = c(0:7)) + |
|  | theme(legend.position = "none") + |
|  | ggtitle("Poisson") + |
|  | annotate(geom = "text", label = dtS$Y, x = 0.6, y = 6.6, fontface = 2) |
|  |  |
|  | p4 <- ggplot(data = dt, aes(x=B1, y=B2), size = 1) + |
|  | geom\_jitter(color="#6285BA", height = .1, width = .1) + |
|  | theme\_ksg("grey95") + |
|  | scale\_x\_continuous(limits = c(-.2, 1.2), breaks = c(0:1)) + |
|  | scale\_y\_continuous(limits = c(-.2, 1.2), breaks = c(0:1)) + |
|  | theme(legend.position = "none") + |
|  | ggtitle("Binary")+ |
|  | annotate(geom = "text", label = dtS$B, x = .5, y = 0.5, fontface = 2) |
|  |  |
|  | gridExtra::grid.arrange(p1, p2, p3, p4, nrow = 2) |

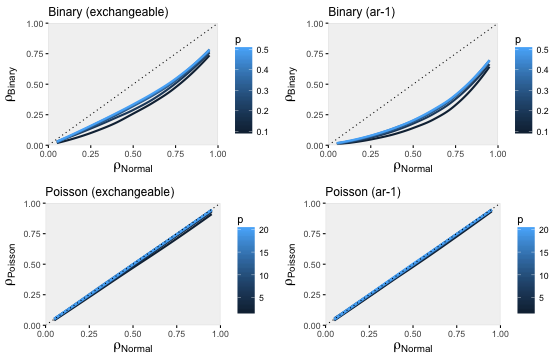


**Some simulation results**

A series of simulations shows how well the estimates of \rho*ρ* compare across a set of different assumptions. In each of the plots below, we see how \rho*ρ* for the non-normal data changes as a function of \rho*ρ* from the original normally distributed data. For each value of \rho*ρ*, I varied the parameter of the non-normal distribution (in the case of the binary data, I varied the probability of the outcome; in the case of the Poisson data, I varied the parameter \lambda*λ* which defines the mean and variance). I also considered both covariance structures, exchangeable and ar-1.

