Package 'forestBalance'

November 9, 2026

Title Balancing Confounder Distributions with Forest Energy Balancing

Version 0.1.1

Description Estimates average treatment effects using kernel energy balancing with random forest similarity kernels. A multivariate random forest jointly models covariates, outcome, and treatment to build a similarity kernel between observations. This kernel is then used for energy balancing to create weights that control for confounding. The method is described in De and Huling (2025) <doi:10.48550/arXiv.2512.18069>.

License GPL (>= 3)

URL https://github.com/jaredhuling/forestBalance

BugReports https://github.com/jaredhuling/forestBalance/issues

Encoding UTF-8

RoxygenNote 7.3.3

LinkingTo Rcpp, RcppEigen

Imports grf (>= 2.3.0), MASS, Matrix, methods, Rcpp

Suggests ggplot2, knitr, osqp, rmarkdown, testthat (>= 3.0.0), WeightIt

VignetteBuilder knitr

NeedsCompilation yes

Author Jared Huling [aut, cre], Simion De [aut]

Maintainer Jared Huling <jaredhuling@gmail.com>

Repository CRAN

Date/Publication 2026-11-09 20:20:15 UTC

Contents

forestBalance-package 2
compute_balance 3
forest_balance 5

forest_kernel 9
get_leaf_node_matrix 10
kernel_balance 11
leaf_node_kernel 13
leaf_node_kernel_Z 14
print.forest_balance 15
simulate_data 16
summary.forest_balance 17

Index **19**

`forestBalance-package` *forestBalance: Forest Kernel Energy Balancing for Causal Inference*

Description

Estimates average treatment effects (ATE) using kernel energy balancing with random forest similarity kernels. A multivariate random forest jointly models covariates, treatment, and outcome to build a proximity kernel that captures confounding structure. Balancing weights are obtained via a closed-form kernel energy distance solution.

Main function

`forest_balance` is the primary interface. It fits the forest, constructs the kernel, computes balancing weights, and estimates the ATE. By default it uses K-fold cross-fitting and an adaptive leaf size to minimize overfitting bias.

Key features

- Adaptive `min.node.size` that scales with n and p
- K-fold cross-fitting to reduce kernel overfitting bias
- Rcpp-accelerated leaf node extraction
- Sparse kernel construction via single `tcrossprod`
- Conjugate gradient solver for large n (avoids forming the kernel matrix entirely)

Lower-level interface

For more control, the pipeline can be run step by step: `get_leaf_node_matrix`, `leaf_node_kernel`, `kernel_balance`.

Author(s)

Maintainer: Jared Huling <jaredhuling@gmail.com>

Authors:

- Simion De

References

De, S. and Huling, J.D. (2025). Data adaptive covariate balancing for causal effect estimation for high dimensional data. *arXiv preprint arXiv:2512.18069*.

See Also

Useful links:

- https://github.com/jaredhuling/forestBalance
- Report bugs at https://github.com/jaredhuling/forestBalance/issues

---

`compute_balance` *Compute covariate balance diagnostics for a set of weights*

Description

Computes standardized mean differences (SMD), effective sample sizes (ESS), and optionally the weighted energy distance for a given set of balancing weights. Can also assess balance on user-supplied nonlinear transformations of the covariates.

Usage

```
compute_balance(X, trt, weights, X.trans = NULL, energy.dist = TRUE)

## S3 method for class 'forest_balance_diag'
print(x, threshold = 0.1, ...)
```

Arguments

`X`	An $n \times p$ numeric covariate matrix.
`trt`	A binary (0/1) vector of treatment assignments of length n.
`weights`	A numeric weight vector of length n. Treated weights should sum to n_1 and control weights to n_0 (as returned by `kernel_balance` or `forest_balance`).
`X.trans`	An optional matrix of nonlinear or transformed covariates ($n \times q$) on which to additionally assess balance (e.g., interactions, squared terms). If `NULL` (default), only linear covariate balance is reported.
`energy.dist`	Logical; if `TRUE`, compute the weighted energy distance between the treated and control groups. This requires computing an $n \times n$ distance matrix and is only feasible for moderate n (automatically skipped when $n > 5000$). Default is `TRUE`.
`x`	A `forest_balance_diag` object.
`threshold`	SMD threshold for flagging imbalanced covariates. Default is 0.1, a standard threshold in the causal inference literature.
`...`	Ignored.

