Package 'LalondeBayesBorrow'

May 4, 2025

Type Package

Title What the Package Does (Title Case)

Version 1.1.0

Author Zhining Sui

Maintainer The package maintainer <yourself@somewhere.net>

Description More about what it does (maybe more than one line)
Use four spaces when indenting paragraphs within the Description.

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Imports dplyr,tidyr,knitr,RBesT,parallel,doParallel,foreach,data.table,stats,rlang

VignetteBuilder knitr

R topics documented:

`bayesian_lalonde_decision` *Bayesian Decision Framework based on Lalonde Criteria*

Description

This function simulates a Bayesian trial across multiple scenarios (simulations), calculating posterior distributions for treatment and control arms (with optional historical borrowing), performing Bayesian inference (estimates, credible intervals, posterior probabilities), and applying decision rules based on the Lalonde framework.

Usage

```
bayesian_lalonde_decision(
  endpoint,
  data_summary,
  prior_params,
  lrv = 0,
  tv = 0,
  fgr = 0.2,
  fsr = 0.1,
  arm_names,
  posterior_infer = TRUE,
  Lalonde_decision = TRUE,
  EXP_TRANSFORM = FALSE
)
```

Arguments

`endpoint`	A character string. The type of endpoint. Must be ‘"continuous"‘ or ‘"binary"‘. This determines the likelihood and prior structure used by ‘posterior_distribution‘.
`data_summary`	A data frame. Contains summarized data for each arm for each simulation repetition. Must include an ‘nsim‘ column identifying the simulation repetition number. Data for each arm and its parameters should be in columns prefixed by the arm name and a dot (e.g., ‘control.n‘, ‘treatment.mu_hat‘, ‘control_h.count‘). • For ‘"continuous"‘ endpoint, required columns per arm (e.g., ‘control.‘) are ‘n‘ (sample size), ‘mu_hat‘ (sample mean), ‘s‘ (standard deviation of sample mean). • For ‘"binary"‘ endpoint, required columns per arm (e.g., ‘control.‘) are ‘n‘ (sample size), ‘count‘ (number of responses/events). Historical arm data (e.g., ‘control_h.n‘, ‘treatment_h.count‘) are optional.
`prior_params`	A named list. Contains prior distribution and borrowing parameters for each arm (treatment and control, potentially historical). Names must follow the convention ‘[arm_name].[parameter]‘, matching the parameters expected by ‘posterior_distribution‘. • For ‘"continuous"‘ arms, includes ‘[arm_name].theta0‘, ‘[arm_name].s0‘. • For ‘"binary"‘ arms, includes ‘[arm_name].a‘, ‘[arm_name].b‘, ‘[arm_name].a0‘, ‘[arm_name].b0‘.

- For arms with potential historical borrowing, includes ‘[arm_name].delta‘, ‘[arm_name].w‘, ‘[arm_name].ess_h‘.

Example names: ‘treatment.delta‘, ‘control.w‘, ‘treatment.theta0‘, ‘control.s0‘, ‘treatment.a‘, ‘control.b‘, ‘treatment.a0‘, ‘control.b0‘, ‘control.ess_h‘.

`lrv` A scalar numeric value. The Lower Reference Value for the comparison metric (difference or ratio). Used in the Lalonde decision criteria.

`tv` A scalar numeric value. The Target Value for the comparison metric. Used in the Lalonde decision criteria.

`fgr` A scalar numeric value between 0 and 1. The False Go Rate. Used in the Lalonde probability and quantile-based decision criteria.

`fsr` A scalar numeric value between 0 and 1. The False Stop Rate. Used in the Lalonde probability and quantile-based decision criteria.

`arm_names` A character vector. A list of arm names present in the simulation. This parameter is used internally to identify which arm data columns to look for in ‘data_summary‘ and which prior parameters to expect in ‘prior_params‘. Expected names are typically ‘"treatment"‘, ‘"control"‘, ‘"treatment_h"‘, ‘"control_h"‘.

`posterior_infer` A logical value. If ‘TRUE‘, compute posterior inference statistics (95% credible intervals and posterior probabilities ‘pr_m‘, ‘pr_l‘, ‘pr_t‘). Defaults to ‘TRUE‘.

`Lalonde_decision` A logical value. If ‘TRUE‘, apply the Lalonde decision rules based on the calculated posterior probabilities and quantiles. Defaults to ‘TRUE‘. Note: Requires ‘posterior_infer = TRUE‘ to make decisions.

`EXP_TRANSFORM` A logical value. If ‘TRUE‘, results for individual arm means, credible intervals, and the comparison estimate (‘est_compare‘) are exponentiated. This implies that the underlying parameters in ‘post_t_mix‘ and ‘post_c_mix‘ (and hence ‘mu_hat‘, ‘mu‘ in ‘data_summary‘ and ‘prior_params‘) are on a log scale, and the comparison is interpreted as a ratio. Defaults to ‘FALSE‘.

Details

It leverages ‘posterior_distribution‘ to calculate individual arm posteriors, ‘posterior_inference‘ to summarize these posteriors and their difference/ratio, and ‘posterior_prob‘ to calculate probabilities against thresholds.

The function is designed to work with data summarized per arm per simulation repetition, typically the output of a data generation step. Parallel processing is used to speed up calculations across simulations.

Value

A list containing three data frames, each row corresponding to a simulation repetition (‘nsim‘):

`post_params` Contains the raw posterior parameters (e.g., ‘w_post‘, ‘mu1_post‘, ‘sigma1_post‘, etc.) for the treatment and control arms, prefixed with ‘treatment.‘ or ‘control.‘, and the prior weights (‘treatment.w_prior‘, ‘control.w_prior‘).