R Code

|  |
| --- |
|  |
| library(geepack) |
|  | genFromU <- function(U, dist, param) { |
|  |  |
|  | if (dist == "binary") colVal <- qbinom(U, 1, param[1]) |
|  | else if (dist == "poisson") colVal <- qpois(U, param[1]) |
|  |  |
|  | return(colVal) |
|  |  |
|  | } |
|  |  |
|  | # need to make a little more flexible - to include covariates in mean model, and link |
|  |  |
|  | genCopulaCor <- function(n, nvars, params, dist, rho, corstr) { |
|  |  |
|  | mu <- rep(0, nvars) |
|  | dt <- genCorData(n, mu, sigma = 1, rho = rho, corstr = corstr ) |
|  | dtM <- melt(dt, id.vars = "id", variable.factor = TRUE, value.name = "Y", variable.name = "seq") |
|  | dtM[, period := as.integer(seq) - 1] |
|  | setkey(dtM, "id") |
|  | dtM[, seqid := .I] |
|  | dtM[, param1 := params[seq], keyby = seqid] |
|  | dtM[, U := pnorm(Y)] |
|  | dtM[, X := genFromU(U, dist, param1), keyby = seqid] |
|  |  |
|  | return(dtM) |
|  |  |
|  | } |
|  |  |
|  | #### binary outcomes |
|  |  |
|  | dist <- "binary" |
|  | # corStr <- "cs" |
|  | corStr <- "ar1" |
|  | results <- data.table() |
|  | n = 5 |
|  |  |
|  | for (i in seq(0.05, 0.95, by = 0.01)) { |
|  | for (p in seq(0.1, 0.5, 0.1)) { |
|  |  |
|  | params <- rep(p, n) |
|  |  |
|  | dtBin <- genCopulaCor(n=1500, nvars = n, params = params, dist = dist, rho = i, corstr = corStr) |
|  | geefit <- geepack::geeglm(X ~ period, data = dtBin, id = id, corstr = "exchangeable") |
|  |  |
|  | results <- rbind(results, data.table(rho = i, p, brho = geefit$geese$alpha)) |
|  | } |
|  | } |
|  |  |
|  | results1 <- copy(results) |
|  | results2 <- copy(results) |
|  |  |
|  | p2 <- ggplot(data = results2, aes(x = rho, y = brho, group = p)) + |
|  | geom\_smooth(se = FALSE, aes(color=p)) + |
|  | geom\_abline(slope = 1, intercept = 0, lty = 3) + |
|  | theme\_ksg("grey95") + |
|  | scale\_x\_continuous(expand = c(0,0), limits = c(0,1)) + |
|  | scale\_y\_continuous(expand = c(0,0), limits = c(0,1)) + |
|  | xlab(expression(rho[Normal])) + |
|  | ylab(expression(rho[Binary])) + |
|  | theme(axis.title = element\_text(size = 14)) + |
|  | ggtitle("Binary (ar-1)") |
|  |  |
|  | png("static/img/post-copula/binaryAR1.png", height = 360, width = 400) |
|  | p1 |
|  | dev.off() |
|  |  |
|  | dist <- "poisson" |
|  | corStr = "cs" |
|  | corStr = "ar1" |
|  | results <- data.table() |
|  | n = 5 |
|  |  |
|  | for (i in seq(0.05, 0.95, by = 0.01)) { |
|  | for (p in seq(2, 20, by = 2)) { |
|  |  |
|  | params <- rep(p, n) |
|  |  |
|  | dtPois <- genCopulaCor(n=1500, nvars = n, params = params, dist = dist, rho = i, corstr = corStr) |
|  | geefit <- geepack::geeglm(X ~ period, data = dtPois, id = id, corstr = corStr) |
|  |  |
|  | results <- rbind(results, data.table(rho = i, p, brho = geefit$geese$alpha)) |
|  | } |
|  | } |
|  |  |
|  | results3 <- copy(results) |
|  | results4 <- copy(results) |
|  |  |
|  | p4 <- ggplot(data = results4[n !=1], aes(x = rho, y = brho, group = p)) + |
|  | geom\_smooth(se = FALSE, aes(color=p)) + |
|  | geom\_abline(slope = 1, intercept = 0, lty = 3) + |
|  | theme\_ksg("grey95") + |
|  | scale\_x\_continuous(expand = c(0,0), limits = c(0,1)) + |
|  | scale\_y\_continuous(expand = c(0,0), limits = c(0,1)) + |
|  | xlab(expression(rho[Normal])) + |
|  | ylab(expression(rho[Poisson])) + |
|  | theme(axis.title = element\_text(size = 14)) + |
|  | ggtitle("Poisson (ar-1)") |
|  |  |
|  | png("static/img/post-copula/dists.png", height = 360, width = 560) |
|  | gridExtra::grid.arrange(p1, p2, p3, p4, nrow = 2) |
|  | dev.off() |
|  |  |



These simulations confirm what we saw earlier. The Poisson data generating process recovers the original \rho*ρ* under both covariance structures reasonably well. The binary data generating process is less successful, with the exchangeable structure doing slightly better than then auto-regressive structure.

# A simstudy update provides an excuse to generate and display Likert-type data

### Defining the data

The proportional odds model assumes a baseline distribution of probabilities. In the case of a survey item, this baseline is the probability of responding at a particular level - in this example I assume a range of 1 (strongly disagree) to 4 (strongly agree) - given a value of zero for all of the covariates. In this example, there is a single predictor x*x* that ranges from -0.5 to 0.5. The baseline probabilities of the response variable r*r* will apply in cases where x = 0*x*=0. In the proportional odds data generating process, the covariates “influence” the response through an additive shift (either positive or negative) on the logistic scale. Here, this additive shift is represented by the variable z*z*, which is a function of x*x*.

**library**(simstudy)

baseprobs<-c(0.40, 0.25, 0.15, 0.20)

def <- defData(varname="x", formula="-0.5;0.5", dist = "uniform")

def <- defData(def, varname = "z", formula = "2\*x", dist = "nonrandom")

### Generate data

The ordinal data is generated after a data set has been created with an adjustment variable. We have to provide the data.table name, the name of the adjustment variable, and the base probabilities. That’s really it.

set.seed(2017)

dx <- genData(2500, def)

dx <- genOrdCat(dx, adjVar = "z", baseprobs, catVar = "r")

dx <- genFactor(dx, "r", c("Strongly disagree", "Disagree",

"Agree", "Strongly agree"))

print(dx)

