Package ‘booami’

September 4, 2025

Language en-US

Type Package

Title Component-Wise Gradient Boosting after Multiple Imputation

Version 0.1.0

Description Component-wise gradient boosting for analysis of multiply imputed datasets. Implements the algorithm Boosting after Multiple Imputation (MIBoost), which enforces uniform variable selection across imputations and provides utilities for pooling. Includes a cross-validation workflow that first splits the data into training and validation sets and then performs imputation on the training data, applying the learned imputation models to the validation data to avoid information leakage. Supports Gaussian and logistic loss. Methods relate to gradient boosting and multiple imputation as in Buehlmann and Hothorn (2007) <doi:10.1214/07-STS242>, Friedman (2001) <doi:10.1214/aos/1013203451>, and van Buuren (2018, ISBN:9781138588318) and Groothuis-Oudshoorn (2011) <doi:10.18637/jss.v045.i03>; see also Kuchen (2025) <doi:10.48550/arXiv.2507.21807>.

License MIT + file LICENSE

URL https://arxiv.org/abs/2507.21807,
https://github.com/RobertKuchen/booami

BugReports https://github.com/RobertKuchen/booami/issues

Encoding UTF-8

Depends R (>= 4.0)

Imports MASS, stats, utils, withr

Suggests mice, miceadds, Matrix, knitr, rmarkdown, testthat (>= 3.0.0), spelling

Config/testthat/edition 3

RoxygenNote 7.3.2

LazyData true

NeedsCompilation no

Author Robert Kuchen [aut, cre]

Maintainer Robert Kuchen <rokuchen@uni-mainz.de>

Repository CRAN

Date/Publication 2025-09-04 14:10:06 UTC

Contents

`booami_predict` *Predict with booami models*

Description

Minimal, dependency-free predictor for models fitted by `cv_boost_raw`, `cv_boost_imputed`, or a *pooled* `impu_boost` fit. Supports Gaussian (identity) and logistic (logit) models, returning either the linear predictor or, for logistic, predicted probabilities.

Usage

```
booami_predict(
  object,
  X_new,
  family = NULL,
  type = c("response", "link"),
  center_means = NULL
)
```

Arguments

`object`	A fit returned by `cv_boost_raw()`, `cv_boost_imputed()`, or a *pooled* `impu_boost()` (i.e., `pool = TRUE` so that `$BETA` is a length-p vector and `$INT` is a scalar).
`X_new`	New data (matrix or data.frame) with the same p predictors the model was trained on. If column names are present in the model, `X_new` will be aligned by name; otherwise it must be in the same order.
`family`	Model family; one of `c("gaussian","logistic")`. If `NULL`, the function tries to infer from `object$type` or attributes; otherwise defaults to `"gaussian"`.

`type` Prediction type; one of `c("response","link")`. For `"gaussian"`, both are identical. For `"logistic"`, `"response"` returns probabilities via the inverse-logit.

`center_means` Optional numeric vector of length p with training means used to center predictors during fitting. If provided, `X_new` is centered as `X_new - center_means` before prediction. If the model stores means by name, pass a named vector whose names match predictor names.

Details

This function is deterministic and involves no random number generation. Coefficients are extracted from either `$final_model` (intercept first, then coefficients) or from `$INT+$BETA` (pooled `impu_boost`). If `X_new` has column names and the model has named coefficients, columns are aligned by name; otherwise they are used in order.

If your training pipeline centered covariates (e.g., `center = "auto"`), providing the same `center_means` here yields numerically consistent predictions. If not supplied but `object$center_means` exists, it will be used automatically. If both are supplied, the explicit `center_means` argument takes precedence.

Value

A numeric vector of predictions (length `nrow(X_new)`). If `X_new` has row names, they are propagated to the returned vector.

See Also

`cv_boost_raw`, `cv_boost_imputed`, `impu_boost`

Examples

```
# 1) Fit on data WITH missing values
set.seed(123)
sim_tr <- simulate_booami_data(
  n = 120, p = 12, p_inf = 3,
  type = "gaussian",
  miss = "MAR", miss_prop = 0.20
)
X_tr <- sim_tr$data[, 1:12]
y_tr <- sim_tr$data$y

fit <- cv_boost_raw(
  X_tr, y_tr,
  k = 2, mstop = 50, seed = 123,
  impute_args    = list(m = 2, maxit = 1, printFlag = FALSE, seed = 1),
  quickpred_args = list(method = "spearman", mincor = 0.30, minpuc = 0.60),
  show_progress  = FALSE
)

# 2) Predict on a separate data set WITHOUT missing values (same p)
```

```
sim_new <- simulate_booami_data(
  n = 5, p = 12, p_inf = 3,
  type = "gaussian",
  miss = "MCAR", miss_prop = 0   # <- complete data with existing API
)
X_new <- sim_new$data[, 1:12, drop = FALSE]

preds <- booami_predict(fit, X_new = X_new, family = "gaussian", type = "response")
round(preds, 3)
```

---

booami_sim *Example dataset for 'booami' (Gaussian, MAR)*

Description

A simulated dataset with predictors `X1`...`X25` and a continuous outcome `y`, with missing values generated under a MAR mechanism. The object is a `data.frame` and carries attributes describing the data-generating process (true coefficients, informative indices, etc.).