`post_est_ci` Contains standard posterior inference (mean, SD, 95% CI) for each arm and their comparison, postfixed with ‘_95ci‘. Values are exponentiated if ‘EXP_TRANSFORM = TRUE‘ (except SDs).

`post_inference` Contains inference results relevant to the Lalonde decision framework. Includes posterior probabilities (‘pr_m‘, ‘pr_l‘, ‘pr_t‘) and comparison quantiles derived from ‘fgr‘/‘fsr‘ (‘_lalonde‘). If ‘Lalonde_decision = TRUE‘ and ‘posterior_infer = TRUE‘, includes decision columns (‘decision_pr‘, ‘decision_ci‘).

See Also

`posterior_distribution`, `posterior_inference`, `posterior_prob`, `convert_RBesT_mix`, `mix`, `mixnorm`, `mixbeta`, `pmixdiff`, `qmixdiff`

`calc_post_dist_metrics`

Calculate Posterior Distribution Metrics

Description

This function calculates various evaluation metrics for the posterior distribution, including the average bias, standard error, empirical standard deviation, and nominal coverage probability. For the "g-score" endpoint, it calculates the log-median ratio, and for the "OR" endpoint, it calculates the rate difference.

Usage

```
calc_post_dist_metrics(endpoint, true_value, post_est_ci, remove.na = FALSE)
```

Arguments

`endpoint`	A string. Specifies the type of endpoint ("g-score" or "OR").
`true_value`	A scalar. The true median ratio for the "g-score" endpoint or the true rate difference for the "OR" endpoint.
`post_est_ci`	A data frame. Contains posterior inference obtained from ‘bayesian_lalonde_decision()‘, including posterior estimates, standard errors, and 95% credible intervals for each simulation repetition.
`remove.na`	

Value

A data frame containing the evaluation metrics for the posterior distribution, including: - ‘bias_avg‘: The average bias of the estimate. - ‘sd_avg‘: The average standard error of the estimate. - ‘sd_empirical‘: The empirical standard deviation of the estimate. - ‘cp‘: The nominal coverage probability (95%). - Additional metrics specific to the "g-score" endpoint. More details can be found in ‘eval_gscore_approx_dist()‘.

convert_RBesT_mix *Convert to RBesT Mixture Distribution*

Description

This function generates a mixture distribution object compatible with the RBesT package from a data frame containing component parameters. It supports the creation of both normal (‘mixnorm‘) and beta (‘mixbeta‘) mixture distributions based on the specified endpoint type. The function is designed to handle mixture models with up to two components based on the provided structure of the input data frame.

Usage

```
convert_RBesT_mix(post, endpoint)
```

Arguments

post	A data frame or a list of parameters. This object should contain the parameters defining the mixture components. The expected structure depends on the ‘endpoint‘ type: - For ‘endpoint = "continuous"‘ (Normal mixture): Expected columns/elements are ‘w‘ (weight of the first component), ‘mu1‘ (mean of the first component), ‘sigma1‘ (standard deviation of the first component), ‘mu2‘ (mean of the second component), and ‘sigma2‘ (standard deviation of the second component). The weight of the second component is implicitly ‘1 - w‘. - For ‘endpoint = "binary"‘ (Beta mixture): Expected columns/elements are ‘w‘ (weight of the first component), ‘a1‘ (alpha parameter of the first component), ‘b1‘ (beta parameter of the first component), ‘a2‘ (alpha parameter of the second component), and ‘b2‘ (beta parameter of the second component). The weight of the second component is implicitly ‘1 - w‘. The function currently assumes a single row/set of parameters for the mixture, representing either a one- or two-component mixture.
endpoint	A character string. Specifies the type of endpoint and thus the type of mixture distribution to create. Must be either ‘"continuous"‘ for a normal mixture (‘mixnorm‘) or ‘"binary"‘ for a beta mixture (‘mixbeta‘).

Value

An RBesT mixture distribution object, specifically of class ‘mixnorm‘ if ‘endpoint‘ is ‘"continuous"‘ or ‘mixbeta‘ if ‘endpoint‘ is ‘"binary"‘. Returns an error if the ‘endpoint‘ is not specified correctly or if required parameters are missing or invalid.

See Also

mixnorm, @seealso mixbeta, @seealso mix

Examples

```
# Example for a continuous endpoint (Normal mixture)
post_continuous <- data.frame(w = 0.6, mu1 = 10, sigma1 = 2, mu2 = 15, sigma2 = 3)
mix_norm <- convert_RBesT_mix(post_continuous, endpoint = "continuous")
print(mix_norm)
```

```
# Example for a binary endpoint (Beta mixture)
post_binary <- list(w = 0.8, a1 = 5, b1 = 15, a2 = 10, b2 = 10)
mix_beta <- convert_RBesT_mix(post_binary, endpoint = "binary")
print(mix_beta)

# Example for a single-component mixture (weight = 1)
post_single <- data.frame(w = 1, mu1 = 12, sigma1 = 1.5, mu2 = NA, sigma2 = NA)
mix_single <- convert_RBesT_mix(post_single, endpoint = "continuous")
print(mix_single)

# Example for a single-component mixture (weight = 0)
post_single_comp2 <- list(w = 0, mu1 = NA, sigma1 = NA, mu2 = 8, sigma2 = 2.5)
mix_single_comp2 <- convert_RBesT_mix(post_single_comp2, endpoint = "continuous")
print(mix_single_comp2)
```

`create_data_gen_params`

Create Data Generation Parameters

Description

This function processes and validates a set of parameters for generating data for different study arms (treatment, control, and their historical counterparts). It checks for missing parameters, replaces them with ‘NA‘ while issuing warnings, and structures the valid parameters into a nested list format suitable for data simulation functions. The function supports various endpoint types, each requiring a specific set of parameters and potentially applying transformations (like logarithm for means in "g-score").

