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

#

# DESCRIPTION

# Functions for determination of R-indicators and coefficients of variation.

# A manual of the functions is found in

#

# de Heij, V., Schouten, B., and Shlomo, N. (2015), RISQ manual 2.1, Tools in

# SAS and R for the computation of R-indicators, partial R-indicators

# and partial coefficients of variation, available at www.risq-project.eu

#

# HISTORY

# 2010/05/10 1.0 V. de Heij ---

# 2011/02/16 1.1 V. de Heij Option withBiasAndVar added.

# 2013/07/25 2.0 V. de Heij Standard error approximations for partial R-indicators,

# coefficient of variation and increased flexibility in models

# 2014/12/15 2.1 B. Schouten Inclusion of partial coefficients of variation

#

#

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

getRIndicator <-

function(formula,

sampleData,

sampleWeights = rep(1, nrow(sampleData)),

sampleStrata = NULL,

family = c('binomial', 'gaussian'),

withPartials = TRUE,

withPartialCV = TRUE,

otherVariables = character())

{ #++

# Determines the R-indicators and the partial R-indicators for a sample.

#

# ARGUMENTS

# formula : the respons model which will be used to determine the

# R-indicators; the left hand side of the formula states

# the respons variabele, the right hand side states the

# lineair model of auxiliary variabeles which will be

# used to describe the respons;

#

# sampleData : a data frame containing the sample data;

#

# sampleWeights : (optional) a vector with the inclusion weights of the

# sampling units;

#

# sampleStrata : (optional) a vector with the strata membership of the

# sampling units;

#

# family : (optional) a string either 'binomial' for logistic

# regression or 'gaussian' for lineair regression;

#

# withPartials : (optional) a boolean value, indicating if partial

# R-indicators have to be determined (TRUE) or

# not (FALSE);

# withPartialCV : (optional) a boolean value, indicating if partial

# coefficients of variation have to be determined (TRUE) or

# not (FALSE);

#

# otherVariables : (optional).

#

# VALUE

# getRIndicator returns a list of which the most important components

# are described in the manual.

nSample = nrow(sampleData)

stopifnot(length(sampleWeights) == nSample)

stopifnot(is.numeric(sampleWeights))

# If sampleStrata is not defined, use sampleWeights to guess the values of

# sampleStrata.

if (is.null(sampleStrata))

sampleStrata <- getSampleStrata(sampleWeights)

stopifnot(is.factor(sampleStrata))

stopifnot(length(sampleStrata) == nSample)

sampleDesign <- getSampleDesign(sampleWeights, sampleStrata)

family <- match.arg(family)

model <- switch(family,

'binomial' = list(

formula = formula,

grad = function(mu) exp(mu) / (1 + exp(mu))^2,

family = binomial(link = 'logit')),

'gaussian' = list(

formula = formula,

grad = function(mu) 1,

family = gaussian(link = 'identity')))

indicator <- getRSampleBased(model, sampleData, sampleDesign)

if ((withPartials) | (withPartialCV))

estpartials <- getPartialRs(

indicator, sampleData, sampleDesign, otherVariables)

if (withPartials)

indicator$partialR <- estpartials$partialR

if (withPartialCV)

indicator$partialCV <- estpartials$partialCV

return (indicator)

} #--

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

#

# Private functions for the estimation of the sample-based indicators.

#

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

getRSampleBased <-

function(model,

sampleData,

sampleDesign)

{ #++

# Determines the sample-based R-indicator.

# See 9.9 from Regression Modelling Strategies (Harrell, 2001) for a

# motivation for the factor

#

# mean(sampleWeights) = sum(sampleWeights) / length(sampleWeights).

sampleData <- within(sampleData, {

sampleWeights <- sampleDesign$weights

sampleWeights <- sampleWeights / mean(sampleWeights) })

modelfit <- glm(model$formula, model$family, sampleData, sampleWeights)

prop <- predict(modelfit, type = 'response')

propMean <- weighted.mean(prop, sampleDesign$weights)

propVar <- weightedVar(prop, sampleDesign$weights)

