Package: DirichletRF (via r-universe)

June 4, 2026

Type Package

Title Dirichlet Random Forest

Version 0.1.0

Description Implementation of the Dirichlet Random Forest algorithm for compositional response data. Supports maximum likelihood estimation ('MLE') and method-of-moments ('MOM') parameter estimation for the Dirichlet distribution. Provides two prediction strategies; averaging-based predictions (average of responses within terminal nodes) and parameter-based predictions (expected value derived from the estimated Dirichlet parameters within terminal nodes). For more details see Masoumifard, van der Westhuizen, and Gardner-Lubbe (2026, ISBN:9781032903910).

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports Rcpp (>= 1.0.0), parallel

LinkingTo Rcpp

RoxygenNote 7.3.3

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

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

Date/Publication 2026-06-04 09:25:11 UTC

RemoteUrl https://github.com/xaleed/dirichletrf

RemoteRef HEAD

RemoteSha 812ce362fb262616c72f85da34ab07d491a371ff

Contents

`DirichletRF-package` *DirichletRF: Dirichlet Random Forest for Compositional Data*

Description

This package implements Dirichlet Random Forest for modeling and predicting compositional data using maximum likelihood estimation or method of moments.

`DirichletRF` *Build a Dirichlet Random Forest for Compositional Responses*

Description

`DirichletRF` fits a random forest tailored to compositional responses, i.e.\ non-negative vectors that sum to one and therefore reside in the unit simplex. Each tree is grown by recursively partitioning the covariate space using a **Dirichlet log-likelihood splitting criterion**: at every internal node the candidate split that maximises the gain in Dirichlet log-likelihood is selected.

Usage

```
DirichletRF(
  X,
  Y,
  num.trees = 100,
  max.depth = 10,
  min.node.size = 5,
  mtry = -1,
  seed = 123,
  est.method = "mom",
  distributional = FALSE,
  num.cores = -1,
  replace = FALSE,
  sample.fraction = 1,
  compute.oob = FALSE
)
```

Arguments

`X`	A numeric (n x p) matrix of covariates. Note that the current version only allows numeric covariates. Users may use one-hot encoding to possibly include categorical covariates.
`Y`	A numeric (n x k) matrix of compositional responses. Each row should sum to 1. That is, data should already be normalised if needed.
`num.trees`	Number of trees grown in the forest. Default is 100.
`max.depth`	Maximum depth of trees. Default is 10.
`min.node.size`	Minimum size of observations in each tree leaf. Default is 5. Note that nodes with sizes smaller than `min.node.size` can occur.
`mtry`	Number of covariates randomly selected as candidates at each split. Default is `sqrt(p)`, indicated by `-1`.
`seed`	The seed of the C++ random number generator.
`est.method`	Parameter estimation method for the Dirichlet distribution when splitting is done. Users may either use maximum likelihood (`"mle"`) or method of moments (`"mom"`). Default is `"mom"`.
`distributional`	Logical. If `TRUE`, the forest stores the training indices at every leaf node, enabling distributional (weight-based) predictions at test time. For each test point, the forest computes a weighted empirical distribution over training observations — weights are proportional to co-occurrence in the same leaf across trees — and a new Dirichlet is fitted to that weighted distribution via `est.method`. This produces a full predictive distribution rather than a single point estimate, at the cost of higher memory usage since leaf sample indices are retained for all trees. Default is `FALSE`.
`num.cores`	Number of OpenMP threads used for parallel tree building. The default is `-1` which uses all the cores on the system minus 1. Users may also specify `1` which means that the forest will be built sequentially.
`replace`	Logical. If `TRUE`, each tree is grown on a bootstrap sample drawn with replacement. If `FALSE` (default), each tree is grown on a subsample drawn without replacement. When `replace = FALSE` and `sample.fraction = 1` (the default), every tree sees all `n` observations and tree diversity comes entirely from random feature subsetting controlled by `mtry`. When `replace = FALSE` and `sample.fraction < 1`, each tree sees a different random subset of the data, enabling out-of-bag estimation.
`sample.fraction`	Numeric. Fraction of observations used to grow each tree, as a proportion of `n`. Default is `1.0`. When `replace = FALSE`, must be in `(0, 1]`; values greater than 1 are not allowed since you cannot draw more unique observations than available. When `replace = TRUE`, values greater than 1 are allowed (e.g. `1.5` draws `1.5n` bootstrap observations), though values in `(0, 1]` are most common. A warning is issued when `sample.fraction < 0.1` regardless of `replace`, as trees grown on very few observations tend to be unreliable.