Usage

```
create_data_gen_params(params, endpoint)
```

Arguments

`params`	A list. Contains the parameters for data generation. Expected parameters follow a naming convention: ‘[arm_prefix]_[parameter_name]‘, where ‘[arm_prefix]‘ can be ‘trt‘ (treatment), ‘ctrl‘ (control), ‘trt_h‘ (historical treatment), or ‘ctrl_h‘ (historical control). Required ‘[parameter_name]‘s depend on the ‘endpoint‘. Examples include ‘n‘ (sample size), ‘p‘ (probability), ‘mu‘ (mean), ‘sigma‘ (standard deviation).
`endpoint`	A character string. Specifies the type of endpoint. Must be one of ‘"continuous"‘, ‘"binary"‘, ‘"DpR"‘, or ‘"g-score"‘.

Details

Arms (treatment, control, historical treatment, historical control) are included in the output list only if all their required parameters for the specified ‘endpoint‘ are present (not ‘NULL‘ and not resulting in ‘NA‘ after initial checks), and if the sample size ‘n‘ is positive.

The interpretation of parameters depends on the ‘endpoint‘:

- **continuous**: Parameters for a normal continuous outcome include a mean (‘mu‘) and a standard deviation (‘sigma‘). Required parameters per arm: ‘n‘, ‘mu‘, ‘sigma‘.

- **binary**: Parameters for a binary outcome include a probability of the event occurring (‘p‘). Required parameters per arm: ‘n‘, ‘p‘.
- **DpR** (Depth of Response): Parameters include a probability of complete tumor shrinkage (‘p‘), a mean (‘mu‘), and a standard deviation (‘sigma‘) for the continuous outcome part. Required parameters per arm: ‘n‘, ‘p‘, ‘mu‘, ‘sigma‘.
- **g-score**: Parameters for zero-inflated continuous g-scores include a probability of observing a zero value (‘p‘), a mean (‘mu‘, log-transformed internally) and a standard deviation (‘sigma‘) for the non-zero part. Required parameters per arm: ‘n‘, ‘p‘, ‘mu‘, ‘sigma‘.

Value

A named list where each element corresponds to a study arm (‘treatment‘, ‘control‘, ‘treatment_h‘, ‘control_h‘). Each arm’s element is a list containing its data generating parameters (‘n‘, ‘p‘, ‘mu‘, ‘sigma‘, etc., depending on the endpoint) and its ‘name‘. Arms for which required parameters were missing (‘NULL‘ or resulted in ‘NA‘) or ‘n‘ was non-positive are excluded from the list. Missing parameters for included arms are reported via warnings and set to ‘NA‘.

Examples

```
# Example for 'continuous' endpoint with missing historical data
params_continuous <- list(
  trt_n = 150, trt_mu = 75, trt_sigma = 10,
  ctrl_n = 150, ctrl_mu = 70, ctrl_sigma = 11,
  trt_h_n = 80, trt_h_mu = 76 # Missing sigma, this arm will be excluded
)
data_params_continuous <- create_data_gen_params(params_continuous, "continuous")
str(data_params_continuous)

# Example for 'binary' endpoint
params_binary <- list(
  trt_n = 200, trt_p = 0.7,
  ctrl_n = 200, ctrl_p = 0.5,
  ctrl_h_n = 100, ctrl_h_p = 0.55
)
data_params_binary <- create_data_gen_params(params_binary, "binary")
str(data_params_binary)

# Example for 'DpR' endpoint
params_DpR <- list(
  trt_n = 120, trt_p = 0.9, trt_mu = 5, trt_sigma = 1.2, # p=prob of complete shrinkage
  ctrl_n = 120, ctrl_p = 0.8, ctrl_mu = 4.5, ctrl_sigma = 1.5
)
data_params_DpR <- create_data_gen_params(params_DpR, "DpR")
str(data_params_DpR)

# Example for 'g-score' endpoint
params_gscore <- list(
  trt_n = 100, trt_p = 0.2, trt_mu = 1.5, trt_sigma = 0.5, # p=prob of zero g-score
  ctrl_n = 100, ctrl_p = 0.4, ctrl_mu = 1.2, ctrl_sigma = 0.6
)
data_params_gscore <- create_data_gen_params(params_gscore, "g-score")
str(data_params_gscore)
```

`data_gen_binary` *Generate Binary Response Data*

Description

This function simulates binary endpoint data (0/1) for multiple study arms across several simulation repetitions. Each arm's data is generated based on the specified probabilities of response using a binomial distribution. The function returns individual-level data, true response rates, and response frequencies per arm per simulation.

Usage

```
data_gen_binary(
  n_arms = 2,
  nsim = 10,
  n_list = NULL,
  prob_list = NULL,
  arm_names = NULL
)
```

Arguments

`n_arms`	Number of arms (optional, inferred from arm_names if not provided).
`nsim`	A scalar. The number of repetitions of the simulation to be performed.
`n_list`	A named list. Each element represents the number of patients in the corresponding arm (names must match arm_names). Must contain positive integers.
`prob_list`	A named list. Each element represents the probability of response (binary outcome = 1) for the corresponding arm (names must match arm_names). Must contain numeric values between 0 and 1.
`arm_names`	A character vector of arm names.