Format

A data frame with 300 rows and 26 variables:

X1 numeric

X2 numeric

X3 numeric

X4 numeric

X5 numeric

X6 numeric

X7 numeric

X8 numeric

X9 numeric

X10 numeric

X11 numeric

X12 numeric

X13 numeric

X14 numeric

X15 numeric

X16 numeric

X17 numeric

X18 numeric

X19 numeric

X20 numeric

X21 numeric

X22 numeric

X23 numeric

X24 numeric

X25 numeric

y numeric outcome

Details

Generated by `simulate_booami_data` with typical settings (see `?simulate_booami_data`). The following attributes are attached to `booami_sim`:

- `"true_beta"`: numeric length-25 vector of true coefficients (non-zeros in positions 1-5).
- `"informative"`: integer vector `1:5`.
- `"type"`: `"gaussian"`.
- `"corr_structure"`: `"all_ar1"`; `"rho"`: 0.3.
- `"intercept"`: 1; `"noise_sd"`: 1 (Gaussian; `NA` otherwise).
- `"mar_scale"`: `TRUE`; `"keep_mar_drivers"`: `TRUE`.

See Also

`simulate_booami_data`, `impu_boost`, `cv_boost_raw`, `cv_boost_imputed`

Examples

```
## \donttest{
utils::data(booami_sim)
dim(booami_sim)
mean(colSums(is.na(booami_sim)) > 0)  # fraction of columns with any NAs
head(attr(booami_sim, "true_beta"))
attr(booami_sim, "informative")
## }
```

`cv_boost_imputed` *Cross-validated boosting on already-imputed data*

Description

Performs k-fold cross-validation for `impu_boost` to determine the optimal value of `mstop` before fitting the final model on the full dataset. This function should only be used when data have already been imputed. In most cases, it is preferable to provide unimputed data and use `cv_boost_raw` instead.

Usage

```
cv_boost_imputed(
  X_train_list,
  y_train_list,
  X_val_list,
  y_val_list,
  X_full,
  y_full,
  ny = 0.1,
  mstop = 250,
  type = c("gaussian", "logistic"),
  MIBoost = TRUE,
  pool = TRUE,
  pool_threshold = 0,
  show_progress = TRUE,
  center = c("auto", "off", "force")
)
```

Arguments

`X_train_list`	A list of length k. Element `i` is itself a list of length M containing the $n_{train} \times p$ numeric design matrices for each imputed dataset in CV fold `i`.
`y_train_list`	A list of length k. Element `i` is a list of length M, where each element is a length-n_{train} numeric response vector aligned with `X_train_list[[i]][[m]]`.
`X_val_list`	A list of length k. Element `i` is a list of length M containing the $n_{val} \times p$ numeric validation matrices matched to the corresponding imputed training dataset in fold `i`.
`y_val_list`	A list of length k. Element `i` is a list of length M, where each element is a length-n_{val} continuous response vector aligned with `X_val_list[[i]][[m]]`.
`X_full`	A list of length M with the $n \times p$ numeric full-data design matrices (one per imputed dataset) used to fit the final model.
`y_full`	A list of length M, where each element is a length-n continuous response vector corresponding to the imputed dataset in `X_full`.
`ny`	Learning rate. Defaults to `0.1`.
`mstop`	Maximum number of boosting iterations to evaluate during cross-validation. The selected `mstop` is the value that minimizes the mean CV error over `1:mstop`. Default is `250`.
`type`	Type of loss function. One of: `"gaussian"` (mean squared error) for continuous responses, or `"logistic"` (binomial deviance) for binary responses.
`MIBoost`	Logical. If `TRUE`, applies the MIBoost algorithm, which enforces uniform variable selection across all imputed datasets. If `FALSE`, variables are selected independently within each imputed dataset, and pooling is governed by `pool_threshold`.
`pool`	Logical. If `TRUE`, models across the M imputed datasets are aggregated into a single final model. If `FALSE`, M separate models are returned.