Details

The standardized mean difference for covariate j is defined as

$$\text{SMD}_j = \frac{|\bar{X}^w_{j,1} - \bar{X}^w_{j,0}|}{s_j},$$

where $\bar{X}^w_{j,a}$ is the weighted mean of covariate j in group a and s_j is the pooled (unweighted) standard deviation.

The effective sample size for a group is

$$\text{ESS} = \frac{(\sum_i w_i)^2}{\sum_i w_i^2},$$

reported as a fraction of the group size.

The weighted energy distance is

$$E = 2\sum_{i,j} p_i q_j \|X_i - X_j\| - \sum_{i,j} p_i p_j \|X_i - X_j\| - \sum_{i,j} q_i q_j \|X_i - X_j\|,$$

where p and q are the normalized treated and control weights.

Value

An object of class `"forest_balance_diag"` containing:

smd Named vector of |SMD| for each covariate.

max_smd Maximum |SMD| across covariates.

smd_trans Named vector of |SMD| for transformed covariates (if `X.trans` was supplied), otherwise `NULL`.

max_smd_trans Maximum |SMD| for transformed covariates, or `NA`.

energy_dist Weighted energy distance, or `NA` if not computed.

ess_treated Effective sample size for the treated group as a fraction of n_1.

ess_control Effective sample size for the control group as a fraction of n_0.

n Total sample size.

n1 Number of treated units.

n0 Number of control units.

The input `x`, invisibly. Called for its side effect of printing balance diagnostics to the console.

Examples

```
n <- 500; p <- 10
X <- matrix(rnorm(n * p), n, p)
A <- rbinom(n, 1, plogis(0.5 * X[, 1]))
Y <- X[, 1] + rnorm(n)

fit <- forest_balance(X, A, Y)
```

```
bal <- compute_balance(X, A, fit$weights)
bal

# With nonlinear features
X.nl <- cbind(X[,1]^2, X[,1]*X[,2])
colnames(X.nl) <- c("X1^2", "X1*X2")
bal2 <- compute_balance(X, A, fit$weights, X.trans = X.nl)
bal2
```

`forest_balance` *Estimate ATE using forest-based kernel energy balancing*

Description

Fits a multivariate random forest that jointly models the relationship between covariates, treatment, and outcome, computes a random forest proximity kernel, and then uses kernel energy balancing to produce weights for estimating the average treatment effect (ATE). By default, K-fold cross-fitting is used to avoid overfitting bias from estimating the kernel on the same data used for treatment effect estimation.

Usage

```
forest_balance(
  X,
  A,
  Y,
  num.trees = 1000,
  min.node.size = NULL,
  estimand = c("ATE", "ATT", "ATC"),
  cross.fitting = TRUE,
  num.folds = 2,
  augmented = FALSE,
  mu.hat = NULL,
  scale.outcomes = TRUE,
  solver = c("auto", "direct", "cg", "bj"),
  lambda = 0.1,
  kernel.response = c("joint", "treatment", "outcome"),
  crossfit.balance = c("fold", "full"),
  tol = 1e-08,
  parallel = FALSE,
  maxiter = 2000L,
  ...
)
```

