Package 'baysc'

May 13, 2024

Type Package

Title Bayesian Survey Clustering

Version 1.0

Date 2023-10-24

Description An R package for running Bayesian clustering methods using survey data.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports dplyr,
e1071,
ggplot2,
gtools,
LaplacesDemon,
magrittr,
Matrix,
matrixStats,
methods,
plyr,
Rcpp (>= 1.0.11),
RcppArmadillo,
RcppTN,
rlang,
rstan (>= 2.21.8),
SimCorMultRes,
stats,
stringr,
survey (>= 4.2.1),
tidyr,
tidyselect,
truncnorm

LinkingTo BH (>= 1.66.0),
Rcpp (>= 0.12.0),
RcppArmadillo,
RcppEigen (>= 0.3.3.3.0),
RcppParallel (>= 5.0.1),
RcppTN,
rstan (>= 2.18.1),
StanHeaders (>= 2.18.0)

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.0

Depends R (>= 3.5.0)

Biarch true

SystemRequirements GNU make

Suggests testthat (>= 3.0.0),
rstantools (>= 2.3.1.1),
knitr,
rmarkdown,
devtools

Config/testthat/edition 3

VignetteBuilder knitr

R topics documented:

`create_acfplot` *Create ACF plots*

Description

Create ACF plots

Usage

```
create_acfplot(param_mcmc, param_names)
```

Arguments

`param_mcmc`	List output from `get_param_mcmc()` containing:
	`pi_mcmc` MxK dataframe of the class membership probability parameters, π, where M is the number of iterations and K is the number of latent classes.
	`theta_mcmc` Mx(JxKxR) dataframe of the item level probability parameters, θ, where J is the number of items and R is the maximum number of item levels.
	`xi_mcmc` If output for a `"swolca"` object, Mx(KxQ) dataframe of the regression parameters, ξ, where Q is the number of covariates, excluding latent class indicators, in the regression.
`param_names`	String vector of parameter names to create plots for. All names must be found as column names of the dataframes in `param_mcmc`.

Value

Creates a grid of ACF plots for the specified parameters

See Also

`get_param_mcmc()` create_traceplot

Examples

```
data(run_nhanes_swolca_results)
param_mcmc <- get_param_mcmc(res = run_nhanes_swolca_results)
# Specify selection of pi_1 to pi_5
param_names <- colnames(param_mcmc$pi_mcmc)
# Create ACF plots
create_acfplot(param_mcmc = param_mcmc, param_names = param_names)
```

`create_traceplot` *Create traceplots*

Description

Create traceplots

Usage

```
create_traceplot(param_mcmc, param_names)
```

Arguments

`param_mcmc`	List output from `get_param_mcmc()` containing: `pi_mcmc` MxK dataframe of the class membership probability parameters, π, where M is the number of iterations and K is the number of latent classes. `theta_mcmc` Mx(JxKxR) dataframe of the item level probability parameters, θ, where J is the number of items and R is the maximum number of item levels. `xi_mcmc` If output for a `"swolca"` object, Mx(KxQ) dataframe of the regression parameters, ξ, where Q is the number of covariates, excluding latent class indicators, in the regression.
`param_names`	String vector of parameter names to create plots for. All names must be found as column names of the dataframes in `param_mcmc`.

Value

Creates a grid of traceplots for the specified parameters

See Also

`get_param_mcmc()` create_acfplot

Examples

```
data(run_nhanes_swolca_results)
param_mcmc <- get_param_mcmc(res = run_nhanes_swolca_results)
# Specify selection of pi_1 to pi_5
param_names <- colnames(param_mcmc$pi_mcmc)
# Create traceplots
create_traceplot(param_mcmc = param_mcmc, param_names = param_names)
```

data_nhanes *NHANES diet and hypertension for low-income women*

Description

Cleaned dataset containing NHANES 2015-2018 data on dietary intake and hypertension among adult women aged 20 or over who are classified as low-income (reported household income at or below 185\ poverty level).

Usage

```
data(data_nhanes)
```

Format

A dataframe with 2004 rows and 71 variables:

`SEQN` 5- or 6-digit unique individual identifier

`stratum_id` Survey design stratum indicator

`cluster_id` Survey design cluster indicator, modified to be unique across strata using the formula: SDMVSTRA*10 + SDMVPSU

`sample_wt` Individual survey sampling weights

`age_cat` Categorical age variable with three categories: 1: 20-39, 2: 40-59, 3: >=60, created from the RIAGEYR variable

`racethnic` Race and Hispanic origin ethnicity with 5 categories: 1: NH White, 2: NH Black, 3: NH Asian, 4: Hispanic, 5: Other/Mixed, created from the RIDRETH3 variable

`educ` Education level with 3 categories: 1: at least some college, 2: high school/GED, 3: less than high school, created from the DMDEDUC2 variable

`smoker` Binary current smoking status, defined as 1 (yes) if having smoked 100 cigarettes in lifetime and currently smokes cigarettes, and 0 (no) otherwise, created from SMQ040 and SMQ020 variables

`physactive` Physically active as a factor with levels "Inactive" and "Active", with active defined as having at least 150 minutes of moderate or vigorous exercise per week

`BP_flag` Binary 0/1 hypertension variable, with 1 (yes) defined as having average systolic blood pressure (BP) above 130, having average diastolic BP above 80, currently taking BP medication, or having been told have high BP at least two times

`citrus` Consumption of citrus, melons, or berries with 4 categories: 1: None, 2: Low, 3: Med, 4: High

`oth_fruit` Consumption of other fruits with 4 categories: 1: None, 2: Low, 3: Med, 4: High

`fruit_juice` Consumption of fruit juices with 4 categories: 1: None, 2: Low, 3: Med, 4: High

`dark_green` Consumption of dark green vegetables with 4 categories: 1: None, 2: Low, 3: Med, 4: High

`tomatoes` Consumption of tomatoes and tomato products with 4 categories: 1: None, 2: Low, 3: Med, 4: High

`oth_red` Consumption of other red and orange vegetables with 4 categories: 1: None, 2: Low, 3: Med, 4: High

potatoes Consumption of white potatoes with 4 categories: 1: None, 2: Low, 3: Med, 4: High

oth_starchy Consumption of other starchy vegetables with 4 categories: 1: None, 2: Low, 3: Med, 4: High

oth_veg Consumption of other vegetables with 4 categories: 1: None, 2: Low, 3: Med, 4: High

whole_grain Consumption of whole grains with 4 categories: 1: None, 2: Low, 3: Med, 4: High

ref_grain Consumption of refined grains with 4 categories: 1: None, 2: Low, 3: Med, 4: High

meat Consumption of beef, veal, pork, lamb, and game meat, excluding organ and cured meat, with 4 categories: 1) None, 2) Low, 3) Med, 4) High

cured_meats Consumption of cured meats with 4 categories: 1: None, 2: Low, 3: Med, 4: High

organ Consumption of organ meat from beef, veal, pork, lamb, game, and poultry with 4 categories: 1: None, 2: Low, 3: Med, 4: High

poultry Consumption of poultry, excluding organ and cured meat, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

seafood_high Consumption of seafood high in n-3 fatty acids, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

seafood_low Consumption of seafood low in n-3 fatty acids, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

eggs Consumption of eggs and egg substitutdes, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

soybean Consumption of soy products, excluding soymilk and products made with raw soybean, with 4 categories: 1) None, 2) Low, 3) Med, 4) High

nuts Consumption of nuts and seeds with 4 categories: 1: None, 2: Low, 3: Med, 4: High

leg_protein Consumption of legumes (beans, peas, lentils) computed as protein foods, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

milk Consumption of milk and soymilk, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

yogurt Consumption of yogurt, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

cheese Consumption of cheese, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

oils Consumption of oils, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

solid_fats Consumption of solid fats, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

add_sugars Consumption of added sugars, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

drinks Consumption of alcoholic beverages and alcohol added to foods after cooking, with 4 categories: 1: None, 2: Low, 3: Med, 4: High

cycle NHANES two-year cycle, either "2015-2016" or "2017-2018"

nrecall Number of 24-hr dietary recalls completed by the individual

SDMVSTRA Original NHANES stratum indicator

SDMVPSU Original NHANES cluster indicator

RIAGENDR Original NHANES gender variable. Equal to 2 for all individuals because restricted to women.

RIDAGEYR Original NHANES age in years. Restricted to those age 20 or over.

RIDRETH3 Original NHANES race/ethnicty variable with 7 categories: 1=Mex_Amer, 2=Other_Hisp, 3=NH_White, 4=NH_Black, 6=NH_Asian, 7=Other/Mixed

DMDEDUC2 Original NHANES education level with categories: 1= <9th, 2=9-11th, 3=HS/GED, 4=Some college/AA, 5=college grad or above, 7=refused, 9=don't know

`SMQ020` Original NHANES binary variable of having smoked at least 100 cigarettes in life

`SMQ040` Original NHANES binary variable of currently smoking cigarettes

`Mins_Active` Number of minutes of moderate or vigorous exercise per week, created from NHANES variables PAQ610, PAQ615, PAQ625, PAQ630, PAQ640, PAQ645, PAQ655, PAQ660, PAQ670, PAQ675

`SBP_avg` Average systolic BP over four readings, created from NHANES variables BPXSY1, BPXSY2, BPXSY3, BPXSY4

`DBP_avg` Average diastolic BP over four readings, created from NHANES variables BPXDI1, BPXDI2, BPXDI3, BPXDI4

`BPQ030` Original NHANES binary variable of having been told have high BP at least two times

`BPQ050A` Original NHANES binary variable of currently taking prescribed medication for high BP

`BPQ020` Original NHANES binary variable of having ever been told have high BP

Source

https://github.com/smwu/SWOLCA/tree/main/Data. For more information on dataset preparation, see Wu et al. (2023)

References

Wu, S. M., Williams, M. R., Savitsky, T. D., & Stephenson, B. J. (2023). Derivation of outcome-dependent dietary patterns for low-income women obtained from survey data using a Supervised Weighted Overfitted Latent Class Analysis. arXiv preprint arXiv:2310.01575.

Examples

```
data(data_nhanes)
x_mat <- data_nhanes %>% dplyr::select(citrus:drinks)
y_all <- data_nhanes$BP_flag
```

---

get_betas_c *Obtain betas matrix for generating categorical latent class assignment*

Description

Obtain matrix of betas that can be used to generate the categorical latent class assignment variable C_i using a multinomial logistic regression where C_i may depend on a categorical covariate such as the stratum variable S_i.

Usage

```
get_betas_c(pi_mat, formula_c, V_unique)
```

Arguments

`pi_mat`	Matrix where each row is the class membership probabilities for a level of the categorical covariate. SxK, where S is the number of levels of the categorical covariate and K is the number of latent classes. Rows of `pi_mat` must sum to 1.
`formula_c`	String specifying formula for multinomial logistic regression to create category latent class assignment C_i.
`V_unique`	Dataframe with one column containing the unique values of the categorical covariate specified in `formula_c`. If `formula_c = "~1"`, set `V_unique = NULL`.

Value

Returns `beta_mat` matrix of betas to be used in a multinomial logistic regression to generate a categorical variable C_i. `beta_mat` has K rows and number of columns equal to the number of levels in the categorical covariate.

Examples

```
## Example 1: latent class C_i depends on stratum variable S_i
# Number of latent classes and number of levels of S_i
K <- 3; S <- 2
# Formula specifying that C_i depends on S_i
formula_c <- "~ s_all"
# Dataframe with unique values of S_i
V_unique <- data.frame(s_all = as.factor(1:S))
# Matrix of class membership probabilities for each level of S_i
pi_mat <- matrix(c(0.3, 0.5, 0.2,   # class membership probs for S_i=1
                   0.1, 0.6, 0.3),  # class membership probs for S_i=2
                 byrow = TRUE, nrow = S, ncol = K)
# Get matrix of betas for generating C_i
beta_mat_c <- get_betas_c(pi_mat = pi_mat, formula_c = formula_c,
                          V_unique = V_unique)
beta_mat_c

## Example 2: latent class is generated independently of other variables
# Matrix of class membership probabilities
pi_mat <- matrix(c(0.3, 0.5, 0.2), nrow = 1)
formula_c <- "~ 1"
V_unique <- NULL
get_betas_c(pi_mat = pi_mat, formula_c = formula_c, V_unique = V_unique)
```

---

get_betas_x *Obtain list of beta matrices for generating multivariate categorical exposure X_i*

Description

Obtain list of beta matrices that can be used to generate the multivariate categorical exposure variable X_i using a multinomial logistic regression where X_i depends on categorical covariate composed of latent class assignment C_i.

Usage

```
get_betas_x(profiles, R, modal_theta_prob = 0.85, formula_x, V_unique)
```

Arguments

`profiles`	Matrix where each column is a latent class pattern profile and each row is the item level for all classes. JxK, where J is the number of exposure items and K is the number of latent classes. The item levels must range from 1 to R, where R is the number of levels for all items.
`R`	Number of exposure levels. Fixed across exposure items.

`modal_theta_prob`
: Probability of true exposure level. Default is 0.85.

`formula_x`
: String specifying formula for multinomial logistic regression to create multivariate categorical exposure X_i.

`V_unique`
: Dataframe with one column containing the unique values of the categorical covariate specified in `formula_x`.

Value

Returns list `beta_list_x` of length J with each element a matrix of betas to be used in a multinomial logistic regression to generate a categorical exposure variable for that item. Each matrix of betas has R rows and number of columns equal to the number of columns in the design matrix.

See Also

`simulate_pop()`

Examples

```
## Example 1: X_i ~ C_i
# Number of items, exposure levels, latent classes
J <- 30; R <- 4; K <- 3
# Formula specifying that X_i depends on C_i
formula_x <- "~ c_all"
# Dataframe with unique values of C_i
V_unique <- data.frame(c_all = as.factor(1:K))
# Matrix of pattern profiles for each latent class
profiles <- as.matrix(data.frame(C1 = c(rep(1, times = 0.5 * J),
                                        rep(3, times = 0.5 * J)),
                                 C2 = c(rep(4, times = 0.2 * J),
                                        rep(2, times = 0.8 * J)),
                                 C3 = c(rep(3, times = 0.3 * J),
                                        rep(4, times = 0.4 * J),
                                        rep(1, times = 0.3 * J))))
# True level probability
modal_theta_prob <- 0.85
# Get matrix of betas for generating C_i
beta_list_x <- get_betas_x(profiles = profiles, R = R,
                           modal_theta_prob = modal_theta_prob,
                           formula_x = formula_x, V_unique = V_unique)
# Beta matrix for item j=1
beta_list_x[[1]]

## Example 2: X_i ~ C_i + S_i
# Update formula_x
formula_x <- "~ c_all + s_all"
# Update beta_list_x by adding in coefficients for S_i to each j matrix
beta_list_x <- lapply(1:J, function(j) cbind(beta_list_x[[j]],
                                             s_all = c(0, 0.5, 0, 0)))
beta_list_x[[1]]
```

`get_categ_probs` *Get unique categorical probabilities for multinomial logistic regresion*

Description

Get categorical probabilities for a unique design matrix corresponding to specified multinomial logistic regression formula and coefficient parameters

Usage

```
get_categ_probs(beta_mat, formula, V_unique)
```

Arguments

`beta_mat`	Matrix of coefficient parameters for the linear predictor terms of a multinomial logistic regression for generating a categorical variable. Number of rows is equal to the number of levels in the categorical variable. Number of columns is equal to the number of covariate terms in the regression.
`formula`	Formula for multinomial logistic regression. Should start with `"~ c_all"` if generating exposure X_i. All variables must be found in `V_unique`.
`V_unique`	Dataframe with containing the unique values of the variables specified in `formula`.

Value

Matrix `categ_probs` where the first columns are the values of the covariates for the multinomial logistic regression, and the remaining columns are the corresponding probability of each category of the categorical outcome variable.

