Package: dubicube (via r-universe)

October 27, 2026

Title Calculation and Interpretation of Data Cube Indicator Uncertainty

Version 0.12.5

Description This R package provides functions to explore data cubes using simple measures and cross-validation techniques. It can also be used for uncertainty calculation using the bootstrap resampling method, and functionality is provided for efficient interpretation and visualisation of uncertainty related to indicators based on occurrence cubes.

License MIT + file LICENSE

URL https://github.com/b-cubed-eu/dubicube, https://b-cubed-eu.github.io/dubicube/, https://doi.org/10.5281/zenodo.14850237

BugReports https://github.com/b-cubed-eu/dubicube/issues

Depends R (>= 3.5.0)

Imports assertthat, boot, data.table, dplyr, effectclass, ggplot2, modelr, purrr, rlang, tibble

Suggests b3gbi, frictionless, testthat (>= 3.0.0), tidyr

Remotes github::b-cubed-eu/b3gbi, github::inbo/effectclass

Additional_repositories https://inbo.r-universe.dev, https://b-cubed-eu.r-universe.dev/

Config/checklist/communities b3; inbo

Config/checklist/keywords uncertainty quantification; uncertainty visualisation; biodiversity indicators; data cubes

Config/Needs/website rmarkdown

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev

Repository https://b-cubed-eu.r-universe.dev

Date/Publication 2026-07-01 11:39:59 UTC

RemoteUrl https://github.com/b-cubed-eu/dubicube

RemoteRef HEAD

RemoteSha 54a6f91b55f6cd61243aa7dce15930707ece7e6b

Contents

add_effect_classification . . . 3
basic_ci . . . 4
basic_cube_rules . . . 6
bca_ci . . . 7
boot_list_to_dataframe . . . 9
bootstrap_cube . . . 10
bootstrap_cube_raw . . . 14
calculate_acceleration . . . 17
calculate_boot_ci_from_boot . . . 20
calculate_bootstrap_ci . . . 22
cross_validate_cube . . . 26
diagnose_cube . . . 29
filter_cube . . . 30
norm_ci . . . 32
perc_ci . . . 34
plot.cube_diagnostics . . . 35
print.cube_diagnostics . . . 36
resolve_bootstrap_method . . . 37
rule_obs_min_records . . . 39
rule_obs_min_total . . . 40
rule_spatial_max_uncertainty . . . 41
rule_spatial_min_cells . . . 41
rule_spatial_miss_uncertainty . . . 42
rule_taxon_min_taxa . . . 43
rule_temporal_min_years . . . 43
rule_temporal_missing_years . . . 44
summary.cube_diagnostics . . . 45

Index **46**

`add_effect_classification` *Add effect classifications to a dataframe by comparing the confidence intervals with a reference and thresholds*

Description

This function adds classified effects to a dataframe as ordered factor variables by comparing the confidence intervals with a reference and thresholds.

Usage

```
add_effect_classification(
  df,
  cl_columns,
  threshold,
  reference = 0,
  coarse = TRUE
)
```

Arguments

`df`	A dataframe containing summary data of confidence limits. Two columns are required containing lower and upper limits indicated by the `cl_columns` argument. Any other columns are optional.
`cl_columns`	A vector of 2 column names in `df` indicating respectively the lower and upper confidence limits (e.g. `c("lcl", "ucl")`).
`threshold`	A vector of either 1 or 2 thresholds. A single threshold will be transformed into `reference + c(-abs(threshold), abs(threshold))`.
`reference`	The null hypothesis value to compare confidence intervals against. Defaults to 0.
`coarse`	Logical, defaults to `TRUE`. If `TRUE`, add a coarse classification to the dataframe.

Details

This function is a wrapper around `effectclass::classify()` and `effectclass::coarse_classification()` from the **effectclass** package (Onkelinx, 2023). They classify effects in a stable and transparent manner.

Symbol	Fine effect / trend	Coarse effect / trend	Rule
++	strong positive effect / strong increase	positive effect / increase	confidence interval above the upper th
+	positive effect / increase	positive effect / increase	confidence interval above reference an
+~	moderate positive effect / moderate increase	positive effect / increase	confidence interval between reference
~	no effect / stable	no effect / stable	confidence interval between threshold
-~	moderate negative effect / moderate decrease	negative effect / decrease	confidence interval between reference
-	negative effect / decrease	negative effect / decrease	confidence interval below reference an

-	strong negative effect / strong decrease	negative effect / decrease	confidence interval below the lower th
?+	potential positive effect / potential increase	unknown effect / unknown	confidence interval contains reference
?-	potential negative effect / potential decrease	unknown effect / unknown	confidence interval contains reference
?	unknown effect / unknown	unknown effect / unknown	confidence interval contains the lower

Value

The returned value is a modified version of the original input dataframe `df` with additional columns `effect_code` and `effect` containing respectively the effect symbols and descriptions as ordered factor variables. In case of `coarse = TRUE` (by default) also `effect_code_coarse` and `effect_coarse` containing the coarse classification effects.

References

Onkelinx, T. (2023). effectclass: Classification and visualisation of effects [Computer software]. https://inbo.github.io/effectclass/

Examples

```
# Example dataset
ds <- data.frame(
  mean = c(0, 0.5, -0.5, 1, -1, 1.5, -1.5, 0.5, -0.5, 0),
  sd = c(1, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.25, 0.25, 0.5)
)
ds$lcl <- qnorm(0.05, ds$mean, ds$sd)
ds$ucl <- qnorm(0.95, ds$mean, ds$sd)

add_effect_classification(
 df = ds,
 cl_columns = c("lcl", "ucl"),
 threshold = 1,
 reference = 0,
 coarse = TRUE
)
```

---

basic_ci *Calculate basic bootstrap confidence interval*

Description

This function calculates a basic confidence interval from a bootstrap sample. It is used by `calculate_bootstrap_ci()`.

