# Appendix

# R Code Used in Study

# Description ##############################################################################

# The R code below is not comprehensive, but contains key portions of the code used in Student (2021).

# This entails code to calculate one of the statistics in the Snijders (2001) family; to calculate AUROCS;

# to simulate item responses; to summarize calculated person fit statistics; and to construct 200,000-respondent

# mixed datasets as in Sinharay (2017).

# The replication portion of the study can be conducted with nearly identical code; all that changes

# is the crossed design of the simulation portion of the code. I also refer readers to Sinharay (2017),

# which links to a Github repository containing Dr. Sinharay’s original R code on which this study is based.

##############################################################################

setwd("") # this line should be updated to reflect current working directory.

set.seed(138123) # this should also be updated.

library(MESS)

library(tidyverse)

library(ltm)

library(irtoys)

library(PerFit)

library(ROCR)

library(metafor)

## from Magis (2012): most of the code to calculate lz, eci4z, and other stats in this family

# such as those in Xia & Zheng (2018), shown below.

## th: ability value

## it: matrix of item parameters: one row per item, three columns

## (discrimination, difficulty, pseudo-guessing)

Pi<-function(th,it){

 res1<-res2<-res3<-NULL

 for (i in 1:nrow(it)) {

 res1[i]<-

 it[i,3]+(1-it[i,3])\*exp(it[i,1]\*(th-it[i,2]))/(1+exp(it[i,1]\*(th-it[

 i,2])))

 res2[i]<-(1-it[i,3])\*it[i,1]\*exp(it[i,1]\*(th-it[i,2]))/(1+exp(it[i,1]\*(th-it[

 i,2])))^2

 res3[i]<-(1-it[i,3])\*it[i,1]^2\*exp(it[i,1]\*(th-it[i,2]))\*(1-exp(it[i,1]\*(th-it[

 i,2])))/(1+exp(it[i,1]\*(th-it[i,2])))^3

 }

 RES<-list(Pi=res1,dPi=res2,d2Pi=res3)

 return(RES)

}

## Functions ri and r0 (equations 24, 27 and 33)

## method: « ML » for maximum likelihood, « BM » for Bayesian modal (or MAp), « WL » for weighted likelihood

## mu, sigma: prior mean and standard deviation parameters of the normal distribution

ri<-function(th,it) Pi(th,it)$dPi/(Pi(th,it)$Pi\*(1-Pi(th,it)$Pi))

r0<-function(method="ML",th,it,mu=0,sigma=1){

 res<-switch(method,

 ML=0,

 BM=(mu-th)/sigma^2,

 WL=sum(ri(th,it)\*Pi(th,it)$d2Pi)/(2\*sum(ri(th,it)\*Pi(th,it)$dPi)))

 return(res)}

## Ability estimation (constrained to range [-4; 4]) (equation 14)

# thetaEst<-function(x,it,method="ML",mu=0,sigma=1){

# f<-function(th){

# r0(method=method,th,it=it,mu=mu,sigma=sigma)+sum((x-Pi(th,it)$Pi)\*ri(th,it))

# }

# if (f(-4)<0 & f(4)<0) {res <- -4} else{

# if (f(-4)>0 & f(4)>0) {res <- 4} else res<-uniroot(f,c(-4,4))$root

# }

# return(res)}

# SHa(1/2) from Xia and Zheng (2018).

# this is based upon the above Magis (2012) code as suggested by Xia & Zheng (2018).

# Other statistics from Xia & Zheng, as well as eci4z-star, can be calculated using a function

# identical to this one except for the weighting function wi. Weight functions can be found in Xia & Zheng (2018).

# responses: matrix of 0/1 item responses, columns are items and rows are persons

# Ability: vector of theta estimates

# IP: item parameters. disc, diff, guess.

sha1.2<- function(responses, theta, it){

 ## Weight function for SHa(1/2)

 wi<-function(th,it) (1-2\* Pi(th,it)$Pi)/sqrt(Pi(th,it)$Pi)

 ## Function Wn(theta) (equation 9)

 ## x: response pattern (same length as nrow(it), zeros and ones as entries)

 Wn<-function(x,th,it) sum((x-Pi(th,it)$Pi)\*wi(th,it))