Arguments

X	A numeric matrix or data frame of covariates ($n \times p$).
A	A binary (0/1) vector of treatment assignments.
Y	A numeric vector of outcomes.
num.trees	Number of trees to grow in the forest. Default is 1000.
min.node.size	Minimum number of observations per leaf node. If NULL (default), an adaptive heuristic is used: max(20, min(floor(n/200) + p, floor(n/50))). This scales the leaf size with both the sample size and the number of covariates, which empirically yields low bias. See Details.
estimand	Target estimand: "ATE" (default), "ATT", or "ATC".
cross.fitting	Logical; if TRUE (default), use K-fold cross-fitting to construct the kernel from held-out data, reducing overfitting bias. If FALSE, the kernel is estimated on the full sample.
num.folds	Number of cross-fitting folds. Default is 2. Only used when cross.fitting = TRUE.
augmented	Logical; if TRUE, use an augmented (doubly-robust) estimator that combines the kernel energy balancing weights with group-specific outcome regression models. This reduces bias when either the kernel or the outcome models are correctly specified. Default is FALSE. See Details.
mu.hat	Optional list with components mu1 and mu0, each a numeric vector of length n, containing user-supplied predictions of $E[Y \mid X, A = 1]$ and $E[Y \mid X, A = 0]$. When provided, these are used instead of fitting internal outcome models. If NULL (default) and augmented = TRUE, two regression_forest models are fit automatically (one on treated, one on control). When supplying mu.hat with cross.fitting = TRUE, the user is responsible for ensuring the predictions were cross-fitted externally.
scale.outcomes	If TRUE (default), the joint outcome matrix cbind(A, Y) is column-standardized before fitting the forest. This ensures that treatment and outcome contribute equally to the splits.
solver	Which linear solver to use for the balancing weights. "auto" (default) selects "direct" for small fold sizes and "cg" for large fold sizes. See kernel_balance for details.
lambda	Nonnegative ridge penalty on the balancing weights; see kernel_balance. Default 0.1.
kernel.response	Which responses the kernel-defining forest is fit to: "joint" (default; treatment and outcome, the proposed method), "treatment" (treatment only), or "outcome" (outcome only). The latter two exist for ablation studies.
crossfit.balance	Under cross-fitting, whether the balancing problem for fold v is solved within the fold ("fold", default) or on the full sample with the kernel learned on the other folds, keeping only fold v's weights ("full").
tol	Convergence tolerance for the CG solver. Default is 5e-11.

`parallel`	Logical or integer. If `FALSE` (default), folds are processed sequentially. If `TRUE`, folds are processed in parallel using all available cores via `mclapply`. An integer value specifies the exact number of cores. Only used when `cross.fitting = TRUE`. Note: parallel processing is not supported on Windows.
`maxiter`	Maximum iterations for each iterative balancing solve. Default 2000.
`...`	Additional arguments passed to `multi_regression_forest`.

Details

The method proceeds in three steps:

1. A `multi_regression_forest` is fit on covariates `X` with a bivariate response `(A, Y)`. This jointly models the relationship between covariates, treatment assignment, and outcome.
2. The forest's leaf co-membership structure defines a proximity kernel: $K(i, j)$ is the proportion of trees where i and j share a leaf. Because the forest splits on both A and Y, this kernel captures confounding structure.
3. `kernel_balance` computes balancing weights via the closed-form kernel energy distance solution. The ATE is then estimated using the Hajek (ratio) estimator with these weights.

Cross-fitting (default): For each fold k, the forest is trained on all data *except* fold k, and the kernel for fold k is built from that held-out forest's leaf predictions. This breaks the dependence between the kernel and the outcomes, reducing overfitting bias. The final ATE is the average of the per-fold Hajek estimates (DML1).

Augmented estimator: When `augmented = TRUE`, two group-specific outcome models $\hat{\mu}_1(X) = E[Y|X, A = 1]$ and $\hat{\mu}_0(X) = E[Y|X, A = 0]$ are fit, and the ATE is estimated via the doubly-robust formula:

$$\hat{\tau} = \frac{1}{n}\sum_i [\hat{\mu}_1(X_i) - \hat{\mu}_0(X_i)] + \frac{\sum w_i A_i (Y_i - \hat{\mu}_1(X_i))}{\sum w_i A_i} - \frac{\sum w_i (1 - A_i)(Y_i - \hat{\mu}_0(X_i))}{\sum w_i (1 - A_i)}.$$

The first term is the regression-based estimate of the ATE; the remaining terms are weighted bias corrections. This is consistent if either the kernel (balancing weights) or the outcome models are correctly specified. When combined with cross-fitting, the outcome models are automatically cross-fitted in lockstep with the kernel.