Usage

```
basic_ci(t0, t, conf = 0.95, h = function(t) t, hinv = function(t) t)
```

Arguments

`t0`	Original statistic.
`t`	Numeric vector of bootstrap replicates.
`conf`	A numeric value specifying the confidence level of the interval. Default is `0.95` (95 % confidence level).
`h`	A function defining a transformation. The intervals are calculated on the scale of `h(t)` and the inverse function `hinv` applied to the resulting intervals. It must be a function of one variable only. The default is the identity function.
`hinv`	A function, like `h`, which returns the inverse of `h`. It is used to transform the intervals calculated on the scale of `h(t)` back to the original scale. The default is the identity function. If `h` is supplied but `hinv` is not, then the intervals returned will be on the transformed scale.

Details

$$CI_{basic} = \left[2\hat{\theta} - \hat{\theta}^*_{(1-\alpha/2)}, 2\hat{\theta} - \hat{\theta}^*_{(\alpha/2)}\right]$$

where $\hat{\theta}^*_{(\alpha/2)}$ and $\hat{\theta}^*_{(1-\alpha/2)}$ are the $\alpha/2$ and $1-\alpha/2$ percentiles of the bootstrap distribution, respectively.

Value

A matrix with four columns:

- `conf`: confidence level
- `rk_lower`: rank of lower endpoint (interpolated)
- `rk_upper`: rank of upper endpoint (interpolated)
- `ll`: lower confidence limit
- `ul`: lower confidence limit

Note

This function is adapted from the internal function `basic.ci()` in the **boot** package (Canty & Ripley, 1999).

References

Canty, A., & Ripley, B. (1999). boot: Bootstrap Functions (Originally by Angelo Canty for S) [Computer software]. https://CRAN.R-project.org/package=boot

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

See Also

Other interval_calculation: `bca_ci()`, `norm_ci()`, `perc_ci()`

Examples

```
set.seed(123)
boot_reps <- rnorm(1000)
t0 <- mean(boot_reps)

# Basic bootstrap CI
basic_ci(t0, boot_reps, conf = 0.95)
```

`basic_cube_rules` *Basic diagnostic rules for data cubes*

Description

Returns basic diagnostic rules used by `diagnose_cube()`. Each rule defines how a specific data quality metric is computed and evaluated.

Usage

```
basic_cube_rules()
```

Details

Rules are implemented as lists containing:

- `id` – name of the diagnostic metric
- `dimension` – cube dimension being evaluated (e.g. temporal)
- `thresholds` – reference values used to determine severity
- `compute()` – function that calculates the metric
- `severity()` – function assigning a severity level
- `message()` – function generating a human-readable message

Contains the following rules:

- `rule_temporal_min_years()`: Number of years
- `rule_temporal_missing_years()`: Missing years
- `rule_spatial_min_cells()`: Number of grid cells
- `rule_spatial_max_uncertainty()`: Number of records where coordinate uncertainty is larger than grid resolution
- `rule_spatial_miss_uncertainty`: Number of records with missing coordinate uncertainty
- `rule_taxon_min_taxa()`: Number of taxa
- `rule_obs_min_records()`: Number of records (rows)
- `rule_obs_min_total()`: Total number of observations (sum)

Default thresholds are used.

Value

A list of diagnostic rule definitions.

bca_ci *Calculate Bias-Corrected and Accelerated (BCa) bootstrap confidence interval*

Description

This function calculates a Bias-Corrected and Accelerated (BCa) confidence interval from a bootstrap sample. It is used by `calculate_bootstrap_ci()`.

Usage

```
bca_ci(t0, t, a, conf = 0.95, h = function(t) t, hinv = function(t) t)
```

Arguments

`t0`	Original statistic.
`t`	Numeric vector of bootstrap replicates.
`a`	Acceleration constant. See also `calculate_acceleration()`.
`conf`	A numeric value specifying the confidence level of the interval. Default is `0.95` (95 % confidence level).
`h`	A function defining a transformation. The intervals are calculated on the scale of `h(t)` and the inverse function `hinv` applied to the resulting intervals. It must be a function of one variable only. The default is the identity function.
`hinv`	A function, like `h`, which returns the inverse of `h`. It is used to transform the intervals calculated on the scale of `h(t)` back to the original scale. The default is the identity function. If `h` is supplied but `hinv` is not, then the intervals returned will be on the transformed scale.

Details

Adjusts for bias and acceleration. Bias refers to the systematic difference between the observed statistic from the original dataset and the center of the bootstrap distribution of the statistic. The bias correction term is calculated as follows:

$$\hat{z}_0 = \Phi^{-1}\left(\frac{\#(\hat{\theta}_b^* < \hat{\theta})}{B}\right)$$

where $\#$ is the counting operator, counting the number of times $\hat{\theta}_b^*$ is smaller than $\hat{\theta}$, and Φ^{-1} the inverse cumulative density function of the standard normal distribution.B is the number of bootstrap samples.

Acceleration quantifies how sensitive the variability of the statistic is to changes in the data. See `calculate_acceleration()` on how this is calculated.

- $a = 0$: The statistic's variability does not depend on the data (e.g., symmetric distribution)

- $a > 0$: Small changes in the data have a large effect on the statistic's variability (e.g., positive skew)
- $a < 0$: Small changes in the data have a smaller effect on the statistic's variability (e.g., negative skew).

The bias and acceleration estimates are then used to calculate adjusted percentiles.

$\alpha_1 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{\alpha/2}}{1 - \hat{a}(\hat{z}_0 + z_{\alpha/2})}\right)$, $\alpha_2 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{1-\alpha/2}}{1 - \hat{a}(\hat{z}_0 + z_{1-\alpha/2})}\right)$

So, we get

$$CI_{bca} = \left[\hat{\theta}^*_{(\alpha_1)}, \hat{\theta}^*_{(\alpha_2)}\right]$$

Value

A matrix with four columns:

- `conf`: confidence level
- `rk_lower`: rank of lower endpoint (interpolated)
- `rk_upper`: rank of upper endpoint (interpolated)
- `ll`: lower confidence limit
- `ul`: lower confidence limit

Note

This function is adapted from the internal function `bca.ci()` in the **boot** package (Canty & Ripley, 1999).