# Because estimaters of bias and variance both use the following vectors

# and matrix, they are calculated only once and passed to the functions.

sigma <- vcov(modelfit)

x <- model.matrix(model$formula, sampleData)[, colnames(sigma)]

z <- model$grad(predict(modelfit, type = 'link')) \* x

withBiasAndVar <- !is.null(sigma) && all(!is.na(sigma))

if (withBiasAndVar) {

RBias <- getBiasRSampleBased(prop, z, sigma, sampleDesign)

RVar <- getVarianceRSampleBased(prop, z, sigma, sampleDesign)

# To simplify formulas the bias correction of the variance will be

# written as a factor, 1 - bias / (variance of propensities).

if (RBias > propVar)

RBiasFactor <- 0

else

RBiasFactor <- 1 - RBias / propVar

} else {

RBias <- NA

RBiasFactor <- NA

RVar <- NA

}

CVUnadj <- sqrt(propVar) / propMean

CV <- sqrt(propVar \* RBiasFactor)/propMean

CVVar <- 0.25 \* RVar / propMean^2 + CV^4 / nrow(sampleData)

indicator <- list(

type = 'R-indicator, sample based',

sampleDesign = sampleDesign,

prop = prop,

propMean = propMean,

model = model,

modelfit = modelfit,

sigma = sigma,

z = z,

R = 1 - 2 \* sqrt(propVar \* RBiasFactor),

RUnadj = 1 - 2 \* sqrt(propVar),

RSE = sqrt(RVar),

RBiasFactor = RBiasFactor,

CVUnadj = CVUnadj,

CV = CV,

CVSE = sqrt(CVVar))

return (indicator)

} #--

getBiasRSampleBased <-

function(prop,

z,

sigma,

sampleDesign)

{ #++

# Estimates the bias of the estimator for the variance of the

# propensities.

nPopulation <- sum(sampleDesign$weights)

propVar <- sampleDesign$getVarTotal(sampleDesign, prop)

z <- z \* sqrt(sampleDesign$weights)

lambda1 <- sum(apply(z, 1, function(zi) return(t(zi) %\*% sigma %\*% zi)))

lambda2 <- propVar / nPopulation

bias <- (lambda1 - lambda2) / nPopulation

return (bias)

} #--

getVarianceRSampleBased <-

function(prop,

z,

sigma,

sampleDesign)

{ #++

# Estimates the variance of the estimator for the R-indicator.

weights <- sampleDesign$weights

nSample <- length(weights)

nPopulation <- sum(weights)

propMean <- weighted.mean(prop, weights)

propVar <- weightedVar(prop, weights, method = 'ML')

propZ <- cbind(prop, z)

A <- cov.wt(propZ, wt = weights, method = 'ML')$cov[-1, 1]

B <- cov.wt(z, wt = weights, method = 'ML')$cov

C <- sampleDesign$getVarTotal(sampleDesign, (prop - propMean)^2)

variance <- numeric()

variance[1] <- 4 \* t(A) %\*% sigma %\*% A

variance[2] <- 2 \* getTrace(B %\*% sigma %\*% B %\*% sigma)

variance[3] <- C / nPopulation^2

variance <- sum(variance) / propVar

return (variance)

} #--

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

#

# Private functions for the estimation of the sample-based, partial

# indicators.

#

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

getPartialRs <-

function(indicator,

sampleData,

sampleDesign,

otherVariables = character())

{ #++

# Estimates both unconditional and conditional partial R-indicators.

RR <- indicator$propMean

modelVariables <- getVariables(indicator$model$formula, FALSE)

variables <- unique(c(modelVariables, otherVariables))

byVariablesR <- NULL

byVariablesCV <- NULL

byCategoriesR <- list()

byCategoriesCV <- list()

for (variable in variables) {

pConditional <-

getPartialRConditional(

indicator, variable, sampleData, sampleDesign)

pUnconditional <-

getPartialRUnconditional(

indicator, variable, sampleData, sampleDesign)