Value

A list containing:

- `data`: A data.table with columns nsim, id, arm, resp, containing all simulated binary outcomes (0/1) for all arms and all repetitions.
- `true_value`: A data frame of true response rates for each arm, along with the difference in response rates between the treatment and control arms (if both are present).
- `freq`: A data.table of response frequencies per arm per simulation, with columns for nsim, arm, count (number of responses), and n (total patients).

`data_gen_continuous` *Generate continuous outcome data for multiple arms across multiple simulations*

Description

Generates continuous outcome data by sampling from Normal distributions for each arm in each simulation replicate.

Usage

```
data_gen_continuous(
  nsim = 10,
  n_list,
  mu_list,
  sigma_list,
  arm_names,
  n_arms = length(arm_names)
)
```

Arguments

`nsim`	Number of simulations
`n_list`	Named list of sample sizes per arm (names should match arm_names). Must contain positive integers.
`mu_list`	Named list of means per arm (names should match arm_names). Must contain numeric values.
`sigma_list`	Named list of standard deviations per arm (names should match arm_names). Must contain non-negative numeric values.
`arm_names`	Character vector of arm names.
`n_arms`	Number of arms (optional, inferred from arm_names if not provided).

Value

A list containing:

- `data`: A data.table with columns nsim, id, arm, value, containing all simulated continuous values.
- `summary`: A data.table with columns nsim and arm-specific summary statistics (n, mu_hat, sd, s).
- `true_value`: A data frame with true mean values per arm and potentially a true mean difference (treatment - control).

data_gen_DpR — *Generate Constrained Depth of Response (DpR) data for multiple arms across multiple simulations*

Description

Generate Constrained Depth of Response (DpR) data for multiple arms across multiple simulations

Usage

```
data_gen_DpR(n_arms, nsim, n_list, p_list, arm_names, mu_list, sigma_list)
```

Arguments

n_arms	Number of arms (can be inferred from arm_names, but kept for consistency with template)
nsim	Number of simulations
n_list	Named list of sample sizes per arm (names should match arm_names)
p_list	Named list of proportion of -1 per arm (names should match arm_names)
arm_names	Character vector of arm names
mu_list	Named list of means per arm (theta_x) (names should match arm_names)
sigma_list	Named list of standard deviations per arm (sigma_x) (names should match arm_names)

Value

A list containing:

- data: A data.table with columns nsim, id, arm, value, containing all simulated DpR values.
- summary: A data.table with columns nsim and arm-specific summary statistics (n, mu_hat, sd, s).
- true_value: A data frame with true mean differences (currently only 'treatment' vs 'control').

data_gen_gscore — *Generate g-scores*

Description

This function generates g-scores for multiple arms of a study across multiple simulation repetitions. It simulates individual-level g-scores for each arm based on specified probabilities, means, and standard deviations for the g-score distributions. The function outputs true median values and adjusted median estimates along with their variance.

Usage

```
data_gen_gscore(
  n_arms = 2,
  nsim = 10,
  n_list = NULL,
  prob_list = NULL,
  mu_list = NULL,
  sigma_list = NULL,
  arm_names = NULL
)
```

Arguments

n_arms	A scalar. Number of arms to be generated.
nsim	A scalar. Number of repetitions for the simulation.
n_list	A named vector. Each element represents the number of patients in each arm.
prob_list	A named vector. Each element represents the probability of a g-score being zero in each arm.
mu_list	A named vector. Each element represents the mean of the normal distribution for generating the log of positive g-scores in each arm.
sigma_list	A named vector. Each element represents the standard deviation of the normal distribution for generating the log of positive g-scores in each arm.
arm_names	A named vector. Each element represents the name of each arm.

Value

A list of data frames: - data: A data frame containing individual-level g-scores for all arms and all repetitions of the simulation. - true_value: A data frame with the true medians for all arms and the true median ratio for treatment vs. control (if both are present). - median_est: A data frame with adjusted median estimates for each arm, along with the variance estimators for the median estimates.

---

eval_gscore_approx_dist

Evaluate g-score Median Ratio Estimation

Description

This function evaluates the reliability of the normal approximation for g-score median estimates. For each simulation repetition, it computes the log-median ratio and evaluates metrics such as bias, standard error, and coverage probability.

Usage

```
eval_gscore_approx_dist(sim_data)
```

Arguments

sim_data	A data frame. Data simulated by the ‘data_gen_gscore()‘ function.

Value

A data frame containing evaluation metrics for the g-score median ratio: - bias_avg: Average bias of the log-median ratio estimates. - bias_avg_median_ratio1: Average bias of the median ratio, obtained by exponentiating the average of log median ratio estimates and subtracting the true median ratio - bias_avg_median_ratio2: Average bias of the median ratio, obtained by exponentiating the log median estimates for both arms to obtain the median ratio estimates for each repetition and taking the average bias over all repetitions of simulation - sd_avg: Average standard error of the log-median ratio estimates. - sd_empirical: Empirical standard deviation of the log-median ratio estimates. - cp: Coverage probability for the 95

Examples

```
# Generating sim_data by function data_gen_gscore()
n_list <- c(treatment = 100, control = 100)
prob_list <- c(treatment = 0.2, control = 0.3)
mu_list <- c(treatment = 0.5, control = 0.6)
sigma_list <- c(treatment = 0.1, control = 0.2)
arm_names <- c(treatment = "treatment", control = "control")
sim_data <- data_gen_gscore(n_arms = 2, nsim = 10, n_list = n_list,
                            prob_list = prob_list, mu_list = mu_list,
                            sigma_list = sigma_list, arm_names = arm_names)

eval_gscore_approx_dist(sim_data = sim_data)
```

---

get_all_functions *Retrieve All Functions from an Environment*

Description

This function returns the names of all functions within a specified environment.