References

Canty, A., & Ripley, B. (1999). boot: Bootstrap Functions (Originally by Angelo Canty for S) [Computer software]. https://CRAN.R-project.org/package=boot

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

See Also

Other interval_calculation: `basic_ci()`, `norm_ci()`, `perc_ci()`

Examples

```
set.seed(123)
boot_reps <- rnorm(1000)
t0 <- mean(boot_reps)

# Example acceleration value (normally estimated via jackknife)
a <- 0.01

# BCa bootstrap CI
bca_ci(t0, boot_reps, a, conf = 0.95)
```

boot_list_to_dataframe *Convert a list of 'boot' objects to a tidy dataframe*

Description

This function converts a named list of "boot" objects (typically produced by bootstrap_cube() into a single long-format dataframe. Each element of the list is assumed to correspond to one group, with the list names defining the values of the grouping variable.

Usage

```
boot_list_to_dataframe(boot_list, grouping_var)
```

Arguments

boot_list	A named list of objects of class "boot", as returned by boot::boot(). Each list element must correspond to exactly one group, and the list names are used as the values of the grouping variable.
grouping_var	A character string giving the name of the grouping variable (e.g. "year"). This will be used as the column name in the returned dataframe.

Details

This function is primarily intended for use with bootstrapping using the bootstrap_cube() function generated with boot methods.

The function assumes that each boot object in boot_list contains a single bootstrap statistic per replicate (i.e. boot$t is a vector or a one-column matrix).

Value

A dataframe with the following columns:

- sample: Sample ID of the bootstrap replicate
- est_original: The statistic based on the full dataset per group
- rep_boot: The statistic based on a bootstrapped dataset (bootstrap replicate)
- est_boot: The bootstrap estimate (mean of bootstrap replicates per group)
- se_boot: The standard error of the bootstrap estimate (standard deviation of the bootstrap replicates per group)
- bias_boot: The bias of the bootstrap estimate per group

See Also

Other indicator_uncertainty_helper: bootstrap_cube_raw(), calculate_acceleration(), calculate_boot_ci_from_b
resolve_bootstrap_method()

Examples

```
## Not run:
# After processing a data cube with b3gbi::process_cube()

# Function to calculate statistic of interest
# Mean observations per year
mean_obs <- function(x) {
  out_df <- aggregate(obs ~ year, x, mean) # Calculate mean obs per year
  names(out_df) <- c("year", "diversity_val") # Rename columns
  return(out_df)
}
mean_obs(processed_cube$data)

# Perform bootstrapping
bootstrap_mean_obs <- bootstrap_cube(
  data_cube = processed_cube,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  method = "boot_group_specific",
  seed = 123
)

bootstrap_df <- boot_list_to_dataframe(
  boot_list = bootstrap_mean_obs,
  grouping_var = "year"
)

head(bootstrap_df)

## End(Not run)
```

`bootstrap_cube` *Perform bootstrapping over a data cube for a calculated statistic*

Description

This function generate `samples` bootstrap replicates of a statistic applied to a data cube. It resamples the data cube and computes a statistic `fun` for each bootstrap replicate, optionally comparing the results to a reference group (`ref_group`).

Usage

```
bootstrap_cube(
  data_cube,
  fun,
  ...,
  grouping_var,
  samples = 1000,
```

```
  ref_group = NA,
  seed = NA,
  processed_cube = TRUE,
  method = "smart",
  progress = FALSE,
  boot_args = list()
)
```

Arguments

data_cube	A data cube object (class 'processed_cube' or 'sim_cube', see `b3gbi::process_cube()`) or a dataframe (cf. `$data` slot of 'processed_cube' or 'sim_cube'). If `processed_cube = TRUE` (default), this must be a processed or simulated data cube that contains a `$data` element.
fun	A function which, when applied to `data_cube$data` returns the statistic(s) of interest (or just `data_cube` in case of a dataframe). This function must return a dataframe with a column `diversity_val` containing the statistic of interest.
...	Additional arguments passed on to `fun`.
grouping_var	A character vector specifying the grouping variable(s) for the bootstrap analysis. The function `fun(data_cube$data, ...)` should return a row per group. The specified variables must not be redundant, meaning they should not contain the same information (e.g., `"time_point"` (1, 2, 3) and `"year"` (2000, 2001, 2002) should not be used together if `"time_point"` is just an alternative encoding of `"year"`).
samples	The number of bootstrap replicates. A single positive integer. Default is 1000.
ref_group	A string indicating the reference group to compare the statistic with. Default is `NA`, meaning no reference group is used.
seed	A positive numeric value setting the seed for random number generation to ensure reproducibility. If `NA` (default), then `set.seed()` is not called at all. If not `NA`, then the random number generator state is reset (to the state before calling this function) upon exiting this function.
processed_cube	Logical. If `TRUE` (default), the function expects `data_cube` to be a data cube object with a `$data` slot. If `FALSE`, the function expects `data_cube` to be a dataframe.
method	A character string specifying the bootstrap method. Options include: • `"smart"`: Automatically select the appropriate bootstrap method based on indicator behaviour and the presence of a reference group (default). • `"boot_whole_cube"`: Perform whole-cube bootstrap using `boot::boot()`. Cannot be used with `ref_group`. • `"boot_group_specific"`: Perform group-specific bootstrap using `boot::boot()`. Cannot be used with `ref_group`. • `"whole_cube"`: Perform whole-cube bootstrap without using the **boot** package. Can be used with `ref_group`. • `"group_specific"`: Perform group-specific bootstrap without using the **boot** package. Can be used with `ref_group`.

progress	Logical. Whether to show a progress bar. Set to TRUE to display a progress bar, FALSE (default) to suppress it.
boot_args	Named list of additional arguments passed to boot::boot().