Examples

```
# Get pi probabilities for C_i ~ S_i
K <- 3; S <- 2
formula_c <- "~ s_all"
V_unique <- data.frame(s_all = as.factor(1:S))
pi_mat <- matrix(c(0.3, 0.5, 0.2,   # class membership probs for S_i=1
                   0.1, 0.6, 0.3),  # class membership probs for S_i=2
                 byrow = TRUE, nrow = S, ncol = K)
beta_mat_c <- get_betas_c(pi_mat = pi_mat, formula_c = formula_c,
                          V_unique = V_unique)
categ_probs_pi <- get_categ_probs(beta_mat = beta_mat_c, formula = formula_c,
                                  V_unique = V_unique)
categ_probs_pi

# Get theta probabilities for X_i ~ C_i
J <- 30; R <- 4; K <- 3
formula_x <- "~ c_all"
V_unique <- data.frame(c_all = as.factor(1:K))
profiles <- as.matrix(data.frame(C1 = c(rep(1, times = 0.5 * J),
                                        rep(3, times = 0.5 * J)),
                                 C2 = c(rep(4, times = 0.2 * J),
                                        rep(2, times = 0.8 * J)),
                                 C3 = c(rep(3, times = 0.3 * J),
                                        rep(4, times = 0.4 * J),
```

```
                                        rep(1, times = 0.3 * J))))
modal_theta_prob <- 0.85
beta_list_x <- get_betas_x(profiles = profiles, R = R,
                           modal_theta_prob = modal_theta_prob,
                           formula_x = formula_x, V_unique = V_unique)
categ_probs_theta <- get_categ_probs(beta_mat = beta_list_x[[1]],
                                     formula = formula_x, V_unique = V_unique)
categ_probs_theta

# Get theta probabilities for X_i ~ C_i + S_i
formula_x <- "~ c_all + s_all"
beta_list_x <- lapply(1:J, function(j) cbind(beta_list_x[[j]],
                                             s_all = c(0, 0.5, 0, 0)))
V_unique <- expand.grid(c_all = as.factor(1:K), s_all = as.factor(1:S))
categ_probs_theta_s <- get_categ_probs(beta_mat = beta_list_x[[1]],
                                       formula = formula_x, V_unique = V_unique)
categ_probs_theta
```

`get_dic` *Obtain DIC-6*

Description

`get_dic` returns the DIC-6 for the model to be used as a measure of model fit.

Usage

```
get_dic(res)
```

Arguments

`res`	An object of class `"swolca"` or `"wolca"`, resulting from a call to `swolca()`, `swolca_var_adjust()`, `wolca()`, or `wolca_var_adjust()`.

Details

The deviance information criterion (DIC) can be used as a metric for model goodness of fit for mixture models (Spiegelhater et al., 2002; Celeux et al., 2006). It compares two values:

1. the median of the likelihood over all iterations
2. the likelihood calculated with the posterior median estimates of all parameters. We use the DIC-6, which increases the penalty of complexity to reduce overfitting (Stephenson et al., 2022). The DIC-6 is calculated as $3\bar{D}(\theta) - 2D(\bar{\theta})) = -6E[\log L(D|\theta)|D] + 4\log L(y|\bar{\theta})$. where $E[\cdot]$ denotes the expectation, $\bar{D}(\theta)$ is the posterior median observed deviance and $D(\bar{\theta})$ is the deviance of the posterior median.

Value

Returns the DIC-6.

References

Celeux, G., Forbes, F., Robert, C. P., Titterington, D. M. et al. (2006) Deviance information criteria for missing data models. Bayesian Analysis, 1, 651-673.

Spiegelhalter, D. J., Best, N. G., Carlin, B. P. and Van Der Linde, A. (2002) Bayesian measures of model complexity and fit. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 64, 583-639.

Stephenson, B. J., Herring, A. H., and Olshan, A. F. (2022). Derivation of maternal dietary patterns accounting for regional heterogeneity. Journal of the Royal Statistical Society Series C: Applied Statistics 71, 1957–1977.

See Also

summarize_res()

Examples

```
data(run_nhanes_swolca_results)
dic <- get_dic(res = run_nhanes_swolca_results)
```

---

get_estimates *Get posterior estimates*

Description

get_estimates obtains posterior parameter samples and estimates prior to variance adjustment

Usage

```
get_estimates(MCMC_out, post_MCMC_out, n, J, V, y_all, x_mat)
```

Arguments

MCMC_out	Output from run_MCMC_Rcpp() containing: pi_MCMC Matrix of posterior samples for pi. (n_iter)xK theta_MCMC Array of posterior samples for theta. (n_iter)xJxKxR xi_MCMC Array of posterior samples for xi. (n_iter)xKxQ c_all_MCMC Matrix of posterior samples for c_all. (n_iter)xn z_all_MCMC Matrix of posterior samples for z_all. (n_iter)xn loglik_MCMC Vector of posterior samples for log-likelihood. (n_iter)x1
post_MCMC_out	output from post_process() containing: K_med Median, across iterations, of number of classes with at least class_cutoff percent of individuals pi Matrix of reduced and relabeled posterior samples for pi. (n_iter)x(K_med) theta Array of reduced and relabeled posterior samples for theta. (n_iter)xJx(K_med)xR xi Array of reduced and relabeled posterior samples for xi. (n_iter)x(K_med)xQ dendrogram Hierarchical clustering dendrogram used for relabeling
n	Number of individuals

J	Number of exposure items
V	Regression design matrix without class assignment. nxQ
y_all	Vector of outcomes. nx1
x_mat	Matrix of multivariate categorical exposures. nxJ

Details

First, duplicate classes that have the same modal exposure categories for all items are combined to obtain the number of unique classes, K_red. Parameters are then renormalized for the unique classes and posterior median estimates are computed across MCMC iterations. Using these median estimates, class assignments c_all, the regression mean, and the individual log-likelihood are derived.

Value

Returns list estimates containing:

K_red Number of unique classes

pi_red Matrix of final posterior samples for pi. Mx(K_red), where M is the number of MCMC iterations after burn-in and thinning.

theta_red Array of final posterior samples for theta. MxJx(K_red)xR

xi_red Array of final posterior samples for xi. Mx(K_red)xQ

pi_med Vector of posterior median estimates for pi. (K_red)x1

theta_med Array of posterior median estimates for theta. Jx(K_red)xR

xi_med Matrix of posterior median estimates for xi. (K_red)xQ

Phi_med Vector of final individual outcome probabilities. nx1

c_all Vector of final individual class assignments. nx1

pred_class_probs Matrix of individual posterior class probabilities. nx(K_red)

loglik_med Vector of final indiviudal log-likehoods. nx1

References

Williams, M. R. and Savitsky, T. D. (2021). Uncertainty estimation for pseudo-bayesian inference under complex sampling. International Statistical Review 89, 72–107.

See Also

run_MCMC_Rcpp() post_process() swolca_var_adjust() swolca()

Examples

```
# Load data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
y_all <- c(data_vars$Y_data)         # Binary outcome vector, nx1
cluster_id <- data_vars$cluster_id  # Cluster indicators, nx1
sampling_wt <- data_vars$sample_wt

# Obtain dimensions
n <- dim(x_mat)[1]        # Number of individuals
```

```
J <- dim(x_mat)[2]        # Number of exposure items
R_j <- apply(x_mat, 2,    # Number of exposure categories for each item
             function(x) length(unique(x)))
R <- max(R_j)             # Maximum number of exposure categories across items
# Obtain normalized weights
kappa <- sum(sampling_wt) / n   # Weights norm. constant
w_all <- c(sampling_wt / kappa) # Weights normalized to sum to n, nx1

# Probit model only includes latent class
V_data <- as.data.frame(matrix(1, nrow = n)) # Additional regression covariates
glm_form <- "~ 1"
# Obtain probit regression design matrix without class assignment
V <- model.matrix(as.formula(glm_form), data = V_data)
# Number of regression covariates excluding class assignment
Q <- ncol(V)

# Set hyperparameters
K <- 30
alpha <- rep(1, K) / K
eta <- matrix(0.01, nrow = J, ncol = R)
for (j in 1:J) {
  eta[j, 1:R_j[j]] <- rep(1, R_j[j])
}
mu0 <- Sig0 <- vector("list", K)
for (k in 1:K) {
  # MVN(0,1) hyperprior for prior mean of xi
  mu0[[k]] <- stats::rnorm(n = Q)
  # InvGamma(3.5, 6.25) hyperprior for prior variance of xi. Assume uncorrelated
  # components and mean variance 2.5 for a weakly informative prior on xi
  Sig0[[k]] <- diag(LaplacesDemon::rinvgamma(n = Q, shape = 3.5, scale = 6.25),
  nrow = Q, ncol = Q)
}

# First initialize OLCA params
OLCA_params <- init_OLCA(K = K, n = n, J = J, R = R, alpha = alpha, eta = eta)

# Then initialize probit params
probit_params <- init_probit(K = K, n = n, Q = Q, V = V, mu0 = mu0,
Sig0 = Sig0, y_all = y_all, c_all = OLCA_params$c_all)

# Then run MCMC sampling
MCMC_out <- run_MCMC_Rcpp(OLCA_params = OLCA_params,
probit_params = probit_params, n_runs = 50, burn = 25, thin = 5,
K = K, J = J, R = R, n = n, Q = Q, w_all = w_all, x_mat = x_mat,
y_all = y_all, V = V, alpha = alpha, eta = eta, Sig0 = Sig0, mu0 = mu0)

# Then run post-process relabeling
post_MCMC_out <- post_process(MCMC_out = MCMC_out, J = J, R = R, Q = Q,
class_cutoff = 0.05)

# Then obtain posterior estimates
estimates <- get_estimates(MCMC_out = MCMC_out, post_MCMC_out = post_MCMC_out,
                           n = n, J = J, V = V, y_all = y_all, x_mat = x_mat)
```

`get_estimates_wolca` *Get posterior estimates for WOLCA*

Description

`get_estimates_wolca` obtains posterior parameter samples and estimates prior for the unsupervised WOLCA model

Usage

```
get_estimates_wolca(MCMC_out, post_MCMC_out, n, J, x_mat)
```

Arguments

`MCMC_out`	Output from `run_MCMC_Rcpp_wolca` containing: `pi_MCMC` Matrix of posterior samples for pi. (n_iter)xK `theta_MCMC` Array of posterior samples for theta. (n_iter)xJxKxR `c_all_MCMC` Matrix of posterior samples for c_all. (n_iter)xn
`post_MCMC_out`	output from `post_process_wolca` containing: `K_med` Median, across iterations, of number of classes with at least `class_cutoff` percent of individuals `pi` Matrix of reduced and relabeled posterior samples for pi. (n_iter)x(K_med) `theta` Array of reduced and relabeled posterior samples for theta. (n_iter)xJx(K_med)xR `dendrogram` Hierarchical clustering dendrogram used for relabeling
`n`	Number of individuals
`J`	Number of exposure items
`x_mat`	Matrix of multivariate categorical exposures. nxJ

Details

First, duplicate classes that have the same modal exposure categories for all items are combined to obtain the number of unique classes, `K_red`. Parameters are then renormalized for the unique classes and posterior median estimates are computed across MCMC iterations. Using these median estimates, class assignments `c_all` are derived.

Value

Returns list `estimates` containing:

`K_red` Number of unique classes

`pi_red` Matrix of final posterior samples for pi. Mx(K_red)

`theta_red` Array of final posterior samples for theta. MxJx(K_red)xR

`pi_med` Vector of posterior median estimates for pi. (K_red)x1

`theta_med` Array of posterior median estimates for theta. Jx(K_red)xR

`c_all` Vector of final individual class assignments. nx1

`pred_class_probs` Matrix of individual posterior class probabilities. nx(K_red)

See Also

get_estimates() run_MCMC_Rcpp_wolca() post_process_wolca() wolca_svyglm() wolca()

Examples

```
# Load data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
cluster_id <- data_vars$cluster_id  # Cluster indicators, nx1
sampling_wt <- data_vars$sample_wt

# Obtain dimensions
n <- dim(x_mat)[1]        # Number of individuals
J <- dim(x_mat)[2]        # Number of exposure items
R_j <- apply(x_mat, 2,    # Number of exposure categories for each item
             function(x) length(unique(x)))
R <- max(R_j)             # Maximum number of exposure categories across items
# Obtain normalized weights
kappa <- sum(sampling_wt) / n   # Weights norm. constant
w_all <- c(sampling_wt / kappa) # Weights normalized to sum to n, nx1

# Set hyperparameters for fixed sampler
K <- 3
alpha <- rep(1, K) / K
eta <- matrix(0.01, nrow = J, ncol = R)
for (j in 1:J) {
  eta[j, 1:R_j[j]] <- rep(1, R_j[j])
}

# First initialize OLCA params
OLCA_params <- init_OLCA(K = K, n = n, J = J, R = R, alpha = alpha, eta = eta)

# Then run MCMC sampling
MCMC_out <- run_MCMC_Rcpp_wolca(OLCA_params = OLCA_params, n_runs = 50,
burn = 25, thin = 5, K = K, J = J, R = R, n = n, w_all = w_all, x_mat = x_mat,
alpha = alpha, eta = eta)

# Then run post-process relabeling
post_MCMC_out <- post_process_wolca(MCMC_out = MCMC_out, J = J, R = R,
class_cutoff = 0.05)

# Then obtain posterior estimates
estimates <- get_estimates_wolca(MCMC_out = MCMC_out,
post_MCMC_out = post_MCMC_out, n = n, J = J, x_mat = x_mat)
```

---

get_param_mcmc *Obtain parameter MCMC iterations*

Description

get_param_mcmc extracts the MCMC iteration values for all parameters obtained through the MCMC sampler.

Usage

```
get_param_mcmc(res)
```

Arguments

res	An object of class "swolca" or "wolca", resulting from a call to swolca(), swolca_var_adjust(), wolca(), or wolca_var_adjust().

Value

Returns list param_mcmc containing the following:

pi_mcmc MxK dataframe of the class membership probability parameters, π, where M is the number of iterations and K is the number of latent classes.

theta_mcmc Mx(JxKxR) dataframe of the item level probability parameters, θ, where J is the number of items and R is the maximum number of item levels.

If res is a "swolca" object, param_mcmc also contains xi_mcmc, a Mx(KxQ) dataframe of the regression parameters, ξ, where Q is the number of covariates, excluding latent class indicators, in the regression.

See Also

summarize_res()

Examples

```
data(run_nhanes_swolca_results)
param_mcmc <- get_param_mcmc(res = run_nhanes_swolca_results)
```

---

get_regr_coefs *Obtains table of regression coefficients*

Description

get_regr_coefs produces a summary table of the regression coefficients, converted to standard reference cell coding.

Usage

```
get_regr_coefs(res, ci_level = 0.95, digits = 2)
```

Arguments

res	An object of class "swolca" or "wolca", resulting from a call to swolca(), swolca_var_adjust(), wolca(), or wolca_var_adjust().
ci_level	Numeric from 0 to 1 specifying the credible interval level. Default is 0.95, which gives a 95\ 2.5\ ci_level parameter in the main function.
digits	Integer indicating the number of decimal places to be used. Default is 2, which rounds to the nearest hundredth.

Details

If `res` is a `swolca` object, any latent class can be chosen as the reference class level for which to display regression coefficients. Simply run `reorder_classes()` with the desired reference level as the first class in `new_order`, and then run `get_regr_coefs()` using the reordered `res` object.

If `res` is a `wolca` object, choosing a different reference level can only be done by re-running `wolca_svyglm()` with the new ordering. To do this, run `reorder_classes()`, then run `wolca_svyglm()` to obtain regression estimates with a different reference class, and then finally use the `get_regr_coefs()` function.

Value

Returns dataframe `beta` containing the following columns:

`Covariate` Names of covariate terms.