Usage

```
get_all_functions(env = environment())
```

Arguments

env	An environment. The environment to search for functions. The default is the current environment.

Value

A character vector. Names of all functions found in the specified environment.

get_true_gscore_median *Calculate True Median of g-scores*

Description

This function calculates the true median of g-scores generated by a log-normal distribution, accounting for zero inflation. When p = 0, i.e., no zero g-scores, the true median is just e^mu.

Usage

```
get_true_gscore_median(p, mu, sigma)
```

Arguments

p	A scalar. Probability of a g-score being zero.
mu	A scalar. Mean value of the normal distribution for generating log of positive g-scores.
sigma	A scalar. Standard deviation of the normal distribution for generating log of positive g-scores.

Value

A scalar. True median of the g-scores.

Examples

```
get_true_gscore_median(p = 0.3, mu = 1, sigma = 0.5)
```

obtain_oc *Obtain Operating Characteristics (OC)*

Description

This function calculates the operating characteristics (OC) by computing the proportion of decisions made based on the posterior probability and the credible interval from a set of decision results. It combines these proportions with the provided settings data to return a comprehensive summary.

Usage

```
obtain_oc(decisions)
```

Arguments

decisions	A data frame. Contains the decisions made in each simulation or trial, with columns for posterior probability-based decisions (‘decision_pr‘) and credible interval-based decisions (‘decision_ci‘).

Value

A data frame that includes the original settings along with the calculated proportions of decisions based on the posterior probability and credible interval. The output includes: - ‘proportion_pr‘: Proportion of decisions made based on the posterior probability. - ‘proportion_ci‘: Proportion of decisions made based on the credible interval.

`posterior_distribution` *Obtain Posterior Distribution for Single Arm*

Description

This function calculates the posterior distribution for one-arm trial data with either continuous or binary endpoints, incorporating current data and optionally borrowing from historical data. It supports the SAM prior (Self-Adaptive Mixture) for automatic conflict detection when borrowing is enabled (‘w = NULL‘) and allows specification of a fixed mixture weight for the informative component (‘w‘ is numeric). A power prior mechanism is included via the ‘ess_h‘ parameter to control the influence of historical data.

Usage

```
posterior_distribution(
  endpoint,
  current,
  historical = NULL,
  delta = NULL,
  w = NULL,
  a = 0.01,
  b = 0.01,
  a0 = 0.01,
  b0 = 0.01,
  theta0 = 0,
  s0 = 100,
  ess_h = NULL
)
```

Arguments

`endpoint`	A string. The type of endpoint, either ‘"continuous"‘ (assumes Normal likelihood and Normal-Inverse-Gamma conjugacy on mean/variance) or ‘"binary"‘ (assumes Binomial likelihood and Beta conjugacy).
`current`	A named list or vector containing current trial data: • For ‘"continuous"‘: must include ‘n‘ (number of patients), ‘mu_hat‘ (sample mean), ‘s‘ (sample standard deviation the mean). • For ‘"binary"‘: must include ‘n‘ (number of patients), ‘count‘ (number of responses/events). Sample size ‘n‘ must be positive.

`historical` A named list or vector containing historical trial data in the same format as the ‘current‘ parameter. If ‘NULL‘ (default), no historical data is used, and borrowing is disabled (‘w‘ is effectively 0). If provided, historical sample size ‘n‘ must be non-negative.

`delta` A scalar positive value, or ‘NULL‘. The clinical significant difference threshold used in the SAM prior calculation to detect prior-data conflict. Only used if ‘w‘ is ‘NULL‘. The interpretation depends on the ‘endpoint‘: for ‘"continuous"‘, it's on the mean scale; for ‘"binary"‘, it's on the probability scale. If ‘NULL‘ and ‘w‘ is ‘NULL‘, default values (0.2 for continuous, 0.1 for binary) are used.

`w` A scalar value between 0 and 1, or ‘NULL‘.

- If a numeric value (0 to 1): Specifies a fixed initial weight for the informative (historical) component of the mixture prior. Borrowing occurs according to this fixed weight.
- If ‘NULL‘ (default): The SAM prior is used. The initial weight for the informative component is automatically calculated based on the agreement between historical and current data, using ‘delta‘.

Note: If ‘historical‘ is ‘NULL‘ or ‘ess_h‘ is 0, ‘w‘ is effectively forced to 0, and no borrowing occurs regardless of the input ‘w‘.

`a` A scalar positive value. The alpha parameter for the *non-informative* Beta prior component (used only for ‘endpoint = "binary"‘). Defaults to 0.01.

`b` A scalar positive value. The beta parameter for the *non-informative* Beta prior component (used only for ‘endpoint = "binary"‘). Defaults to 0.01.

`a0` A scalar positive value. The base alpha parameter for the *informative* Beta prior component (used only for ‘endpoint = "binary"‘). This parameter (along with ‘b0‘) serves as a base prior which is updated by the discounted historical data to form the informative prior component parameters. Defaults to 0.01.