byVariableR <- data.frame(

variable = variable,

Pu = pUnconditional$Pu,

PuUnadj = pUnconditional$PuUnadj,

PuSE = pUnconditional$PuSE,

Pc = pConditional$Pc,

PcUnadj = pConditional$PcUnadj,

PcSEApprox = pUnconditional$PuSE)

byVariableCV <- data.frame(

variable = variable,

CVu = pUnconditional$Pu/RR,

CVuUnadj = pUnconditional$PuUnadj/RR,

CVuSE = pUnconditional$PuSE/RR,

CVc = pConditional$Pc/RR,

CVcUnadj = pConditional$PcUnadj/RR,

CVcSEApprox = pUnconditional$PuSE/RR)

byVariablesR <- rbind(byVariablesR, byVariableR)

byVariablesCV <- rbind(byVariablesCV, byVariableCV)

byCategoryR <- merge(

pUnconditional$byCategory,

pConditional$byCategory)

byCategoryCV <- byCategoryR

names(byCategoryCV) <- c("category","CVuUnadj","CVuUnadjSE","CVcUnadj","CVcUnadjSE")

byCategoryCV$CVuUnadj <- byCategoryCV$CVuUnadj/RR

byCategoryCV$CVuUnadjSE <- byCategoryCV$CVuUnadjSE/RR

byCategoryCV$CVcUnadj <- byCategoryCV$CVcUnadj/RR

byCategoryCV$CVcUnadjSE <- byCategoryCV$CVcUnadjSE/RR

byCategoriesR <- c(byCategoriesR, list(byCategoryR))

byCategoriesCV <- c(byCategoriesCV, list(byCategoryCV))

}

names(byCategoriesR) <- byVariablesR$variable

names(byCategoriesCV) <- byVariablesCV$variable

partialR <- list(

byVariables = byVariablesR,

byCategories = byCategoriesR)

partialCV <- list(

byVariables = byVariablesCV,

byCategories = byCategoriesCV)

partials <- list(

partialR = partialR,

partialCV = partialCV)

return (partials)

} #--

getPartialRUnconditional <-

function(indicator,

variable,

sampleData,

sampleDesign)

{ #++

# Estimates unconditional partial R-indicators.

stopifnot(variable %in% names(sampleData))

categories <- sampleData[[variable]]

RBiasFactor <- indicator$RBiasFactor

nPopulation <- sum(indicator$sampleDesign$weights)

propMean <- indicator$propMean

arg <- with(indicator,

data.frame(

n = sampleDesign$weights,

prop = sampleDesign$weights \* prop))

byCategory <- within(

aggregate(arg, list(category = categories), sum), {

prop <- prop / n

propSign <- sign(n \* (prop - propMean))

propVar <- n \* (prop - propMean)^2 / nPopulation

PuUnadj <- propSign \* sqrt(propVar) })

model <- within(indicator$model,

formula <- replaceRHSByVariable(formula, variable))

propVar <- sum(byCategory$propVar)

Pu <- sqrt(propVar \* RBiasFactor)

PuUnadj <- sqrt(propVar)

PuSE <- 0.5 \* getRSampleBased(model, sampleData, sampleDesign)$RSE

partialIndicator <- list(

type = 'Unconditional partial R-indicator, sample based',

variable = variable,

Pu = Pu,

PuUnadj = PuUnadj,

PuSE = PuSE,

byCategory = byCategory)

partialIndicator <- getVariancePartialRUnconditional(

partialIndicator, indicator, sampleData, sampleDesign)

partialIndicator$byCategory <-

partialIndicator$byCategory[c('category', 'PuUnadj', 'PuUnadjSE')]

return (partialIndicator)

} #--

getVariancePartialRUnconditional <-

function(partialIndicator,

indicator,

sampleData,

sampleDesign)

{ #++

# Calculate the variance of the partial-R indicators.

nPopulation <- sum(sampleDesign$weights)

variable <- partialIndicator$variable

byCategory <- partialIndicator$byCategory

prop <- indicator$prop

V1 <- numeric()