`Estimate` Regression estimates in reference cell coding format.

`LB` Regression estimate lower bound corresponding to the level specified in `ci_level`. Calculated using posterior samples for `swolca` objects and confidence intervals for `wolca` objects.

`UB` Regression estimate upper bound corresponding to the level specified in `ci_level`. Calculated using posterior samples for `swolca` objects and confidence intervals for `wolca` objects.

`P(xi>0)` **or** `p-value` For `swolca` objects, the proportion of posterior samples greater than 0. For `wolca` objects, the p-value obtained from the `wolca_svyglm()` function

See Also

`reorder_classes()` `plot_outcome_probs()` `summarize_res()`

Examples

```
data(run_nhanes_swolca_results)
get_regr_coefs(res = run_nhanes_swolca_results, ci_level = 0.95, digits = 2)
```

`init_OLCA` *Initialize OLCA model*

Description

`init_OLCA` initializes priors and variables for an overfitted latent class analysis (OLCA) given hyperparameters.

Usage

```
init_OLCA(K, n, J, R, alpha, eta)
```

Arguments

K	Number of classes
n	Number of individuals
J	Number of exposure items
R	Maximum number of exposure categories
alpha	Kx1 vector of hyperparameters for prior for class membership probabilities π
eta	JxR matrix of hyperparameters for prior for item consumption level probabilities $\theta_{jk\cdot}$ for each item j and class k, assumed to be the same across classes.

Details

First, class membership probabilities π are initialized by drawing from a Dirichlet distribution with hyperparameter $\alpha = 1/K$ for each of the K components. Then, class assignments `c_all` are initialized for each individual by drawing from a Categorical distribution with parameter π. Finally, item level probabilities θ are initialized by drawing from a Dirichlet distribution with hyperparameter η, independently for each exposure item and latent class.

Value

Returns list `OLCA_params` containing:

`pi` Vector parameter pi for class membership probabilities. Kx1

`theta` Array parameter theta for item level probabilities. JxKxR

`c_all` Vector of random initial class assignments. nx1

See Also

`init_probit()` `swolca()` `wolca()`

Examples

```
K <- 30; n <- 4000; J <- 30; R <- 4
alpha <- rep(1, K) / K
eta <- matrix(1, nrow = J, ncol = R)
OLCA_params <- init_OLCA(K = K, n = n, J = J, R = R, alpha = alpha, eta = eta)
# OLCA_params
```

`init_probit` *Initialize probit model*

Description

`init_probit` initializes priors and variables for the probit regression model given hyperparameters.

Usage

```
init_probit(K, n, Q, V, mu0, Sig0, y_all, c_all)
```

Arguments

K	Number of classes
n	Number of individuals
Q	Number of regression covariates excluding class assignment
V	Regression design matrix without class assignment. nxQ
mu0	List of K Qx1 vectors of mean hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k.
Sig0	List of K QxQ matrices of variance hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k.
y_all	Vector of outcomes. nx1
c_all	Vector of random initial class assignments. nx1

Details

First, regression coefficients ξ are initialized by drawing independently for each latent class from a Multivariate Normal distribution with mean vector hyperparameter μ_0 drawn from a Normal(0,1) hyperprior and variance matrix hyperparameter Σ_0 set to be a diagonal matrix with diagonal components drawn from InvGamma(shape=5/2, scale=2/5) distributions. Then, the latent probit variable z_all is initialized by drawing from a Truncated Normal distribution with mean $V\xi$ and variance 1 and truncation boundary 0, where negative values are drawn if the outcome is 0 and positive values are drawn if the outcome is 1.

Value

Returns list probit_params containing:

xi Matrix parameter xi for probit regression coefficients. KxQ

z_all Vector of latent variables in the probit model. nx1

See Also

init_OLCA() swolca()

Examples

```
# Load data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
y_all <- c(data_vars$Y_data)         # Binary outcome vector, nx1
n <- dim(x_mat)[1]        # Number of individuals
J <- dim(x_mat)[2]        # Number of exposure items
R_j <- apply(x_mat, 2,    # Number of exposure categories for each item
             function(x) length(unique(x)))
R <- max(R_j)             # Maximum number of exposure categories across items

# Probit model only includes latent class
V_data <- as.data.frame(matrix(1, nrow = n)) # Additional regression covariates
glm_form <- "~ 1"
# Obtain probit regression design matrix without class assignment
V <- model.matrix(as.formula(glm_form), data = V_data)
# Number of regression covariates excluding class assignment
```

```
Q <- ncol(V)

# Set hyperparameters
K <- 30
alpha <- rep(1, K) / K
eta <- matrix(0.01, nrow = J, ncol = R)
for (j in 1:J) {
  eta[j, 1:R_j[j]] <- rep(1, R_j[j])
}
mu0 <- Sig0 <- vector("list", K)
for (k in 1:K) {
  # MVN(0,1) hyperprior for prior mean of xi
  mu0[[k]] <- stats::rnorm(n = Q)
  # InvGamma(3.5, 6.25) hyperprior for prior variance of xi. Assume uncorrelated
  # components and mean variance 2.5 for a weakly informative prior on xi
  Sig0[[k]] <- diag(LaplacesDemon::rinvgamma(n = Q, shape = 3.5, scale = 6.25),
  nrow = Q, ncol = Q)
}

# First initialize OLCA params
OLCA_params <- init_OLCA(K = K, n = n, J = J, R = R, alpha = alpha, eta = eta)

# Initialize probit model
probit_params <- init_probit(K = K, n = n, Q = Q, V = V, mu0 = mu0,
Sig0 = Sig0, y_all = y_all, c_all = OLCA_params$c_all)
# probit_params
```

`plot_class_dist` *Plot distribution of classes in the population across posterior sample iterations*

Description

`plot_class_dist` plots a boxplot of the class membership probabilities, π, in the posterior samples, for each latent class.

Usage

```
plot_class_dist(
  res,
  class_labels = NULL,
  class_title = "Dietary Pattern",
  y_title = "Class Membership Probability",
  ...
)
```

Arguments

`res`	An object of class `"swolca"` or `"wolca"`, resulting from a call to `swolca()`, `swolca_var_adjust()`, `wolca()`, or `wolca_var_adjust()`.
`class_labels`	String vector of names for the latent classes. Kx1. If `NULL` (default), numbers from 1 to K are used, where K is the final determined number of latent classes.

class_title	String specifying the title for the latent classes. Default is "Dietary Pattern".
y_title	String specifying the title for the y-axis. Default is "Class Membership Probability".
...	Additional arguments passed

Value

Returns a ggplot2 object displaying a boxplot of the distribution of class membership probabilities in the posterior samples, for each latent class.

See Also

plot_pattern_probs() plot_pattern_profiles() plot_outcome_probs()

Examples

```
data(run_nhanes_swolca_results)

# Default labels
plot_class_dist(res = run_nhanes_swolca_results)

# Specifying labels
class_labels <- paste0("Class ", 1:5)
plot_class_dist(res = run_nhanes_swolca_results, class_labels = class_labels)
```

---

plot_outcome_probs *Plot conditional probability of outcome for each latent class for one or two categorical covariates.*

Description

plot_outcome_probs plots the conditional probability of the outcome, ranging from 0 to 1 and obtained by transforming the probit regression coefficients to the probability scale, for up to two categorical covariates. Point estimates and error bars are colored by latent class.

Usage

```
plot_outcome_probs(
  res,
  cov_name,
  ci_level = 0.95,
  add_lines = FALSE,
  cov_labels = NULL,
  class_labels = NULL,
  class_title = "Dietary Pattern",
  x_title = NULL,
  y_title = "Probability of Outcome",
  ...
)
```

Arguments

res	An object of class "swolca" or "wolca", resulting from a call to swolca(), swolca_var_adjust(), wolca(), or wolca_var_adjust().
cov_name	String specifying the covariate to plot. To plot interactions between two covariates, specify a string vector with both covariate names. Covariates must be included in glm_form and V_data.
ci_level	Optional number from 0 to 1 specifying the confidence/credible interval level. Default is 0.95, which gives a 95\ composed of the 2.5\ match the ci_level parameter in the main wolca() function. Set to NULL to exclude credible intervals from the plot.
add_lines	Boolean specifying whether lines should be added to connect the points between categories. Default is FALSE.
cov_labels	Optional list of 1 or 2 string vectors specifying the corresponding category labels for the covariate(s) of interest. Must be the same length as cov_name and have the appropriate number of categories. If NULL (default), the covariate categories from the data are used.
class_labels	String vector of names for the latent classes. Kx1. If NULL (default), numbers from 1 to K are used, where K is the final determined number of latent classes.
class_title	String specifying the title for the latent classes. Default is "Dietary Pattern".
x_title	Optional string specifying x-axix label. If NULL (default), the first string in cov_nameis used.
y_title	Optional string specifying the title for the y-axis. Default is "Probability of Outcome".
...	Additional arguments passed

Value

Returns a ggplot2 object displaying the point estimates and error bars of the conditional probability of the outcome across categories of one or two covariates of interest, colored by latent class.

See Also

get_regr_coefs() plot_pattern_probs() plot_class_dist() plot_pattern_profiles()

Examples

```
data(run_nhanes_swolca_results)
res <- run_nhanes_swolca_results

# Default labels
plot_outcome_probs(res = res, cov_name = "racethnic")

# Specify labels
cov_labels <- c("NH White", "NH Black", "NH Asian", "Hispanic/Latino",
                "Other/Mixed")
class_labels <- paste0("Class", 1:5)
x_title <- "Race and Ethnicity"
plot_outcome_probs(res = res, cov_name = "racethnic", cov_labels = cov_labels,
                   class_labels = class_labels, x_title = x_title)

# Two covariates, remove error bars, add lines
```

```
age_cat_categs <- c("(20,40)", "(40,60)", ">=60")
racethnic_categs <- c("NH White", "NH Black", "NH Asian", "Hispanic/Latino",
                      "Other/Mixed")
plot_outcome_probs(res = res, cov_name = c("age_cat", "racethnic"),
                   cov_labels = list(age_cat_categs, racethnic_categs),
                   class_labels = class_labels, x_title = "Age Group",
                   ci_level = NULL, add_lines = TRUE)
## Not run:
# Multiple plots for various covariates
educ_categs <- c("Some College", "HS/GED", "<HS")
smoker_categs <- c("Non-Smoker", "Smoker")
physactive_categs <- c("Inactive", "Active")
p1 <- plot_outcome_probs(res = res, cov_name = "age_cat",
                         cov_labels = age_cat_categs,
                         class_labels = class_labels,
                         x_title = "Age Group")
p2 <- plot_outcome_probs(res = res, cov_name = "racethnic",
                         cov_labels = racethnic_categs,
                         class_labels = class_labels,
                         x_title = "Race and Ethnicity")
p3 <- plot_outcome_probs(res = res, cov_name = "smoker",
                         cov_labels = smoker_categs,
                         class_labels = class_labels,
                         x_title = "Current Smoking Status")
p4 <- plot_outcome_probs(res = res, cov_name = "physactive",
                         cov_labels = physactive_categs,
                         class_labels = class_labels,
                         x_title = "Physical Activity")
ggpubr::ggarrange(p1, p2, p3, p4, common.legend = TRUE, legend = "top",
                  nrow = 1, ncol = 4, widths = c(0.7, 1, 0.45, 0.45))

## End(Not run)
```

`plot_pattern_probs` *Plot probabilities of exposure categories, theta, for each latent class*

Description

`plot_pattern_probs` plots a grouped barplot of the probability of the exposure categories, for each exposure item and each latent class.

Usage

```
plot_pattern_probs(
  res,
  item_labels = NULL,
  categ_labels = NULL,
  categ_title = "Consumption Level",
  class_labels = NULL,
  class_title = "Dietary Pattern",
  y_title = "Consumption Level Probability",
  ...
)
```

Arguments

res	An object of class "swolca" or "wolca", resulting from a call to swolca(), swolca_var_adjust(), wolca(), or wolca_var_adjust().
item_labels	String vector of names for the exposure items. Jx1. If NULL (default), numbers from 1 to J are used.
categ_labels	String vector of names for the item categories. Rx1. If NULL (default), numbers from 1 to R are used.
categ_title	String specifying the title for the item categories. Default is "Consumption Level".
class_labels	String vector of names for the latent classes. Kx1. If NULL (default), numbers from 1 to K are used, where K is the final determined number of latent classes.
class_title	String specifying the title for the latent classes. Default is "Dietary Pattern".
y_title	String specifying the title for the y-axis. Default is "Consumption Level Probability".
...	Additional arguments passed

Value

Returns a ggplot2 object displaying a grouped barplot of the probability of the exposure categories, for each exposure item and each latent class

See Also

plot_pattern_profiles() plot_class_dist() plot_outcome_probs()

Examples

```
data(run_nhanes_swolca_results)

# Default labels
plot_pattern_probs(res = run_nhanes_swolca_results)

# Specifying labels
item_labels <- c("Citrus/Melon/Berries", "Other Fruits", "Fruit Juice",
                 "Dark Green Vegs", "Tomatoes", "Oth Red/Orange Vegs",
                 "Potatoes", "Other Starchy Vegs", "Other Vegetables",
                 "Whole Grains", "Refined Grains", "Meat", "Cured Meats",
                 "Organ Meat", "Poultry", "Seafood (High n-3)", "Seafood (Low n-3)",
                 "Eggs", "Soybean Products", "Nuts and Seeds", "Legumes (Protein)",
                 "Milk", "Yogurt", "Cheese", "Oils", "Solid Fats", "Added Sugar",
                 "Alcoholic Drinks")
categ_labels <- c("None", "Low", "Med", "High")
class_labels <- paste0("C", 1:5)
plot_pattern_probs(res = run_nhanes_swolca_results, item_labels = item_labels,
                categ_labels = categ_labels, class_labels = class_labels)
```

`plot_pattern_profiles` *Plot theta modal exposure categories for each latent class*

Description

`plot_pattern_profiles` plots a heatmap of the latent class patterns, where the patterns are defined by the category with the highest probability (i.e., the model category) for each exposure item.

Usage

```
plot_pattern_profiles(
  res,
  item_labels = NULL,
  item_title = "Item",
  categ_labels = NULL,
  categ_title = "Consumption Level",
  class_labels = NULL,
  class_title = "Dietary Pattern",
  ...
)
```

Arguments

`res`	An object of class `"swolca"` or `"wolca"`, resulting from a call to `swolca()`, `swolca_var_adjust()`, `wolca()`, or `wolca_var_adjust()`.
`item_labels`	String vector of names for the exposure items. Jx1. If `NULL` (default), numbers from 1 to J are used.
`item_title`	String specifying the title for the exposure items. Default is `"Item"`.
`categ_labels`	String vector of names for the item categories. Rx1. If `NULL` (default), numbers from 1 to R are used.
`categ_title`	String specifying the title for the item categories. Default is `"Consumption Level"`.
`class_labels`	String vector of names for the latent classes. Kx1. If `NULL` (default), numbers from 1 to K are used, where K is the final determined number of latent classes.
`class_title`	String specifying the title for the latent classes. Default is `"Dietary Pattern"`.
`...`	Additional arguments passed

Value

Returns a `ggplot2` object displaying a heatmap of the latent class patterns.