`compute.oob` Logical. If `TRUE`, computes out-of-bag (OOB) predictions after the forest is built, using only trees for which each observation was not in training. Both the OOB prediction matrix and a scalar MSE are returned via `$oob$predictions` and `$oob$mse`. Not available when `replace = FALSE` and `sample.fraction = 1` since no held-out observations exist. Default is `FALSE`.

Details

Predictions. The fitted forest produces two complementary point-prediction surfaces: a *mean-based* prediction (the sample mean of training responses in the matched leaf) and a *parameter-based* prediction. The forest is also able to produce **full distributional (weight-based) predictions**: Trains a Distributional Random Forest which estimates the full conditional distribution $P(Y|X)$ for possibly multivariate response Y and predictors X. The conditional distribution estimate is represented as a weighted distribution of the training data. The weights can be conveniently used in the downstream analysis to estimate any quantity of interest $\tau(P(Y|X))$.

Out-of-bag (OOB) evaluation. Each observation is predicted exclusively by trees for which it was held out. The OOB prediction matrix and scalar OOB MSE are returned in `$oob`; the full prediction matrix is also available so users may apply any alternative compositional error measure such as the Aitchison distance.

Feature importance. Three complementary importance measures are computed automatically:

Gain (raw \& normalised) Total Dirichlet log-likelihood gain accumulated over every split where a feature was chosen, summed across all trees. The normalised version sums to 1, facilitating comparison across forests.

Split count Number of times a feature was selected as the best split variable across all internal nodes and all trees.

Permutation importance Computed post-hoc via `permutation_importance`: the mean increase in OOB loss when a feature's values are randomly permuted within each tree's OOB sample, with a scaled (t-statistic-like) variant that accounts for tree-to-tree variability. Supports Aitchison distance, MSE, and KL divergence as loss functions.

The implementation delegates all tree-building to compiled C++ code and uses **OpenMP** for parallel construction of trees.

Out-of-Bag (OOB) Predictions

When `compute.oob = TRUE`, each observation is predicted by averaging over only the trees for which it was out-of-bag. This requires `replace = TRUE` or `replace = FALSE` with `sample.fraction < 1`. The reported `$oob$mse` is the MSE between OOB predictions and true responses, averaged over components and OOB observations. Note that MSE is not universally accepted for compositional data since it ignores the simplex geometry — the Aitchison distance, which operates in log-ratio space, is an alternative. The full OOB prediction matrix `$oob$predictions` (n x k, with `NA` for observations never out-of-bag) is returned so users can apply any alternative error measure directly.

Value

A list of class `DirichletRF` which contains the following elements:

`type` Parallelisation type used: `"openmp"` or `"sequential"`.

`num.cores` Number of cores used.

`num.trees` Total number of trees in the forest.

`replace` Logical indicating whether bootstrap sampling was used.

`sample.fraction` The fraction of observations used per tree.

`compute.oob` Logical indicating whether OOB prediction was computed.

`distributional` Logical indicating whether the forest was built in distributional mode (leaf sample indices retained).

`est.method` The estimation method used (`"mom"` or `"mle"`).

`Y_train` The training compositional response matrix.