 ## Function sig2n (equation 11)

 sig2n<-function(th,it)

 sum(wi(th,it)^2\*Pi(th,it)$Pi\*(1-Pi(th,it)$Pi))/nrow(it)

 ## Function cn (equation 15)

 cn<-function(th,it) sum(Pi(th,it)$dPi\*wi(th,it))/sum(Pi(th,it)$dPi\*ri(th,it))

 ## Function wi ‘tilde’ (equation 16)

 wiTilde<-function(th,it) wi(th,it)-cn(th,it)\*ri(th,it)

 ## Function tau2n (equation 18)

 tau2n<-function(th,it)

 sum(wiTilde(th,it)^2\*Pi(th,it)$Pi\*(1-Pi(th,it)$Pi))/nrow(it)

 ## snijders: logical argument: FALSE returns standardized, TRUE returns asympotically corrected

 ## so for SH's, we want it TRUE

 ## added th argument for pre-estimated theta

 SHa1.2<-function(x,it,th,method="ML",mu=0,sigma=1,snijders=TRUE){

 if (snijders==TRUE){

 EWn<-cn(th,it)\*r0(method=method,th=th,it=it,mu=mu,sigma=sigma)

 VWn<-nrow(it)\*tau2n(th,it)

 } else{

 EWn<-0

 VWn<-nrow(it)\*sig2n(th,it)

 }

 res<-(Wn(x,th,it)-EWn)/sqrt(VWn)

 return(res)

 }

 stats <- vector()

 for(row in 1:nrow(responses)){

 th <- theta[row]

 x <- as.numeric(responses[row,][1,])

 stat <- SHa1.2(x, it, th)

 stats <- append(stats, stat)

 }

 return(stats)

}

# from Sinharay, 2017: Function to compute an ROC Area and standard error

ROC = function(good,bad,x) {

 Areas=rep(0,ncol(good))

 SE=Areas

 F=matrix(0,length(x),ncol(good))

 H=F

 for (j in 1:ncol(good)){

 goodstat=good[,j]

 badstat=bad[,j]

 for (i in 1:length(x)){

 # here, x[i] is the cutoff. it starts at -5 and ends at 5. all stats are standardized, and stats

 # where large values indicate aberrance are flipped

 F[i,j]=length(goodstat[goodstat<x[i]])/length(goodstat)

 H[i,j]=length(badstat[badstat<x[i]])/length(badstat)

 }

 #Use the R function `auc' to compute the ROC area

 a=auc(F[,j],H[,j],type='spline')

 Areas[j]=ifelse(a>1,0.999,a)

 a=Areas[j]

 ng=nrow(good)

 nb=nrow(bad)

 q1=a/(2-a)

 q2=2\*a\*a/(1+a)

 SE[j]=sqrt(( a\*(1-a)+(nb-1)\*(q1-a\*a)+(ng-1)\*(q2-a\*a) )/(ng\*nb))

 }

 return(rbind(Areas,SE,F,H))

}

# Get everything in place for the analysis.

ItemResponses <- list()

# Cheating-specific output

Output.ch <- NULL

SEs.ch <- NULL

x=seq(-5, 5, length.out=100)

# General aberrance detection output

Output.all <- NULL

SEs.all <- NULL

n <- 10000

# Simulation conditions

# Test lengths

lengths = c(17, 33, 65)

# Cheating only, cheating plus... creative, guessing, careless, random.

types = c("ch", "cr","g","ca","r")

# Proportions of aberrance

amounts = c(0.01, 0.05, 0.1, 0.25)

# construct the matrix of test conditions

ds\_conditions <- data.frame(

 length = c(

 rep(lengths[1], 20),

 rep(lengths[2], 20),

 rep(lengths[3], 20)

 ),

 type = c(

 rep(

 c(

 rep(types[1], 4),

 rep(types[2], 4),

 rep(types[3], 4),

 rep(types[4], 4),

 rep(types[5], 4)

 ),

 3

 )

 ),

 amount = c(

 rep(amounts, 15)

 )