See Also

`plot_pattern_probs()` `plot_class_dist()` `plot_outcome_probs()`

Examples

```
data(run_nhanes_swolca_results)

# Default labels
plot_pattern_profiles(res = run_nhanes_swolca_results)

# Specifying labels
item_labels <- c("Citrus/Melon/Berries", "Other Fruits", "Fruit Juice",
                 "Dark Green Vegs", "Tomatoes", "Oth Red/Orange Vegs",
                 "Potatoes", "Other Starchy Vegs", "Other Vegetables",
                 "Whole Grains", "Refined Grains", "Meat", "Cured Meats",
                 "Organ Meat", "Poultry", "Seafood (High n-3)", "Seafood (Low n-3)",
                 "Eggs", "Soybean Products", "Nuts and Seeds", "Legumes (Protein)",
                 "Milk", "Yogurt", "Cheese", "Oils", "Solid Fats", "Added Sugar",
                 "Alcoholic Drinks")
categ_labels <- c("None", "Low", "Med", "High")
class_labels <- paste0("Class ", 1:5)
plot_pattern_profiles(res = run_nhanes_swolca_results, item_labels = item_labels,
                categ_labels = categ_labels, class_labels = class_labels)
```

---

plot_regr_coefs *Plot regression coefficients*

Description

`plot_regr_coefs` plots point estimates and error bars for all the regression coefficients in reference cell coding format, using the output from the `get_regr_coefs()` function. Estimates corresponding to the same latent class are displayed with the same color.

Usage

```
plot_regr_coefs(regr_coefs, res, cov_labels = NULL, ...)
```

Arguments

`regr_coefs`	Dataframe output from `get_regr_coefs()` containing at least the following columns: `Covariate` Names of covariate terms. `Estimate` Regression estimates in reference cell coding format. `LB` Regression estimate lower bound `UB` Regression estimate upper bound
`res`	An object of class `"swolca"` or `"wolca"`, resulting from a call to `swolca()`, `swolca_var_adjust()`, `wolca()`, or `wolca_var_adjust()`.
`cov_labels`	Optional string vector of labels whose order must correspond exactly with that of the `Covariate` column in `regr_coefs`. Default is `NULL` and original covariate names are used.
`...`	Additional arguments passed

Value

Returns a ggplot2 object displaying point estimates and error bars for the probit regression coefficients in reference cell coding format. Estimates and intervals are obtained from the get_regr_coefs() function using the interval level specified there.

See Also

get_regr_coefs() plot_pattern_probs() plot_class_dist() plot_pattern_profiles()

Examples

```
data(run_nhanes_swolca_results)
res <- run_nhanes_swolca_results

# Get table of regression coefficients
regr_coefs <- get_regr_coefs(res = res, ci_level = 0.95, digits = 2)

# Define new vector of covariate labels
class_dummies <- paste0("C", 2:5)
reps <- length(class_dummies)
age_dummies <- paste0("Age", c("40_60", "60"))
race_dummies <- paste0("RaceEth", c("NH_Black", "NH_Asian", "Hisp", "Other"))
cov_labels <- c("Intercept", class_dummies, age_dummies, race_dummies,
                "SmokerYes", "PhysActive",
                paste0(class_dummies, ":", rep(age_dummies, each = reps)),
                paste0(class_dummies, ":", rep(race_dummies, each = reps)),
                paste0(class_dummies, ":", rep("SmokerYes", each = reps)),
                paste0(class_dummies, ":", rep("PhysActive", each = reps)))

# Plot regression coefficient estimates and error bars
plot_regr_coefs(regr_coefs = regr_coefs, res = res, cov_labels = cov_labels)
```

---

post_process *Post-process posterior samples to relabel and reduce classes*

Description

post_process conducts post-processing to cluster individuals into a reduced number of classes and reorder posterior parameter samples according to the reduced number of classes.

Usage

```
post_process(MCMC_out, J, R, Q, class_cutoff)
```

Arguments

MCMC_out	Output from run_MCMC_Rcpp() containing: pi_MCMC Matrix of posterior samples for pi. (n_iter)xK theta_MCMC Array of posterior samples for theta. (n_iter)xJxKxR xi_MCMC Array of posterior samples for xi. (n_iter)xKxQ c_all_MCMC Matrix of posterior samples for c_all. (n_iter)xn

	`z_all_MCMC` Matrix of posterior samples for z_all. (n_iter)xn
	`loglik_MCMC` Vector of posterior samples for log-likelihood. (n_iter)x1
`J`	Number of exposure items
`R`	Maximum number of exposure categories
`Q`	Number of regression covariates excluding class assignment
`class_cutoff`	Minimum class size proportion when determining number of classes. Default is 0.05.

Details

First, `K_med`, the median number of classes with at least the `class_cutoff` proportion of individuals, is obtained over all MCMC iterations. Then, label switching is addressed through a relabeling procedure, where agglomerative clustering with Hamming distance is used to group individuals into `K_med` clusters and labels are re-assigned based on these clusters. Finally, parameter estimates are reordered using the relabeled classes so that posterior output can be averaged across MCMC iterations.

Value

Returns list `post_MCMC_out` containing:

`K_med` Median, across iterations, of number of classes with at least `class_cutoff` percent of individuals

`pi` Matrix of reduced and relabeled posterior samples for pi. (n_iter)x(K_med)

`theta` Array of reduced and relabeled posterior samples for theta. (n_iter)xJx(K_med)xR

`xi` Array of reduced and relabeled posterior samples for xi. (n_iter)x(K_med)xQ

`dendrogram` Hierarchical clustering dendrogram used for relabeling

See Also

`run_MCMC_Rcpp()` `get_estimates()` `swolca_var_adjust()` `swolca()`

Examples

```
# Load data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
y_all <- c(data_vars$Y_data)         # Binary outcome vector, nx1
cluster_id <- data_vars$cluster_id  # Cluster indicators, nx1
sampling_wt <- data_vars$sample_wt

# Obtain dimensions
n <- dim(x_mat)[1]        # Number of individuals
J <- dim(x_mat)[2]        # Number of exposure items
R_j <- apply(x_mat, 2,    # Number of exposure categories for each item
             function(x) length(unique(x)))
R <- max(R_j)             # Maximum number of exposure categories across items
# Obtain normalized weights
kappa <- sum(sampling_wt) / n   # Weights norm. constant
w_all <- c(sampling_wt / kappa) # Weights normalized to sum to n, nx1

# Probit model only includes latent class
```

```
V_data <- as.data.frame(matrix(1, nrow = n)) # Additional regression covariates
glm_form <- "~ 1"
# Obtain probit regression design matrix without class assignment
V <- model.matrix(as.formula(glm_form), data = V_data)
# Number of regression covariates excluding class assignment
Q <- ncol(V)

# Set hyperparameters
K <- 30
alpha <- rep(1, K) / K
eta <- matrix(0.01, nrow = J, ncol = R)
for (j in 1:J) {
  eta[j, 1:R_j[j]] <- rep(1, R_j[j])
}
mu0 <- Sig0 <- vector("list", K)
for (k in 1:K) {
  # MVN(0,1) hyperprior for prior mean of xi
  mu0[[k]] <- stats::rnorm(n = Q)
  # InvGamma(3.5, 6.25) hyperprior for prior variance of xi. Assume uncorrelated
  # components and mean variance 2.5 for a weakly informative prior on xi
  Sig0[[k]] <- diag(LaplacesDemon::rinvgamma(n = Q, shape = 3.5, scale = 6.25),
  nrow = Q, ncol = Q)
}

# First initialize OLCA params
OLCA_params <- init_OLCA(K = K, n = n, J = J, R = R, alpha = alpha, eta = eta)

# Then initialize probit params
probit_params <- init_probit(K = K, n = n, Q = Q, V = V, mu0 = mu0,
Sig0 = Sig0, y_all = y_all, c_all = OLCA_params$c_all)

# Then run MCMC sampling
MCMC_out <- run_MCMC_Rcpp(OLCA_params = OLCA_params,
probit_params = probit_params, n_runs = 50, burn = 25, thin = 5,
K = K, J = J, R = R, n = n, Q = Q, w_all = w_all, x_mat = x_mat,
y_all = y_all, V = V, alpha = alpha, eta = eta, Sig0 = Sig0, mu0 = mu0)

# Then run post-process relabeling
post_MCMC_out <- post_process(MCMC_out = MCMC_out, J = J, R = R, Q = Q,
class_cutoff = 0.05)
# post_MCMC_out
# plot(post_MCMC_out$dendrogram)
```

`post_process_wolca` *Post-process posterior samples to relabel and reduce classes for WOLCA*

Description

`post_process_wolca` conducts post-processing to cluster individuals into a reduced number of classes and reorder posterior parameter samples according to the reduced number of classes for the unsupervised WOLCA model.

Usage

```
post_process_wolca(MCMC_out, J, R, class_cutoff)
```

Arguments

MCMC_out	Output from run_MCMC_Rcpp_wolca containing: pi_MCMC Matrix of posterior samples for pi. (n_iter)xK theta_MCMC Array of posterior samples for theta. (n_iter)xJxKxR c_all_MCMC Matrix of posterior samples for c_all. (n_iter)xn
J	Number of exposure items
R	Maximum number of exposure categories
class_cutoff	Minimum class size proportion when determining number of classes. Default is 0.05.

Details

First, K_med, the median number of classes with at least the class_cutoff proportion of individuals, is obtained over all MCMC iterations. Then, label switching is addressed through a relabeling procedure, where agglomerative clustering with Hamming distance is used to group individuals into K_med clusters and labels are re-assigned based on these clusters. Finally, parameter estimates are reordered using the relabeled classes so that posterior output can be averaged across MCMC iterations.

Value

Returns list post_MCMC_out containing:

K_med Median, across iterations, of number of classes with at least class_cutoff percent of individuals

pi Matrix of reduced and relabeled posterior samples for pi. (n_iter)x(K_med)

theta Array of reduced and relabeled posterior samples for theta. (n_iter)xJx(K_med)xR

dendrogram Hierarchical clustering dendrogram used for relabeling

See Also

post_process() run_MCMC_Rcpp_wolca() get_estimates_wolca() wolca_svyglm() wolca()

Examples

```
# Load data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
cluster_id <- data_vars$cluster_id  # Cluster indicators, nx1
sampling_wt <- data_vars$sample_wt

# Obtain dimensions
n <- dim(x_mat)[1]        # Number of individuals
J <- dim(x_mat)[2]        # Number of exposure items
R_j <- apply(x_mat, 2,    # Number of exposure categories for each item
             function(x) length(unique(x)))
R <- max(R_j)             # Maximum number of exposure categories across items
```

```
# Obtain normalized weights
kappa <- sum(sampling_wt) / n   # Weights norm. constant
w_all <- c(sampling_wt / kappa) # Weights normalized to sum to n, nx1

# Set hyperparameters for fixed sampler
K <- 3
alpha <- rep(1, K) / K
eta <- matrix(0.01, nrow = J, ncol = R)
for (j in 1:J) {
  eta[j, 1:R_j[j]] <- rep(1, R_j[j])
}

# First initialize OLCA params
OLCA_params <- init_OLCA(K = K, n = n, J = J, R = R, alpha = alpha, eta = eta)

# Then run MCMC sampling
MCMC_out <- run_MCMC_Rcpp_wolca(OLCA_params = OLCA_params, n_runs = 50,
burn = 25, thin = 5, K = K, J = J, R = R, n = n, w_all = w_all, x_mat = x_mat,
alpha = alpha, eta = eta)

# Then run post-process relabeling
post_MCMC_out <- post_process_wolca(MCMC_out = MCMC_out, J = J, R = R,
class_cutoff = 0.05)
# post_MCMC_out
# plot(post_MCMC_out$dendrogram)
```

`reorder_classes` *Reorder classes*

Description

`reorder_classes` changes the order of the latent classes to help with plotting and to allow visualization of a different reference class for the regression coefficients.

Usage

```
reorder_classes(res, new_order)
```

Arguments

`res`	An object of class `"swolca"` or `"wolca"`, resulting from a call to `swolca()`, `swolca_var_adjust()`, `wolca()`, or `wolca_var_adjust()`.
`new_order`	Numeric vector specifying the new ordering of the classes. For example, if there are three classes and the new desired order is to have class 2 as the reference, followed by class 3 and class 1, then `new_order` would be `c(2, 3, 1)`.

Details

Latent class assignment variable `c_all` is also changed, so that if `new_order = c(2, 3, 1)`, then `c_all == 2` will become `c_all == 1`, `c_all = 3` will become `c_all = 2`, and `c_all == 1` will become `c_all = 3`.

Value

Returns object `res_new` that is identical to input `res` but has updated latent class ordering for `pi_red`, `theta_red`, `pi_med`, `theta_med`, and `c_all`. If `res` is a `"swolca"` object, `res_new` also includes updated latent class ordering for `xi_red` and `xi_med`. If `res` is a `"wolca"` object, reordering should be done prior to running `wolca_svyglm()` to obtain regression estimates with a different reference class.

See Also

`plot_outcome_probs()` `plot_class_dist()` `summarize_res()`

Examples

```
# Load NHANES data
data(run_nhanes_swolca_results)
# Reorder latent classes
res_new <- reorder_classes(res = run_nhanes_swolca_results,
                           new_order = c(3, 2, 5, 4, 1))
# Get posterior estimates for xi with class 3 as the reference
get_regr_coefs(res = res_new, ci_level = 0.95, digits = 2)
```

---

run_MCMC_Rcpp *Run MCMC to get posterior samples*

Description

`run_MCMC_Rcpp` runs the Gibbs sampler MCMC algorithm using Rcpp to obtain posterior samples.

Usage

```
run_MCMC_Rcpp(
  OLCA_params,
  probit_params,
  n_runs,
  burn,
  thin,
  K,
  J,
  R,
  n,
  Q,
  w_all,
  x_mat,
  y_all,
  V,
  alpha,
  eta,
  mu0,
  Sig0,
  update = 10
)
```

Arguments

OLCA_params	Output list from `init_OLCA()` containing: `pi` Vector parameter pi for class membership probabilities. Kx1 `theta` Array parameter theta for item level probabilities. JxKxR `c_all` Vector of random initial class assignments. nx1
probit_params	Output list from `init_probit()` containing: `xi` Matrix parameter xi for probit regression coefficients. KxQ `z_all` Vector of latent variables in the probit model. nx1
n_runs	Number of MCMC iterations. Default is 20000.
burn	Number of MCMC iterations to drop as a burn-in period. Default is 10000.
thin	Thinning factor for MCMC iterations. Default is 5.
K	Number of classes
J	Number of exposure items
R	Maximum number of exposure categories
n	Number of individuals
Q	Number of regression covariates excluding class assignment
w_all	Weights normalized to sum to n. nx1
x_mat	Matrix of multivariate categorical exposures. nxJ
y_all	Vector of outcomes. nx1
V	Regression design matrix without class assignment. nxQ
alpha	Kx1 vector of hyperparameters for prior for class membership probabilities π
eta	JxR matrix of hyperparameters for prior for item consumption level probabilities $\theta_{jk\cdot}$ for each item j and class k, assumed to be the same across classes.
mu0	List of K Qx1 vectors of mean hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k.
Sig0	List of K QxQ matrices of variance hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k.
update	Number specifying that MCMC progress updates should be printed every `update` iterations. Default is 10.

Details

A Gibbs sampler updates the parameters and variables in the following order: π, `c_all`, θ, ξ, `z_all`. Class assignments are permuted every 10 iterations to encourage mixing, according to a random permutation sampler (Fruhwirth-Schnatter, 2001).

Value

Returns list `MCMC_out` containing:

`pi_MCMC` Matrix of posterior samples for pi. (n_iter)xK

`theta_MCMC` Array of posterior samples for theta. (n_iter)xJxKxR

`xi_MCMC` Array of posterior samples for xi. (n_iter)xKxQ

`c_all_MCMC` Matrix of posterior samples for c_all. (n_iter)xn

`z_all_MCMC` Matrix of posterior samples for z_all. (n_iter)xn

`loglik_MCMC` Vector of posterior samples for log-likelihood. (n_iter)x1

References

Fruhwirth-Schnatter, S. (2001). Markov chain monte carlo estimation of classical and dynamic switching and mixture models. Journal of the American Statistical Association 96, 194–209.