`fitted` A list of fitted values on the training data:

- `alpha_hat` Estimated Dirichlet alpha parameters (n x k matrix).
- `mean_based` Mean-based fitted values (n x k matrix), derived from sample means at each leaf.
- `param_based` Parameter-based fitted values (n x k matrix), obtained by normalising `alpha_hat` so rows sum to 1.

`residuals` A list of residuals (Y - fitted values):

- `mean_based` Residuals from mean-based predictions.
- `param_based` Residuals from parameter-based predictions.

`importance` A list of feature importance measures:

- `gain` Raw total likelihood gain per feature, summed over all trees and all splits where the feature was selected.
- `gain_normalised` Gain divided by total gain across all features, summing to 1. Recommended for interpretation and comparison across forests.
- `count` Number of times each feature was selected as the best split variable across all trees and all internal nodes.

`oob` A list of OOB results. All elements are `NA` or `NULL` when `compute.oob = FALSE`:

- `mse` Scalar OOB mean squared error, averaged over all components and all observations that appeared OOB at least once.
- `predictions` An (n x k) matrix of OOB predictions. Rows corresponding to observations that never appeared OOB are `NA`.
- `alpha_predictions` An (n x k) matrix of OOB Dirichlet alpha parameter estimates, averaged over OOB trees. `NA` for observations never out-of-bag.
- `weights` An (n x n) matrix of OOB proximity weights. `weights[i, j]` is the average fraction of OOB trees in which observations `i` and `j` landed in the same leaf. Only available when both `distributional = TRUE` and `compute.oob = TRUE`; `NULL` otherwise. The matrix is generally asymmetric since row `i` is averaged only over trees where `i` was out-of-bag.

References

Masoumifard, K., van der Westhuizen, S., & Gardner-Lubbe, S. (2026). Dirichlet random forest for predicting compositional data. In A. Bekker, P. Nagar, J. Ferreira, B. Erasmus, & A. Ramoelo (Eds.), Environmental Modelling with Contemporary Statistics: Learning, Directionality, and Space-Time Dynamics. Chapman & Hall/CRC. ISBN: 9781032903910.

See Also

`predict.DirichletRF` for point predictions on new data (call as `predict(forest, newdata)`, documented under `?predict.DirichletRF`). `print.DirichletRF` for a summary of the fitted object (call as `print(forest)` or just `forest`). `sample_conditional` for drawing compositional samples from the conditional predictive distribution (requires `distributional = TRUE`). `importance.DirichletRF` for impurity-based (gain and count) feature importance. `permutation_importance` for permutation-based OOB feature importance (requires `compute.oob = TRUE`). `predict_weights` for proximity weights for new observations (requires `distributional = TRUE`).

