Package: BATTS (via r-universe)

November 14, 2026

Title Boosting and Bayesian Additive Trees for Two-Sample Comparison

Version 0.3.0

Description Implements boosting and Bayesian additive tree methods for two-sample comparison through density ratio estimation, following Awaya, Xu, and Ma (2025). The package supports both point estimation and uncertainty quantification via Bayesian additive tree ensembles.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, RcppArmadillo

Suggests ggplot2, knitr, mvtnorm, rmarkdown, scales

SystemRequirements C++17

URL https://github.com/nawaya040/BATTS

BugReports https://github.com/nawaya040/BATTS/issues

Repository https://nawaya040.r-universe.dev

Date/Publication 2026-10-19 10:38:28 UTC

RemoteUrl https://github.com/nawaya040/batts

RemoteRef HEAD

RemoteSha 9ade23e222852048e3784e79f1aec325b1e5644c

Contents

BATTS-package 2
batts 2
boots 4
eval_balance_weight 6

Index **8**

BATTS-package *Boosting and Bayesian Additive Trees for Two-Sample Comparison*

Description

Estimate the square root of a density ratio using boosting or generalized Bayesian additive trees. See `boots()`, `batts()`, and `eval_balance_weight()` for the public interface.

Author(s)

Maintainer: Naoki Awaya <nawaya@waseda.jp>

Authors:

- Yuliang Xu
- Li Ma

References

Awaya, N., Xu, Y., and Ma, L. (2025). Two-sample comparison through additive tree models for density ratios. https://arxiv.org/abs/2508.03059.

See Also

Useful links:

- https://github.com/nawaya040/BATTS
- Report bugs at https://github.com/nawaya040/BATTS/issues

batts *Bayesian additive trees for a two-sample density ratio*

Description

Draw generalized Bayesian balancing weights w = sqrt(p/q) by backfitting.

Usage

```
batts(
  data,
  group_labels,
  num_trees = 200,
  max_resol = 0,
  learn_rate = 0.01,
  n_bins = 100,
  n_min_obs_per_node = 1,
```

```
  n_ratio_per_node = 1e-100,
  margin_scale = 0.1,
  use_gradient = FALSE,
  size_burnin = 2000,
  size_backfitting = 1000,
  thin = 1,
  prob_moves = c(1/3, 1/3, 1/3),
  lambda_0 = 5,
  omega_prior_parameters = c(1, 1),
  tree_priors = c(0.95, 2),
  output_BART_ensembles = FALSE,
  quiet = FALSE
)
```

Arguments

`data`	Numeric matrix with observations in rows and variables in columns. Values must be finite. With nonnegative `margin_scale`, each column must vary.
`group_labels`	Numeric vector of zeros and ones, one per row of `data`, with both groups present. Group 0 has density p and group 1 has density q.
`num_trees`	Number of trees in the Bayesian ensemble.
`max_resol`	Maximum boosting tree depth. For `batts`, must be zero so that all initial trees are unsplit.
`learn_rate`	Boosting shrinkage factor; use a positive value.
`n_bins`	Number of bins used to construct the boosting cut grid. For root-initialized `batts`, retained for interface compatibility.
`n_min_obs_per_node`	Minimum count from each group in a boosting child.
`n_ratio_per_node`	Reserved argument. Only the default `1e-100` is supported.
`margin_scale`	Nonnegative fraction of each training range added at both ends of the fitted domain. A negative value uses the fixed unit box and requires that the supplied data already lie in that box.
`use_gradient`	Logical; use gradient boosting when TRUE and forward-stagewise boosting when FALSE. Initialization in `batts` is unsplit.
`size_burnin`	Nonnegative integer number of discarded backfitting iterations (default 2000). `NULL` uses `floor(size_backfitting / 2)`.
`size_backfitting`	Positive integer number of saved draws (default 1000).
`thin`	Positive integer giving the number of backfitting sweeps per iteration, including burn-in iterations.
`prob_moves`	Probabilities of GROW, PRUNE, and CHANGE, in that order; supply three finite positive values summing to one (tolerance 1e-12).
`lambda_0`	Positive leaf-prior scale per tree; ensemble scale is `num_trees * lambda_0` and remains fixed during sampling.

`omega_prior_parameters`
: Shape and rate of the Gamma prior on the loss temperature, as a positive length-two vector.

`tree_priors`
: Two tree-depth prior parameters: split probability at depth d is `tree_priors[1] / (1 + d)^tree_priors[2]`.

`output_BART_ensembles`
: Logical; save posterior forests for subsequent evaluation on new points. Training-point draws are returned in either case.

`quiet`
: Logical; suppress progress messages.

Details

The defaults (200 trees, 2000 burn-in iterations, 1000 saved draws, `lambda_0 = 5` and margin 0.1) are the settings of the paper's simulations.

Each fit starts with unsplit trees (`max_resol = 0`). Use multiple chains and convergence diagnostics for substantive applications. A short successful run does not establish convergence or frequentist coverage. Sparse overlap and complete separation can produce prior-sensitive estimates and slow mixing. The returned intervals are pointwise; zero inclusion does not establish equality of distributions.

Value

A list containing `balance_weight_BART_data`, an observations-by-draws matrix; `forest_list`, saved forests when requested; `omega_store`, temperature draws; and `lambda_store`, fixed leaf-scale values. Also includes `tree_list`, `c`, `data_info`, and `Omega` for initialization and evaluation.