See Also

post_process() get_estimates() swolca_var_adjust() swolca() run_MCMC_Rcpp_wolca()

Examples

```
# Load data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
y_all <- c(data_vars$Y_data)         # Binary outcome vector, nx1
cluster_id <- data_vars$cluster_id  # Cluster indicators, nx1
sampling_wt <- data_vars$sample_wt

# Obtain dimensions
n <- dim(x_mat)[1]        # Number of individuals
J <- dim(x_mat)[2]        # Number of exposure items
R_j <- apply(x_mat, 2,    # Number of exposure categories for each item
             function(x) length(unique(x)))
R <- max(R_j)             # Maximum number of exposure categories across items
# Obtain normalized weights
kappa <- sum(sampling_wt) / n   # Weights norm. constant
w_all <- c(sampling_wt / kappa) # Weights normalized to sum to n, nx1

# Probit model only includes latent class
V_data <- as.data.frame(matrix(1, nrow = n)) # Additional regression covariates
glm_form <- "~ 1"
# Obtain probit regression design matrix without class assignment
V <- model.matrix(as.formula(glm_form), data = V_data)
# Number of regression covariates excluding class assignment
Q <- ncol(V)

# Set hyperparameters
K <- 30
alpha <- rep(1, K) / K
eta <- matrix(0.01, nrow = J, ncol = R)
for (j in 1:J) {
  eta[j, 1:R_j[j]] <- rep(1, R_j[j])
}
mu0 <- Sig0 <- vector("list", K)
for (k in 1:K) {
  # MVN(0,1) hyperprior for prior mean of xi
  mu0[[k]] <- stats::rnorm(n = Q)
  # InvGamma(3.5, 6.25) hyperprior for prior variance of xi. Assume uncorrelated
  # components and mean variance 2.5 for a weakly informative prior on xi
  Sig0[[k]] <- diag(LaplacesDemon::rinvgamma(n = Q, shape = 3.5, scale = 6.25),
  nrow = Q, ncol = Q)
}

# First initialize OLCA params
OLCA_params <- init_OLCA(K = K, n = n, J = J, R = R, alpha = alpha, eta = eta)
```

```
# Then initialize probit params
probit_params <- init_probit(K = K, n = n, Q = Q, V = V, mu0 = mu0,
Sig0 = Sig0, y_all = y_all, c_all = OLCA_params$c_all)

# Then run MCMC sampling
MCMC_out <- run_MCMC_Rcpp(OLCA_params = OLCA_params,
probit_params = probit_params, n_runs = 50, burn = 25, thin = 5,
K = K, J = J, R = R, n = n, Q = Q, w_all = w_all, x_mat = x_mat,
y_all = y_all, V = V, alpha = alpha, eta = eta, Sig0 = Sig0, mu0 = mu0)
# MCMC_out
```

`run_MCMC_Rcpp_wolca` *Run MCMC to get posterior samples for WOLCA*

Description

`run_MCMC_Rcpp_wolca` runs the Gibbs sampler MCMC algorithm using Rcpp to obtain posterior samples for the two-step unsupervised WOLCA model.

Usage

```
run_MCMC_Rcpp_wolca(
  OLCA_params,
  n_runs,
  burn,
  thin,
  K,
  J,
  R,
  n,
  w_all,
  x_mat,
  alpha,
  eta,
  update = 10
)
```

Arguments

`OLCA_params`	Output list from `init_OLCA()` containing: `pi` Vector parameter pi for class membership probabilities. Kx1 `theta` Array parameter theta for item level probabilities. JxKxR `c_all` Vector of random initial class assignments. nx1
`n_runs`	Number of MCMC iterations. Default is 20000.
`burn`	Number of MCMC iterations to drop as a burn-in period. Default is 10000.
`thin`	Thinning factor for MCMC iterations. Default is 5.
`K`	Number of classes
`J`	Number of exposure items

R	Maximum number of exposure categories
n	Number of individuals
w_all	Weights normalized to sum to n. nx1
x_mat	Matrix of multivariate categorical exposures. nxJ
alpha	Kx1 vector of hyperparameters for prior for class membership probabilities π
eta	JxR matrix of hyperparameters for prior for item consumption level probabilities $\theta_{jk\cdot}$ for each item j and class k, assumed to be the same across classes.
update	Number specifying that MCMC progress updates should be printed every update iterations. Default is 10.

Details

A Gibbs sampler updates the parameters and variables in the following order: π, c_all, θ. Class assignments are permuted every 10 iterations to encourage mixing, according to a random permutation sampler.

Value

Returns list MCMC_out containing:

pi_MCMC Matrix of posterior samples for pi. (n_iter)xK

theta_MCMC Array of posterior samples for theta. (n_iter)xJxKxR

c_all_MCMC Matrix of posterior samples for c_all. (n_iter)xn

See Also

run_MCMC_Rcpp() post_process_wolca() get_estimates_wolca() wolca_svyglm() wolca()

Examples

```
# Load data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
cluster_id <- data_vars$cluster_id   # Cluster indicators, nx1
sampling_wt <- data_vars$sample_wt

# Obtain dimensions
n <- dim(x_mat)[1]        # Number of individuals
J <- dim(x_mat)[2]        # Number of exposure items
R_j <- apply(x_mat, 2,    # Number of exposure categories for each item
             function(x) length(unique(x)))
R <- max(R_j)             # Maximum number of exposure categories across items
# Obtain normalized weights
kappa <- sum(sampling_wt) / n   # Weights norm. constant
w_all <- c(sampling_wt / kappa) # Weights normalized to sum to n, nx1

# Set hyperparameters
K <- 30
alpha <- rep(1, K) / K
eta <- matrix(0.01, nrow = J, ncol = R)
for (j in 1:J) {
  eta[j, 1:R_j[j]] <- rep(1, R_j[j])
```

```
}

# First initialize OLCA params
OLCA_params <- init_OLCA(K = K, n = n, J = J, R = R, alpha = alpha, eta = eta)

# Then run MCMC sampling
MCMC_out <- run_MCMC_Rcpp_wolca(OLCA_params = OLCA_params, n_runs = 50,
burn = 25, thin = 5, K = K, J = J, R = R, n = n, w_all = w_all, x_mat = x_mat,
alpha = alpha, eta = eta)
# MCMC_out
```

`run_nhanes_swolca_results` *NHANES SWOLCA results*

Description

Results from running `swolca()` on the `data_nhanes` dataset

Usage

```
data(run_nhanes_swolca_results)
```

Format

List `res` containing:

`estimates` List of unadjusted posterior model results

`runtime` Total runtime for model

`data_vars` List of data variables used

`MCMC_out` List of full MCMC output

`post_MCMC_out` List of MCMC output after relabeling

`K_fixed` Number of classes used for the fixed sampler

`estimates_adjust` List of adjusted posterior model results with correct uncertainty estimation

Source

Result from running `swolca()` on `data_nhanes`

Examples

```
data(run_nhanes_swolca_results)
```

simulate_pop *Create and save simulated population data*

Description

simulate_pop creates and saves simulated population data according to input specifications

Usage

```
simulate_pop(
  N = 80000,
  S = 2,
  J = 30,
  K = 3,
  R = 4,
  N_s = c(60000, 20000),
  modal_theta_prob = 0.85,
  formula_c = "~ s_all",
  formula_x = "~ c_all",
  formula_y = "~ c_all * s_all",
  beta_mat_c = NULL,
  beta_list_x = NULL,
  beta_vec_y = NULL,
  xi_mat_y = NULL,
  V_additional = NULL,
  cluster_size = 80,
  pop_seed = 1,
  save_res = TRUE,
  save_path = NULL
)
```

Arguments

N	Population size. Default is 80000.
S	Number of subpopulations. Default is 2.
J	Number of exposure items. Default is 30.
K	Number of latent classes. Default is 3.
R	Number of exposure categories for all items. Default is 4.
N_s	Population size for each level of categorical variable S_i. Default is c(60000, 20000), corresponding to a binary S_i with S=2 levels.
modal_theta_prob	Probability between 0 and 1 for the most likely exposure category, assumed to be the same for all items. For all other categories, 1 - modal_theta_prob is evenly split across them. Default is 0.85, so if there are R=4 exposure categories, then the 3 non-modal categories occur with probability $(1 - 0.85)/3 = 0.05$
formula_c	String specifying formula for multinomial logistic regression to create category latent class assignment C_i. Default is "~ s_all", which generates C_i dependent on stratum variable S_i. All variables in the formula must be "c_all", "s_all", or specified in V_additional.

formula_x	String specifying formula for multinomial logistic regression to create multivariate categorical exposure X_i. Default is "~ c_all", which generates X_i dependent on latent class C_i. All variables in the formula must be "c_all", "s_all", or specified in V_additional.
formula_y	String specifying formula for logistic regression to create binary outcome Y_i. Default is "~ c_all * s_all", which generates Y_i dependent on latent class C_i, stratum S_i, and their interaction. All variables in the formula must be "c_all", "s_all", or specified in V_additional.
beta_mat_c	Coefficient parameters for a multinomial logistic regression to generate latent class assignment C_i. Kx(Q_full), where K is the number of categories and Q_full is the number of covariate terms in the regression. Default is NULL and default values are used. See details.
beta_list_x	Coefficient parameters for a multinomial logistic regression to generate multivariate exposure X_i. List of J matrices, each of dimension Kx(Q_full). Default is NULL and default values are used. See details.
beta_vec_y	Coefficient parameters for a logistic regression to generate binary outcome Y_i. (Q_full)x1. Default is NULL and default values are used. See details.
xi_mat_y	Alternative to xi_mat_y but in mixture reference coding. Default is NULL. If specified, must have K rows and Q columns, where Q is the number of covariate terms in the regression, excluding all terms involving C_i.
V_additional	Dataframe with any additional variables other than C_i and S_i to be used to generate variables. Number of rows should be N. Default is NULL.
cluster_size	Size of clusters for clustering in the outcome, Y_i, assumed to be the same for all clusters. Default is 80, corresponding to 1000 clusters if N is 80000, assuming exchangeable latent correlation matrix with correlation 0.5 on the off-diagonals.
pop_seed	Numeric seed for population generation. Default is 1.
save_res	Boolean specifying if results should be saved. Default = TRUE.
save_path	String specifying directory and file name to save results, e.g., "~/Documents/stratified_classes". Default is NULL.

Details

If NULL (default) is used for beta_mat_c, beta_list_x, or beta_mat_y, the following default values are used, corresponding to the scenario with K=3 latent class and S=2 levels for variable S_i: beta_mat_c is a 3x2 matrix with values c(0, 0, 0.5, 1.3, -0.4, 1.5) by row; beta_list_x is a list of J=30 4x4 matrices as in Example 1 provided below; and beta_vec_y is a vector of length $3 * 2 = 6$ with values c(1, -0.7, -1.5, -0.5, -0.5, -0.3). The generation of default values is demonstrated in Example 1 provided below.

To save the simulated population, set save_res = TRUE (default) and save_path to a string that specifies both the location and the beginning of the file name (e.g., "~/Documents/stratified_classes"). The file name will have "sim_pop.RData" appended to it.

Value

Returns list sim_pop containing:

N Population size

J Number of exposure items

R Number of exposure categories

`S` Number of stratum (i.e., levels of S_i)

`N_s` Vector of population sizes for levels of S_i. Sx1

`true_K` True number of latent classes, K

`true_Ai` `NULL` or vector of additional continuous variable if `a_all` is provided in `V_additional`. Nx1

`true_Bi` `NULL` or vector of additional binary variable if `b_all` is provided in `V_additional`. Nx1

`true_Si` Vector of true stratum indicators. Nx1

`true_Ci` Vector of true latent class indicators. Nx1

`true_pi` Vector of true pi values overall in the population. Kx1

`true_pi_s` `NULL` or SxK matrix of true pi values within each level of S_i

`X_data` Matrix of multivariate categorical exposure for all individuals. NxJ

`true_global_patterns` Matrix of true global exposure patterns defined by modal category. JxK

`true_global_thetas` Array of true thetas. JxKxR

`Y_data` Vector of binary outcome for all individuals. Nx1

`cluster_id` Vector of true cluster indicators. Nx1

`cluster_size` Cluster size

`true_xi` Matrix of probit regression coefficients in mixture reference coding. KxQ, where Q is the number of covariate terms in the regression, excluding all terms involving C_i

`true_Phi` Vector of true outcome probabilities for all individuals. Nx1

`true_Phi_mat` `NULL` or matrix of true outcome probabilities for individuals aggregated by C_i and S_i. KxS

If `save_res` = `TRUE` (default), also saves `sim_pop` as `[save_path]_sim_pop.RData`.

See Also

`simulate_samp()` `create_categ_var()` `get_betas_x()`

Examples

```
## Not run:
### Example 1: Default values
sim_pop <- simulate_pop(save_res = FALSE)

### Example 2: Similar to default but smaller population size
# Population size and strata dimensions
N = 800; S = 2; N_s = c(600, 200)

# Generate C_i ~ S_i
K <- 3
formula_c <- "~ s_all"
V_unique <- data.frame(s_all = as.factor(1:S))
pi_mat <- matrix(c(0.3, 0.5, 0.2,
                   0.1, 0.6, 0.3),
                 byrow = TRUE, nrow = S, ncol = K)
beta_mat_c <- get_betas_c(pi_mat = pi_mat, formula_c = formula_c,
                          V_unique = V_unique)

# Generate X_i ~ C_i
J <- 30; R <- 4
```

```
formula_x <- "~ c_all"
V_unique <- data.frame(c_all = as.factor(1:K))
profiles <- as.matrix(data.frame(C1 = c(rep(1, times = 0.5 * J),
                                        rep(3, times = 0.5 * J)),
                                 C2 = c(rep(4, times = 0.2 * J),
                                        rep(2, times = 0.8 * J)),
                                 C3 = c(rep(3, times = 0.3 * J),
                                        rep(4, times = 0.4 * J),
                                        rep(1, times = 0.3 * J))))
modal_theta_prob <- 0.85
beta_list_x <- get_betas_x(profiles = profiles, R = R,
                           modal_theta_prob = modal_theta_prob,
                           formula_x = formula_x, V_unique = V_unique)

# Generate Y_i ~ C_i + S_i + C_i:S_i
formula_y <- "~ c_all * s_all"
beta_vec_y <- c(1, -0.7, -1.5, -0.5, -0.5, -0.3)
cluster_size <- 80

# Simulate population
pop_seed <- 1  # Set seed
sim_pop <- simulate_pop(N = N, J = J, K = K, R = R, N_s = N_s,
                        modal_theta_prob = modal_theta_prob,
                        formula_c = formula_c, formula_x = formula_x,
                        formula_y = formula_y, beta_mat_c = beta_mat_c,
                        beta_list_x = beta_list_x, beta_vec_y = beta_vec_y,
                        cluster_size = cluster_size,
                        pop_seed = pop_seed, save_res = FALSE)

### Example 2: Selection-dependent pattern profiles
# Generate X_i ~ C_i + S_i
 formula_x <- "~ c_all + s_all"
 beta_list_x <- lapply(1:J, function(j) cbind(beta_list_x[[j]],
                                              s_all = c(0, 0.5, 0, 0)))

# Create population
sim_pop <- simulate_pop(N = N, S=S, J = J, K = K, R = R, N_s = N_s,
                        modal_theta_prob = modal_theta_prob,
                        formula_c = formula_c, formula_x = formula_x,
                        formula_y = formula_y, beta_mat_c = beta_mat_c,
                        beta_list_x = beta_list_x, beta_vec_y = beta_vec_y,
                        cluster_size = cluster_size,
                        pop_seed = pop_seed, save_res = FALSE)

### Example 3: Additional effect modifiers for Y_i and no clustering
# Continuous variable A_i for age centered about 0
a_all <- stats::rnorm(n = N, mean = 0, sd = 5)
# Binary variable B_i for physically inactive or active
b_all <- as.factor(stats::rbinom(n = N, size = 1, prob = 0.3) + 1)
# Create dataframe of additional variables A_i and B_i
V_additional <- data.frame(a_all, b_all)

# Generate Y_i ~ C_i + S_i + A_i + B_i + C_i:S_i + C_i:A_i + C_i:B_i
formula_y <- "~ c_all * (s_all + a_all + b_all)"
beta_vec_y <- c(1, -0.7, -1.5, -0.5, -0.5, -0.3, -0.04, 0.09, 0.08, 0.4,
                -0.7, -0.6)
cluster_size <- 1
```

```
# Create population
sim_pop <- simulate_pop(N = N, S=S, J = J, K = K, R = R, N_s = N_s,
                        modal_theta_prob = modal_theta_prob,
                        formula_c = formula_c, formula_x = formula_x,
                        formula_y = formula_y, beta_mat_c = beta_mat_c,
                        beta_list_x = beta_list_x, beta_vec_y = beta_vec_y,
                        V_additional = V_additional, cluster_size = cluster_size,
                        pop_seed = pop_seed, save_res = FALSE)

## End(Not run)
```

`simulate_samp` *Create and save simulated sample data*

Description

`simulate_samp` creates and saves simulated sample data by sampling from the simulated population according to input specifications.