)

trueitemdifficulties.17 = seq(-2, 2, length.out = 17)

trueitemdifficulties.33 = seq(-2, 2, length.out = 33)

trueitemdifficulties.65 = seq(-2, 2, length.out = 65)

statsK03=list() #PFSs of all examinees over 60 data sets

# Now, construct datasets and calculate stats

# Data generation will take several hours, even on a very powerful computer

for(i in 1:60){

 condition <- ds\_conditions[i,]

 length <- condition$length

 diffs <- switch(

 as.character(length),

 "17"=trueitemdifficulties.17,

 "33"=trueitemdifficulties.33,

 "65"=trueitemdifficulties.65

 )

 type = condition$type

 amt = 10000\*condition$amount

 examinees <- data.frame(

 aberrant = rep(FALSE, n),

 type = rep(type, n)

 )

 # assign true thetas by a standard normal distribution

 examinees$theta <- rnorm(n)

 # insert columns for item responses

 for(j in 1:length){

 examinees[[paste("i", j, sep="")]] <- -1

 }

 # sort by ability

 examinees <- examinees[order(examinees$theta),]

 examinees$type = "normal"

 # next, designate a certain number of examinees as aberrant according to amt,

 # and simulate their aberrant responses

 if(type=="ch"){

 # select amt cheaters from the examinees with ability below -0.5

 examinees[sample(1:round(n\*pnorm(-0.5)), amt, replace = FALSE),]$aberrant <- TRUE

 examinees[which(examinees$aberrant==TRUE),]$type <- "cheating"

 # select a random subset of harder 50% of items as "compromised"

 compromised <- sample(as.integer(length/2):length, size = as.integer(0.18\*length), replace = FALSE)

 # simulate responses

 for(k in 1:length){

 item <- paste("i", k, sep="")

 examinees[[item]] <- apply(

 examinees,

 1,

 function(drow){

 switch(

 drow["type"],

 "cheating" = ifelse(

 k %in% compromised,

 1,

 rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 ),

 "normal" = rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 )

 }

 )

 }

 } else if(type=="cr") { # creative

 # select amt/2 creative from the examinees with ability above about 0.5

 crs <- sample(round(n\*pnorm(0.5)):n, amt/2, replace = FALSE)

 examinees[crs,]$aberrant <- TRUE

 examinees[which(examinees$aberrant==TRUE),]$type <- "creative"

 # select amt/2 cheating from the examinees with ability below -0.5

 examinees[sample(round(n\*pnorm(-0.5)):n, amt/2, replace = FALSE),]$aberrant <- TRUE

 examinees[which(examinees$aberrant==TRUE & examinees$type != "creative"),]$type <- "cheating"

 # select a random subset of items as "compromised"

 compromised <- sample(as.integer(length/2):length, size = as.integer(0.18\*length), replace = FALSE)

 # now, simulate answering creatively/cheating

 # then simulate the normal respondents via the Rasch model

 for(k in 1:length){

 item <- paste("i", k, sep="")

 examinees[[item]] <- apply(

 examinees,

 1,

 function(drow){

 switch(

 drow["type"],

 "creative" = ifelse(

 diffs[k] <= -1.5,

 0,

 rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 ),

 "cheating" = ifelse(

 k %in% compromised,

 1,

 rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 ),

 "normal" = rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 )

 }

 )

 }

 } else if(type=="g") { # guessing

 # select amt/2 each guessing and cheating from the examinees with ability below -0.5

 examinees[sample(1:round(n\*pnorm(-0.5)), amt, replace = FALSE),]$aberrant <- TRUE

 examinees[which(examinees$aberrant==TRUE),]$type <- "guessing"

 examinees[sample(which(examinees$aberrant==TRUE), size = amt/2, replace = FALSE),]$type <- "cheating"

 # select a random subset of 18% of items as "compromised"

 compromised <- sample(as.integer(length/2):length, size = as.integer(0.18\*length), replace = FALSE)

 # now, simulate guessing and cheating

 # then simulate the normal respondents via the Rasch model

 for(k in 1:length){

 item <- paste("i", k, sep="")

 examinees[[item]] <- apply(

 examinees,

 1,

 function(drow){

 switch(

 drow["type"],

 "guessing" = ifelse(

 diffs[k] >= 0.5,

 rbernoulli(1, 0.25) %>% as.numeric(),

 rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 ),

 "cheating" = ifelse(

 k %in% compromised,

 1,

 rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 ),

 "normal" = rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 )