`pool_threshold`	Only used when `MIBoost = FALSE` and `pool = TRUE`. Controls the pooling rule when aggregating the M models obtained from the imputed datasets into a single final model. A candidate variable is included only if it is selected in at least `pool_threshold` (a value in (0, 1]) proportion of the imputed datasets; coefficients of all other variables are set to zero. A value of `0` corresponds to estimate-averaging, while values > `0` correspond to selection-frequency thresholding.
`show_progress`	Logical; print fold-level progress and summary timings. Default `TRUE`.
`center`	One of `c("auto", "off", "force")`. Controls centering of `X` within each imputed dataset. With `"auto"` (recommended), centering is applied only if the training matrix is not already centered. With `"force"`, centering is always applied. With `"off"`, centering is skipped. If `X_val_list` is provided, validation sets are centered using the means from the corresponding training set.

Details

To avoid data leakage, each CV fold should first be split into training and validation subsets, after which imputation is performed. For the final model, all data should be imputed independently.

The recommended workflow is illustrated in the examples.

Centering affects only `X`; `y` is left unchanged. For `type = "logistic"`, responses are treated as numeric `0/1` via the logistic link. Validation loss is averaged over imputations and then over folds.

Value

A list with:

- `CV_error`: numeric vector of length `mstop` with the mean cross-validated loss across folds (and imputations).
- `best_mstop`: integer index of the minimizing entry in `CV_error`.
- `final_model`: numeric vector of length `1 + p` containing the intercept followed by p coefficients of the final pooled model fitted at `best_mstop` on `X_full`/`y_full`.

References

Kuchen, R. (2025). *MIBoost: A Gradient Boosting Algorithm for Variable Selection After Multiple Imputation*. arXiv:2507.21807. doi:10.48550/arXiv.2507.21807 https://arxiv.org/abs/2507.21807.

See Also

`impu_boost`, `cv_boost_raw`

Examples

```
set.seed(123)
utils::data(booami_sim)
k <- 2; M <- 2
```

```
n <- nrow(booami_sim); p <- ncol(booami_sim) - 1
folds <- sample(rep(seq_len(k), length.out = n))

X_train_list <- vector("list", k)
y_train_list <- vector("list", k)
X_val_list   <- vector("list", k)
y_val_list   <- vector("list", k)

for (cv in seq_len(k)) {
  tr <- folds != cv
  va <- !tr
  dat_tr <- booami_sim[tr, , drop = FALSE]
  dat_va <- booami_sim[va, , drop = FALSE]
  pm_tr  <- mice::quickpred(dat_tr, method = "spearman", mincor = 0.30, minpuc = 0.60)
 imp_tr <- mice::mice(dat_tr, m = M, predictorMatrix = pm_tr, maxit = 1, printFlag = FALSE)
  imp_va <- mice::mice.mids(imp_tr, newdata = dat_va, maxit = 1, printFlag = FALSE)
  X_train_list[[cv]] <- vector("list", M)
  y_train_list[[cv]] <- vector("list", M)

  X_val_list[[cv]]   <- vector("list", M)
  y_val_list[[cv]]   <- vector("list", M)
  for (m in seq_len(M)) {
    tr_m <- mice::complete(imp_tr, m)
    va_m <- mice::complete(imp_va, m)
    X_train_list[[cv]][[m]] <- data.matrix(tr_m[, 1:p, drop = FALSE])
    y_train_list[[cv]][[m]] <- tr_m$y
    X_val_list[[cv]][[m]]   <- data.matrix(va_m[, 1:p, drop = FALSE])
    y_val_list[[cv]][[m]]   <- va_m$y
  }
}

pm_full  <- mice::quickpred(booami_sim, method = "spearman", mincor = 0.30, minpuc = 0.60)
imp_full <- mice::mice(booami_sim, m = M, predictorMatrix = pm_full, maxit = 1, printFlag = FALSE)
X_full <- lapply(seq_len(M),
function(m) data.matrix(
mice::complete(imp_full, m)[, 1:p, drop = FALSE]))
y_full <- lapply(seq_len(M), function(m) mice::complete(imp_full, m)$y)

res <- cv_boost_imputed(
  X_train_list, y_train_list,
  X_val_list,   y_val_list,
  X_full,       y_full,
  ny = 0.1, mstop = 50, type = "gaussian",
  MIBoost = TRUE, pool = TRUE, center = "auto",
  show_progress = FALSE
)



set.seed(2025)
utils::data(booami_sim)
k <- 5; M <- 10
```

```
n <- nrow(booami_sim); p <- ncol(booami_sim) - 1
folds <- sample(rep(seq_len(k), length.out = n))

X_train_list <- vector("list", k)
y_train_list <- vector("list", k)
X_val_list   <- vector("list", k)
y_val_list   <- vector("list", k)
for (cv in seq_len(k)) {
  tr <- folds != cv; va <- !tr
  dat_tr <- booami_sim[tr, , drop = FALSE]
  dat_va <- booami_sim[va, , drop = FALSE]
  pm_tr  <- mice::quickpred(dat_tr, method = "spearman", mincor = 0.20, minpuc = 0.40)
 imp_tr <- mice::mice(dat_tr, m = M, predictorMatrix = pm_tr, maxit = 5, printFlag = TRUE)
  imp_va <- mice::mice.mids(imp_tr, newdata = dat_va, maxit = 1, printFlag = FALSE)
  X_train_list[[cv]] <- vector("list", M)
  y_train_list[[cv]] <- vector("list", M)
  X_val_list[[cv]]   <- vector("list", M)
  y_val_list[[cv]]   <- vector("list", M)
  for (m in seq_len(M)) {
    tr_m <- mice::complete(imp_tr, m); va_m <- mice::complete(imp_va, m)
    X_train_list[[cv]][[m]] <- data.matrix(tr_m[, 1:p, drop = FALSE])
    y_train_list[[cv]][[m]] <- tr_m$y
    X_val_list[[cv]][[m]]   <- data.matrix(va_m[, 1:p, drop = FALSE])
    y_val_list[[cv]][[m]]   <- va_m$y
  }
}
pm_full  <- mice::quickpred(booami_sim, method = "spearman", mincor = 0.20, minpuc = 0.40)
imp_full <- mice::mice(booami_sim, m = M, predictorMatrix = pm_full, maxit = 5, printFlag = TRUE)
X_full <- lapply(seq_len(M),
function(m) data.matrix(mice::complete(imp_full, m)[, 1:p, drop = FALSE]))
y_full <- lapply(seq_len(M),
function(m) mice::complete(imp_full, m)$y)

res_heavy <- cv_boost_imputed(
  X_train_list, y_train_list,
  X_val_list,   y_val_list,
  X_full,       y_full,
  ny = 0.1, mstop = 250, type = "gaussian",
  MIBoost = TRUE, pool = TRUE, center = "auto",
  show_progress = TRUE
)
str(res_heavy)
```