Adaptive leaf size: The default `min.node.size` is set adaptively via `max(20, min(floor(n/200) + p, floor(n/50)))`. Larger leaves produce smoother kernels that generalize better, while the cap at `n/50` prevents kernel degeneracy.

Value

An object of class `"forest_balance"` (a list) with the following elements:

ate The estimated average treatment effect. When cross-fitting is used, this is the average of per-fold Hajek estimates (DML1).

weights The balancing weight vector (length n). When cross-fitting is used, these are the concatenated per-fold weights.

mu1.hat Predictions of $E[Y|X, A = 1]$ (length n), or `NULL` if `augmented = FALSE`.

mu0.hat Predictions of $E[Y|X, A = 0]$ (length n), or `NULL` if `augmented = FALSE`.

kernel The $n \times n$ forest proximity kernel (sparse matrix), or `NULL` when cross-fitting or the CG solver is used.

forest The trained forest object. When cross-fitting is used, this is the last fold's forest.

forests All trained forests, one per fold.

fold_diagnostics Per-fold index maps, complete balancing weights, and solver residual checks. Failed folds are errors.

X, A, Y The input data.

n, n1, n0 Total, treated, and control sample sizes.

solver The solver that was used (`"direct"` or `"cg"`).

crossfit Logical indicating whether cross-fitting was used.

augmented Logical indicating whether augmentation was used.

num.folds Number of folds (if cross-fitting was used).

fold_ates Per-fold ATE estimates (if cross-fitting was used).

fold_ids Fold assignments (if cross-fitting was used).

The object has `print` and `summary` methods. Use `summary.forest_balance` for covariate balance diagnostics.

References

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W. and Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1), C1–C68.

De, S. and Huling, J.D. (2025). Data adaptive covariate balancing for causal effect estimation for high dimensional data. *arXiv preprint arXiv:2512.18069*.

Examples

```
n <- 500
p <- 10
X <- matrix(rnorm(n * p), n, p)
A <- rbinom(n, 1, plogis(0.5 * X[, 1]))
Y <- X[, 1] + rnorm(n)  # true ATE = 0

# Default: cross-fitting with adaptive leaf size
result <- forest_balance(X, A, Y)
result

# Augmented (doubly-robust) estimator
result_aug <- forest_balance(X, A, Y, augmented = TRUE)

# Without cross-fitting
result_nocf <- forest_balance(X, A, Y, cross.fitting = FALSE)
```

`forest_kernel` *Compute random forest proximity kernel from a GRF forest*

Description

Extracts the leaf node assignments for each observation across all trees in a trained GRF forest, then computes the $n \times n$ proximity kernel matrix where entry (i, j) is the proportion of trees in which observations i and j share a leaf node.

Usage

```
forest_kernel(forest, newdata = NULL)
```

Arguments

`forest`	A trained forest object from the **grf** package.
`newdata`	A numeric matrix of observations. If `NULL` (default), uses the original training data.

Details

This is a convenience function that calls `get_leaf_node_matrix` followed by `leaf_node_kernel`. If you need both the leaf matrix and the kernel, it is more efficient to call them separately.

Value

A symmetric numeric matrix of dimension $n \times n$.

Examples

```
library(grf)
n <- 100
p <- 5
X <- matrix(rnorm(n * p), n, p)
Y <- cbind(X[, 1] + rnorm(n), X[, 2] + rnorm(n))
forest <- multi_regression_forest(X, Y, num.trees = 50)

K <- forest_kernel(forest)
```

`get_leaf_node_matrix` *Extract leaf node membership matrix from a GRF forest*

Description

For each observation and each tree in a trained GRF forest, determines which leaf node the observation falls into. The tree traversal is implemented in C++ for speed, directly reading the forest's internal tree structure and avoiding the overhead of `get_tree` and `get_leaf_node`.