`b0` A scalar positive value. The base beta parameter for the *informative* Beta prior component (used only for ‘endpoint = "binary"‘). Defaults to 0.01.

`theta0` A scalar. The mean parameter for the *non-informative* Normal prior component (used only for ‘endpoint = "continuous"‘). Defaults to 0.

`s0` A scalar positive value. The standard deviation parameter for the *non-informative* Normal prior component (used only for ‘endpoint = "continuous"‘). Defaults to 100.

`ess_h` A scalar non-negative finite value, or ‘NULL‘. The Effective Sample Size to use for the historical data. This implements a power prior approach, where the historical data likelihood is raised to the power of ‘ess_h / nh‘. If ‘NULL‘ (default), the power is 1 (full historical data influence). If 0, historical data is fully discounted, equivalent to no borrowing.

Value

A list containing the results of the posterior calculation:

- ‘w_prior‘: The initial mixture weight used for the informative prior component *before* updating with current data. This is either the input ‘w‘ or the weight calculated by the SAM prior if ‘w‘ was ‘NULL‘. Returns 0 if no historical data was provided, historical ‘n‘ was 0, or ‘ess_h‘ was 0.
- ‘post‘: A named list containing the parameters of the resulting two-component mixture posterior distribution, suitable for use with RBesT:

- ‘w‘: The *posterior* weight of the informative component. This is calculated by updating the ‘w_prior‘ with the current data via Likelihood Ratio Test.
- For continuous (‘mixnorm‘ format): ‘mu1‘, ‘sigma1‘ (mean and SD of the informative posterior component), ‘mu2‘, ‘sigma2‘ (mean and SD of the non-informative posterior component). These will be ‘NA_real_‘ if the corresponding posterior component has zero weight (‘post$w‘ is 0 or 1).
- For binary (‘mixbeta‘ format): ‘a1‘, ‘b1‘ (parameters of the informative Beta posterior component), ‘a2‘, ‘b2‘ (parameters of the non-informative Beta posterior component). These will be ‘NA_real_‘ if the corresponding posterior component has zero weight (‘post$w‘ is 0 or 1).

Returns an error if inputs are invalid or required parameters are missing.

See Also

posterior_inference, @seealso posterior_prob, @seealso mix, @seealso mixnorm, @seealso mixbeta

Examples

```
# Continuous Endpoint - SAM prior
current_cont <- list(n = 50, mu_hat = 1.8, s = 0.5)
historical_cont <- list(n = 100, mu_hat = 1.5, s = 0.4)
# delta = 0.2 for continuous is on the mean scale
post_sam_cont <- posterior_distribution(endpoint = "continuous",
                                        current = current_cont,
                                        historical = historical_cont,
                                        delta = 0.2, theta0 = 1.6, s0 = 2)
str(post_sam_cont)

# Continuous Endpoint - Fixed weight (no SAM), with ESS discounting
post_fixed_cont_ess <- posterior_distribution(endpoint = "continuous",
                                              current = current_cont,
                                              historical = historical_cont,
                                         w = 0.6, ess_h = 40, # Use ESS=40 for historical
                                              theta0 = 1.6, s0 = 2)
str(post_fixed_cont_ess)

# Continuous Endpoint - No historical data (w is ignored)
post_nocorr_cont <- posterior_distribution(endpoint = "continuous",
                                           current = current_cont,
                                           historical = NULL,
                                           theta0 = 1.6, s0 = 2)
str(post_nocorr_cont)

# Binary Endpoint - SAM prior
current_bin <- list(n = 60, count = 45)
historical_bin <- list(n = 100, count = 60)
# delta = 0.1 for binary is on the probability scale (0 to 1)
post_sam_bin <- posterior_distribution(endpoint = "binary",
                                       current = current_bin,
                                       historical = historical_bin,
                              delta = 0.1, a = 0.5, b = 0.5, # Non-informative Beta(0.5, 0.5)
                                       a0 = 1, b0 = 1) # Base for informative Beta(1, 1)
str(post_sam_bin)
```

```
# Binary Endpoint - Fixed weight with ESS discounting
post_fixed_ess_bin <- posterior_distribution(endpoint = "binary",
                                             current = current_bin,
                                             historical = historical_bin,
                                          w = 0.8, ess_h = 30, # Use ESS=30 for historical
                                             a = 0.5, b = 0.5, a0 = 1, b0 = 1)
str(post_fixed_ess_bin)

# Binary Endpoint - No borrowing (w=0 explicitly, historical data still checked for structure)
post_noborrow_bin <- posterior_distribution(endpoint = "binary",
                                            current = current_bin,
                                historical = historical_bin, # Still provide historical if available
                                w = 0, a = 0.5, b = 0.5) # Only non-informative prior params needed
str(post_noborrow_bin)
```

`posterior_inference` *Perform Posterior Inference*

Description

This function calculates key summary statistics (posterior mean, standard deviation, and credible intervals) for one or two RBesT mixture posterior distributions. When two distributions are provided (e.g., for treatment and control arms), it also computes these statistics for the difference or ratio between them. The function supports exponentiating the results, useful when the original parameter of interest is on a log scale (e.g., log-odds, log-mean ratio).