 }

 )

 }

 } else if(type=="ca") { # careless

 # select amt/2 careless from the examinees with ability above about 0.5

 examinees[sample(round(n\*pnorm(0.5)):n, amt/2, replace = FALSE),]$aberrant <- TRUE

 examinees[which(examinees$aberrant==TRUE),]$type <- "careless"

 # select amt/2 cheating from the examinees with ability below -0.5

 examinees[sample(round(n\*pnorm(-0.5)):n, amt/2, replace = FALSE),]$aberrant <- TRUE

 examinees[which(examinees$aberrant==TRUE & examinees$type != "careless"),]$type <- "cheating"

 # select a random subset of 18% of items as "compromised"

 compromised <- sample(as.integer(length/2):length, size = as.integer(0.18\*length), replace = FALSE)

 # now, simulate answering carelessly/cheating

 # then simulate the normal respondents via the Rasch model

 for(k in 1:length){

 item <- paste("i", k, sep="")

 examinees[[item]] <- apply(

 examinees,

 1,

 function(drow){

 switch(

 drow["type"],

 "careless" = ifelse(

 diffs[k] <= -0.5,

 rbernoulli(

 1,

 0.5

 ) %>% as.numeric(),

 rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 ),

 "cheating" = ifelse(

 k %in% compromised,

 1,

 rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 ),

 "normal" = rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 )

 }

 )

 }

 } else { # random

 # select amt/2 random and cheating from the examinees with ability below -0.5

 examinees[sample(1:round(n\*pnorm(-0.5)), amt, replace = FALSE),]$aberrant <- TRUE

 examinees[which(examinees$aberrant==TRUE),]$type <- "random"

 examinees[sample(which(examinees$aberrant==TRUE), size = amt/2, replace = FALSE),]$type <- "cheating"

 compromised <- sample(as.integer(length/2):length, size = as.integer(0.18\*length), replace = FALSE)

 # now, simulate them answering at random

 # then simulate the normal respondents via the Rasch model

 for(k in 1:length){

 item <- paste("i", k, sep="")

 examinees[[item]] <- apply(

 examinees,

 1,

 function(drow){

 switch(

 drow["type"],

 "random" = rbernoulli(1, 0.25) %>% as.numeric(),

 "cheating" = ifelse(

 k %in% compromised,

 1,

 rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 ),

 "normal" = rbernoulli(

 1,

 1/(1+exp(diffs[k]-as.numeric(drow["theta"])))

 ) %>% as.numeric()

 )

 }

 )

 }

 }

 # Here, I analyze simulated responses

 # fit the Rasch model using ltm as in Sinharay (2017)

 responses <- examinees[,4:(length+3)]

 scores.rasch = rasch(responses, constraint = cbind(length + 1, 1)) #fit the model

 itparm=coef(scores.rasch)#Estimated item parameters

 b=itparm[,1]#Difficulty parameters

 a=itparm[,2]#Common slope parameter

 raw=apply(responses,1,sum)

 responses <- responses[raw>0 & raw<length,]#Remove those with 0 or full score: HT undefined for them

 examinees <- examinees[raw>0 & raw<length,]

 theta=examinees$theta[raw>0 & raw<length]

 #Estimate the examinee abilities using package 'irtoys' as in Sinharay (2017)

 itparms=cbind(a,b,rep(0,length))

 thetaest=mlebme(responses,itparms)[,1]

 #Compute lz\*, HT, U3 using R package Perfit as in Sinharay (2017)

 lzs=lzstar(responses,Ability=thetaest,IP=itparms)

 Hts = Ht(responses)

 U3s=U3(responses,Ability=thetaest,IP=itparms)#U3 is large for aberrant examinees

 # The functions below need to be defined above; sha1.2 was provided as an example.

 # compute eci4z\*

 eci4zs <- eci4zstar(responses, thetaest, itparms)