Usage

```
get_leaf_node_matrix(forest, newdata = NULL)
```

Arguments

`forest`	A trained forest object from the **grf** package (e.g., from `multi_regression_forest`, `regression_forest`, etc.).
`newdata`	A numeric matrix of observations to predict leaf membership for. Must have the same number of columns as the training data. If `NULL` (default), uses the original training data stored in the forest.

Details

This function reads the internal tree vectors stored in a GRF forest object (`_child_nodes`, `_split_vars`, `_split_values`, `_root_nodes`, `_send_missing_left`) and traverses each tree in compiled C++ code. This is dramatically faster than the per-observation R-level loop in `grf::get_leaf_node`.

Value

An integer matrix of dimension `nrow(newdata)` by `num.trees`, where entry `[i, b]` is the internal node index (1-based) of the leaf that observation `i` falls into in tree `b`.

Examples

```
library(grf)
n <- 200
p <- 5
X <- matrix(rnorm(n * p), n, p)
Y <- cbind(X[, 1] + rnorm(n), X[, 2] + rnorm(n))
forest <- multi_regression_forest(X, Y, num.trees = 100)

# Leaf membership for training data
leaf_mat <- get_leaf_node_matrix(forest)

# Leaf membership for new data
X.test <- matrix(rnorm(50 * p), 50, p)
leaf_mat_test <- get_leaf_node_matrix(forest, X.test)
```

kernel_balance *Kernel energy balancing weights via closed-form solution*

Description

Computes balancing weights that minimize a kernelized energy distance between the weighted treated and control distributions and the overall sample. The weights are obtained via a closed-form solution to a linear system derived from the kernel energy distance objective.

Usage

```
kernel_balance(
  trt,
  kern = NULL,
  Z = NULL,
  leaf_matrix = NULL,
  num.trees = NULL,
  estimand = c("ATE", "ATT", "ATC"),
  solver = c("auto", "direct", "cg", "bj"),
  lambda = 0.1,
  tol = 1e-08,
  maxiter = 2000L
)
```

Arguments

`trt`	A binary (0/1) integer or numeric vector indicating treatment assignment (`1` = treated, `0` = control).
`kern`	A symmetric $n \times n$ kernel matrix (dense or sparse), or `NULL` if `Z` is provided. Required for `solver = "direct"` (if not provided but `Z` is available, the kernel is formed automatically, though this is $O(n^2)$ and may be slow for large n).
`Z`	Optional sparse indicator matrix from `leaf_node_kernel_Z` such that $K = ZZ^\top/B$. When supplied, the iterative solvers (`"cg"`, `"bj"`) can perform matrix-free products without forming the full kernel. Required for `solver = "cg"` and `solver = "bj"`.
`leaf_matrix`	Optional integer matrix of leaf node assignments (observations x trees), as returned by `get_leaf_node_matrix`. Required for `solver = "bj"` (Block Jacobi preconditioner uses tree 1's leaf partition). If `NULL` and `solver = "bj"`, falls back to `"cg"` with a warning.
`num.trees`	Number of trees B. Required when `Z` is provided.
`estimand`	Target estimand: `"ATE"` (default), `"ATT"`, or `"ATC"`.
`solver`	Which linear solver to use. `"auto"` (default) selects the best available solver based on the inputs: `"cg"` when `Z` is available and $n > 5000$, or `"direct"` otherwise. See Details for solver requirements.

`lambda` Nonnegative ridge penalty on the weights. The group-a problem solved is $\frac{1}{2}\|\sum_{i\in S_a} w_i K(X_i,\cdot) - n_a P_n\|^2 + \frac{\lambda}{2}\|w_a\|^2$ subject to $\sum_{i\in S_a} w_i = n_a$, whose KKT system is $(K_{aa} + \lambda I)w_a = b_a + \gamma_a 1$. Default `0.1`. The ridge makes the weights unique when the kernel block is singular, which happens when several units share leaves in nearly every tree. With `lambda = 0` a numerically singular block raises an error.

`tol` Convergence tolerance for iterative solvers. Default is `1e-8`.

`maxiter` Maximum iterations for iterative solvers. Default is 2000.