Details

Bootstrapping is a statistical technique used to estimate the distribution of a statistic by resampling with replacement from the original data (Davison & Hinkley, 1997; Efron & Tibshirani, 1994). In the case of data cubes, each row is sampled with replacement. Below are the common notations used in bootstrapping:

1. **Original Sample Data**: $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$
   - The initial set of data points. Here, n is the sample size. This corresponds to the number of cells in a data cube or the number of rows in tabular format.
2. **Statistic of Interest**: θ
   - The parameter or statistic being estimated, such as the mean $\bar{X}$, variance σ^2, or a biodiversity indicator. Let $\hat{\theta}$ denote the estimated value of θ calculated from the complete dataset $\mathbf{X}$.
3. **Bootstrap Sample**: $\mathbf{X}^* = \{X_1^*, X_2^*, \ldots, X_n^*\}$
   - A sample of size n drawn with replacement from the original sample $\mathbf{X}$. Each X_i^* is drawn independently from $\mathbf{X}$.
   - A total of B bootstrap samples are drawn from the original data. Common choices for B are 1000 or 10,000 to ensure a good approximation of the distribution of the bootstrap replications (see further).
4. **Bootstrap Replication**: $\hat{\theta}_b^*$
   - The value of the statistic of interest calculated from the b-th bootstrap sample $\mathbf{X}_b^*$. For example, if θ is the sample mean, $\hat{\theta}_b^* = \bar{X}_b^*$.
5. **Bootstrap Statistics**:

- **Bootstrap Estimate of the Statistic**: $\hat{\theta}_{\text{boot}}$
  - The average of the bootstrap replications:

$$\hat{\theta}_{\text{boot}} = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}_b^*$$

- **Bootstrap Bias**: $\text{Bias}_{\text{boot}}$
  - This bias indicates how much the bootstrap estimate deviates from the original sample estimate. It is calculated as the difference between the average bootstrap estimate and the original estimate:

$$\text{Bias}_{\text{boot}} = \frac{1}{B} \sum_{b=1}^{B} (\hat{\theta}_b^* - \hat{\theta}) = \hat{\theta}_{\text{boot}} - \hat{\theta}$$

- **Bootstrap Standard Error**: SE_{boot}

  - The standard deviation of the bootstrap replications, which estimates the variability of the statistic.

There are two methods for bootstrapping:

- Whole-cube bootstrapping: resampling all rows in the cube, regardless of grouping. For indicators that are use data across groups.
- Group-specific bootstrapping: resampling rows only within a group of interest (e.g., a species, year, or habitat). For indicators that are calculated independently per group.

The default smart option (`method = "smart"`) determines both (i) whether the indicator is group-specific or whole-cube, and (ii) whether the **boot** package should be used.

The decision is made by calculating the statistic on larger and smaller subsets of the data (containing respectively more and fewer groups in `grouping_var`). If indicator values for the common groups are identical, the indicator is treated as group-specific; otherwise, it is treated as whole-cube.

If no reference group is used (`ref_group = NA`), `method = "smart"` resolves to `"boot_group_specific"` or `"boot_whole_cube"`, both of which use `boot::boot()`. If a reference group is specified, `method = "smart"` resolves to `"group_specific"` or `"whole_cube"` and bootstrapping is handled internally.

Value

A dataframe containing the bootstrap results with the following columns:

- `sample`: Sample ID of the bootstrap replicate
- `est_original`: The statistic based on the full dataset per group
- `rep_boot`: The statistic based on a bootstrapped dataset (bootstrap replicate)
- `est_boot`: The bootstrap estimate (mean of bootstrap replicates per group)
- `se_boot`: The standard error of the bootstrap estimate (standard deviation of the bootstrap replicates per group)
- `bias_boot`: The bias of the bootstrap estimate per group

If `method` resolves to `"boot_whole_cube"` or `"boot_group_specific"`, the returned value is a named list of objects of class `"boot"`, as produced by `boot::boot()`.

References

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

Efron, B., & Tibshirani, R. J. (1994). An Introduction to the Bootstrap (1st ed.). Chapman and Hall/CRC. doi:10.1201/9780429246593

See Also

Other indicator_uncertainty: `calculate_bootstrap_ci()`

Examples

```
## Not run:
# After processing a data cube with b3gbi::process_cube()

# Function to calculate statistic of interest
# Mean observations per year
mean_obs <- function(x) {
  out_df <- aggregate(obs ~ year, x, mean) # Calculate mean obs per year
  names(out_df) <- c("year", "diversity_val") # Rename columns
  return(out_df)
}
mean_obs(processed_cube$data)

# Perform bootstrapping
bootstrap_mean_obs <- bootstrap_cube(
  data_cube = processed_cube,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123
)

## End(Not run)
```

`bootstrap_cube_raw` *Perform bootstrapping over a dataframe for a calculated statistic*

Description

This function generate `samples` bootstrap replicates of a statistic applied to a dataframe. It resamples the data cube and computes a statistic `fun` for each bootstrap replicate, optionally comparing the results to a reference group (`ref_group`). Bootstrapping happens over the whole dataset `data_cube`.

Usage

```
bootstrap_cube_raw(
  data_cube,
  fun,
  ...,
  grouping_var,
  samples = 1000,
  ref_group = NA,
  seed = NA,
  progress = FALSE
)
```

Arguments

data_cube	A dataframe.
fun	A function which, when applied to data_cube$data returns the statistic(s) of interest (or just data_cube in case of a dataframe). This function must return a dataframe with a column diversity_val containing the statistic of interest.
...	Additional arguments passed on to fun.
grouping_var	A character vector specifying the grouping variable(s) for the bootstrap analysis. The function fun(data_cube$data, ...) should return a row per group. The specified variables must not be redundant, meaning they should not contain the same information (e.g., "time_point" (1, 2, 3) and "year" (2000, 2001, 2002) should not be used together if "time_point" is just an alternative encoding of "year").
samples	The number of bootstrap replicates. A single positive integer. Default is 1000.
ref_group	A string indicating the reference group to compare the statistic with. Default is NA, meaning no reference group is used.
seed	A positive numeric value setting the seed for random number generation to ensure reproducibility. If NA (default), then set.seed() is not called at all. If not NA, then the random number generator state is reset (to the state before calling this function) upon exiting this function.
progress	Logical. Whether to show a progress bar. Set to TRUE to display a progress bar, FALSE (default) to suppress it.