Examples

```
#   Minimal example (auto-tested)
set.seed(42)
n <- 50; p <- 2
X <- matrix(rnorm(n * p), n, p)
colnames(X) <- paste0("X", 1:p)
G <- matrix(rgamma(n * 3, shape = rep(c(2, 3, 4), each = n)), n, 3)
Y <- G / rowSums(G)

# Default: no bootstrap, no OOB, fastest configuration
forest <- DirichletRF(X, Y, num.trees = 5, num.cores = 1)
print(forest)

# Feature importance
importance(forest)

# Prediction on new data
Xtest <- matrix(rnorm(5 * p), 5, p)
colnames(Xtest) <- paste0("X", 1:p)
pred  <- predict(forest, Xtest)
pred$mean_predictions


#   Larger example with informative and noise covariates
set.seed(42)
n <- 200; p <- 6
X <- matrix(rnorm(n * p), n, p)
colnames(X) <- paste0("X", 1:p)

# X1 and X2 are informative, X3-X6 are noise
alpha_mat <- cbind(
  2 + 3 * (X[, 1] > 0),
  3 + 3 * (X[, 2] > 0),
```

```
  rep(4, n)
)
G <- matrix(rgamma(n * 3, shape = as.vector(t(alpha_mat))), n, 3,
            byrow = TRUE)
Y <- G / rowSums(G)

# Default: no bootstrap, no OOB
forest <- DirichletRF(X, Y, num.trees = 100, num.cores = 1)

# Feature importance - X1 and X2 should dominate
importance(forest)

# Fitted values and residuals
head(forest$fitted$mean_based)
head(forest$residuals$mean_based)

#   Bootstrap with OOB
forest_oob <- DirichletRF(X, Y, num.trees = 100, num.cores = 1,
                          replace = TRUE, sample.fraction = 1.0,
                          compute.oob = TRUE)
forest_oob$oob$mse
head(forest_oob$oob$predictions)

#   Subsampling without replacement with OOB
forest_sub <- DirichletRF(X, Y, num.trees = 100, num.cores = 1,
                          replace = FALSE, sample.fraction = 0.632,
                          compute.oob = TRUE)
forest_sub$oob$mse

#   Prediction
Xtest <- matrix(rnorm(10 * p), 10, p)
colnames(Xtest) <- paste0("X", 1:p)
pred <- predict(forest, Xtest)
head(pred$mean_predictions)
param_pred <- pred$alpha_predictions / rowSums(pred$alpha_predictions)

 #   Distributional forest with OOB weight matrix
forest_dist <- DirichletRF(X, Y, num.trees = 100, num.cores = 1,
                           replace = TRUE, sample.fraction = 1.0,
                           compute.oob = TRUE, distributional = TRUE)

# OOB weight matrix: n x n, W[i,j] = proximity of i to j via OOB trees
W <- forest_dist$oob$weights
dim(W)

# Symmetrise if a symmetric proximity matrix is preferred
W_sym <- (W + t(W)) / 2

# Weights for new observations
Xtest <- matrix(rnorm(5 * p), 5, p)
colnames(Xtest) <- paste0("X", 1:p)
W_new <- predict_weights(forest_dist, Xtest)  # 5 x n_train
```

`importance` *Feature Importance for a Dirichlet Forest*

Description

Returns a data frame summarising feature importance from a fitted `DirichletRF` object. Two measures are provided:

`gain` Total likelihood gain accumulated across all splits where this feature was selected (raw, summed over all trees).

`gain_normalised` Same as `gain` but normalised to sum to 1 across all features, making values comparable across forests of different sizes.

`count` Number of times the feature was chosen as the best split variable across all trees and all internal nodes.

The data frame is sorted by `gain_normalised` in descending order.

Usage

```
importance(object, ...)

## S3 method for class 'DirichletRF'
importance(object, ...)
```

Arguments

`object`	A `DirichletRF` object returned by `DirichletRF`.
`...`	Currently unused.

Value

A data frame with columns `feature`, `gain`, `gain_normalised`, and `count`, sorted by `gain_normalised` descending.

See Also

`predict.DirichletRF` for point predictions on new data. `print.DirichletRF` for a summary of the fitted object (call as `print(forest)` or just `forest`). `sample_conditional` for drawing compositional samples from the conditional predictive distribution (requires `distributional = TRUE`). `permutation_importance` for permutation-based OOB feature importance (requires `compute.oob = TRUE`). `predict_weights` for proximity weights for new observations (requires `distributional = TRUE`).

Examples

```
set.seed(42)
n <- 50; p <- 4
X <- matrix(rnorm(n * p), n, p)
colnames(X) <- paste0("X", 1:p)
G <- matrix(rgamma(n * 3, shape = rep(c(2, 3, 4), each = n)), n, 3)
Y <- G / rowSums(G)
forest <- DirichletRF(X, Y, num.trees = 10, num.cores = 1)
importance(forest)
```

---

permutation_importance

Permutation Feature Importance for a Dirichlet Forest

Description

Computes permutation-based variable importance (VI) for each feature. For each tree b and feature j, the OOB error is measured before and after randomly permuting column j within the OOB sample of that tree. The importance of feature j is:

Usage

```
permutation_importance(
  object,
  X,
  loss = c("aitchison", "mse", "kl"),
  num.permutations = 5L,
  seed = 42L
)
```

Arguments

`object`	A `DirichletRF` object fitted with `compute.oob = TRUE`.
`X`	The training covariate matrix (n x p) passed to `DirichletRF`. Required because leaf predictions are recomputed on permuted copies.
`loss`	Loss function used to measure OOB error. One of:
	`"aitchison"` (default) Mean Aitchison distance between predicted and true compositions. Respects simplex geometry. $d_A(y, \hat{y}) = \|\text{clr}(y) - \text{clr}(\hat{y})\|_2$.
	`"mse"` Mean squared error, averaged over components.
	`"kl"` Mean KL divergence $\sum_k y_k \log(y_k/\hat{y}_k)$.
`num.permutations`	Number of random permutations to average over per feature per tree. Higher values reduce Monte Carlo noise. Default is `5`.
`seed`	Integer random seed for reproducibility of permutations. Default is `42L`.

Details

$$\mathrm{VI}(j) = \frac{1}{B}\sum_{b=1}^{B}\left[\frac{1}{R}\sum_{r=1}^{R} L\Big(S_b,\, \tilde{X}^{(j,r)}\Big) - L(S_b,\, X)\right]$$

where S_b is the OOB index set of tree b, R is `num.permutations`, and $\tilde{X}^{(j,r)}$ is the data matrix with column j randomly permuted on replicate r, holding all other columns fixed.

The scaled version divides by the **population** standard deviation of the per-tree importances (denominator B, not $B-1$):

$$\mathrm{VI}_{\mathrm{scaled}}(j) = \frac{\mathrm{VI}(j)}{\widehat{\sigma}_j}, \quad \widehat{\sigma}_j^2 = \frac{1}{B}\sum_{b=1}^{B}\mathrm{VI}_{b,j}^2 - \mathrm{VI}(j)^2$$

where $\mathrm{VI}_{b,j}$ denotes the bracketed quantity above for a single tree.

Loss functions for compositional data

MSE ignores the simplex constraint and treats the components independently. The Aitchison distance operates in the log-ratio space that is natural for compositions and is the recommended default. KL divergence is asymmetric but common in information-theoretic contexts.

Small predicted values near zero can cause numerical issues for Aitchison and KL losses. A small constant (`1e-10`) is added to all predictions before computing these losses.

Interpretation

A feature with `importance` near zero (or negative, due to Monte Carlo noise) does not contribute to predictive accuracy. Features with large positive `importance_scaled` are robustly important across trees.

Value

A data frame with one row per feature and columns:

`feature` Feature name.

`importance` Mean increase in OOB loss when the feature is permuted ($\mathrm{VI}(j)$). Larger = more important.

`importance_scaled` Importance divided by its standard deviation across trees ($\mathrm{VI}_{\mathrm{scaled}}(j)$). Analogous to a t-statistic; values > 1 suggest a feature contributes meaningfully.

`importance_sd` Standard deviation of the per-tree importance values, giving a sense of variability.

Sorted by `importance` descending.

See Also

predict.DirichletRF for point predictions on new data. print.DirichletRF for a summary of the fitted object (call as `print(forest)` or just `forest`). sample_conditional for drawing compositional samples from the conditional predictive distribution (requires `distributional = TRUE`). importance.DirichletRF for impurity-based (gain and count) feature importance. predict_weights for proximity weights for new observations (requires `distributional = TRUE`).