Details

The modified kernel K_q used in the optimization is block-diagonal: the treated–control cross-blocks are zero because $K_q(i,j) = 0$ whenever $A_i \neq A_j$. All solvers exploit this structure by working on the treated and control blocks independently.

Solver requirements:

Solver	Required inputs	Optional inputs
`"direct"`	`kern` (or `Z` + `num.trees`)	
`"cg"`	`Z` + `num.trees`	
`"bj"`	`Z` + `num.trees` + `leaf_matrix`	(falls back to `"cg"` if `leaf_matrix` is missing)

The **direct** solver extracts sub-blocks of the kernel and solves via sparse Cholesky. If only `Z` is provided, the kernel is formed as $K = ZZ^\top/B$, which requires $O(n^2)$ time and memory.

The **CG** solver uses the factored representation $K = ZZ^\top/B$ to perform matrix–vector products without forming any kernel matrix.

The **Block Jacobi** solver (`"bj"`) uses the first tree's leaf partition (from `leaf_matrix`) to define a block-diagonal preconditioner for CG. Each leaf block is a small dense system that is cheap to factor.

Only 2 linear solves per block are needed (not 3) because the third right-hand side is a linear combination of the first two.

Value

A list with the following elements:

weights A numeric vector of length n containing the balancing weights. Treated weights sum to n_1 and control weights sum to n_0.

solver The solver that was used.

diagnostics Residuals and convergence checks for every linear solve, arm-total errors, and projected KKT residuals. For positive ridge and a positive semidefinite kernel, the reported weight error bound includes the arm-total correction. Failed accuracy checks raise an error carrying diagnostics.

References

De, S. and Huling, J.D. (2025). Data adaptive covariate balancing for causal effect estimation for high dimensional data. *arXiv preprint arXiv:2512.18069*.

Examples

```
library(grf)
n <- 200
p <- 5
X <- matrix(rnorm(n * p), n, p)
A <- rbinom(n, 1, plogis(X[, 1]))
Y <- X[, 1] + rnorm(n)

# --- Direct solver (using the kernel matrix) ---
forest <- multi_regression_forest(X, cbind(A, Y), num.trees = 500)
K <- forest_kernel(forest)
bal_direct <- kernel_balance(A, kern = K, solver = "direct")

# --- CG solver (using the Z matrix, avoids forming K) ---
# Step 1: extract leaf node assignments (n x B matrix)
leaf_mat <- get_leaf_node_matrix(forest, X)

# Step 2: build sparse indicator matrix Z such that K = Z Z' / B
Z <- leaf_node_kernel_Z(leaf_mat)

# Step 3: solve with CG (matrix-free, no kernel formed)
bal_cg <- kernel_balance(A, Z = Z, num.trees = 500, solver = "cg")

# Both solvers give the same weights
max(abs(bal_direct$weights - bal_cg$weights))

# Weighted ATE estimate
w <- bal_cg$weights
ate <- weighted.mean(Y[A == 1], w[A == 1]) -
       weighted.mean(Y[A == 0], w[A == 0])
```

`leaf_node_kernel` *Compute random forest proximity kernel from a leaf node matrix*

Description

Given a matrix of leaf node assignments (observations x trees), computes the $n \times n$ kernel matrix where entry (i, j) is the proportion of trees in which observations i and j fall into the same leaf node.

Usage

```
leaf_node_kernel(leaf_matrix, sparse = TRUE)
```

Arguments

`leaf_matrix`	An integer matrix of dimension $n \times B$, where `leaf_matrix[i, b]` is the leaf node ID for observation `i` in tree `b`. Typically produced by `get_leaf_node_matrix`.
`sparse`	Logical; if `TRUE` (default), returns a sparse `dgCMatrix` from the **Matrix** package. If `FALSE`, returns a dense base R matrix.

Details

For each tree b, the leaf assignment defines a sparse $n \times L_b$ indicator matrix Z_b. This function stacks all B indicator matrices column-wise into a single sparse matrix $Z = [Z_1|Z_2|\cdots|Z_B]$ of dimension $n \times \sum_b L_b$. The kernel is then obtained via a single sparse cross-product: $K = ZZ^\top/B$. Leaf ID remapping is done in C++ for speed.