Details

Bootstrapping is a statistical technique used to estimate the distribution of a statistic by resampling with replacement from the original data (Davison & Hinkley, 1997; Efron & Tibshirani, 1994). In the case of data cubes, each row is sampled with replacement. Below are the common notations used in bootstrapping:

1. **Original Sample Data**: $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$
   - The initial set of data points. Here, n is the sample size. This corresponds to the number of cells in a data cube or the number of rows in tabular format.
2. **Statistic of Interest**: θ
   - The parameter or statistic being estimated, such as the mean $\bar{X}$, variance σ^2, or a biodiversity indicator. Let $\hat{\theta}$ denote the estimated value of θ calculated from the complete dataset $\mathbf{X}$.
3. **Bootstrap Sample**: $\mathbf{X}^* = \{X_1^*, X_2^*, \ldots, X_n^*\}$
   - A sample of size n drawn with replacement from the original sample $\mathbf{X}$. Each X_i^* is drawn independently from $\mathbf{X}$.
   - A total of B bootstrap samples are drawn from the original data. Common choices for B are 1000 or 10,000 to ensure a good approximation of the distribution of the bootstrap replications (see further).
4. **Bootstrap Replication**: $\hat{\theta}_b^*$

- The value of the statistic of interest calculated from the b-th bootstrap sample $\mathbf{X}_b^*$. For example, if θ is the sample mean, $\hat{\theta}_b^* = \bar{X}_b^*$.

5. **Bootstrap Statistics**:

- **Bootstrap Estimate of the Statistic**: $\hat{\theta}_{\text{boot}}$
  - The average of the bootstrap replications:

$$\hat{\theta}_{\text{boot}} = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}_b^*$$

- **Bootstrap Bias**: $\text{Bias}_{\text{boot}}$
  - This bias indicates how much the bootstrap estimate deviates from the original sample estimate. It is calculated as the difference between the average bootstrap estimate and the original estimate:

$$\text{Bias}_{\text{boot}} = \frac{1}{B} \sum_{b=1}^{B} (\hat{\theta}_b^* - \hat{\theta}) = \hat{\theta}_{\text{boot}} - \hat{\theta}$$

- **Bootstrap Standard Error**: SE_{boot}
  - The standard deviation of the bootstrap replications, which estimates the variability of the statistic.

Value

A dataframe containing the bootstrap results with the following columns:

- `sample`: Sample ID of the bootstrap replicate
- `est_original`: The statistic based on the full dataset per group
- `rep_boot`: The statistic based on a bootstrapped dataset (bootstrap replicate)
- `est_boot`: The bootstrap estimate (mean of bootstrap replicates per group)
- `se_boot`: The standard error of the bootstrap estimate (standard deviation of the bootstrap replicates per group)
- `bias_boot`: The bias of the bootstrap estimate per group

References

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

Efron, B., & Tibshirani, R. J. (1994). An Introduction to the Bootstrap (1st ed.). Chapman and Hall/CRC. doi:10.1201/9780429246593

See Also

Other indicator_uncertainty_helper: `boot_list_to_dataframe()`, `calculate_acceleration()`, `calculate_boot_ci_from_boot()`, `resolve_bootstrap_method()`

Examples

```
## Not run:
# Function to calculate statistic of interest
# Mean observations per year
mean_obs <- function(x) {
  out_df <- aggregate(obs ~ year, x, mean) # Calculate mean obs per year
  names(out_df) <- c("year", "diversity_val") # Rename columns
  return(out_df)
}
mean_obs(data)

# Perform bootstrapping
bootstrap_mean_obs <- bootstrap_cube_raw(
  data_cube = data,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123
)
head(bootstrap_mean_obs)

## End(Not run)
```

---

calculate_acceleration

Calculate acceleration for a statistic in a dataframe

Description

This function calculates acceleration values, which quantify the sensitivity of a statistic's variability to changes in the dataset. Acceleration is used for bias-corrected and accelerated (BCa) confidence intervals in `calculate_bootstrap_ci()`.

Usage

```
calculate_acceleration(
  data_cube,
  fun,
  ...,
  grouping_var,
  ref_group = NA,
  influence_method = "usual",
  processed_cube = TRUE,
  progress = FALSE
)
```

Arguments

data_cube	A data cube object (class 'processed_cube' or 'sim_cube', see b3gbi::process_cube()) or a dataframe (cf. $data slot of 'processed_cube' or 'sim_cube'). If processed_cube = TRUE (default), this must be a processed or simulated data cube that contains a $data element.
fun	A function which, when applied to data_cube$data returns the statistic(s) of interest (or just data_cube in case of a dataframe). This function must return a dataframe with a column diversity_val containing the statistic of interest. As used by bootstrap_cube().
...	Additional arguments passed on to fun.
grouping_var	A character vector specifying the grouping variable(s) for the bootstrap analysis. The function fun(data_cube$data, ...) should return a row per group. The specified variables must not be redundant, meaning they should not contain the same information (e.g., "time_point" (1, 2, 3) and "year" (2000, 2001, 2002) should not be used together if "time_point" is just an alternative encoding of "year"). This variable is used to split the dataset into groups for separate acceleration calculations.
ref_group	A string indicating the reference group to compare the statistic with. Default is NA, meaning no reference group is used. As used by bootstrap_cube().
influence_method	A string specifying the method used for calculating the influence values. • "usual": Negative jackknife (default if BCa is selected). • "pos": Positive jackknife
processed_cube	Logical. If TRUE (default), the function expects data_cube to be a data cube object with a $data slot. If FALSE, the function expects data_cube to be a dataframe.
progress	Logical. Whether to show a progress bar for jackknifing. Set to TRUE to display a progress bar, FALSE (default) to suppress it.