`cv_boost_raw` *Cross-Validated Component-Wise Gradient Boosting with Multiple Imputation Performed Inside Each Fold*

Description

Performs k-fold cross-validation for `impu_boost` on data with missing values. Within each fold, multiple imputation, centering, model fitting, and validation are performed in a leakage-avoiding manner to select the optimal number of boosting iterations (`mstop`). The final model is then fitted on multiple imputations of the full dataset at the selected stopping iteration.

Usage

```
cv_boost_raw(
  X,
  y,
  k = 5,
  ny = 0.1,
  mstop = 250,
  type = c("gaussian", "logistic"),
  MIBoost = TRUE,
  pool = TRUE,
  pool_threshold = 0,
  impute_args = list(m = 10, maxit = 5, printFlag = FALSE),
  impute_method = NULL,
  use_quickpred = TRUE,
 quickpred_args = list(mincor = 0.1, minpuc = 0.5, method = NULL, include = NULL,
    exclude = NULL),
  seed = 123,
  show_progress = TRUE,
  return_full_imputations = FALSE,
  center = "auto"
)
```

Arguments

`X`	A data.frame or matrix of predictors of size $n \times p$ containing missing values. Column names are preserved. If no missing values are present in `X` or `y`, use `cv_boost_imputed` instead.
`y`	A vector of length n with the outcome (numeric for `type = "gaussian"`; numeric `0/1` or a 2-level factor for `type = "logistic"`). Must align with `X` rows.
`k`	Number of cross-validation folds. Default is `5`.
`ny`	Learning rate. Defaults to `0.1`.
`mstop`	Maximum number of boosting iterations to evaluate during cross-validation. The selected `mstop` is the value minimizing the mean CV error over `1:mstop`. Default is `250`.
`type`	Type of loss function. One of: `"gaussian"` (mean squared error) for continuous responses, or `"logistic"` (binomial deviance) for binary responses.
`MIBoost`	Logical. If `TRUE`, applies the MIBoost algorithm, which enforces uniform variable selection across all imputed datasets. If `FALSE`, variables are selected independently within each imputed dataset, and pooling is governed by `pool_threshold`.