## id x z r fr

## 1: 1 0.42424261 0.84848522 2 Disagree

## 2: 2 0.03717641 0.07435283 3 Agree

## 3: 3 -0.03080435 -0.06160871 3 Agree

## 4: 4 -0.21137382 -0.42274765 1 Strongly disagree

## 5: 5 0.27008816 0.54017632 1 Strongly disagree

## ---

## 2496: 2496 -0.32250407 -0.64500815 4 Strongly agree

## 2497: 2497 -0.10268875 -0.20537751 2 Disagree

## 2498: 2498 -0.17037112 -0.34074223 2 Disagree

## 2499: 2499 0.14778233 0.29556465 2 Disagree

## 2500: 2500 0.10665252 0.21330504 3 Agree

The expected cumulative log odds when x=0*x*=0 can be calculated from the base probabilities:

dp <- data.table(baseprobs,

cumProb = cumsum(baseprobs),

cumOdds = cumsum(baseprobs)/(1 - cumsum(baseprobs))

)

dp[, cumLogOdds := log(cumOdds)]

dp

## baseprobs cumProb cumOdds cumLogOdds

## 1: 0.40 0.40 0.6666667 -0.4054651

## 2: 0.25 0.65 1.8571429 0.6190392

## 3: 0.15 0.80 4.0000000 1.3862944

## 4: 0.20 1.00 Inf Inf

If we fit a cumulative odds model (using package ordinal), we recover those cumulative log odds (see the output under the section labeled “Threshold coefficients”). Also, we get an estimate for the coefficient of x*x* (where the true value used to generate the data was 2.00):

**library**(ordinal)

model.fit <- clm(fr ~ x, data = dx, link = "logit")

summary(model.fit)

## formula: fr ~ x

## data: dx

##

## link threshold nobs logLik AIC niter max.grad cond.H

## logit flexible 2500 -3185.75 6379.51 5(0) 3.19e-11 3.3e+01

##

## Coefficients:

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

## x 2.096 0.134 15.64 <2e-16 \*\*\*

## ---

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

##

## Threshold coefficients:

## Estimate Std. Error z value

## Strongly disagree|Disagree -0.46572 0.04243 -10.98

## Disagree|Agree 0.60374 0.04312 14.00

## Agree|Strongly agree 1.38954 0.05049 27.52

### Looking at the data

Below is a plot of the response as a function of the predictor x*x*. I “jitter” the data prior to plotting; otherwise, individual responses would overlap and obscure each other.

**library**(ggplot2)

dx[, rjitter := jitter(as.numeric(r), factor = 0.5)]

ggplot(data = dx, aes(x = x, y = rjitter)) +

geom\_point(color = "forestgreen", size = 0.5) +

scale\_y\_continuous(breaks = c(1:4),

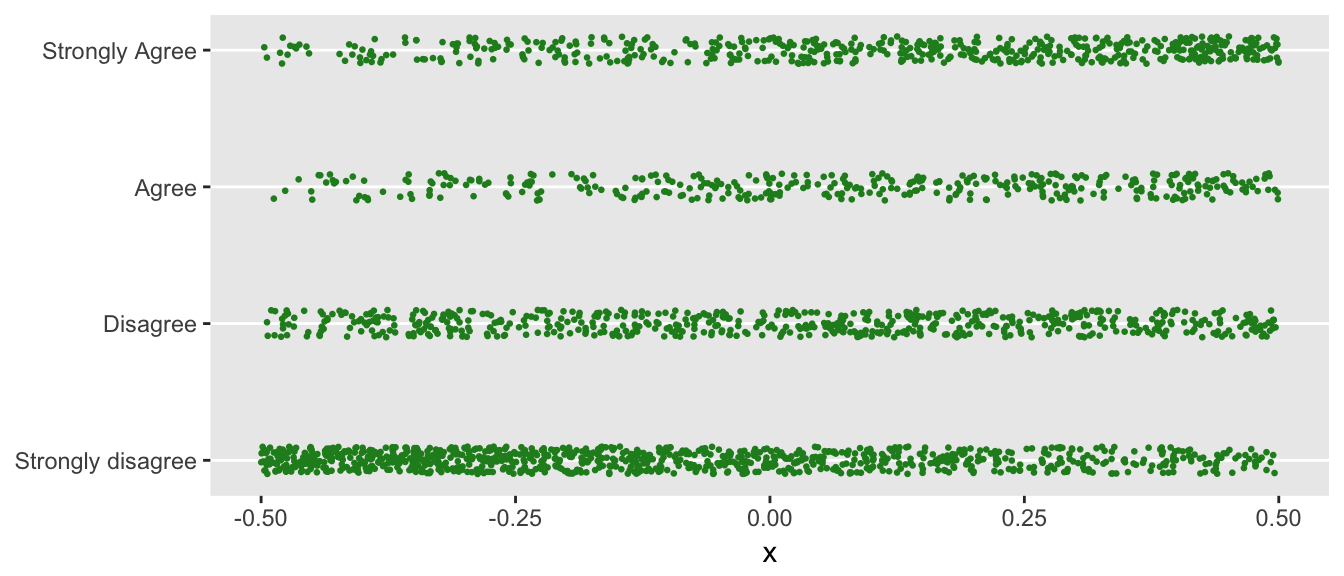
labels = c("Strongly disagree", "Disagree",