Details

Acceleration quantifies how sensitive the variability of a statistic θ is to changes in the data.

- $a = 0$: The statistic's variability does not depend on the data (e.g., symmetric distribution)
- $a > 0$: Small changes in the data have a large effect on the statistic's variability (e.g., positive skew)
- $a < 0$: Small changes in the data have a smaller effect on the statistic's variability (e.g., negative skew).

It is used for BCa confidence interval calculation, which adjust for bias and skewness in bootstrapped distributions (Davison & Hinkley, 1997, Chapter 5). See also the empinf() function of the **boot** package in R (Canty & Ripley, 1999)). The acceleration is calculated as follows:

$$\hat{a} = \frac{1}{6} \frac{\sum_{i=1}^{n} (I_i^3)}{\left(\sum_{i=1}^{n} (I_i^2)\right)^{3/2}}$$

where I_i denotes the influence of data point x_i on the estimation of θ. I_i can be estimated using jackknifing. Examples are (1) the negative jackknife: $I_i = (n-1)(\hat{\theta} - \hat{\theta}_{-i})$, and (2) the positive jackknife $I_i = (n+1)(\hat{\theta}_{-i} - \hat{\theta})$ (Frangos & Schucany, 1990). Here, $\hat{\theta}_{-i}$ is the estimated value leaving out the i'th data point x_i. The **boot** package also offers infinitesimal jackknife and regression estimation. Implementation of these jackknife algorithms can be explored in the future.

If a reference group is used, jackknifing is implemented in a different way. Consider $\hat{\theta} = \hat{\theta}_1 - \hat{\theta}_2$ where $\hat{\theta}_1$ is the estimate for the indicator value of a non-reference period (sample size n_1) and $\hat{\theta}_2$ is the estimate for the indicator value of a reference period (sample size n_2). The acceleration is now calculated as follows:

$$\hat{a} = \frac{1}{6} \frac{\sum_{i=1}^{n_1+n_2} (I_i^3)}{\left(\sum_{i=1}^{n_1+n_2} (I_i^2)\right)^{3/2}}$$

I_i can be calculated using the negative or positive jackknife. Such that

$\hat{\theta}_{-i} = \hat{\theta}_{1,-i} - \hat{\theta}_2$ for $i = 1, \ldots, n_1$, and

$\hat{\theta}_{-i} = \hat{\theta}_1 - \hat{\theta}_{2,-i}$ for $i = n_1 + 1, \ldots, n_1 + n_2$

Value

A dataframe containing the acceleration values per `grouping_var`.

References

Canty, A., & Ripley, B. (1999). boot: Bootstrap Functions (Originally by Angelo Canty for S) [Computer software]. https://CRAN.R-project.org/package=boot

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

Frangos, C. C., & Schucany, W. R. (1990). Jackknife estimation of the bootstrap acceleration constant. Computational Statistics & Data Analysis, 9(3), 271–281. doi:10.1016/01679473(90)90109U

See Also

Other indicator_uncertainty_helper: `boot_list_to_dataframe()`, `bootstrap_cube_raw()`, `calculate_boot_ci_from_b`
`resolve_bootstrap_method()`

Examples

```
## Not run:
# After processing a data cube with b3gbi::process_cube()

# Function to calculate statistic of interest
# Mean observations per year
mean_obs <- function(x) {
  out_df <- aggregate(obs ~ year, x, mean) # Calculate mean obs per year
  names(out_df) <- c("year", "diversity_val") # Rename columns
  return(out_df)
}
mean_obs(processed_cube$data)
```

```
# Calculate acceleration
acceleration_df <- calculate_acceleration(
  data_cube = processed_cube,
  fun = mean_obs,
  grouping_var = "year",
  progress = FALSE
)
acceleration_df

## End(Not run)
```

`calculate_boot_ci_from_boot` *Calculate confidence intervals from a 'boot' object*

Description

This function calculates multiple types of confidence intervals (normal, basic, percentile, BCa) for a **boot** object using `boot::boot.ci()`.

Usage

```
calculate_boot_ci_from_boot(
  boot_obj,
  type = c("norm", "basic", "perc", "bca"),
  conf = 0.95,
  h = function(t) t,
  hinv = function(t) t,
  boot_args = list()
)
```

Arguments

`boot_obj`	A `boot` object (from the **boot** package).
`type`	A character vector specifying the type(s) of confidence intervals to compute. Options include: • `"perc"`: Percentile interval • `"bca"`: Bias-corrected and accelerated interval • `"norm"`: Normal interval • `"basic"`: Basic interval • `"all"`: Compute all available interval types (default)
`conf`	A numeric value specifying the confidence level of the intervals. Default is `0.95` (95 % confidence level).
`h`	A function defining a transformation. The intervals are calculated on the scale of `h(t)` and the inverse function `hinv` applied to the resulting intervals. It must be a function of one variable only. The default is the identity function.

hinv	A function, like h, which returns the inverse of h. It is used to transform the intervals calculated on the scale of h(t) back to the original scale. The default is the identity function. If h is supplied but hinv is not, then the intervals returned will be on the transformed scale.
boot_args	Named list of additional arguments to pass to boot::boot.ci().