 # and xia & zheng's asymmetrically-weighted PFSs

 sha1.2s <- sha1.2(responses, thetaest, itparms)

 sha1s <- sha1(responses, thetaest, itparms)

 shb2s <- shb2(responses, thetaest, itparms)

 shb3s <- shb3(responses, thetaest, itparms)

 # combine them all together; add a negative sign for PFSs where aberrance is indicated with a large value

 stats=cbind(

 lzs[[1]]$PFscores,

 Hts[[1]]$PFscores,

 -U3s[[1]]$PFscores,

 -eci4zs,

 -sha1.2s,

 -sha1s,

 -shb2s,

 -shb3s

 )

 statsK03[[paste(type, length, amt)]] <- stats

 stats=scale(stats,center=TRUE,scale=TRUE)#Standardize each PFS

 ## cheating detection rates

 good=stats[which(examinees$type != "cheating"),]#PFSs for the non-cheating examinees

 bad=stats[which(examinees$type == "cheating"),]#PFSs for the cheating examinees

 ROCOut=ROC(good,bad,x)#Compute ROC Areas using the function `ROC'

 Areas=ROCOut[1,]#ROC Areas of the PFSs

 F=ROCOut[3:(length(x)+2),]#False-alarm rates

 H=ROCOut[(length(x)+3):nrow(ROCOut),]#Hit rates

 SEs.ch=rbind(SEs.ch,ROCOut[2,])

 Output.ch=rbind(Output.ch,c(length,as.character(type),100\*condition$amount,Areas))

 ## general aberrance detection rates

 good=stats[which(examinees$aberrant == FALSE),]#PFSs for the non-aberrant examinees

 bad=stats[which(examinees$aberrant == TRUE),]#PFSs for the aberrant examinees

 ROCOut=ROC(good,bad,x)#Compute ROC Areas using the function `ROC'

 Areas=ROCOut[1,]#ROC Areas of the PFSs

 F=ROCOut[3:(length(x)+2),]#False-alarm rates

 H=ROCOut[(length(x)+3):nrow(ROCOut),]#Hit rates

 SEs.all=rbind(SEs.all,ROCOut[2,])

 Output.all=rbind(Output.all,c(length,as.character(type),100\*condition$amount,Areas))

 ## save item responses for later

 ItemResponses[[paste(type, length, amt)]] <- examinees

}

# I recommend writing results to RDS files because data simulation takes so long.

write\_rds(ItemResponses, ./raw\_item\_responses.csv")

write\_rds(statsK03, ./rawstats.csv")

write.csv(Output.ch, ./output\_ch.csv")

write.csv(Output.all, ./output\_all.csv")

statsK03 <- readRDS(./rawstats.csv")

ItemResponses <- readRDS(./raw\_item\_responses.csv")

end<-Sys.time()

print(end - start)

# The code below shows how to calculate average AUROCs over all conditions using the

# Dersimonian-Laird algorithm, as in Sinharay (2017). Analogous code can be written to

# summarize over specific lengths, proportions of aberrance or aberrance types.

out.ch.overall.dsl <- NULL

Areas <- out.ch

Areas <- Areas %>% pivot\_longer(names\_to = "stat", values\_to = "AUC", cols = c("lz\*", "ht", "u3", "eci4z\*", "sha1/2", "sha1", "shb2", "shb3"))

se.df <- data.frame(SEs.ch)

names(se.df) <- c("lz\*", "ht", "u3", "eci4z\*", "sha1/2", "sha1", "shb2", "shb3")

se.df <- se.df %>% pivot\_longer(names\_to = "st", values\_to = "se", cols = c("lz\*", "ht", "u3", "eci4z\*", "sha1/2", "sha1", "shb2", "shb3"))

dsl.dat <- cbind(Areas, se.df)

#Use DerSimonian-Laird algorithm

for (stat in unique(dsl.dat$stat)){

 out.ch.overall.dsl=c(

 out.ch.overall.dsl,

 rma(

 yi = as.numeric(dsl.dat[which(dsl.dat$stat==stat),]$AUC),

 sei = as.numeric(dsl.dat[which(dsl.dat$stat==stat),]$se),

 # data = data.frame(yi = Areas[,i], sei = SEs.ch[,i]),

 method="DL"

 )$beta

 )