Usage

```
posterior_inference(post1, post2 = NULL, quantiles, EXP_TRANSFORM = FALSE)
```

Arguments

`post1`	An RBesT mixture distribution object (class 'mix', 'mixnorm', 'mixbeta', etc.). Represents the posterior distribution for the first entity (e.g., treatment arm).
`post2`	An RBesT mixture distribution object (class 'mix', 'mixnorm', 'mixbeta', etc.), or 'NULL'. Represents the posterior distribution for the second entity (e.g., control arm). If 'NULL' (default), inference is performed only for 'post1'.
`quantiles`	A numeric vector of length 2 specifying the lower and upper quantiles for the credible interval (e.g., 'c(0.025, 0.975)' for a 95% credible interval). Must be sorted and within (0, 1).
`EXP_TRANSFORM`	A logical value. If 'TRUE', the calculated mean, credible interval bounds, and the comparison estimate are exponentiated ('exp(x)'). This is typically used when the mixture distribution models a parameter on the log scale (e.g., log-mean, log-odds). Note that the standard deviation ('sd1', 'sd2', 'sd_compare') is always calculated on the original scale of the input mixture distributions.

Value

A data frame with exactly one row. Contains the following columns:

`est1` Posterior mean for 'post1'. Exponentiated if 'EXP_TRANSFORM = TRUE'.

`sd1` Posterior standard deviation for ‘post1‘. Always on the original scale of ‘post1‘.

`ci1_l` Lower bound of the credible interval for ‘post1‘. Exponentiated if ‘EXP_TRANSFORM = TRUE‘.

`ci1_u` Upper bound of the credible interval for ‘post1‘. Exponentiated if ‘EXP_TRANSFORM = TRUE‘.

`est2` Posterior mean for ‘post2‘ (if ‘post2‘ is not ‘NULL‘). Exponentiated if ‘EXP_TRANSFORM = TRUE‘. ‘NA‘ if ‘post2‘ is ‘NULL‘.

`sd2` Posterior standard deviation for ‘post2‘ (if ‘post2‘ is not ‘NULL‘). Always on the original scale of ‘post2‘. ‘NA‘ if ‘post2‘ is ‘NULL‘.

`ci2_l` Lower bound of the credible interval for ‘post2‘ (if ‘post2‘ is not ‘NULL‘). Exponentiated if ‘EXP_TRANSFORM = TRUE‘. ‘NA‘ if ‘post2‘ is ‘NULL‘.

`ci2_u` Upper bound of the credible interval for ‘post2‘ (if ‘post2‘ is not ‘NULL‘). Exponentiated if ‘EXP_TRANSFORM = TRUE‘. ‘NA‘ if ‘post2‘ is ‘NULL‘.

`est_compare` Estimated treatment effect. If ‘EXP_TRANSFORM = FALSE‘, this is ‘est1 - est2‘ (difference of means/parameters). If ‘EXP_TRANSFORM = TRUE‘, this is ‘exp(est1) / exp(est2)‘ (ratio of exponentiated means/parameters). ‘NA‘ if ‘post2‘ is ‘NULL‘.

`sd_compare` Standard deviation of the *difference* (‘post1 - post2‘) on the original scale. Calculated as ‘sqrt(sd1^2 + sd2^2)‘, assuming independence. This is *not* the standard deviation of the ratio if ‘EXP_TRANSFORM = TRUE‘. ‘NA‘ if ‘post2‘ is ‘NULL‘.

`compare_ci_l` Lower bound of the credible interval for the treatment effect (‘post1 - post2‘). Exponentiated if ‘EXP_TRANSFORM = TRUE‘. ‘NA‘ if ‘post2‘ is ‘NULL‘.

`compare_ci_u` Upper bound of the credible interval for the treatment effect (‘post1 - post2‘). Exponentiated if ‘EXP_TRANSFORM = TRUE‘. ‘NA‘ if ‘post2‘ is ‘NULL‘.

See Also

`posterior_distribution`, @seealso `posterior_prob`, @seealso `summary.mix`, @seealso `qmix`, @seealso `qmixdiff`

Examples

```
# Assuming RBesT is installed and loaded
library(RBesT) # Load RBesT for example
# Example 1: Single arm, continuous (Normal mixture)
post_single_norm <- mixnorm(c(0.6, 10, 2), c(0.4, 15, 3))
inference_single <- posterior_inference(post1 = post_single_norm, quantiles = c(0.025, 0.975))
print(inference_single)

# Example 2: Two arms, continuous (Difference)
post_ctrl_norm <- mixnorm(c(1, 12, 2.5)) # Control arm posterior
inference_diff <- posterior_inference(post1 = post_single_norm, post2 = post_ctrl_norm,
                                      quantiles = c(0.025, 0.975))
print(inference_diff)

# Example 3: Two arms, binary (Beta mixture), inference on probability scale
post_trt_beta <- mixbeta(c(0.7, 10, 5), c(0.3, 2, 8)) # Treatment posterior for probability
post_ctrl_beta <- mixbeta(c(1, 8, 12)) # Control posterior for probability
inference_prob_diff <- posterior_inference(post1 = post_trt_beta, post2 = post_ctrl_beta,
                                           quantiles = c(0.05, 0.95))
print(inference_prob_diff)

# Example 4: Two arms, log-transformed continuous data (e.g., log-expression), inference on ratio scale
```

```
# Assume post_log_trt and post_log_ctrl are posteriors on the log scale
post_log_trt <- mixnorm(c(1, log(10), 0.5))
post_log_ctrl <- mixnorm(c(1, log(5), 0.6))
inference_log_ratio <- posterior_inference(post1 = post_log_trt, post2 = post_log_ctrl,
                                    quantiles = c(0.025, 0.975), EXP_TRANSFORM = TRUE)
# Note: est_compare is exp(log(10) - log(5)) = 10/5 = 2
# credible interval is exp(CI_log_diff) = exp(CI_log_trt - CI_log_ctrl) = CI_ratio
print(inference_log_ratio)
```

`posterior_prob` *Calculate Posterior Probability*

Description

This function calculates the posterior probability of a parameter estimate (or the difference between two parameter estimates) falling within a specified range, based on provided RBesT mixture posterior distributions. For a single posterior distribution, it computes the probability of the parameter being greater than, less than, or between specified values. For two posterior distributions, it computes the probability of the difference between the first and second distributions falling within the range.