"Agree", "Strongly Agree")) +

theme(panel.grid.minor = element\_blank(),

panel.grid.major.x = element\_blank(),

axis.title.y = element\_blank())



You can see that when x*x* is smaller (closer to -0.5), a response of “Strongly disagree” is more likely. Conversely, when x*x* is closer to +0.5, the proportion of folks responding with “Strongly agree” increases.

If we “bin” the individual responses by ranges of x*x*, say grouping by tenths, -0.5 to -0.4, -0.4 to -0.3, all the way to 0.4 to 0.5, we can get another view of how the probabilities shift with respect to x*x*.

The likert package requires very little data manipulation, and once the data are set, it is easy to look at the data in a number of different ways, a couple of which I plot here.

**library**(likert)

bins <- cut(dx$x, breaks = seq(-.5, .5, .1), include.lowest = TRUE)

dx[ , xbin := bins]

item <- data.frame(dx[, fr])

names(item) <- "r"

bin.grp <- factor(dx[, xbin])

likert.bin <- likert(item, grouping = bin.grp)

likert.bin

## Group Item Strongly disagree Disagree Agree Strongly agree

## 1 [-0.5,-0.4] r 65.63877 18.50220 7.048458 8.810573

## 2 (-0.4,-0.3] r 53.33333 27.40741 8.888889 10.370370

## 3 (-0.3,-0.2] r 52.84553 19.51220 10.975610 16.666667

## 4 (-0.2,-0.1] r 48.00000 22.80000 12.800000 16.400000

## 5 (-0.1,0] r 40.24390 24.39024 17.886179 17.479675

## 6 (0,0.1] r 35.20599 25.46816 15.355805 23.970037

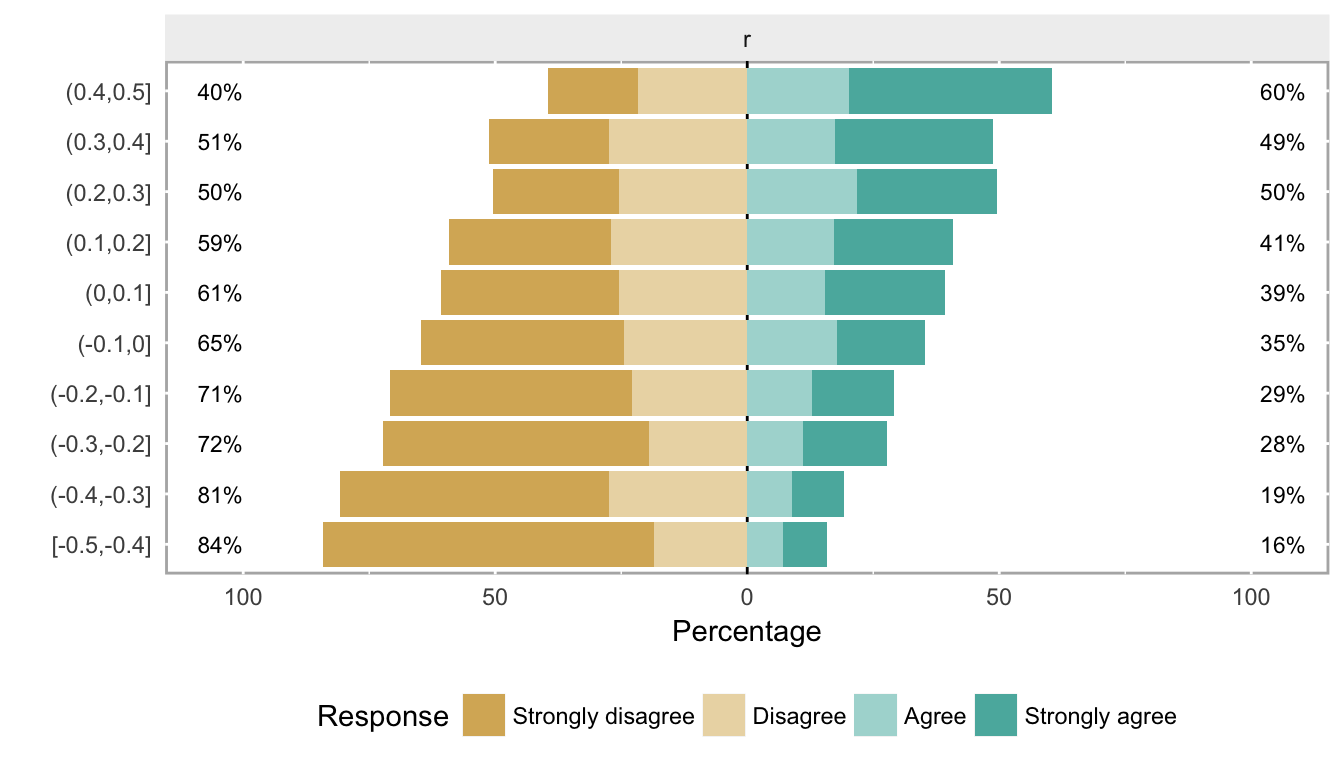
## 7 (0.1,0.2] r 32.06107 27.09924 17.175573 23.664122