pool	Logical. If TRUE, models across the M imputed datasets are aggregated into a single final model. If FALSE, M separate models are returned.
pool_threshold	Only used when MIBoost = FALSE and pool = TRUE. Controls the pooling rule when aggregating the M models obtained from the imputed datasets into a single final model. A candidate variable is included only if it is selected in at least pool_threshold (a value in (0, 1) proportion of the imputed datasets; coefficients of all other variables are set to zero. A value of 0 corresponds to estimate-averaging, while values > 0 correspond to selection-frequency thresholding.
impute_args	A named list of arguments forwarded to mice::mice() both inside CV and on the full dataset (e.g., m, maxit, seed, printFlag, etc.). Internally, data, predictorMatrix, and ignore are set by the function and will override any values supplied here. If m is missing, a default of 10 is used.
impute_method	Optional *named* character vector passed to mice::mice(method = ...) to control per-variable methods (e.g., "pmm", "logreg"). This may be a *partial* vector: entries are merged by name into a full default method vector derived from the data; unmatched names are ignored with a warning. If NULL (default), numeric columns use "pmm"; for type = "logistic", the outcome uses "logreg" (coerced to a 2-level factor if needed).
use_quickpred	Logical. If TRUE (default), build the predictorMatrix via mice::quickpred() on the training data within each fold; otherwise use mice::make.predictorMatrix().
quickpred_args	A named list of arguments forwarded to mice::quickpred() (e.g., mincor, minpuc, method, include, exclude). Ignored when use_quickpred = FALSE.
seed	Base random seed for fold assignment. If impute_args$seed is *not* supplied, this value also seeds imputation; otherwise the user-specified impute_args$seed is respected and deterministically offset per fold. RNG state is restored on exit. Default 123.
show_progress	Logical. If TRUE (default), print progress for the imputation and boosting phases, plus a summary at completion.
return_full_imputations	Logical. If TRUE, attach the list of full-data imputations used for the final fit as $full_imputations = list(X = <list length m>, y = <list length m>). Default is FALSE.
center	One of c("auto", "off", "force"). Controls centering of X within each imputed dataset. With "auto" (recommended), centering is applied only if the training matrix is not already centered. With "force", centering is always applied. With "off", centering is skipped. Validation sets are always centered using the means from the corresponding training set.

Details

Within each CV fold, the data are first split into a training subset and a validation subset. The training subset is multiply imputed M times using **mice**, producing M imputed training datasets. Covariates in each training dataset are centered. The corresponding validation subset is then imputed M times using the imputation models learned from the training imputations, ensuring consistency between training and validation. These validation datasets are centered using the variable means from their associated training datasets.

impu_boost is run on the imputed training datasets for up to mstop boosting iterations. At each iteration, prediction errors are computed on the corresponding validation datasets and averaged across imputations. This yields an aggregated error curve per fold, which is then averaged across folds. The optimal stopping iteration is chosen as the mstop value minimizing the mean CV error.

Finally, the full dataset is multiply imputed M times and centered independently within each imputed dataset. impu_boost is applied to these datasets for the selected number of boosting iterations to obtain the final model.

Imputation control. All key **mice** settings can be passed via impute_args (a named list forwarded to mice::mice()) and/or impute_method (a named character vector of per-variable methods). Internally, the function builds a full default method vector from the actual data given to mice(), then *merges* any user-supplied entries by name. *The names in* impute_method *must exactly match the column names in* data.frame(y = y, X) *(i.e., the data passed to* mice()*).* Partial vectors are allowed; variables not listed fall back to defaults; unknown names are ignored with a warning. The function sets and may override data, method (after merging overrides), predictorMatrix, and ignore (to enforce train-only learning). Predictor matrices can be built with mice::quickpred() (see use_quickpred, quickpred_args) or with mice::make.predictorMatrix().

Value

A list with:

- CV_error: numeric vector (length mstop) of mean CV loss.
- best_mstop: integer index minimizing CV_error.
- final_model: numeric vector of length 1 + p with the intercept and pooled coefficients of the final fit on full-data imputations at best_mstop.
- full_imputations: (optional) when return_full_imputations=TRUE, a list list(X = <list length m>, y = <list length m>) containing the full-data imputations used for the final model.
- folds: integer vector of length n giving the CV fold id for each observation (1..k).

References

Kuchen, R. (2025). *MIBoost: A Gradient Boosting Algorithm for Variable Selection After Multiple Imputation*. arXiv:2507.21807. doi:10.48550/arXiv.2507.21807 https://arxiv.org/abs/2507.21807.