}

out.ch.overall.dsl <- data.frame(out.ch.overall.dsl %>% t())

names(out.ch.overall.dsl) <- c("lz\*", "ht", "u3", "eci4z\*", "sha1/2", "sha1", "shb2", "shb3")

# The code below calculates the values found in Table 12. Code is also included to

# calculate sensitivity and specificity, though these have problems described in the paper.

thetastats <- list()

counts.flagged <- list()

for(r in names(statsK03)){

 # 1) join examinees' theta and response type to table of raw statistics

 st <- statsK03[[r]]

 th <- ItemResponses[[r]][,1:3]

 thetastats[[r]] <- cbind(th, st)

 names(thetastats[[r]]) <- c("aberrant", "type", "theta", "lz\*", "ht", "u3", "eci4z\*", "sha1/2", "sha1", "shb2", "shb3")

 condition <- strsplit(r, " ")

 counts.flagged[[r]] <- data.frame(

 type = condition[[1]][1],

 naber = condition[[1]][3],

 length = condition[[1]][2],

 # flagged and cheating

 sha1.2.cheating = length(which(thetastats[[r]]$type=="cheating" & thetastats[[r]]$`sha1/2` < -1.64)),

 # flagged and not cheating

 sha1.2.not = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$`sha1/2` < -1.64)),

 # neither flagged nor cheating

 sha1.2.neither = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$`sha1/2` >= -1.64)),

 sha1.cheating = length(which(thetastats[[r]]$type=="cheating" & thetastats[[r]]$sha1 < -1.64)),

 sha1.not = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$sha1 < -1.64)),

 sha1.neither = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$sha1 >= -1.64)),

 shb2.cheating = length(which(thetastats[[r]]$type=="cheating" & thetastats[[r]]$shb2 < -1.64)),

 shb2.not = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$shb2 < -1.64)),

 shb2.neither = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$shb2 >= -1.64)),

 shb3.cheating = length(which(thetastats[[r]]$type=="cheating" & thetastats[[r]]$shb3 < -1.64)),

 shb3.not = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$shb3 < -1.64)),

 shb3.neither = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$shb3 >= -1.64)),

 lzstar.cheating = length(which(thetastats[[r]]$type=="cheating" & thetastats[[r]]$`lz\*` < -1.64)),

 lzstar.not = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$`lz\*` < -1.64)),

 lzstar.neither = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$`lz\*` >= -1.64)),

 ht.cheating = length(which(thetastats[[r]]$type=="cheating" & scale(thetastats[[r]]$ht) < -1.64)),

 ht.not = length(which(thetastats[[r]]$type!="cheating" & scale(thetastats[[r]]$ht) < -1.64)),

 ht.neither = length(which(thetastats[[r]]$type!="cheating" & scale(thetastats[[r]]$ht) >= -1.64)),

 ht.cheatingnoflag = length(which(thetastats[[r]]$type=="cheating" & scale(thetastats[[r]]$ht) >= -1.64)),

 eci.cheating = length(which(thetastats[[r]]$type=="cheating" & thetastats[[r]]$`eci4z\*` < -1.64)),

 eci.not = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$`eci4z\*` < -1.64)),

 eci.neither = length(which(thetastats[[r]]$type!="cheating" & thetastats[[r]]$`eci4z\*` >= -1.64)),

 u3.cheating = length(which(thetastats[[r]]$type=="cheating" & scale(thetastats[[r]]$u3) < -1.64)),

 u3.not = length(which(thetastats[[r]]$type!="cheating" & scale(thetastats[[r]]$u3) < -1.64)),

 u3.neither = length(which(thetastats[[r]]$type!="cheating" & scale(thetastats[[r]]$u3) >= -1.64)),

 u3.cheatingnoflag = length(which(thetastats[[r]]$type=="cheating" & scale(thetastats[[r]]$u3) >= -1.64))

 )