Examples

```
set.seed(42)
n <- 100; p <- 4
X <- matrix(rnorm(n * p), n, p)
colnames(X) <- paste0("X", 1:p)
alpha_mat <- cbind(2 + 3 * (X[, 1] > 0), 3 + 3 * (X[, 2] > 0), rep(4, n))
G <- matrix(rgamma(n * 3, shape = as.vector(t(alpha_mat))), n, 3, byrow = TRUE)
Y <- G / rowSums(G)

forest <- DirichletRF(X, Y, num.trees = 50, num.cores = 1,
                      replace = TRUE, compute.oob = TRUE)
permutation_importance(forest, X)
```

`predict.DirichletRF` *Predict with a Dirichlet Forest*

Description

Makes predictions using a fitted `DirichletRF` object returned by DirichletRF.

Usage

```
## S3 method for class 'DirichletRF'
predict(object, newdata, ...)
```

Arguments

`object`	A `DirichletRF` object.
`newdata`	A numeric matrix of new covariates (n_new x p).
`...`	Currently unused.

Value

A list with the following elements:

`alpha_predictions` Estimated Dirichlet alpha parameters for each new observation (n_new x k matrix).

`mean_predictions` Mean-based compositional predictions (n_new x k matrix).

See Also

print.DirichletRF for a summary of the fitted object (call as `print(forest)` or just `forest`). sample_conditional for drawing compositional samples from the conditional predictive distribution (requires `distributional = TRUE`). importance.DirichletRF for impurity-based (gain and count) feature importance. permutation_importance for permutation-based OOB feature importance (requires `compute.oob = TRUE`). predict_weights for proximity weights for new observations (requires `distributional = TRUE`).

Examples

```
# Small toy example (auto-tested)
set.seed(42)
n <- 50; p <- 2
X <- matrix(rnorm(n * p), n, p)
G <- matrix(rgamma(n * 3, shape = rep(c(2, 3, 4), each = n)), n, 3)
Y <- G / rowSums(G)
forest  <- DirichletRF(X, Y, num.trees = 5, num.cores = 1)
Xtest   <- matrix(rnorm(5 * p), 5, p)
pred    <- predict(forest, Xtest)
pred$mean_predictions


n <- 500; p <- 4
X <- matrix(rnorm(n * p), n, p)
alpha <- c(2, 3, 4)
G <- matrix(rgamma(n * length(alpha), shape = rep(alpha, each = n)),
            n, length(alpha))
Y <- G / rowSums(G)
forest <- DirichletRF(X, Y, num.trees = 50, num.cores = 1)
Xtest  <- matrix(rnorm(10 * p), 10, p)
pred   <- predict(forest, Xtest)
param_pred  <- pred$alpha_predictions / rowSums(pred$alpha_predictions)
single_pred <- predict(forest, Xtest[1, , drop = FALSE])
```

---

predict_weights *Proximity Weights for New Observations*

Description

For each row of `newdata`, computes a normalised weight vector over all training observations, where the weight of training observation `j` is proportional to how often it co-occurs with the new point in the same leaf across all trees. These weights define the forest-weighted empirical distribution over the training responses and can be used to estimate conditional quantities such as means, variances, or probabilities for the new covariate point.

Usage

```
predict_weights(object, newdata)
```

Arguments

`object`	A `DirichletRF` object built with `distributional = TRUE`.
`newdata`	A numeric matrix of new covariates (n_test x p). Column order must match the training matrix passed to `DirichletRF`.

Details

Weights are computed using all trees in the forest (no OOB restriction applies, since new observations were not part of training). This contrasts with `$oob$weights`, which restricts each training observation to its held-out trees only and is available directly on the fitted object when both `distributional = TRUE` and `compute.oob = TRUE`.

Value

A numeric matrix of dimensions n_test x n_train. Row `i` contains the normalised proximity weights of the `i`-th new observation over all `n_train` training observations. Each row sums to 1. Entries are zero for training observations that never shared a leaf with the new point across any tree.