## 8 (0.2,0.3] r 25.00000 25.40984 21.721311 27.868852

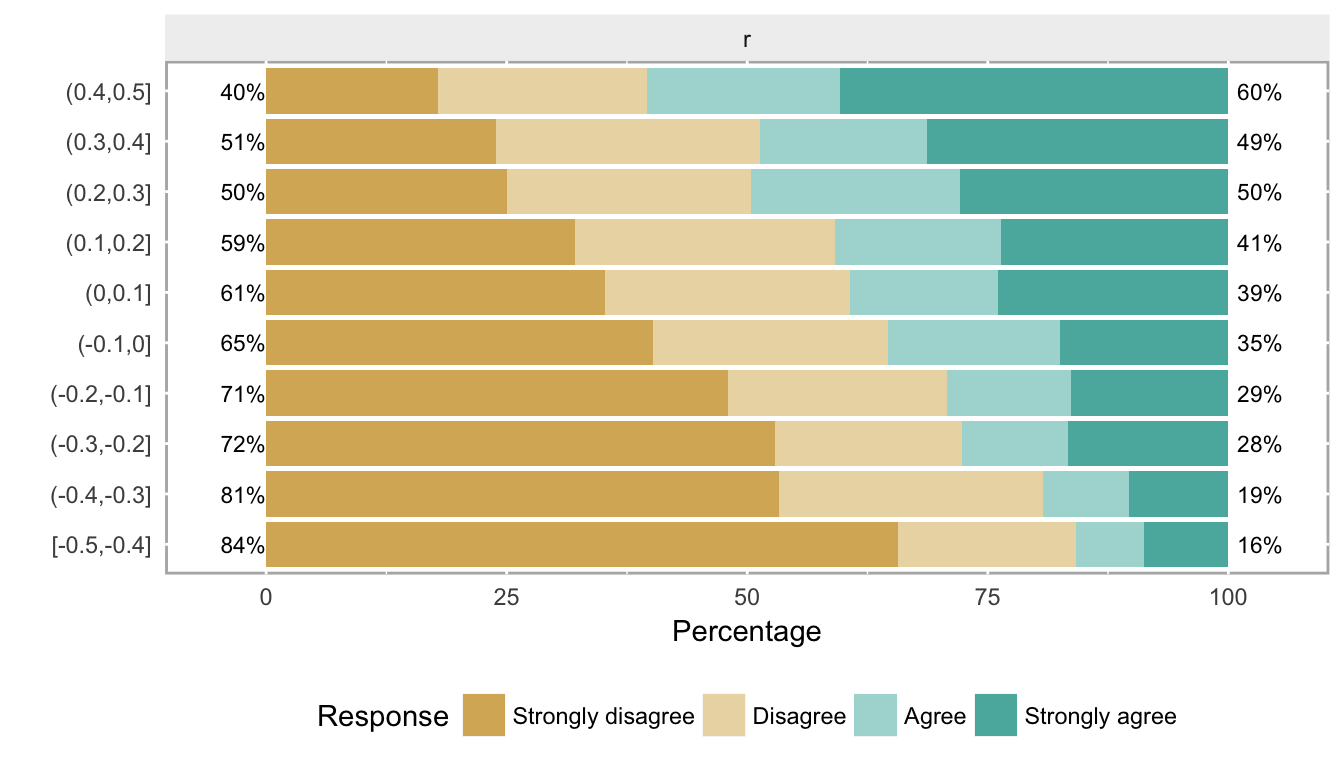
## 9 (0.3,0.4] r 23.91304 27.39130 17.391304 31.304348