See Also

impu_boost, cv_boost_imputed, **mice**

Examples

```
utils::data(booami_sim)
X <- booami_sim[, 1:25]
y <- booami_sim[, 26]

res <- cv_boost_raw(
  X = X, y = y,
```

```
  k = 2, seed = 123,
  impute_args    = list(m = 2, maxit = 1, printFlag = FALSE, seed = 1),
  quickpred_args = list(mincor = 0.30, minpuc = 0.60),
  mstop = 50,
  show_progress = FALSE
)


# Partial custom imputation method override
meth <- c(y = "pmm", X1 = "pmm")
res2 <- cv_boost_raw(
  X = X, y = y,
  k = 2, seed = 123,
  impute_args    = list(m = 2, maxit = 1, printFlag = FALSE, seed = 456),
  quickpred_args = list(mincor = 0.30, minpuc = 0.60),
  mstop = 50,
  impute_method  = meth,
  show_progress = FALSE
)
```

impu_boost *Component-Wise Gradient Boosting Across Multiply Imputed Datasets*

Description

Applies component-wise gradient boosting to multiply imputed datasets. Depending on the settings, either a separate model is reported for each imputed dataset, or the M models are pooled to yield a single final model. For pooling, one can choose the algorithm *MIBoost* (Boosting after Multiple Imputation), which enforces a uniform variable-selection scheme across all imputations, or the more conventional approaches of estimate-averaging and selection-frequency thresholding.

Usage

```
impu_boost(
  X_list,
  y_list,
  X_list_val = NULL,
  y_list_val = NULL,
  ny = 0.1,
  mstop = 250,
  type = c("gaussian", "logistic"),
  MIBoost = TRUE,
  pool = TRUE,
  pool_threshold = 0,
  center = "auto"
)
```

Arguments

X_list	List of length M; each element is an $n \times p$ numeric predictor matrix from one imputed dataset.
y_list	List of length M; each element is a length-n numeric response vector from one imputed dataset.
X_list_val	Optional validation list (same structure as X_list).
y_list_val	Optional validation list (same structure as y_list).
ny	Learning rate. Defaults to 0.1.
mstop	Number of boosting iterations (default 250).
type	Type of loss function. One of: "gaussian" (mean squared error) for continuous responses, or "logistic" (binomial deviance) for binary responses.
MIBoost	Logical. If TRUE, applies the MIBoost algorithm, which enforces uniform variable selection across all imputed datasets. If FALSE, variables are selected independently within each imputed dataset, and pooling is governed by pool_threshold.
pool	Logical. If TRUE, models across the M imputed datasets are aggregated into a single final model. If FALSE, M separate models are returned.
pool_threshold	Only used when MIBoost = FALSE and pool = TRUE. Controls the pooling rule when aggregating the M models obtained from the imputed datasets into a single final model. A candidate variable is included only if it is selected in at least pool_threshold (a value in (0, 1)) proportion of the imputed datasets; coefficients of all other variables are set to zero. A value of 0 corresponds to estimate-averaging, while values > 0 correspond to selection-frequency thresholding.
center	One of c("auto", "off", "force"). Controls centering of X within each imputed dataset. With "auto" (recommended), centering is applied only if the training matrix is not already centered. With "force", centering is always applied. With "off", centering is skipped. If X_list_val is provided, validation sets are centered using the means from the corresponding training set.

Details

This function supports *MIBoost*, which enforces uniform variable selection across multiply imputed datasets. For full methodology, see the references below.

Value

A list with elements:

- INT: intercept(s). A scalar if pool = TRUE, otherwise a length-M vector.
- BETA: coefficient estimates. A length-p vector if pool = TRUE, otherwise an M $\times$ p matrix.
- CV_error: vector of validation errors (if validation data were provided), otherwise NULL.

References

Buehlmann, P. and Hothorn, T. (2007). "Boosting Algorithms: Regularization, Prediction and Model Fitting." doi:10.1214/07STS242

Friedman, J. H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine." doi:10.1214/aos/1013203451

van Buuren, S. and Groothuis-Oudshoorn, K. (2011). "mice: Multivariate Imputation by Chained Equations in R." doi:10.18637/jss.v045.i03

Kuchen, R. (2025). "MIBoost: A Gradient Boosting Algorithm for Variable Selection After Multiple Imputation." doi:10.48550/arXiv.2507.21807