}

counts.concat <- do.call(what = "rbind", counts.flagged)

counts.concat$naber <- as.numeric(as.character(counts.concat$naber))

counts.concat$ncheating <- ifelse(counts.concat$type=="ch", counts.concat$naber, counts.concat$naber/2)

counts.concat$nnormal <- 10000 -counts.concat$ncheating

counts.bypct <- counts.concat %>%

 mutate(

 sha1.2 = sha1.2.cheating + sha1.2.not,

 sha1 = sha1.cheating + sha1.not,

 shb2 = shb2.cheating + shb2.not,

 shb3 = shb3.cheating + shb3.not,

 lzstar = lzstar.cheating + lzstar.not,

 ht = ht.cheating + ht.not,

 eci = eci.cheating + eci.not,

 u3 = u3.cheating + u3.not,

 sha1.2pct = sha1.2.cheating/sha1.2,

 sha1pct = sha1.cheating/sha1,

 shb2pct = shb2.cheating/shb2,

 shb3pct = shb3.cheating/shb3,

 lzstarpct = lzstar.cheating/lzstar,

 htpct = ht.cheating/ht,

 ecipct = eci.cheating/eci,

 u3pct = u3.cheating/u3,

 sha1.2sens = sha1.2.cheating/ncheating,

 sha1sens = sha1.cheating/ncheating,

 shb2sens = shb2.cheating/ncheating,

 shb3sens = shb3.cheating/ncheating,

 lzstarsens = lzstar.cheating/ncheating,

 htsens = ht.cheating/ncheating,

 ecisens = eci.cheating/ncheating,

 u3sens = u3.cheating/ncheating,

 sha1.2spec = sha1.2.neither/(10000-ncheating),

 sha1spec = sha1.neither/(10000-ncheating),

 shb2spec = shb2.neither/(10000-ncheating),

 shb3spec = shb3.neither/(10000-ncheating),

 lzstarspec = lzstar.neither/(10000-ncheating),

 htspec = ht.neither/(10000-ncheating),

 ecispec = eci.neither/(10000-ncheating),

 u3spec = u3.neither/(10000-ncheating)

 ) %>% dplyr::select(

 type, naber, length, sha1.2, sha1, shb2, shb3, lzstar, ht, eci, u3,

 sha1.2pct, sha1pct, shb2pct, shb3pct, lzstarpct, htpct, ecipct, u3pct,

 sha1.2sens, sha1sens, shb2sens, shb3sens, lzstarsens, htsens, ecisens, u3sens,

 sha1.2spec, sha1spec, shb2spec, shb3spec, lzstarspec, htspec, ecispec, u3spec

 ) %>%

 pivot\_longer(

 names\_to = "key",

 values\_to = "value",

 cols = c(

 sha1.2, sha1, shb2, shb3, lzstar, ht, eci, u3,

 sha1.2pct, sha1pct, shb2pct, shb3pct, lzstarpct, htpct, ecipct, u3pct,

 sha1.2sens, sha1sens, shb2sens, shb3sens, lzstarsens, htsens, ecisens, u3sens,

 sha1.2spec, sha1spec, shb2spec, shb3spec, lzstarspec, htspec, ecispec, u3spec

 )

 ) %>% group\_by(

 naber, key

 ) %>% summarise(

 mean = mean(value)

 ) %>% pivot\_wider(

 names\_from = naber, values\_from = mean

 )

# Here, I combine and analyze datasets for each test length into 200,000-examinee datasets as in

# Sinharay (2017). Construction and analysis is only shown for 17-item dataset;

# analogous code for other lengths only requires changing the specified test length.

l17 <- NULL

l17.cheaters <- c()

l17.noncheaters <- c()

# create combined datasets

for(dataset in names(ItemResponses)){

 length <- (strsplit(dataset, " "))[[1]][2]

 type = (strsplit(dataset, " "))[[1]][1]

 dat <- drop\_na(ItemResponses[[dataset]])

 if(length=="17") {

 # note cheaters

 if(type=="ch"){

 l17.cheaters <- append(l17.cheaters, (ifelse(is.null(nrow(l17)), 0, nrow(l17))+which(dat$aberrant==TRUE)))

 l17.noncheaters <- append(l17.noncheaters, (ifelse(is.null(nrow(l17)), 0, nrow(l17))+which(dat$aberrant==FALSE)))

 } else{

 l17.noncheaters <- append(l17.noncheaters, (ifelse(is.null(nrow(l17)), 0, nrow(l17))+1:nrow(dat)))