## 10 (0.4,0.5] r 17.82946 21.70543 20.155039 40.310078

plot(likert.bin)



plot(likert.bin, centered = FALSE)



These plots show what data look like when the cumulative log odds are proportional as we move across different levels of a covariate. (Note that the two center groups should be closest to the baseline probabilities that were used to generate the data.) If you have real data, obviously it is useful to look at it first to see if this type of pattern emerges from the data. When we have more than one or two covariates, the pictures are not as useful, but then it also is probably harder to justify the proportional odds assumption.

**General approach**

The data generation algorithm assumes an underlying latent process logistic process. In the context of a set of multivariate responses, there is a latent process for each of the responses. For a single response, we can randomly select a value from the logistic distribution and determine the response region in which this values falls to assign the randomly generated response. To generate correlated responses, we generate correlated values from the logistic distribution using a standard normal copula-like approach, just as to generate multivariate data from non-normal distributions.

The new function genCorOrdCat requires specification of the baseline probabilities for each of the items in matrix form. The function also provides an argument to incorporate covariates, much like its univariate counterpart genOrdCat. The correlation is specified either with a single correlation coefficient \(\rho\) and a correlation structure (“independence”, “compound symmetry”, or “AR-1”) or by specifying the correlation matrix directly.

**Examples**

In the following examples, I assume four items each with four possible responses – which is different from the EQ-5D.

**High correlation**

In the first simulation items two and three share the same uniform distribution, and items one and four each have their own distribution:

baseprobs <- matrix(c(0.10, 0.20, 0.30, 0.40,

0.25, 0.25, 0.25, 0.25,

0.25, 0.25, 0.25, 0.25,

0.40, 0.30, 0.20, 0.10),

nrow = 4, byrow = TRUE)

# generate the data

set.seed(3333)

dT <- genData(100000)

dX <- genCorOrdCat(dT, adjVar = NULL, baseprobs = baseprobs,

prefix = "q", rho = 0.8, corstr = "cs")

dX

## id q1 q2 q3 q4

## 1: 1 2 1 1 1

## 2: 2 1 1 1 1

## 3: 3 2 2 1 1

## 4: 4 3 3 3 2

## 5: 5 4 2 3 1

## ---

## 99996: 99996 3 4 4 3

## 99997: 99997 2 1 1 2

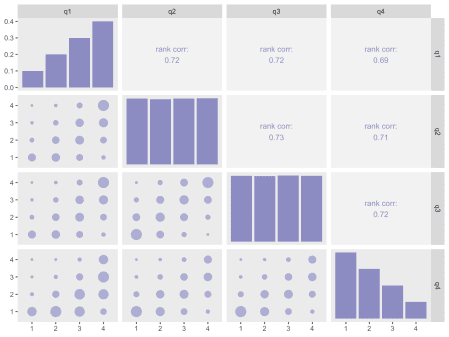
## 99998: 99998 2 2 2 2

## 99999: 99999 3 1 1 1

## 100000: 100000 4 4 4 4

Here is a correlation plot that tries to help us visualize what high correlation looks like under this context. (The plots are generated using function ggpairs from the package GGally. Details of the plot are provided in the addendum.) In the plot, the size of the circles represents the frequency of observations with a particular combination; the larger the circle, the more times we observe a combination. The correlation that is reported is the estimated *Spearman’s Rho*, which is appropriate for ordered or ranked data.

If you look at the plot in the third row and second column of this first example, the observations are mostly located near the diagonal – strong evidence of high correlation.