See Also

simulate_booami_data, cv_boost_raw, cv_boost_imputed

Examples

```
set.seed(123)
utils::data(booami_sim)

M <- 2
n <- nrow(booami_sim)
x_cols <- grepl("^X\\d+$", names(booami_sim))

tr_idx <- sample(seq_len(n), floor(0.8 * n))
dat_tr <- booami_sim[tr_idx, , drop = FALSE]
dat_va <- booami_sim[-tr_idx, , drop = FALSE]

pm_tr <- mice::quickpred(dat_tr, method = "spearman",
                         mincor = 0.30, minpuc = 0.60)

imp_tr <- mice::mice(dat_tr, m = M, predictorMatrix = pm_tr,
                     maxit = 1, printFlag = FALSE)
imp_va <- mice::mice.mids(imp_tr, newdata = dat_va, maxit = 1, printFlag = FALSE)

X_list      <- vector("list", M)
y_list      <- vector("list", M)
X_list_val  <- vector("list", M)
y_list_val  <- vector("list", M)
for (m in seq_len(M)) {
  tr_m <- mice::complete(imp_tr, m)
  va_m <- mice::complete(imp_va, m)
  X_list[[m]]     <- data.matrix(tr_m[, x_cols, drop = FALSE])
  y_list[[m]]     <- tr_m$y
  X_list_val[[m]] <- data.matrix(va_m[, x_cols, drop = FALSE])
  y_list_val[[m]] <- va_m$y
}

fit <- impu_boost(
  X_list, y_list,
  X_list_val = X_list_val, y_list_val = y_list_val,
```

```
  ny = 0.1, mstop = 50, type = "gaussian",
  MIBoost = TRUE, pool = TRUE, center = "auto"
)

which.min(fit$CV_error)
head(fit$BETA)
fit$INT
```

---

predict.booami *Predict from booami objects*

Description

Predict responses (link or response scale) from fitted booami models.

Usage

```
## S3 method for class 'booami_cv'
predict(object, newdata, type = c("link", "response"), ...)

## S3 method for class 'booami_pooled'
predict(object, newdata, type = c("link", "response"), ...)

## S3 method for class 'booami_multi'
predict(object, newdata, type = c("link", "response"), ...)
```

Arguments

`object`	A fitted booami object. One of: • `"booami_cv"` — cross-validated model object. • `"booami_pooled"` — pooled fit from `impu_boost(..., pool = TRUE)`. • `"booami_multi"` — unpooled fit from `impu_boost(..., pool = FALSE)`.
`newdata`	A data.frame or matrix of predictors (same columns/order as training).
`type`	Either `"link"` for the linear predictor, or `"response"` for mean/probability (Gaussian/logistic respectively).
`...`	Passed to `booami_predict`. For `"booami_multi"`, you may use `aggregate = "mean" | "median" | NULL` and/or `which_m = <index>` to control how predictions are aggregated across imputations.

Value

A numeric vector of predictions.

See Also

`booami_predict`

simulate_booami_data *Simulate a Booami Example Dataset with Missing Values*

Description

Generates a dataset with p predictors, of which the first p_inf are informative. Predictors are drawn from a multivariate normal with a chosen correlation structure, and the outcome can be continuous (type = "gaussian") or binary (type = "logistic"). Missing values are introduced via MAR or MCAR.

Usage

```
simulate_booami_data(
  n = 300,
  p = 25,
  p_inf = 5,
  rho = 0.3,
  type = c("gaussian", "logistic"),
  beta_range = c(1, 2),
  intercept = 1,
  corr_structure = c("all_ar1", "informative_cs", "blockdiag", "none"),
  rho_noise = NULL,
  noise_sd = 1,
  miss = c("MAR", "MCAR"),
  miss_prop = 0.25,
  mar_drivers = c(1, 2, 3),
  gamma_vec = NULL,
  calibrate_mar = FALSE,
  mar_scale = TRUE,
  keep_observed = integer(0),
  jitter_sd = 0.25,
  keep_mar_drivers = TRUE
)
```

Arguments

n	Number of observations (default 300).
p	Total number of predictors (default 25).
p_inf	Number of informative predictors (default 5); must satisfy p_inf <= p.
rho	Correlation parameter (interpretation depends on corr_structure).
type	Either "gaussian" or "logistic" (default "gaussian").
beta_range	Length-2 numeric; coefficients for the first p_inf informative predictors are drawn i.i.d. Uniform(beta_range[1], beta_range[2]).
intercept	Intercept added to the linear predictor (default 1).
corr_structure	One of "all_ar1", "informative_cs", "blockdiag", "none".

rho_noise — Optional correlation for the noise block when corr_structure = "blockdiag" (defaults to rho).

noise_sd — Std. dev. of Gaussian noise added to y when type = "gaussian" (default 1); ignored for type = "logistic".

miss — Missingness mechanism: "MAR" or "MCAR" (default "MAR").

miss_prop — Target marginal missingness proportion (default 0.25).

mar_drivers — Indices of predictors that drive MAR (default c(1, 2, 3)). Must lie within 1..p. (Out-of-range indices are ignored; an empty set is not allowed.)

gamma_vec — Coefficients for MAR drivers; length must equal the number of MAR drivers actually used (i.e., length(mar_drivers) after restricting to 1..p). If NULL, heuristic defaults are used (starting from c(0.5, -0.35, 0.15) as available).

calibrate_mar — If TRUE, calibrates the MAR intercept by root-finding so that the average missingness matches miss_prop. If FALSE, uses qlogis(miss_prop).

mar_scale — If TRUE (default), standardize MAR drivers before applying gamma_vec.

keep_observed — Indices of predictors kept fully observed (values outside 1..p are ignored).

jitter_sd — Standard deviation of the per-row jitter added to the MAR logit to induce heterogeneity (default 0.25).

keep_mar_drivers — Logical; if TRUE (default), predictors in mar_drivers are kept fully observed under MAR so that missingness depends only on observed covariates (MAR). If FALSE, those drivers may be masked as well, making the mechanism effectively non-ignorable (MNAR) for variables whose missingness depends on them.