Usage

```
simulate_samp(
  sim_pop,
  samp_prop = 0.05,
  samp_size = NULL,
  strat = TRUE,
  strat_dist = c(0.5, 0.5),
  clust = TRUE,
  samp_seed = 101,
  save_res = TRUE,
  save_path
)
```

Arguments

`sim_pop`	Simulated population output from `sim_pop()` function
`samp_prop`	Proportion of population to sample. Default is 0.05. Either `samp_prop` or `samp_size` must be specified.
`samp_size`	Sample size. Default is `NULL`. Either `samp_prop` or `samp_size` must be specified.
`strat`	Boolean specifying whether to perform stratification by S_i. Default is `TRUE`.
`strat_dist`	Vector of relative stratum sizes in the sample. Components must be proportions that sum to 1. Default is `c(0.5, 0.5)`, which results in equal stratum sizes in the sample.
`clust`	Boolean specifying whether to perform cluster sampling. Default is `TRUE`.
`samp_seed`	Numeric seed for population generation. Default is 1.
`save_res`	Boolean specifying if results should be saved. Default = `TRUE`.
`save_path`	String specifying directory and file name to save results, e.g., "~/Documents/stratified_classes". Default is `NULL`.

Value

Returns list `sim_samp` containing:

`samp_ind` Vector of population indices for sampled individuals. nx1

`sample_wt` Vector of sampling weights for sampled individuals. nx1

`N` Population size

`J` Number of exposure items

`R` Number of exposure categories

`S` Number of stratum (i.e., levels of S_i)

`N_s` Vector of population sizes for levels of S_i. Sx1

`true_K` True number of latent classes, K

`true_Ai` `NULL` or vector of additional continuous variable for sampled individuals. nx1

`true_Bi` `NULL` or vector of additional binary variable for sampled individuals. nx1

`true_Si` Vector of true stratum indicators for sampled individuals. nx1

`true_Ci` Vector of true latent class indicators for sampled individuals. nx1

`true_pi` Vector of true pi values overall in the population. Kx1

`true_pi_s` `NULL` or SxK matrix of true pi values within each level of S_i

`X_data` Matrix of multivariate categorical exposure for sampled individuals. nxJ

`true_global_patterns` Matrix of true global exposure patterns defined by modal category. JxK

`true_global_thetas` Array of true thetas. JxKxR

`Y_data` Vector of binary outcome for sampled individuals. nx1

`cluster_id` Vector of true cluster indicators for sampled individuals. nx1

`cluster_size` Cluster size

`true_xi` Matrix of probit regression coefficients in mixture reference coding. KxQ, where Q is the number of covariate terms in the regression, excluding all terms involving C_i

`true_Phi` Vector of true outcome probabilities for sampled individuals. nx1

`true_Phi_mat` `NULL` or matrix of true outcome probabilities for individuals aggregated by C_i and S_i. KxS

If `save_res` = `TRUE` (default), also saves `sim_samp` as `[save_path]_sim_samp.RData`.

See Also

`simulate_pop()`

Examples

```
## Not run:
### Example 1: Default values
# Create population
sim_pop <- simulate_pop(save_res = FALSE)
sim_samp <- simulate_samp(sim_pop = sim_pop, samp_prop = 0.05, strat = TRUE,
                          strat_dist = c(0.5, 0.5), clust = TRUE,
                          samp_seed = 101, save_res = FALSE)

## End(Not run)
```

`sim_data` *Simulated data*

Description

A dataset simulated using `simulate_pop()` and `simulate_samp()`.

Usage

```
data(sim_data)
```

Format

A list with 23 elements:

`samp_ind` Vector of population indices for sampled individuals. nx1, where n=4000.

`sample_wt` Vector of sampling weights for sampled individuals. nx1

`N` Population size

`J` Number of exposure items

`R` Number of exposure categories

`H` Number of stratum (i.e., levels of S)

`N_s` Vector of population sizes for levels of S. Hx1, where H = 2.

`true_K` True number of latent classes, K

`true_Ai` `NULL` or vector of additional continuous variable for sampled individuals. nx1

`true_Bi` `NULL` or vector of additional binary variable for sampled individuals. nx1

`true_Si` Vector of true stratum indicators for sampled individuals. nx1

`true_Ci` Vector of true latent class indicators for sampled individuals. nx1

`true_pi` Vector of true pi values overall in the population. Kx1, where K = 3.

`true_pi_s` `NULL` or HxK matrix of true pi values within each level of S

`X_data` Matrix of multivariate categorical exposure for sampled individuals. nxJ, where J = 30.

`true_global_patterns` Matrix of true global exposure patterns defined by modal category. JxK

`true_global_thetas` Array of true thetas. JxKxR, where R = 4.

`Y_data` Vector of binary outcome for sampled individuals. nx1

`cluster_id` Vector of true cluster indicators for sampled individuals. nx1

`cluster_size` Cluster size

`true_xi` Matrix of probit regression coefficients in mixture reference coding. Kxq, where q is the number of covariate terms in the regression, excluding all terms involving C; q = 2 for this dataset.

`true_Phi` Vector of true outcome probabilities for sampled individuals. nx1

`true_Phi_mat` `NULL` or matrix of true outcome probabilities for individuals aggregated by C and S. KxH

Source

Simulated data using `simulate_pop()` and `simulate_samp()`.

Examples

```
data(sim_data)
x_mat <- sim_data$X_data
```

`summarize_res` *Obtains table of regression coefficients*

Description

`summarize_res` produces a summary table of the regression coefficients, converted to standard reference cell coding.

Usage

```
summarize_res(res, ci_level = 0.95, digits = 2)
```

Arguments

`res`	An object of class `"swolca"` or `"wolca"`, resulting from a call to `swolca()`, `swolca_var_adjust()`, `wolca()`, or `wolca_var_adjust()`.
`ci_level`	Numeric from 0 to 1 specifying the credible interval level. Default is 0.95, which gives a 95\ 2.5\ `ci_level` parameter in the main function.
`digits`	Integer indicating the number of decimal places to be used. Default is 2, which rounds to the nearest hundredth.

Value

Returns dataframe `estimates_df` of the coefficients estimates and their lower and upper bounds using the `ci_level` specified, rounded to the number of digits specified in `digits`. π (class membership probability) parameters are listed first, followed by θ (item level probability) parameters and ξ (regression coefficient) parameters.

See Also

`get_regr_coefs()`

Examples

```
data(run_nhanes_swolca_results)
estimates_df <- summarize_res(res = run_nhanes_swolca_results, ci_level = 0.95,
                              digits = 2)
```

`swolca` *Run the SWOLCA model*

Description

`swolca` runs a supervised weighted overfitted latent class analysis (SWOLCA) described in Wu et al. (2023) and saves and returns the results. For proper variance estimation, please run `swolca_var_adjust()` after.

Usage

```
swolca(
  x_mat,
  y_all,
  sampling_wt = NULL,
  cluster_id = NULL,
  stratum_id = NULL,
  V_data = NULL,
  glm_form,
  run_sampler = "both",
  K_max = 30,
  adapt_seed = NULL,
  K_fixed = NULL,
  fixed_seed = NULL,
  class_cutoff = 0.05,
  n_runs = 20000,
  burn = 10000,
  thin = 5,
  update = 10,
  save_res = TRUE,
  save_path = NULL,
  alpha_adapt = NULL,
  eta_adapt = NULL,
  mu0_adapt = NULL,
  Sig0_adapt = NULL,
  alpha_fixed = NULL,
  eta_fixed = NULL,
  mu0_fixed = NULL,
  Sig0_fixed = NULL
)
```

Arguments

`x_mat`	Matrix of multivariate categorical exposures. nxJ
`y_all`	Vector of binary outcomes. nx1
`sampling_wt`	Vector of survey sampling weights. nx1. Default is `NULL`, indicating no sampling weights and setting all weights to 1.
`cluster_id`	Vector of individual cluster IDs. nx1. Default is `NULL`, indicating each individual is their own cluster.

stratum_id	Vector of individual stratum IDs. nx1. Default is NULL, indicating no stratification.
V_data	Dataframe of additional regression covariates. nxQ. Factor covariates must be converted to factors. If NULL (default), no additional covariates are to be included. All variables in glm_form must be found in V_data.
glm_form	String specifying formula for probit regression, excluding outcome and latent class. For example, "~ 1" for the model with only latent class as covariates. All variables in glm_form must be found in V_data. Do not specify interaction terms for latent class by additional covariates, as these terms are already included.
run_sampler	String specifying which sampler(s) should be run. Must be one of "both" (default), "fixed", or "adapt". See Details.
K_max	Upper limit for number of classes. Default is 30.
adapt_seed	Numeric seed for adaptive sampler. Default is NULL.
K_fixed	True number of classes, if known. Default is NULL, as it is not necessary for the adaptive sampler. If bypassing the adaptive sampler and running the fixed sampler directly, need to specify a value here. See Details.
fixed_seed	Numeric seed for fixed sampler. Default is NULL.
class_cutoff	Minimum class size proportion when determining number of classes. Default is 0.05.
n_runs	Number of MCMC iterations. Default is 20000.
burn	Number of MCMC iterations to drop as a burn-in period. Default is 10000.
thin	Thinning factor for MCMC iterations. Default is 5.
update	Number specifying that MCMC progress updates should be printed every update iterations. Default is 10.
save_res	Boolean specifying if results should be saved. Default = TRUE.
save_path	String specifying directory and file name to save results, e.g., "~/Documents/run". Default is NULL.
alpha_adapt	Adaptive sampler hyperparameter for prior for class membership probabilities π. Default is NULL and default values are used (see Details). If specified, must be (K_max)x1.
eta_adapt	Adaptive sampler hyperparameter for prior for item level probabilities $\theta_{jk\cdot}$ for each item j and class k, assumed to be the same across classes. Default is NULL and default values are used (see Details). If specified, must be JxR, where J is the number of exposure items and R is the maximum number of categories for the exposure.
mu0_adapt	Adaptive sampler mean hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k. Default is NULL and default values are used (see Details). If specified, must be a list of K_max vectors of dimension Qx1, where Q is the number of regression covariates excluding latent class assignment.
Sig0_adapt	Adaptive sampler variance hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k. Default is NULL and default values are used (see Details). If specified, must be a list of K_max QxQ matrices, where Q is the number of regression covariates excluding latent class assignment.
alpha_fixed	Fixed sampler hyperparameter for prior for π. Default is NULL and default values are used (see Details). If specified, must be (K_fixed)x1.

`eta_fixed` Fixed sampler hyperparameter for prior for item level probabilities $\theta_{jk\cdot}$ for each item j and class k, assumed to be the same across classes. Default is `NULL` and default values are used (see Details). If specified, must be JxR, where J is the number of exposure items and R is the maximum number of categories for the exposure.

`mu0_fixed` Fixed sampler mean hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k. Default is `NULL` and default values are used (see Details). If specified, must be a list of `K_fixed` vectors of dimension Qx1, where Q is the number of regression covariates excluding latent class assignment.

`Sig0_fixed` Fixed sampler variance hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k. Default is `NULL` and default values are used (see Details). If specified, must be a list of `K_fixed` QxQ matrices, where Q is the number of regression covariates excluding latent class assignment.

Details

If no survey sample adjustments are desired, leave `sampling_wt`, `stratum_id`, and `cluster_id` to their default `NULL` values.

By default, the function will run two samplers: the adaptive sampler, followed by the fixed sampler. The adaptive sampler determines the number of latent classes, which is then used in the fixed sampler for parameter estimation. If the number of latent classes is already known and only the fixed sampler needs to be run, specify `"fixed"` for the `run_sampler` argument and specify a number for `K_fixed`. If only the adaptive sampler is to be run, specify `"adapt"` for the `run_sampler` argument. Use `adapt_seed` (default is `NULL`) to specify a seed for the adaptive sampler, and use `fixed_seed` (default is `NULL`) to specify a separate seed for the fixed sampler.

To run the post-processing variance adjustment to prevent underestimation of posterior intervals, run the `swolca_var_adjust()` function after running `swolca()` (see Examples).

`x_mat` is an nxJ matrix with each row corresponding to the J-dimensional categorical exposure for an individual. If there is no clustering present, `cluster_id` should be set to the individual IDs. `V_data` includes all additional covariates other than latent class that are to be included in the probit regression model. If there are no additional covariates, `V_data` should be `NULL` (default). `K_max` is the maximum number of latent classes allowable, to be used for the overfitted latent class model if the adaptive sampler is run. `class_cutoff` is the minimum size of each class as a proportion of the population, used when determining the number of latent classes.

To save results, set `save_res = TRUE` (default) and `save_path` to a string that specifies both the location and the beginning of the file name (e.g., "~/Documents/run"). The file name will have "_swolca_adapt.RData" or "_swolca_results.RData" appended to it.

If hyperparameters for the adaptive or fixed sampler are left as `NULL` (default), the following default values are used. Let K refer to `K_max` for the adaptive sampler and `K_fixed` for the fixed sampler. For π, a Dirichlet prior with hyperparameter $\alpha = 1/K$ for each component. For $\theta_{jk\cdot}$, a Dirichlet prior with hyperparameter η_j equal to `rep(1, R_j)` where `R_j` is the number of levels for exposure item j. If `R_j < R`, the remaining levels have hyperparameter set to 0.01. This is done independently for each exposure item j and is assumed to be the same across latent classes. For $\xi_{k\cdot}$, a Multivariate Normal distribution with mean vector hyperparameter μ_0 drawn from a Normal(0,1) hyperprior for each component, and variance matrix hyperparameter Σ_0 a diagonal matrix with diagonal components drawn from InvGamma(shape=3.5, scale=6.25) distributions. Note that hyperparameters for the fixed sampler should probably only be specified if running the fixed sampler directly, bypassing the adaptive sampler.

To prevent underflow issues, all θ and ξ parameters are restricted to have a minimum value of $1e-8$. If continuous variables are incorporated into the binary outcome model, any such variables

with standard deviation greater than 5 may result in errors in the variance adjustment. To avoid these errors, consider standardizing the variable to have mean 0 and standard deviation 1 or converting the variable into a categorical form.