See Also

`predict.DirichletRF` for point predictions on new data. `print.DirichletRF` for a summary of the fitted object (call as `print(forest)` or just `forest`). `sample_conditional` for drawing compositional samples from the conditional predictive distribution (requires `distributional = TRUE`). `importance.DirichletRF` for impurity-based (gain and count) feature importance. `permutation_importance` for permutation-based OOB feature importance (requires `compute.oob = TRUE`).

Examples

```
set.seed(42)
n <- 100; p <- 4
X <- matrix(rnorm(n * p), n, p)
colnames(X) <- paste0("X", 1:p)
alpha_mat <- cbind(2 + 3 * (X[, 1] > 0), 3 + 3 * (X[, 2] > 0), rep(4, n))
G <- matrix(rgamma(n * 3, shape = as.vector(t(alpha_mat))), n, 3, byrow = TRUE)
Y <- G / rowSums(G)

forest <- DirichletRF(X, Y, num.trees = 50, num.cores = 1,
                      distributional = TRUE)

# Weights for 5 new observations - matrix is 5 x 100
Xtest <- matrix(rnorm(5 * p), 5, p)
colnames(Xtest) <- paste0("X", 1:p)
W <- predict_weights(forest, Xtest)
dim(W)       # 5 x 100
rowSums(W)   # all 1

# Weighted conditional mean for each new observation
Y_hat <- W %*% Y   # 5 x k
```

print.DirichletRF *Custom Print Method for DirichletRF Objects*

Description

Suppresses the display of large data matrices (Y_train, fitted, residuals) when the object is printed, while keeping them accessible via `$`.

Usage

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

Arguments

`x`	A `DirichletRF` object.
`...`	Further arguments passed to or from other methods.

Value

Invisibly returns `x`, the `DirichletRF` object unchanged. Called primarily for its side effect of printing a summary of the model to the console.

See Also

`predict.DirichletRF` for point predictions on new data (call as `predict(forest, newdata)`, documented under `?predict.DirichletRF`). `sample_conditional` for drawing compositional samples from the conditional predictive distribution (requires `distributional = TRUE`). `importance.DirichletRF` for impurity-based (gain and count) feature importance. `permutation_importance` for permutation-based OOB feature importance (requires `compute.oob = TRUE`). `predict_weights` for proximity weights for new observations (requires `distributional = TRUE`).

sample_conditional *Draw Conditional Samples from a Dirichlet Forest*

Description

Given a fitted `DirichletRF` built with `distributional = TRUE` and a single test covariate vector, draws `size` compositional observations from the forest-weighted empirical distribution over the training responses. Each training observation receives a weight proportional to how often it co-occurs with the test point in the same leaf across all trees; the returned rows are a weighted-bootstrap draw from those training Y rows.

Usage

```
sample_conditional(object, x_new, size = 100L)
```

Arguments

object	A DirichletRF object built with distributional = TRUE.
x_new	A numeric vector of length p (a single test covariate point).
size	A positive integer giving the number of compositional observations to draw. Default is 100L.

Value

A numeric matrix of dimensions size x k, where each row is one draw from the conditional distribution of Y given x_new. Row names are draw_1, draw_2, …and column names are inherited from the training Y matrix if available.

See Also

predict.DirichletRF for point predictions on new data. print.DirichletRF for a summary of the fitted object. importance.DirichletRF for impurity-based feature importance. permutation_importance for permutation-based OOB feature importance. predict_weights for proximity weights for new observations.

Examples

```
set.seed(1)
n <- 80; p <- 3
X <- matrix(rnorm(n * p), n, p)
G <- matrix(rgamma(n * 4, shape = rep(c(1, 2, 3, 4), each = n)), n, 4)
Y <- G / rowSums(G)
forest <- DirichletRF(X, Y, num.trees = 20, num.cores = 1,
                      distributional = TRUE)
x_test <- rnorm(p)
draws  <- sample_conditional(forest, x_test, size = 200L)
colMeans(draws)   # estimated conditional mean of Y | x_test
```

Index