Value

A symmetric $n \times n$ matrix (sparse or dense depending on `sparse`) where entry (i, j) equals

$$K(i,j) = \frac{1}{B}\sum_{b=1}^{B} \mathbf{1}(L_b(i) = L_b(j)),$$

i.e., the fraction of trees where i and j share a leaf.

Examples

```
library(grf)
n <- 100
p <- 5
X <- matrix(rnorm(n * p), n, p)
Y <- cbind(X[, 1] + rnorm(n), X[, 2] + rnorm(n))
forest <- multi_regression_forest(X, Y, num.trees = 50)

leaf_mat <- get_leaf_node_matrix(forest)
K <- leaf_node_kernel(leaf_mat)               # sparse
K_dense <- leaf_node_kernel(leaf_mat, sparse = FALSE)  # dense
```

`leaf_node_kernel_Z` *Build the sparse indicator matrix Z from a leaf node matrix*

Description

Returns the sparse $n \times L$ indicator matrix Z such that the proximity kernel is $K = ZZ^\top/B$. This factored representation can be passed to `kernel_balance` to enable the CG solver, which avoids forming the full $n \times n$ kernel.

Usage

```
leaf_node_kernel_Z(leaf_matrix)
```

Arguments

`leaf_matrix`	An integer matrix of dimension $n \times B$, where `leaf_matrix[i, b]` is the leaf node ID for observation `i` in tree `b`. Typically produced by `get_leaf_node_matrix`.

Value

A sparse `dgCMatrix` of dimension $n \times L$, where $L = \sum_b L_b$ is the total number of leaves across all trees. Each row has exactly B nonzero entries (one per tree).

Examples

```
library(grf)
n <- 100
p <- 5
X <- matrix(rnorm(n * p), n, p)
Y <- cbind(X[, 1] + rnorm(n), X[, 2] + rnorm(n))
forest <- multi_regression_forest(X, Y, num.trees = 50)
leaf_mat <- get_leaf_node_matrix(forest)
Z <- leaf_node_kernel_Z(leaf_mat)
```

`print.forest_balance` *Print a forest_balance object*

Description

Displays a concise summary of the forest balance fit, including the ATE estimate, sample sizes, effective sample sizes, and a brief covariate balance overview.

Usage

```
## S3 method for class 'forest_balance'
print(x, ...)
```

Arguments

`x`	A `forest_balance` object returned by `forest_balance`.
`...`	Ignored.

Value

The input `x`, invisibly. Called for its side effect of printing a summary to the console.

`simulate_data` *Simulate observational study data with confounding*

Description

Generates data from a design with nonlinear confounding, where covariates jointly influence both treatment assignment and the outcome through non-trivial functions. The true average treatment effect is known, allowing evaluation of estimator performance.

Usage

```
simulate_data(n = 500, p = 10, ate = 0, rho = -0.25, sigma = 1, dgp = 1)
```

Arguments

`n`	Sample size. Default is 500.
`p`	Number of covariates. Must be at least 5 for `dgp = 1` and at least 8 for `dgp = 2`. Default is 10.
`ate`	True average treatment effect. Default is 0.
`rho`	Correlation parameter for the AR(1) covariance structure among covariates: $\Sigma_{jk} = \rho^{\lvert j-k \rvert}$. Default is -0.25.
`sigma`	Noise standard deviation for the outcome. Default is 1.
`dgp`	Integer selecting the data generating process. Default is 1. See Details.

Details

Both DGPs generate covariates from $X \sim N(0, \Sigma)$ where $\Sigma_{jk} = \rho^{|j-k|}$.

DGP 1 (default): Confounding through X_1 via a Beta density.

- Propensity: $P(A = 1|X) = 0.25(1 + B(X_1; 2, 4))$ where B is the Beta(2,4) density.
- Outcome: $Y = 2(X_1 - 1) + 2B(X_1; 2, 4) + X_2 + 2B(X_5; 2, 4) + \tau A + \varepsilon$.