V2 <- numeric()

nSample <- nrow(sampleData)

for (index in seq(nrow(byCategory))) {

label <- byCategory[index, 'category']

delta <- ifelse(sampleData[[variable]] == label, 1, 0)

deltaC <- 1 - delta

V1[index] <- sampleDesign$getVarTotal(sampleDesign, delta \* prop)

V2[index] <- sampleDesign$getVarTotal(sampleDesign, deltaC \* prop)

}

partialIndicator$byCategory <- within(byCategory, {

PuUnadjSE <- sqrt(n / nPopulation \* (

V1 \* (1 / n - 1 / nPopulation)^2 +

V2 \* (1 / nPopulation)^2)) })

return (partialIndicator)

} #--

getPartialRConditional <-

function(indicator,

variable,

sampleData,

sampleDesign)

{ #++

# Estimates conditional partial R-indicators.

stopifnot(variable %in% names(sampleData))

sampleWeights <- indicator$sampleDesign$weights

modelVariables <- getVariables(indicator$model$formula, FALSE)

otherVariables <- modelVariables %sub% variable

otherCategories <- as.list(sampleData[otherVariables])

propMeanByOthers <- with(indicator,

ave(sampleWeights \* prop, otherCategories, FUN = sum) /

ave(sampleWeights, otherCategories, FUN = sum))

zMeanByOthers <- apply(

indicator$z, 2,

FUN = function(x) return (

ave(sampleWeights \* x, otherCategories, FUN = sum) /

ave(sampleWeights, otherCategories, FUN = sum)))

categories <- sampleData[[variable]]

RBiasFactor <- indicator$RBiasFactor

weights <- sampleWeights / sum(sampleWeights)

arg <- with(indicator,

data.frame(

n = sampleWeights,

propVar = weights \* (prop - propMeanByOthers)^2))

byCategory <- within(

aggregate(arg, list(category = categories), sum), {

PcUnadj <- sqrt(propVar) } )

propVar <- sum(byCategory$propVar)

Pc <- sqrt(propVar \* RBiasFactor)

PcUnadj <- sqrt(propVar)

partialIndicator <- list(

type = 'Conditional partial R-indicator, sample based',

variable = variable,

Pc = Pc,

PcUnadj = PcUnadj,

byCategory = byCategory,

propMeanByOthers = propMeanByOthers,

zMeanByOthers = zMeanByOthers)

partialIndicator <- getVariancePartialRConditional(

partialIndicator, indicator, sampleData, sampleDesign)

partialIndicator$byCategory <-

partialIndicator$byCategory[c('category', 'PcUnadj', 'PcUnadjSE')]

return (partialIndicator)

} #--

getVariancePartialRConditional <-

function(partialIndicator,

indicator,

sampleData,

sampleDesign)

{ #++

byCategory <- partialIndicator$byCategory

variable <- partialIndicator$variable

sigma <- indicator$sigma

weights <- sampleDesign$weights

nPopulation <- sum(weights)

propDelta <- indicator$prop - partialIndicator$propMeanByOthers

zDelta <- indicator$z - partialIndicator$zMeanByOthers

variance <- numeric()

for (index in seq(nrow(byCategory))) {

label <- byCategory[index, 'category']

delta <- ifelse(sampleData[[variable]] == label, 1, 0)

zDeltaWeight <- zDelta \* delta \* weights

propDelta2 <- propDelta \* propDelta \* delta

A <- matrix(propDelta, nrow = 1) %\*% zDeltaWeight

B <- t(zDelta) %\*% zDeltaWeight

V1 <- 4 \* A %\*% sigma %\*% t(A)

V2 <- 2 \* getTrace(B %\*% sigma %\*% B %\*% sigma)

V3 <- sampleDesign$getVarTotal(sampleDesign, propDelta2)

variance[index] <-

0.25 \* (V1 + V2 + V3) /

(nPopulation \* sum(propDelta2 \* weights))

}

partialIndicator$byCategory <- within(byCategory,

PcUnadjSE <- sqrt(variance))

return (partialIndicator)