Usage

```
posterior_prob(
  range_type = "greater",
  post1,
  post2 = NULL,
  value,
  value2 = NULL
)
```

Arguments

`range_type`	A character string. Specifies the type of comparison range. Must be one of ‘"greater"‘, ‘"less"‘, or ‘"between"‘. Defaults to ‘"greater"‘.
`post1`	An RBesT mixture distribution object (class ‘mix‘, ‘mixnorm‘, ‘mixbeta‘, etc.). Represents the first posterior distribution.
`post2`	An RBesT mixture distribution object (class ‘mix‘, ‘mixnorm‘, ‘mixbeta‘, etc.). Optional second posterior distribution object for pairwise comparison of ‘post1 - post2‘. If ‘NULL‘ (default), the probability is calculated for ‘post1‘ itself.
`value`	A scalar numeric value. The threshold used for comparison. If ‘range_type‘ is "greater" or "less", this is the single threshold. If ‘range_type‘ is "between", this is the lower bound of the interval.
`value2`	A scalar numeric value. The upper bound for the "between" comparison. This parameter is required if ‘range_type‘ is ‘"between"‘.

Value

A scalar numeric value representing the calculated posterior probability, between 0 and 1. Returns ‘NA‘ if there is an error during the probability calculation (e.g., ‘pmixdiff‘ fails for two distributions).

See Also

posterior_distribution, @seealso posterior_inference, @seealso pmix, @seealso pmixdiff

Examples

```
# Assuming RBesT is installed and loaded
library(RBesT) # Load RBesT for example
# Example 1: Single arm - Probability of being greater than a value
post_single_norm <- mixnorm(c(0.6, 10, 2), c(0.4, 15, 3))
prob_greater <- posterior_prob(post1 = post_single_norm, value = 12, range_type = "greater")
cat("P(post_single_norm > 12):", prob_greater, "\n")

# Example 2: Single arm - Probability of being less than a value
prob_less <- posterior_prob(post1 = post_single_norm, value = 11, range_type = "less")
cat("P(post_single_norm < 11):", prob_less, "\n")

# Example 3: Single arm - Probability of being between two values
prob_between <- posterior_prob(post1 = post_single_norm, value = 11,
                               range_type = "between", value2 = 14)
cat("P(11 < post_single_norm < 14):", prob_between, "\n")

# Example 4: Two arms - Probability of difference > 0 (post1 > post2)
post_ctrl_norm <- mixnorm(c(1, 12, 2.5)) # Control arm posterior
prob_diff_greater_0 <- posterior_prob(post1 = post_single_norm, post2 = post_ctrl_norm,
                                      value = 0, range_type = "greater")
cat("P(post_single_norm - post_ctrl_norm > 0):", prob_diff_greater_0, "\n")

# Example 5: Two arms - Probability of difference < a value
prob_diff_less_neg1 <- posterior_prob(post1 = post_single_norm, post2 = post_ctrl_norm,
                                      value = -1, range_type = "less")
cat("P(post_single_norm - post_ctrl_norm < -1):", prob_diff_less_neg1, "\n")

# Example 6: Two arms - Probability of difference between two values
prob_diff_between <- posterior_prob(post1 = post_single_norm, post2 = post_ctrl_norm,
                                    value = -2, range_type = "between", value2 = 1)
cat("P(-2 < post_single_norm - post_ctrl_norm < 1):", prob_diff_between, "\n")
```

---

sd_gscore_median *Variance Estimator for g-score Median Estimate*

Description

This function calculates the variance estimator for the median estimate of g-scores, based on the method proposed by Price & Bonett (2001).

Usage

```
sd_gscore_median(x)
```

Arguments

x	A vector of g-scores.

Value

A scalar. Variance estimator for the g-score median.

Examples

```
g_scores <- simulate_gscore(n = 100, mu = 1, sigma = 0.5, prob_zero = 0.2)
sd_gscore_median(g_scores)
```

`simulate_gscore` *Simulate g-scores with Zero Inflation*

Description

This function generates g-scores by simulating positive values from a log-normal distribution, with added zero inflation. The probability of a g-score being zero is controlled by 'prob_zero', while positive values are generated as log-normal.

Usage

```
simulate_gscore(n, mu, sigma, prob_zero)
```

Arguments

`n`	A scalar. Number of patients in the arm of interest.
`mu`	A scalar. Mean value of the normal distribution for generating the log of positive g-scores.
`sigma`	A scalar. Standard deviation of the normal distribution for generating the log of positive g-scores.
`prob_zero`	A scalar. Probability of a g-score being zero.

Details

P(G=0) = p, P(G>0) = 1-p; log(G) | G > 0 ~ N(mu, sigma^2)

Value

A vector. Simulated g-scores.

Examples

```
simulate_gscore(n = 100, mu = 1, sigma = 0.5, prob_zero = 0.2)
```