See Also

`boots()`, `eval_balance_weight()`

Examples

```
set.seed(2)
x <- matrix(c(rnorm(20), rnorm(20, 0.5)), ncol = 1)
g <- rep(0:1, each = 20)
# Small API demonstration; these settings are not a convergence recommendation.
fit <- batts(x, g, num_trees = 4, size_burnin = 5,
             size_backfitting = 10, output_BART_ensembles = TRUE, quiet = TRUE)
dim(fit$balance_weight_BART_data)
```

`boots` *Boosting for a two-sample density ratio*

Description

Fit an additive tree estimate of the balancing weight w = sqrt(p/q). The full log-density ratio is obtained as `2 * log(w)`.

Usage

```
boots(
  data,
  group_labels,
  num_trees_max = 1000,
  K_CV = 5,
  max_resol = 4,
  learn_rate = 0.01,
  n_bins = 32,
  n_min_obs_per_node = 1,
  n_ratio_per_node = 1e-100,
  margin_scale = 0.1,
  use_gradient = TRUE,
  quiet = FALSE,
  subsample_fraction = 0.5
)
```

Arguments

`data`	Numeric matrix with observations in rows and variables in columns. Values must be finite. With nonnegative `margin_scale`, each column must vary.
`group_labels`	Numeric vector of zeros and ones, one per row of `data`, with both groups present. Group 0 has density p and group 1 has density q.
`num_trees_max`	Maximum number of boosting trees.
`K_CV`	Number of group-stratified cross-validation folds, or zero to fit `num_trees_max` trees without cross-validation. A positive value must be an integer between two and the smaller group size. CV selects the minimum mean held-out balancing loss.
`max_resol`	Maximum boosting tree depth. For `batts`, must be zero so that all initial trees are unsplit.
`learn_rate`	Boosting shrinkage factor; use a positive value.
`n_bins`	Number of bins used to construct the boosting cut grid. For root-initialized `batts`, retained for interface compatibility.
`n_min_obs_per_node`	Minimum count from each group in a boosting child.
`n_ratio_per_node`	Reserved argument. Only the default `1e-100` is supported.
`margin_scale`	Nonnegative fraction of each training range added at both ends of the fitted domain. A negative value uses the fixed unit box and requires that the supplied data already lie in that box.
`use_gradient`	Logical; use gradient boosting when TRUE and forward-stagewise boosting when FALSE. Initialization in `batts` is unsplit.
`quiet`	Logical; suppress progress messages.

`subsample_fraction`
: Fraction of the training observations used to construct each tree, in (0, 1]. With the default `0.5`, each tree uses `ceiling(subsample_fraction * m)` of the `m` training observations (the CV training folds, or all data for the final fit), drawn as in `ada` with `bag.frac`: one observation from each group, and the rest uniformly without replacement regardless of group. `1` uses all of them.

Details

The defaults (gradient boosting with at most 1000 trees selected by 5-fold CV, depth 4, learning rate 0.01, 32 bins, margin 0.1 and per-tree subsampling of one half) are the settings of the paper's simulations. Set the seed before fitting; CV folds and subsamples use the R random number generator.

Boosting uses uniformly spaced candidate cuts; Bayesian cut proposals are uniform. Candidate cuts are relative to the current node. Splits requiring children with too few observations from either group are excluded. Under complete separation this restriction can prevent informative splits. Forward-stagewise fits include a multiplicative normalization constant. With subsampling, splits, gradients, and leaf values use the tree's subsample, while the balancing weights of all observations are updated and the forward-stagewise normalization uses the whole training set. CV losses are evaluated on the held-out folds. Subsampling uses the R random number generator; `subsample_fraction = 1` does not.

Value

A list containing `balance_weight_boosting_data` (training weights), `tree_list` (fitted trees), `c` (normalization), `data_info` (training domain and group sizes), and `Omega` (domain bounds). With CV, also returns `loss_CV_store` (folds by candidate tree count).

See Also

`batts()`, `eval_balance_weight()`

Examples

```
set.seed(1)
x <- matrix(c(rnorm(30), rnorm(30, 0.5)), ncol = 1)
g <- rep(0:1, each = 30)
fit <- boots(x, g, num_trees_max = 5, quiet = TRUE)
head(2 * log(fit$balance_weight_boosting_data))
```

---

eval_balance_weight *Evaluate fitted balancing weights*

Description

Evaluate stored trees at points inside the fitted domain. Weights have scale w = sqrt(p/q), with group 0 in the numerator; use `2 * log(w)` for log ratios.

Usage

```
eval_balance_weight(list_result, eval_points, is_Bayes = FALSE)
```

Arguments

list_result	Result of boots() or batts().
eval_points	Finite numeric matrix with the same columns, in the same order, as the training matrix. All points must lie inside Omega.
is_Bayes	Logical; additionally evaluate posterior forests. Requires fitting with output_BART_ensembles = TRUE.

Details

Prediction does not draw random numbers. With is_Bayes = TRUE, the boosting component describes the initialization; the posterior draws are in balancing_weight_BART. Missing posterior forests cause an error.

Value

A list with balancing_weight_boosting, a one-column numeric matrix, and, when is_Bayes = TRUE, balancing_weight_BART, an evaluation-points-by-draws matrix. Both include the fitted normalization constant.

See Also

boots(), batts()

Examples

```
set.seed(3)
x <- matrix(c(rnorm(20), rnorm(20, 0.5)), ncol = 1)
fit <- boots(x, rep(0:1, each = 20), num_trees_max = 5, quiet = TRUE)
pred <- eval_balance_weight(fit, x[1:3, , drop = FALSE])
pred$balancing_weight_boosting
```

Index

BATTS *(*BATTS-package*)*, 2
batts, 2
batts(), *2*, *6*, *7*
BATTS-package, 2
boots, 4
boots(), *2*, *4*, *7*

eval_balance_weight, 6
eval_balance_weight(), *2*, *4*, *6*