} #--

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

#

# Other private functions, ... .

#

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

getSampleStrata <-

function(sampleWeights,

nMaxStrata = 20)

{ #++

# Guesses a definition of the sample strata, using the values of the sample

# weights.

weights <- sort(unique(sampleWeights))

indices <- seq(along = sampleWeights)

strata <- factor(seq(along = weights))

if (length(weights) <= nMaxStrata) {

values <- merge(

data.frame(weight = sampleWeights, index = indices),

data.frame(weight = weights, stratum = strata),

all.x = TRUE)

sampleStrata <- values[order(values$index), 'stratum']

} else

sampleStrata <- factor(rep(1, length(sampleWeights)))

return (sampleStrata)

} #--

getSampleDesign <-

function(sampleWeights,

sampleStrata)

{ #++

# Guesses which type of sample desing is used, using the following rules.

# (1) A single stratum and constant weights implies SI sampling.

# (2) More than one stratum and constant weights per stratum implies STSI

# sampling.

minmax <- sapply(split(sampleWeights, sampleStrata), range)

constantWeights <- all(minmax[1,] == minmax[2,])

nStrata <- length(levels(sampleStrata[, drop = TRUE]))

if (constantWeights) {

type <- ifelse(nStrata == 1, 'SI', 'STSI')

getVarTotal <- getSampleVarTotalSTSI

} else {

type <- ''

getVarTotal <- getSampleVarTotalPPS

}

sampleDesign <- list(

type = type,

weights = sampleWeights,

strata = sampleStrata,

getVarTotal = getVarTotal)

return (sampleDesign)

} #--

getSampleVarTotalSTSI <-

function(sampleDesign,

y)

{ #++

# return (getSampleCovTotalSTSI(sampleDesign, y, y))

getStratumVarTotal <-

function(sample)

{ #++

N <- sum(sample$weights)

n <- nrow(sample)

return (N^2 \* (1 - n / N) \* var(sample$y) / n)

} #--

sample <- data.frame(y = y, weights = sampleDesign$weights)

strataVar <- sapply(split(sample, sampleDesign$strata), getStratumVarTotal)

sampleVar <- sum(strataVar)

return (sampleVar)

} #--

getSampleVarTotalPPS <-

function(sampleDesign,

y)

{ #++

# If the sample design is neither SI nor STSI, use the formulae of the SE

# of a PPS design.

# return (getSampleCovTotalPPS(sampleDesign, y, y))

n <- length(sampleDesign$weights)

y <- y \* sampleDesign$weights

sampleVar <- sum((n \* y - sum(y))^2) / n / (n - 1)

return (sampleVar)

} #--

getVariables <-

function(formula,

leftHandSide = FALSE)

{ #++

# Returns the names of the variables used either in the left hand side of

# the formula or in the right hand side of the formula.

if (leftHandSide)

formula <- update.formula(formula, . ~ 1)

else

formula <- update.formula(formula, 1 ~ .)

variables <- all.vars(formula)

if (length(variables) == 1 && variables == '.')

variables <- NA

return (variables)

} #--

replaceRHSByVariable <-

function(formula,

variable)

{ #++

replacement <- as.formula(paste('. ~ ', variable, sep = ''))

formula <- update.formula(formula, replacement)

return (formula)

} #--

getTrace <-

function(m)

{ #++

# Returns the trace of the matrix m.

return (sum(m[col(m) == row(m)]))

} #--

weightedVar <-

function(x,

weights = rep(1, length(x)),

method = c('unbiased', 'ML'))

{ #++

# Returns the weighted variance of the vector x.

xMean <- weighted.mean(x, weights)

xVar <- sum(weights \* (x - xMean)^2)

xVar <- switch(match.arg(method),

'unbiased' = xVar / (sum(weights) - 1),

'ML' = xVar / sum(weights))

return (xVar)

} #--

'%sub%' <-

function(x,

y)

{ #++

# Returns all elements of the set operation x - y.

# > c(1, 2, 3, 4, 5) %sub% c(2, 4)

# [1] 1 3 5

return (x[! x %in% y])

} #--