Value

A tidy dataframe with columns:

- stat_index: Index of statistic in the boot object
- est_original: Original estimate from full dataset
- int_type: Interval type
- ll: Lower confidence limit
- ul: Upper confidence limit
- conf: Confidence level

See Also

Other indicator_uncertainty_helper: boot_list_to_dataframe(), bootstrap_cube_raw(), calculate_acceleration(), resolve_bootstrap_method()

Examples

```
## Not run:
library(boot)

# Function to compute the mean
mean_fun <- function(data, indices) {
  mean(data[indices])
}

# Bootstrap mean of the 'mpg' variable in mtcars
set.seed(123)
boot_obj <- boot(data = mtcars$mpg, statistic = mean_fun, R = 1000)

# Calculate confidence intervals for all types
ci_df <- calculate_boot_ci_from_boot(
  boot_obj = boot_obj,
  type = "all",
  conf = 0.95
)
ci_df

## End(Not run)
```

calculate_bootstrap_ci *Calculate confidence intervals from bootstrap results*

Description

This function calculates confidence intervals for a dataframe containing bootstrap replicates based on different methods, including percentile (perc), bias-corrected and accelerated (bca), normal (norm), and basic (basic). The function also supports a boot object from the **boot** package.

Usage

```
calculate_bootstrap_ci(
  bootstrap_results,
  grouping_var = NULL,
  type = c("perc", "bca", "norm", "basic"),
  conf = 0.95,
  h = function(t) t,
  hinv = function(t) t,
  no_bias = FALSE,
  aggregate = TRUE,
  data_cube = NA,
  fun = NA,
  ...,
  ref_group = NA,
  influence_method = ifelse(is.element("bca", type), "usual", NA),
  progress = FALSE,
  boot_args = list()
)
```

Arguments

bootstrap_results
: A dataframe with bootstrap replicates, or a boot object, or a list of boot objects. For dataframes, each row is a bootstrap replicate and must include columns rep_boot, est_original, and grouping variables. For boot objects, confidence intervals are either computed directly using boot::boot.ci() or the objects are converted to a dataframe, depending on the value of no_bias.

grouping_var
: A character vector specifying the grouping variable(s) for the bootstrap analysis. The function fun(data_cube$data, ...) should return a row per group. The specified variables must not be redundant, meaning they should not contain the same information (e.g., "time_point" (1, 2, 3) and "year" (2000, 2001, 2002) should not be used together if "time_point" is just an alternative encoding of "year"). This variable is used to split the dataset into groups for separate confidence interval calculations.

type
: A character vector specifying the type(s) of confidence intervals to compute. Options include:

- "perc": Percentile interval
- "bca": Bias-corrected and accelerated interval
- "norm": Normal interval
- "basic": Basic interval
- "all": Compute all available interval types (default)

conf — A numeric value specifying the confidence level of the intervals. Default is 0.95 (95 % confidence level).

h — A function defining a transformation. The intervals are calculated on the scale of h(t) and the inverse function hinv applied to the resulting intervals. It must be a function of one variable only. The default is the identity function.

hinv — A function, like h, which returns the inverse of h. It is used to transform the intervals calculated on the scale of h(t) back to the original scale. The default is the identity function. If h is supplied but hinv is not, then the intervals returned will be on the transformed scale.

no_bias — Logical. If TRUE intervals are centered around the original estimates (bias is ignored). Default is FALSE.

aggregate — Logical. If TRUE (default), the function returns distinct confidence limits per group. If FALSE, the confidence limits are added to the original bootstrap dataframe bootstrap_results.

data_cube — Only used when type = "bca" and no boot method is used. A data cube object (class 'processed_cube' or 'sim_cube', see b3gbi::process_cube()) or a dataframe (cf. $data slot of 'processed_cube' or 'sim_cube'). As used by bootstrap_cube().

fun — Only used when type = "bca" and no boot method is used. A function which, when applied to data_cube$data returns the statistic(s) of interest (or just data_cube in case of a dataframe). This function must return a dataframe with a column diversity_val containing the statistic of interest. As used by bootstrap_cube().

... — Additional arguments passed on to fun.

ref_group — Only used when type = "bca". A string indicating the reference group to compare the statistic with. Default is NA, meaning no reference group is used. As used by bootstrap_cube().

influence_method — A string specifying the method used for calculating the influence values.

- "usual": Negative jackknife (default if BCa is selected).
- "pos": Positive jackknife

progress — Logical. Whether to show a progress bar for jackknifing. Set to TRUE to display a progress bar, FALSE (default) to suppress it.

boot_args — Named list of additional arguments passed to boot::boot.ci().

Details

We consider four different types of intervals (with confidence level α). The choice for confidence interval types and their calculation is in line with the **boot** package in R (Canty & Ripley, 1999) to ensure ease of implementation. They are based on the definitions provided by Davison & Hinkley (1997, Chapter 5) (see also DiCiccio & Efron, 1996; Efron, 1987).

1. **Percentile**: Uses the percentiles of the bootstrap distribution.

$$CI_{perc} = \left[\hat{\theta}^*_{(\alpha/2)}, \hat{\theta}^*_{(1-\alpha/2)}\right]$$

where $\hat{\theta}^*_{(\alpha/2)}$ and $\hat{\theta}^*_{(1-\alpha/2)}$ are the $\alpha/2$ and $1-\alpha/2$ percentiles of the bootstrap distribution, respectively.

2. **Bias-Corrected and Accelerated (BCa)**: Adjusts for bias and acceleration

Bias refers to the systematic difference between the observed statistic from the original dataset and the center of the bootstrap distribution of the statistic. The bias correction term is calculated as follows:

$$\hat{z}_0 = \Phi^{-1}\left(\frac{\#(\hat{\theta}^*_b < \hat{\theta})}{B}\right)$$

where $\#$ is the counting operator, counting the number of times $\hat{\theta}^*_b$ is smaller than $\hat{\theta}$, and Φ^{-1} the inverse cumulative density function of the standard normal distribution.B is the number of bootstrap samples.

Acceleration quantifies how sensitive the variability of the statistic is to changes in the data. See `calculate_acceleration()` on how this is calculated.

- $a = 0$: The statistic's variability does not depend on the data (e.g., symmetric distribution)
- $a > 0$: Small changes in the data have a large effect on the statistic's variability (e.g., positive skew)
- $a < 0$: Small changes in the data have a smaller effect on the statistic's variability (e.g., negative skew).