 }

 l17 <- rbind(l17, dat)

 } else if (length=="33") {

 # removed for brevity; analogous to section for 17 item test

 } else if (length=="65") {

 # removed for brevity; analogous to section for 17 item test

 }

}

# estimate difficulties

responses.17 <- l17[,4:20]

scores.rasch.17 = rasch(responses.17, constraint = cbind(18, 1)) #fit the model

itparm.17=coef(scores.rasch.17)#Estimated item parameters

b.17=itparm.17[,1]#Difficulty parameters

a.17=itparm.17[,2]#Common slope parameter

#Estimate the examinee abilities using package 'irtoys'

itparms.17=cbind(a.17,b.17,rep(0,17))

thetaest.17=mlebme(responses.17,itparms.17)[,1]

#Compute lz\*, HT, U3 using R package Perfit

lzs=lzstar(responses.17,Ability=thetaest.17,IP=itparms.17)

Hts = Ht(responses.17)

U3s=U3(responses.17,Ability=thetaest.17,IP=itparms.17)#U3 is large for aberrant examinees

# compute eci4z\*

eci4zs <- eci4zstar(responses.17, thetaest.17, itparms.17)

# and xia & zheng's asymmetrically-weighted PFSs

sha1.2s <- sha1.2(responses.17, thetaest.17, itparms.17)

sha1s <- sha1(responses.17, thetaest.17, itparms.17)

shb2s <- shb2(responses.17, thetaest.17, itparms.17)

shb3s <- shb3(responses.17, thetaest.17, itparms.17)

# combine them all together; negative for PFSs where aberrance is indicated with a large value

stats.17=cbind(

 lzs[[1]]$PFscores,

 Hts[[1]]$PFscores,

 -U3s[[1]]$PFscores,

 -eci4zs,

 -sha1.2s,

 -sha1s,

 -shb2s,

 -shb3s

)

stats.17=scale(stats.17,center=TRUE,scale=TRUE)#Standardize each PFS

## general aberrance detection rates

good=stats.17[which(l17$aberrant == FALSE),]#PFSs for the non-aberrant examinees

bad=stats.17[which(l17$aberrant == TRUE),]#PFSs for the aberrant examinees

ROCOut=ROC(good,bad,x)#Compute ROC Areas using the function `ROC'

Areas=ROCOut[1,]#ROC Areas of the PFSs

F=ROCOut[3:19,]#False-alarm rates

H=ROCOut[20:nrow(ROCOut),]#Hit rates

SEs.all.17=ROCOut[2,]

Output.all.17=Areas

## cheating detection

good=stats.17[l17.noncheaters,]#PFSs for the non-aberrant examinees

bad=stats.17[l17.cheaters,]#PFSs for the aberrant examinees

ROCOut=ROC(good,bad,x)#Compute ROC Areas using the function `ROC'

Areas=ROCOut[1,]#ROC Areas of the PFSs

F=ROCOut[3:19,]#False-alarm rates

H=ROCOut[20:nrow(ROCOut),]#Hit rates

SEs.ch.17=ROCOut[2,]

Output.ch.17=Areas

# Calculate PPV, # flagged, sensitivity, specificity

combined.17 <- stats.17 %>% data.frame()

names(combined.17) <- c("lz\*", "ht", "u3", "eci4z\*", "sha1/2", "sha1", "shb2", "shb3")

combined.17$cheating <- FALSE

combined.17[l17.cheaters,]$cheating <- TRUE

combined.17 <- combined.17 %>% pivot\_longer(names\_to = "stat", values\_to = "value", cols = c(

 "lz\*", "ht", "u3", "eci4z\*", "sha1/2", "sha1", "shb2", "shb3"

))

combined.17.summary <- combined.17 %>%

 mutate(

 flagged = (value < -1.64)

 ) %>% group\_by(stat) %>%

 summarise(

 fl = sum(flagged),

 ch = sum(cheating),

 both = sum(as.numeric(cheating & flagged)),

 neither = sum(!cheating & !flagged),

 ppv = sum(flagged & cheating)/sum(flagged),

 sens = sum(flagged & cheating)/sum(cheating),

 spec = sum(!flagged & !cheating)/sum(!cheating)

 )