Details

Correlation structures:

- "all_ar1": AR(1) correlation with parameter rho across all p predictors.
- "informative_cs": compound symmetry (exchangeable) within the first p_inf predictors with parameter rho; others independent.
- "blockdiag": block-diagonal AR(1): the informative block (size p_inf) has AR(1) with rho; the noise block (size p - p_inf) has AR(1) with rho_noise (defaults to rho).
- "none": independent predictors.

Missingness:

- "MAR": for each row, a logit missingness score is computed from the selected MAR drivers (see mar_drivers, gamma_vec, mar_scale); an intercept is set via calibrate_mar to target the proportion miss_prop (otherwise qlogis(miss_prop)), and per-row jitter $N(0, jitter_sd)$ adds heterogeneity. The resulting probability is used to mask predictors (except those in keep_observed and—if keep_mar_drivers = TRUE—the drivers themselves). For type = "gaussian" only, y is also subject to the same missingness mechanism.
- "MCAR": each predictor (except those in keep_observed) is masked independently with probability miss_prop. For type = "gaussian" only, y is also masked MCAR with probability miss_prop.

Note: In the simulation, missingness probabilities are computed using the fully observed latent covariates before masking. From an analyst's perspective after masking, allowing the MAR drivers themselves to be missing makes missingness depend on unobserved values—i.e., effectively non-ignorable (MNAR). Setting keep_mar_drivers = TRUE keeps those drivers observed and yields a MAR mechanism.

Value

A list with elements:

- data: data.frame with columns X1..Xp and y, containing NAs per the missingness mechanism.
- beta: numeric length-p vector of true coefficients (non-zeros in the first p_inf positions).
- informative: integer vector 1:p_inf.
- type: character, outcome type ("gaussian" or "logistic").
- intercept: numeric intercept used.

The data element additionally carries attributes: "true_beta", "informative", "type", "corr_structure", "rho", "rho_noise" (if set), "intercept", "noise_sd" (Gaussian; NA otherwise), "mar_scale", and "keep_mar_drivers".

Reproducing the shipped dataset booami_sim

```
set.seed(123)
sim <- simulate_booami_data(
  n = 300, p = 25, p_inf = 5, rho = 0.3,
  type = "gaussian", beta_range = c(1, 2), intercept = 1,
  corr_structure = "all_ar1", rho_noise = NULL, noise_sd = 1,
  miss = "MAR", miss_prop = 0.25,
  mar_drivers = c(1, 2, 3), gamma_vec = NULL,
  calibrate_mar = FALSE, mar_scale = TRUE,
  keep_observed = integer(0), jitter_sd = 0.25,
  keep_mar_drivers = TRUE
)
booami_sim <- sim$data
```

See Also

booami_sim, cv_boost_raw, cv_boost_imputed, impu_boost

Examples

```
set.seed(42)
sim <- simulate_booami_data(
  n = 200, p = 15, p_inf = 4, rho = 0.25,
  type = "gaussian", miss = "MAR", miss_prop = 0.20
)
d <- sim$data
dim(d)
```

```
mean(colSums(is.na(d)) > 0)    # fraction of columns with any NAs
head(attr(d, "true_beta"))
attr(d, "informative")

# Example with block-diagonal correlation and protected MAR drivers
sim2 <- simulate_booami_data(
  n = 150, p = 12, p_inf = 3, rho = 0.40, rho_noise = 0.10,
  corr_structure = "blockdiag", miss = "MAR", miss_prop = 0.30,
  mar_drivers = c(1, 2), keep_mar_drivers = TRUE
)
colSums(is.na(sim2$data))[1:4]

# Binary outcome example
sim3 <- simulate_booami_data(
  n = 100, p = 10, p_inf = 2, rho = 0.2,
  type = "logistic", miss = "MCAR", miss_prop = 0.15
)
table(sim3$data$y, useNA = "ifany")


utils::data(booami_sim)
dim(booami_sim)
head(attr(booami_sim, "true_beta"))
attr(booami_sim, "informative")
```

Index