Value

If the fixed sampler is run, returns an object `res` of class `"swolca"`; a list containing the following:

`estimates` List of posterior model results, resulting from a call to `get_estimates()`

`runtime` Total runtime for model

`data_vars` List of data variables used, including: `n`: Sample size. `J`: Number of exposure items. `R_j`: Vector of number of exposure levels for each item; Jx1. `R`: Maximum number of exposure categories across items. `Q`: Number of regression covariates excluding class assignment. `w_all`: Vector of sampling weights normalized to sum to n; nx1. `sampling_wt`: Vector of survey sampling weights; nx1. `x_mat`: Matrix of multivariate categorical exposures; nxJ. `y_all`: Vector of binary outcomes; nx1. `V_data`: Dataframe of additional regression covariates; nxQ or NULL. `V`: Regression design matrix without class assignment; nxQ. `glm_form`: String specifying formula for probit regression, excluding outcome and latent class. `stratum_id`: Vector of individual stratum IDs; nx1 or NULL. `cluster_id`: Vector of individual cluster IDs; nx1 or NULL.

`MCMC_out` List of full MCMC output, resulting from a call to `run_MCMC_Rcpp()`

`post_MCMC_out` List of MCMC output after relabeling, resulting from a call to `post_process()`

`K_fixed` Number of classes used for the fixed sampler

If `save_res = TRUE` (default), also saves `res` as `[save_path]_swolca_results.RData`.

If only the adaptive sampler is run (i.e., `run_sampler = "adapt"`), returns list `res` containing:

`MCMC_out` List of full MCMC output, resulting from a call to `run_MCMC_Rcpp()`

`K_fixed` Number of classes used for the fixed sampler, obtained from the adaptive sampler

`K_MCMC` Adaptive sampler MCMC output for K; Mx1, where M is the number of MCMC iterations after burn-in and thinning.

If `save_res = TRUE` (default), also saves `res` as `[save_path]_swolca_adapt.RData`.

References

Wu, S. M., Williams, M. R., Savitsky, T. D., & Stephenson, B. J. (2023). Derivation of outcome-dependent dietary patterns for low-income women obtained from survey data using a Supervised Weighted Overfitted Latent Class Analysis. arXiv preprint arXiv:2310.01575.

See Also

`swolca_var_adjust()` `wolca()`

Examples

```
## Not run:
# Load simulated data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
y_all <- c(data_vars$Y_data)         # Binary outcome vector, nx1
cluster_id <- data_vars$cluster_id   # Cluster indicators, nx1
```

```
stratum_id <- data_vars$true_Si      # Stratum indicators, nx1
sampling_wt <- data_vars$sample_wt   # Survey sampling weights, nx1
n <- dim(x_mat)[1]                   # Number of individuals

# Probit model only includes latent class
V_data <- NULL # Additional regression covariates
glm_form <- "~ 1"

# Run swolca
res <- swolca(x_mat = x_mat, y_all = y_all, sampling_wt = sampling_wt,
                    cluster_id = cluster_id, stratum_id = stratum_id, V_data = V_data,
                    run_sampler = "both", glm_form = glm_form, adapt_seed = 1,
                    n_runs = 50, burn = 25, thin = 1, save_res = FALSE)

# Run variance adjustment to prevent underestimation of posterior intervals
res_adjust <- swolca_var_adjust(res = res, num_reps = 100, save_res = FALSE,
                                adjust_seed = 1)

# Run swolca on NHANES data
data("data_nhanes")
x_mat <- as.matrix(dplyr::select(data_nhanes, citrus:drinks))
y_all <- data_nhanes$BP_flag
stratum_id <- data_nhanes$stratum_id
cluster_id <- data_nhanes$cluster_id
sampling_wt <- data_nhanes$sample_wt
V_data <- dplyr::select(data_nhanes, age_cat, racethnic, smoker, physactive)
glm_form <- "~ age_cat + racethnic + smoker + physactive"
res_nhanes <- swolca(x_mat = x_mat, y_all = y_all, sampling_wt = sampling_wt,
                     cluster_id = cluster_id, stratum_id = stratum_id,
                     V_data = V_data, run_sampler = "both",
                     glm_form = glm_form, adapt_seed = 20230225,
                     n_runs = 20000, burn = 19800, thin = 5, save_res = FALSE,
                     save_path = "~/Documents/run")
res_nhanes_adjust <- swolca_var_adjust(res = res_nhanes, num_reps = 100,
                                       save_res = FALSE, adjust_seed = 1)

## End(Not run)
```

`swolca_var_adjust` *Post-processing variance adjustment*

Description

`swolca_var_adj` applies applies the post-processing variance adjustment after a call to `swolca()` to correct for underestimation of posterior intervals.

Usage

```
swolca_var_adjust(
  res,
  alpha = NULL,
  eta = NULL,
  mu0 = NULL,
```

```
  Sig0 = NULL,
  num_reps = 100,
  save_res = TRUE,
  save_path = NULL,
  adjust_seed = NULL
)
```

Arguments

res	An object of class "swolca", resulting from a call to swolca(), containing the unadjusted estimates.
alpha	Hyperparameter for prior for class membership probabilities π. Default is NULL and default values are used (see Details below). If specified, must be (K_max)x1.
eta	Hyperparameter for prior for item consumption level probabilities $\theta_{jk\cdot}$ for each item j and class k, assumed to be the same across classes. Default is NULL and default values are used (see Details below). If specified, must be JxR, where J is the number of exposure items and R is the maximum number of levels for the exposure.
mu0	Mean hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k. Default is NULL and default values are used (see Details below). If specified, must be a list of K_max vectors of dimension Qx1, where Q is the number of regression covariates excluding latent class assignment.
Sig0	Variance hyperparameters for regression coefficients $\xi_{k\cdot}$ for each class k. Default is NULL and default values are used (see Details below). If specified, must be a list of K_max QxQ matrices, where Q is the number of regression covariates excluding latent class assignment.
num_reps	Number of bootstrap replicates to use for the variance adjustment estimate. Default is 100.
save_res	Boolean specifying if results should be saved. Default = TRUE.
save_path	String specifying directory and file name to save results, e.g., "~/Documents/run". Default is NULL. If this is the same as the file name specified in swolca(), the unadjusted results are overwritten with the adjusted results.
adjust_seed	Numeric seed for variance adjustment. Default is NULL.

Details

var_adjust applies a post-processing variance adjustment that rescales the variance to obtain correct coverage of posterior intervals, adapted from Williams and Savitsky (2021). To obtain the rescaling, a sandwich-type variance is estimated. To estimate the Hessian that composes the "bread" of the sandwich, the mixture model is specified in Stan and the parameters are converted to the unconstrained space. Bootstrap replicates are used to estimate the covariance matrix that composes the "meat" of the sandwich. If there are any rounding issues, the resulting matrices are mapped to the nearest positive definite matrix. Next, the rescaling adjustment is derived and applied to the parameters for all MCMC iterations. Finally, the adjusted parameters are converted back to the constrained space and the posterior median parameter estimates are recomputed, now with proper variance estimation.

To save results, set save_res = TRUE (default) and save_path to a string that specifies both the location and the beginning of the file name (e.g., "~/Documents/run"). The file name will have "_swolca_results.RData" appended to it, overwriting the unadjusted results if the file names are the same.

If hyperparameters are left as NULL (default), the following default values are used. Let K refer to the final number of latent class obtained from running swolca(), available at res$estimates$K_red. For π, a Dirichlet prior with hyperparameter $\alpha = 1/K$ for each component. For $\theta_{jk\cdot}$, a Dirichlet prior with hyperparameter η_j equal to rep(1, R_j) where R_j is the number of levels for exposure item j. If R_j < R, the remaining levels have hyperparameter set to 0.01. This is done independently for each exposure item j and is assumed to be the same across latent classes. For $\xi_{k\cdot}$, a Multivariate Normal distribution with mean vector hyperparameter μ_0 drawn from a Normal(0,1) hyperprior for each component, and variance matrix hyperparameter Σ_0 a diagonal matrix with diagonal components drawn from InvGamma(shape=3.5, scale=6.25) distributions.

If the following warning message appears, please run the sampler for more iterations as there is instability in the parameter estimates, usually in the form of large positive or negative values for ξ): "Error in svrVar(thetas, scale, rscales, mse = design$mse, coef = full) : All replicates contained NAs".

Value

Returns an object res of class "swolca", which includes all outputs from swolca() as well as a list estimates_adjust containing:

pi_red Matrix of adjusted posterior samples for pi. Mx(K_red), where M is the number of MCMC iterations after burn-in and thinning.

theta_red Array of adjusted posterior samples for theta. MxJx(K_red)xR

xi_red Array of adjusted posterior samples for xi. Mx(K_red)xQ

pi_med Vector of adjusted posterior median estimates for pi. (K_red)x1

theta_med Array of adjusted posterior median estimates for theta. px(K_red)xR

xi_med Matrix of adjusted posterior median estimates for xi. (K_red)xQ

Phi_med Vector of adjusted individual outcome probabilities. nx1

c_all Vector of final individual class assignments from swolca(). nx1

pred_class_probs Matrix of individual posterior class probabilities from swolca(). nx(K_red)

loglik_med Vector of final indiviudal log-likehoods from swolca(). nx1

The runtime output for res is also updated to include the runtime for the variance adjustment in addition to the runtime for the main swolca() model.

If save_res = TRUE (default), the updated res object is saved as [save_path]_swolca_results.RData, overwriting the unadjusted results if the file names are the same.

References

Williams, M. R., & Savitsky, T. D. (2021). Uncertainty Estimation for Pseudo-Bayesian Inference Under Complex Sampling. International Statistical Review, 89(1), 72-107.

See Also

swolca()

Examples

```
## Not run:
# Load simulated data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
```

```
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
y_all <- c(data_vars$Y_data)         # Binary outcome vector, nx1
cluster_id <- data_vars$cluster_id   # Cluster indicators, nx1
stratum_id <- data_vars$true_Si      # Stratum indicators, nx1
sampling_wt <- data_vars$sample_wt   # Survey sampling weights, nx1
n <- dim(x_mat)[1]                   # Number of individuals

# Probit model only includes latent class
V_data <- NULL # Additional regression covariates
glm_form <- "~ 1"

# Run swolca
res <- swolca(x_mat = x_mat, y_all = y_all, sampling_wt = sampling_wt,
              cluster_id = cluster_id, stratum_id = stratum_id, V_data = V_data,
              run_sampler = "both", glm_form = glm_form, adapt_seed = 1,
              n_runs = 50, burn = 25, thin = 1, save_res = FALSE)

# Apply variance adjustment to posterior estimates
res_adjust <- swolca_var_adjust(res = res, num_reps = 100, save_res = FALSE,
                                adjust_seed = 1)

## End(Not run)
```

`vars_across_class` *Create table of class distributions across variables*

Description

`vars_across_class` displays the distribution of variables across latent classes in the population.

Usage

```
vars_across_class(
  c_all,
  cov_df,
  sampling_wt,
  stratum_id,
  cluster_id,
  digits = 1,
  col_props = TRUE,
  res
)
```

Arguments

`c_all`	nx1 factor vector of latent class assignments, typically obtained from the `swolca()`, `wolca()`, `swolca_var_adjust()` or `wolca_var_adjust()` functions. Factor levels should be labeled with the names that are to appear in the output table.
`cov_df`	Dataframe with n rows, with columns consisting of the variables that are to be shown with their distributions across latent classes. Factor variables should be labeled with the names and levels that are to appear in the output table.
`sampling_wt`	Vector of survey sampling weights. nx1. Default is `NULL`, indicating no sampling weights and setting all weights to 1.

stratum_id	Vector of individual stratum IDs. nx1. Default is NULL, indicating no stratification.
cluster_id	Vector of individual cluster IDs. nx1. Default is NULL, indicating each individual is their own cluster.
digits	Integer indicating the number of decimal places to be used. Default is 1, which rounds to the nearest tenth.
col_props	Boolean indicating if factor variables should have percentages reported as column totals (default TRUE) that provide the percentage of the population in each category for a given latent class, or reported as row totals (FALSE) that provide the percentage of the population in each latent class for a given category.
res	Results from swolca(), wolca(), swolca_var_adjust() or wolca_var_adjust().

Details

The dataframe output includes a first row that presents the population percentage of individuals falling into each latent class, as well as the corresponding posterior standard deviation, obtained from the π parameters from a "swolca" or "wolca" object. The second row is the sample percentage falling into each latent class.

Subsequent rows for continuous variables display the mean values for each latent class in the population, adjusting for survey design. Subsequent rows for factor variables display either column totals (percentage of the population in each category for a given latent class) or row totals (percentage of the population in each latent class for a given category), depending on the col_props input variable, adjusting for survey design. Survey design adjustments are carried out post-hoc using methods provided in the survey R package (Lumley, 2004). Since these methods do not account for variability in the latent classes, their standard error estimates will be underestimates and are not reported.

Value

Outputs dataframe output_df displaying the distribution of variables across the latent classes in the population. output_df has columns for the variable names, the category names for factor variables, the distribution of the variables across the latent classes, and the distribution of the variables overall in the population.

References

Lumley T (2004). "Analysis of Complex Survey Samples." Journal of Statistical Software, 9(1), 1-19. R package verson 2.2.

Examples

```
data("data_nhanes")
data("run_nhanes_swolca_results")
res <- run_nhanes_swolca_results
c_all <- factor(res$estimates_adjust$c_all, levels = 1:5,
                labels = paste0("C", 1:5))
cov_df <- dplyr::select(data_nhanes, RIDAGEYR, age_cat, racethnic, smoker,
                        physactive)
cov_df <- dplyr::rename(cov_df, Age = RIDAGEYR)
cov_df <- dplyr::mutate(cov_df,
   Age_Group = factor(age_cat, levels = c(1, 2, 3),
                      labels = c("[20, 40)", "[40, 60)", ">=60")),
   Race_Ethnicity = factor(racethnic, c(1, 2, 3, 4, 5),
```

```
                    labels = c("NH White", "NH Black", "NH Asian",
                               "Hispanic/Latino", "Other/Mixed")),
   Current_Smoker = factor(smoker, levels = c(0, 1), labels = c("No", "Yes")),
   Physical_Activity = factor(physactive, levels = c("Inactive", "Active")),
   .keep = "unused")
output_df <- vars_across_class(c_all = c_all, cov_df = cov_df,
                               sampling_wt = data_nhanes$sample_wt,
                               stratum_id = data_nhanes$stratum_id,
                               cluster_id = data_nhanes$cluster_id,
                               digits = 1, col_props = TRUE, res = res)
```

`wolca` *Run the WOLCA model*

Description

`wolca` runs an unsupervised weighted overfitted latent class analysis (WOLCA) described in Wu et al. (2023) and Stephenson et al. (2023), then saves and returns the results.

Usage

```
wolca(
  x_mat,
  sampling_wt = NULL,
  cluster_id = NULL,
  stratum_id = NULL,
  run_sampler = "both",
  K_max = 30,
  adapt_seed = NULL,
  K_fixed = NULL,
  fixed_seed = NULL,
  class_cutoff = 0.05,
  n_runs = 20000,
  burn = 10000,
  thin = 5,
  update = 10,
  save_res = TRUE,
  save_path = NULL,
  alpha_adapt = NULL,
  eta_adapt = NULL,
  alpha_fixed = NULL,
  eta_fixed = NULL
)
```

Arguments

`x_mat`	Matrix of multivariate categorical exposures. nxJ
`sampling_wt`	Vector of survey sampling weights. nx1. Default is `NULL`, indicating no sampling weights and setting all weights to 1.
`cluster_id`	Vector of individual cluster IDs. nx1. Default is `NULL`, indicating each individual is their own cluster.

stratum_id	Vector of individual stratum IDs. nx1. Default is NULL, indicating no stratification.
run_sampler	String specifying which sampler(s) should be run. Must be one of "both" (default), "fixed", or "adapt". See Details.
K_max	Upper limit for number of classes. Default is 30.
adapt_seed	Numeric seed for adaptive sampler. Default is NULL.
K_fixed	True number of classes, if known. Default is NULL, as it is not necessary for the adaptive sampler. If bypassing the adaptive sampler and running the fixed sampler directly, need to specify a value here. See Details.
fixed_seed	Numeric seed for fixed sampler. Default is NULL.
class_cutoff	Minimum class size proportion when determining number of classes. Default is 0.05.
n_runs	Number of MCMC iterations. Default is 20000.
burn	Number of MCMC iterations to drop as a burn-in period. Default is 10000.
thin	Thinning factor for MCMC iterations. Default is 5.
update	Number specifying that MCMC progress updates should be printed every update iterations. Default is 10.
save_res	Boolean specifying if results should be saved. Default = TRUE.
save_path	String specifying directory and file name to save results, e.g., "~/Documents/run". Default is NULL.
alpha_adapt	Adaptive sampler hyperparameter for prior for class membership probabilities π. Default is NULL and default values are used (see Details). If specified, must be (K_max)x1.
eta_adapt	Adaptive sampler hyperparameter for prior for item level probabilities $\theta_{jk\cdot}$ for each item j and class k, assumed to be the same across classes. Default is NULL and default values are used (see Details). If specified, must be JxR, where J is the number of exposure items and R is the maximum number of categories for the exposure.
alpha_fixed	Fixed sampler hyperparameter for prior for π. Default is NULL and default values are used (see Details). If specified, must be (K_fixed)x1.
eta_fixed	Fixed sampler hyperparameter for prior for item level probabilities $\theta_{jk\cdot}$ for each item j and class k, assumed to be the same across classes. Default is NULL and default values are used (see Details). If specified, must be JxR, where J is the number of exposure items and R is the maximum number of categories for the exposure.