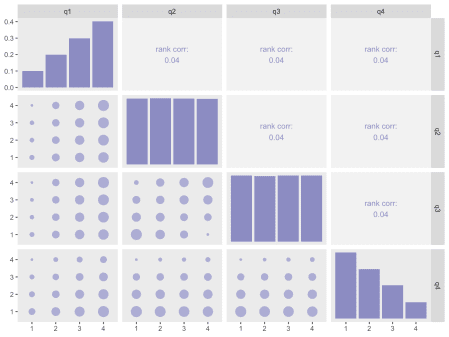


**Low correlation**

dX <- genCorOrdCat(dT, adjVar = NULL, baseprobs = baseprobs,

prefix = "q", rho = 0.05, corstr = "cs")

In this second example with very little correlation, the clustering around the diagonal in the third row/second column is less pronounced.



**Same distribution**

I leave you with two plots that are based on responses that share the same distributions:

baseprobs <- matrix(c(0.1, 0.2, 0.3, 0.4,

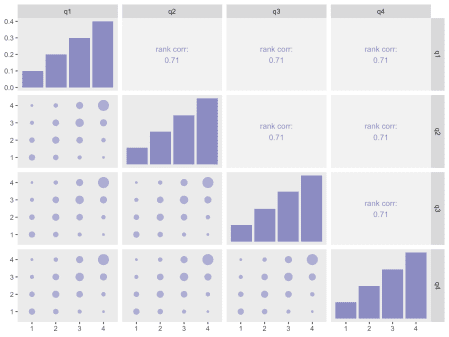
0.1, 0.2, 0.3, 0.4,

0.1, 0.2, 0.3, 0.4,

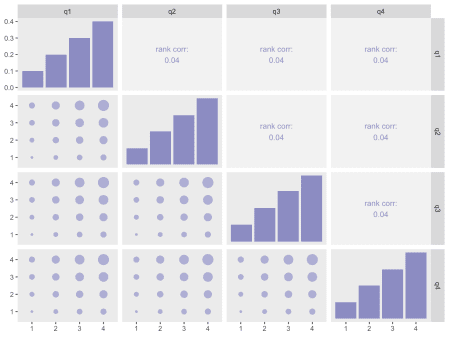
0.1, 0.2, 0.3, 0.4),

nrow = 4, byrow = TRUE)

**High correlation**



**Low correlation**



**Addendum**

In case you are interested in seeing how I generated the correlation plots, here is the code:

library(GGally)

mycor <- function(data, mapping, sgnf=3, size = 8, ...) {

xCol <- as.character(mapping[[1]][[2]])

yCol <- as.character(mapping[[2]][[2]])

xVal <- data[[xCol]]

yVal <- data[[yCol]]

rho <- Hmisc::rcorr(xVal, yVal, type = "spearman")$r[2,1]

loc <- data.table(x=.5, y=.5)

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

xlim(0:1) +

ylim(0:1) +

theme(panel.background = element\_rect(fill = "grey95"),

panel.grid = element\_blank()) +

labs(x = NULL, y = NULL) +

geom\_text(size = size, color = "#8c8cc2",

label =

paste("rank corr:\n", round(rho, sgnf), sep = "", collapse = ""))

p

}

my\_lower <- function(data, mapping, ...){

xCol <- as.character(mapping[[1]][[2]])

yCol <- as.character(mapping[[2]][[2]])

dx <- data.table(data)[ , c(xCol, yCol), with = FALSE]

ds <- dx[, .N,

keyby = .(eval(parse(text=xCol)), eval(parse(text=yCol)))]

setnames(ds, c("parse", "parse.1"), c(xCol, yCol))

p <- ggplot(data = ds, mapping = mapping) +

geom\_point(aes(size = N), color = "#adadd4") +

scale\_x\_continuous(expand = c(.2, 0)) +

scale\_y\_continuous(expand = c(.2, 0)) +

theme(panel.grid = element\_blank())

p

}

my\_diag <- function(data, mapping, ...){

p <- ggplot(data = data, mapping = mapping) +

geom\_bar(aes(y = (..count..)/sum(..count..)), fill = "#8c8cc2") +

theme(panel.grid = element\_blank())

p

}

ggpairs(dX[, -"id"], lower = list(continuous = my\_lower),

diag = list(continuous = my\_diag),

upper = list(continuous = wrap(mycor, sgnf = 2, size = 3.5)))