DGP 2: Rich outcome surface with moderate confounding. Designed to illustrate the benefit of the augmented estimator. Confounding operates through X_1 and X_2, while the outcome depends on $X_1, \ldots, X_8$ with interactions and nonlinearities.

- Propensity: $P(A = 1|X) = \text{logit}^{-1}(0.6X_1 - 0.4X_2 + 0.2X_1X_2)$.
- Outcome: $Y = 2X_1 + X_2^2 - 1.5X_3 + \sin(2X_4) + X_5X_1 + 0.8X_6 - \cos(X_7) + 0.5X_8 + \tau A + \varepsilon$.

Value

A list with the following elements:

X The $n \times p$ covariate matrix.

A Binary (0/1) treatment assignment vector.

Y Observed outcome vector.

propensity True propensity scores $P(A = 1 \mid X)$.

ate The true ATE used in the simulation.

n Sample size.

p Number of covariates.

dgp The DGP that was used.

Examples

```
dat1 <- simulate_data(n = 500, p = 10, dgp = 1)
dat2 <- simulate_data(n = 500, p = 20, dgp = 2)
```

---

summary.forest_balance

Summarize a forest_balance object

Description

Produces a detailed summary of the forest balance fit, including the ATE estimate, covariate balance diagnostics (SMD, ESS, energy distance), and kernel sparsity information.

Usage

```
## S3 method for class 'forest_balance'
summary(object, X.trans = NULL, threshold = 0.1, energy.dist = TRUE, ...)

## S3 method for class 'summary.forest_balance'
print(x, ...)
```

Arguments

`object`	A `forest_balance` object returned by `forest_balance`.
`X.trans`	An optional matrix of nonlinear or transformed covariates ($n \times q$) on which to additionally assess balance (e.g., interactions, squared terms).
`threshold`	SMD threshold for flagging imbalanced covariates. Default is 0.1.
`energy.dist`	Logical; if `TRUE` (default), compute the weighted energy distance (skipped if $n > 5000$).
`...`	Ignored.
`x`	A `summary.forest_balance` object.

Value

Invisibly returns a list of class `"summary.forest_balance"` containing:

ate The estimated ATE.

balance_weighted A `forest_balance_diag` object for the weighted sample.

balance_unweighted A `forest_balance_diag` object for the unweighted sample.

kernel_sparsity Fraction of nonzero entries in the kernel matrix.

n Total sample size.

n1 Number of treated.

n0 Number of control.

num.trees Number of trees in the forest.

threshold The SMD threshold used for flagging imbalanced covariates.

The input `x`, invisibly. Called for its side effect of printing a detailed balance summary to the console.

Examples

```
n <- 500; p <- 10
X <- matrix(rnorm(n * p), n, p)
A <- rbinom(n, 1, plogis(0.5 * X[, 1]))
Y <- X[, 1] + rnorm(n)

fit <- forest_balance(X, A, Y)
summary(fit)

# With nonlinear balance assessment
X.nl <- cbind(X[,1]^2, X[,1]*X[,2])
colnames(X.nl) <- c("X1^2", "X1*X2")
summary(fit, X.trans = X.nl)
```

Index

compute_balance, 3

dgCMatrix, *14*

forest_balance, *2, 3*, 5, *15*, *17*
forest_kernel, 9
forestBalance *(*forestBalance-package*)*, 2
forestBalance-package, 2

get_leaf_node, *10*
get_leaf_node_matrix, *2*, *9*, 10, *11*, *14, 15*
get_tree, *10*

kernel_balance, *2, 3*, *6, 7*, 11, *14*

leaf_node_kernel, *2*, *9*, 13
leaf_node_kernel_Z, *11*, 14

mclapply, *7*
multi_regression_forest, *7*, *10*

print.forest_balance, 15
print.forest_balance_diag
        *(*compute_balance*)*, 3
print.summary.forest_balance
        *(*summary.forest_balance*)*, 17

regression_forest, *6*, *10*

simulate_data, 16
summary.forest_balance, *8*, 17