Details

If no survey sample adjustments are desired, leave sampling_wt, stratum_id, and cluster_id to their default NULL values.

By default, the function will run two samplers: the adaptive sampler, followed by the fixed sampler. The adaptive sampler determines the number of latent classes, which is then used in the fixed sampler for parameter estimation. If the number of latent classes is already known and only the fixed sampler needs to be run, specify "fixed" for the run_sampler argument and specify a number for K_fixed. If only the adaptive sampler is to be run, specify "adapt" for the run_sampler argument. Use adapt_seed (default is NULL) to specify a seed for the adaptive sampler, and use fixed_seed (default is NULL) to specify a separate seed for the fixed sampler.

x_mat is an nxJ matrix with each row corresponding to the J-dimensional categorical exposure for an individual. If there is no clustering present, cluster_id should be set to the individual IDs. K_max is the maximum number of latent classes allowable, to be used for the overfitted latent class model if the adaptive sampler is run. class_cutoff is the minimum size of each class as a proportion of the population, used when determining the number of latent classes.

To save results, set save_res = TRUE (default) and save_path to a string that specifies both the location and the beginning of the file name (e.g., "~/Documents/run"). The file name will have "_wolca_adapt.RData" or "_wolca_results.RData" appended to it.

If hyperparameters for the adaptive or fixed sampler are left as NULL (default), the following default values are used. Let K refer to K_max for the adaptive sampler and K_fixed for the fixed sampler. For π, a Dirichlet prior with hyperparameter $\alpha = 1/K$ for each component. For $\theta_{jk\cdot}$, a Dirichlet prior with hyperparameter η_j equal to rep(1, R_j) where R_j is the number of categories for exposure item j. If R_j < R, the remaining categories have hyperparameter set to 0.01. This is done independently for each exposure item j and is assumed to be the same across latent classes. Note that hyperparameters for the fixed sampler should probably only be specified if running the fixed sampler directly, bypassing the adaptive sampler.

To prevent underflow issues, all θ parameters are restricted to have a minimum value of $1e-8$.

Value

If the fixed sampler is run, returns an object res of class "wolca"; a list containing the following:

estimates List of posterior model results, resulting from a call to get_estimates_wolca()

runtime Total runtime for model

data_vars List of data variables used, including: n: Sample size. J: Number of exposure items. R_j: Number vector of number of exposure categories for each item; Jx1. R: Maximum number of exposure categories across items. w_all: Vector of sampling weights normalized to sum to n; nx1. sampling_wt: Vector of survey sampling weights; nx1. x_mat: Matrix of multivariate categorical exposures; nxJ. stratum_id: Vector of individual stratum IDs; nx1 or NULL. cluster_id: Vector of individual cluster IDs; nx1 or NULL. ci_level: Confidence interval level.

MCMC_out List of full MCMC output, resulting from a call to run_MCMC_Rcpp()

post_MCMC_out List of MCMC output after relabeling, resulting from a call to post_process()

K_fixed Number of classes used for the fixed sampler

If save_res = TRUE (default), also saves res as [save_path]_wolca_results.RData.

If only the adaptive sampler is run (i.e., run_sampler = "adapt"), returns list res containing:

MCMC_out List of full MCMC output, resulting from a call to run_MCMC_Rcpp()

K_fixed Number of classes used for the fixed sampler, obtained from the adaptive sampler

K_MCMC Adaptive sampler MCMC output for K; Mx1, where M is the number of MCMC iterations after burn-in and thinning.

If save_res = TRUE (default), also saves res as [save_path]_wolca_adapt.RData.

References

Stephenson, B. J. K., Wu, S. M., Dominici, F. (2023). Identifying dietary consumption patterns from survey data: a Bayesian nonparametric latent class model. Journal of the Royal Statistical Society Series A: Statistics in Society, qnad135.

Wu, S. M., Williams, M. R., Savitsky, T. D., & Stephenson, B. J. (2023). Derivation of outcome-dependent dietary patterns for low-income women obtained from survey data using a Supervised Weighted Overfitted Latent Class Analysis. arXiv preprint arXiv:2310.01575.

Examples

```
## Not run:
# Load data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
cluster_id <- data_vars$cluster_id   # Cluster indicators, nx1
stratum_id <- data_vars$true_Si      # Stratum indicators, nx1
sampling_wt <- data_vars$sample_wt   # Survey sampling weights, nx1
n <- dim(x_mat)[1]                   # Number of individuals

# Run wolca
res <- wolca(x_mat = x_mat, sampling_wt = sampling_wt,
             cluster_id = cluster_id, stratum_id = stratum_id,
             run_sampler = "both", adapt_seed = 1, n_runs = 50, burn = 25,
             thin = 1, save_res = FALSE)

## End(Not run)
```

---

wolca_svyglm *Fit survey-weighted probit model for WOLCA*

Description

wolca_svyglm relates latent classes patterns derived from wolca() to a binary outcome by fitting a survey-weighted probit model.

Usage

```
wolca_svyglm(
  res,
  y_all,
  V_data = NULL,
  glm_form,
  ci_level = 0.95,
  save_res = TRUE,
  save_path = NULL
)
```

Arguments

res	An object of class "wolca", resulting from a call to wolca() or swolca_var_adjust()
y_all	Vector of binary outcomes. nx1
V_data	Dataframe of additional regression covariates. nxQ. Factor covariates must be converted to factors. If NULL (default), no additional covariates are to be included. All variables in glm_form must be found in V_data.
glm_form	String specifying formula for probit regression, excluding outcome and latent class. For example, "~ 1" for the model with only latent class as covariates. All variables in glm_form must be found in V_data. Do not specify interaction terms for latent class by additional covariates, as these terms are already included.

ci_level	Confidence interval level for probit regression coefficient estimates. Default is 0.95.
save_res	Boolean specifying if results should be saved. Default = TRUE.
save_path	String specifying directory and file name to save results, e.g., "~/Documents/run". Default is NULL.

Details

wolca_svyglm is the second step of a two-step approach that runs an unsupervised WOLCA in the first step to derive latent class patterns and subsequently treats the class assignments as fixed and includes them as covariates in a frequentist survey-weighted probit regression model that uses an asymptotic sandwich estimator for variance estimation.

wolca_svyglm specifies survey design and fits a survey-weighted probit regression model according to the formula specified in glm_form using the svyglm() function from the survey package (Lumley, 2023). Regression coefficients and their confidence intervals are obtained from the svyglm() output. If the residual degrees of freedom is negative, a Wald confidence interval is manually calculated using a t-distribution with degrees of freedom from the survey design, and the fit_summary output also uses the degrees of freedom from the survey design to obtain p-values. This can happen when there are few clusters per stratum and a large number of covariates.

The point and interval estimates are then converted into mixture reference coding format to match the output format from swolca(). V_data includes all covariates to include in the probit regression other than latent class.

To save results, set save_res = TRUE (default) and save_path to a string that specifies both the location and the beginning of the file name (e.g., "~/Documents/run"). The file name will have "_wolca_results.RData" appended to it, overwriting the unsupervised results if the file names are the same.

Value

Returns an object res of class "wolca", which includes all outputs from wolca() or wolca_var_adjust() as well as additional list estimates_syglm containing the following objects:

xi_est Matrix of estimates for xi. (K_red)xQ

xi_est_lb Matrix of confidence interval lower bound estimates for xi. (K_red)xQ

xi_est_ub Matrix of confidence interval upper bound estimates for xi. (K_red)xQ

fit svyglm class object with output from the svyglm regression model

fit_summary summary.svyglm class object with output from the summary() function. If the residual degrees of freedom is negative, the degrees of freedom from the survey design is used.

List data_vars of res is updated to contain the following additional objects:

Q Number of regression covariates excluding class assignment.

y_all Vector of binary outcomes; nx1.

V_data Dataframe of additional regression covariates; nxQ or NULL.

V Regression design matrix without class assignment; nxQ.

glm_form String specifying formula for probit regression, excluding outcome and latent class.

ci_level Confidence interval level for probit regression coefficient estimates.

The runtime output for res is also updated to include the runtime for the probit regression in addition to the runtime for the main wolca() model.

If save_res = TRUE (default), the updated res object is saved as [save_path]_wolca_results.RData, overwriting the unsupervised results if the file names are the same.

References

Lumley T (2023). "survey: analysis of complex survey samples." R package version 4.2.

See Also

wolca()

Examples

```
## Not run:
# Load data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
y_all <- c(data_vars$Y_data)         # Binary outcome vector, nx1
cluster_id <- data_vars$cluster_id   # Cluster indicators, nx1
stratum_id <- data_vars$true_Si      # Stratum indicators, nx1
sampling_wt <- data_vars$sample_wt   # Survey sampling weights, nx1
n <- dim(x_mat)[1]                   # Number of individuals

# Probit model only includes latent class
V_data <- NULL # Additional regression covariates
# Survey-weighted regression formula
glm_form <- "~ 1"

# Run wolca
res <- wolca(x_mat = x_mat, sampling_wt = sampling_wt,
             cluster_id = cluster_id, stratum_id = stratum_id,
             run_sampler = "both", adapt_seed = 1, n_runs = 50, burn = 25,
             thin = 1, save_res = FALSE)

# Apply variance adjustment to posterior estimates
res_adjust <- wolca_var_adjust(res = res, num_reps = 100, save_res = FALSE,
                               adjust_seed = 1)

# Run weighted outcome regression model
res_svyglm <- wolca_svyglm(res = res_adjust, y_all = y_all,
                           glm_form = glm_form, ci_level = 0.95,
                           V_data = V_data, save_res = FALSE)

## End(Not run)
```

---

wolca_var_adjust *Post-processing variance adjustment*

Description

wolca_var_adj applies applies the post-processing variance adjustment after a call to wolca() to correct for underestimation of posterior intervals.

Usage

```
wolca_var_adjust(
  res,
  alpha = NULL,
  eta = NULL,
  num_reps = 100,
  save_res = TRUE,
  save_path = NULL,
  adjust_seed = NULL
)
```

Arguments

res	An object of class "wolca", resulting from a call to wolca(), containing the unadjusted estimates.
alpha	Hyperparameter for prior for class membership probabilities π. Default is NULL and default values are used (see Details below). If specified, must be (K_max)x1.
eta	Hyperparameter for prior for item consumption level probabilities $\theta_{jk\cdot}$ for each item j and class k, assumed to be the same across classes. Default is NULL and default values are used (see Details below). If specified, must be JxR, where J is the number of exposure items and R is the maximum number of levels for the exposure.
num_reps	Number of bootstrap replicates to use for the variance adjustment estimate. Default is 100.
save_res	Boolean specifying if results should be saved. Default = TRUE.
save_path	String specifying directory and file name to save results, e.g., "~/Documents/run". Default is NULL. If this is the same as the file name specified in wolca(), the unadjusted results are overwritten with the adjusted results.
adjust_seed	Numeric seed for variance adjustment. Default is NULL.

Details

wolca_var_adjust applies a post-processing variance adjustment that rescales the variance to obtain correct coverage of posterior intervals, adapted from Williams and Savitsky (2021). To obtain the rescaling, a sandwich-type variance is estimated. To estimate the Hessian that composes the "bread" of the sandwich, the mixture model is specified in Stan and the parameters are converted to the unconstrained space. Bootstrap replicates are used to estimate the covariance matrix that composes the "meat" of the sandwich. If there are any rounding issues, the resulting matrices are mapped to the nearest positive definite matrix. Next, the rescaling adjustment is derived and applied to the parameters for all MCMC iterations. Finally, the adjusted parameters are converted back to the constrained space and the posterior median parameter estimates are recomputed, now with proper variance estimation.

To save results, set save_res = TRUE (default) and save_path to a string that specifies both the location and the beginning of the file name (e.g., "~/Documents/run"). The file name will have "_wolca_results.RData" appended to it, overwriting the unadjusted results if the file names are the same.

If hyperparameters are left as NULL (default), the following default values are used. Let K refer to the final number of latent class obtained from running wolca(), available at res$estimates$K_red. For π, a Dirichlet prior with hyperparameter $\alpha = 1/K$ for each component. For $\theta_{jk\cdot}$, a Dirichlet prior with hyperparameter η_j equal to rep(1, R_j) where R_j is the number of levels for exposure

item j. If R_j < R, the remaining levels have hyperparameter set to 0.01. This is done independently for each exposure item j and is assumed to be the same across latent classes.

Value

Returns an object res of class "wolca", which includes all outputs from wolca() as well as a list estimates_adjust containing:

pi_red Matrix of adjusted posterior samples for pi. Mx(K_red), where M is the number of MCMC iterations after burn-in and thinning.

theta_red Array of adjusted posterior samples for theta. MxJx(K_red)xR

pi_med Vector of adjusted posterior median estimates for pi. (K_red)x1

theta_med Array of adjusted posterior median estimates for theta. px(K_red)xR

c_all Vector of final individual class assignments from wolca(). nx1

pred_class_probs Matrix of individual posterior class probabilities from wolca(). nx(K_red)

The runtime output for res is also updated to include the runtime for the variance adjustment in addition to the runtime for the main wolca() model.

If save_res = TRUE (default), the updated res object is saved as [save_path]_wolca_results.RData, overwriting the unadjusted results if the file names are the same.

References

Williams, M. R., & Savitsky, T. D. (2021). Uncertainty Estimation for Pseudo-Bayesian Inference Under Complex Sampling. International Statistical Review, 89(1), 72-107.

See Also

wolca()

Examples

```
## Not run:
# Load simulated data and obtain relevant variables
data("sim_data")
data_vars <- sim_data
x_mat <- data_vars$X_data            # Categorical exposure matrix, nxJ
cluster_id <- data_vars$cluster_id   # Cluster indicators, nx1
stratum_id <- data_vars$true_Si      # Stratum indicators, nx1
sampling_wt <- data_vars$sample_wt   # Survey sampling weights, nx1
n <- dim(x_mat)[1]                   # Number of individuals

# Run wolca
res <- wolca(x_mat = x_mat, sampling_wt = sampling_wt,
             cluster_id = cluster_id, stratum_id = stratum_id,
             run_sampler = "both", adapt_seed = 1, n_runs = 50, burn = 25,
             thin = 1, save_res = FALSE)

# Apply variance adjustment to posterior estimates
res_adjust <- wolca_var_adjust(res = res, num_reps = 100, save_res = FALSE,
                               adjust_seed = 1)

## End(Not run)
```

Index