The bias and acceleration estimates are then used to calculate adjusted percentiles.
$\alpha_1 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{\alpha/2}}{1-\hat{a}(\hat{z}_0+z_{\alpha/2})}\right)$, $\alpha_2 = \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{1-\alpha/2}}{1-\hat{a}(\hat{z}_0+z_{1-\alpha/2})}\right)$
So, we get

$$CI_{bca} = \left[\hat{\theta}^*_{(\alpha_1)}, \hat{\theta}^*_{(\alpha_2)}\right]$$

3. **Normal**: Assumes the bootstrap distribution of the statistic is approximately normal

$$CI_{norm} = \left[\hat{\theta} - \text{Bias}_{\text{boot}} - \text{SE}_{\text{boot}} \times z_{1-\alpha/2}, \hat{\theta} - \text{Bias}_{\text{boot}} + \text{SE}_{\text{boot}} \times z_{1-\alpha/2}\right]$$

where $z_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the standard normal distribution.

4. **Basic**: Centers the interval using percentiles

$$CI_{basic} = \left[2\hat{\theta} - \hat{\theta}^*_{(1-\alpha/2)}, 2\hat{\theta} - \hat{\theta}^*_{(\alpha/2)}\right]$$

where $\hat{\theta}^*_{(\alpha/2)}$ and $\hat{\theta}^*_{(1-\alpha/2)}$ are the $\alpha/2$ and $1-\alpha/2$ percentiles of the bootstrap distribution, respectively.

Value

A dataframe containing the bootstrap results with the following columns:

- `est_original`: The statistic based on the full dataset per group
- `est_boot`: The bootstrap estimate (mean of bootstrap replicates per group)
- `se_boot`: The standard error of the bootstrap estimate (standard deviation of the bootstrap replicates per group)
- `bias_boot`: The bias of the bootstrap estimate per group
- `int_type`: The interval type
- `ll`: The lower limit of the confidence interval
- `ul`: The upper limit of the confidence interval
- `conf`: The confidence level of the interval When `aggregate = FALSE`, the dataframe contains the columns from `bootstrap_results` with one row per bootstrap replicate.

References

Canty, A., & Ripley, B. (1999). boot: Bootstrap Functions (Originally by Angelo Canty for S) [Computer software]. https://CRAN.R-project.org/package=boot

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

DiCiccio, T. J., & Efron, B. (1996). Bootstrap confidence intervals. Statistical Science, 11(3). doi:10.1214/ss/1032280214

Efron, B. (1987). Better Bootstrap Confidence Intervals. Journal of the American Statistical Association, 82(397), 171–185. doi:10.1080/01621459.1987.10478410

Efron, B., & Tibshirani, R. J. (1994). An Introduction to the Bootstrap (1st ed.). Chapman and Hall/CRC. doi:10.1201/9780429246593

See Also

Other indicator_uncertainty: `bootstrap_cube()`

Examples

```
## Not run:
# After processing a data cube with b3gbi::process_cube()

# Function to calculate statistic of interest
# Mean observations per year
mean_obs <- function(x) {
  out_df <- aggregate(obs ~ year, x, mean) # Calculate mean obs per year
  names(out_df) <- c("year", "diversity_val") # Rename columns
  return(out_df)
}
mean_obs(processed_cube$data)

# Perform bootstrapping
bootstrap_mean_obs <- bootstrap_cube(
```

```
  data_cube = processed_cube,
  fun = mean_obs,
  grouping_var = "year",
  samples = 1000,
  seed = 123
)

# Calculate confidence limits
# Percentile interval
ci_mean_obs <- calculate_bootstrap_ci(
  bootstrap_results = bootstrap_mean_obs,
  grouping_var = "year",
  type = "perc",
  conf = 0.95
)
ci_mean_obs

## End(Not run)
```

`cross_validate_cube` *Leave-one-out cross-validation for data cubes*

Description

This function performs leave-one-out (LOO) or k-fold (experimental) cross-validation (CV) on a biodiversity data cube to assess the performance of a specified indicator function. It partitions the data by a specified variable, calculates the specified indicator on training data, and compares it with the true values to evaluate the influence of one or more categories on the final result.

Usage

```
cross_validate_cube(
  data_cube,
  fun,
  ...,
  grouping_var,
  out_var = "taxonKey",
  crossv_method = c("loo", "kfold"),
  k = ifelse(crossv_method == "kfold", 5, NA),
  max_out_cats = 1000,
  processed_cube = TRUE,
  progress = FALSE
)
```

Arguments

`data_cube`	A data cube object (class 'processed_cube' or 'sim_cube', see `b3gbi::process_cube()`) or a dataframe (cf. `$data` slot of 'processed_cube' or 'sim_cube'). If `processed_cube = TRUE` (default), this must be a processed or simulated data cube that contains a `$data` element.

fun	A function which, when applied to data_cube$data returns the statistic(s) of interest (or just data_cube in case of a dataframe). This function must return a dataframe with a column diversity_val containing the statistic of interest.
...	Additional arguments passed on to fun.
grouping_var	A character vector specifying the grouping variable(s) for fun. The output of fun(data_cube) returns a row per group.
out_var	A string specifying the column by which the data should be left out iteratively. Default is "taxonKey" which can be used for leave-one-species-out CV.
crossv_method	Method of data partitioning. If crossv_method = "loo" (default), S = number of unique values in ou training partitions are created containing S - 1 rows each. If crossv_method = "kfold", the aggregated data is split the data into k exclusive partitions containing S / k rows each. K-fold CV is experimental and results should be interpreted with caution.
k	Number of folds (an integer). Used only if crossv_method = "kfold". Default 5.
max_out_cats	An integer specifying the maximum number of unique categories in out_var to leave out iteratively. Default is 1000. This can be increased if needed, but keep in mind that a high number of categories in out_var may significantly increase runtime.
processed_cube	Logical. If TRUE (default), the function expects data_cube to be a data cube object with a $data slot. If FALSE, the function expects data_cube to be a dataframe.
progress	Logical. Whether to show a progress bar. Set to TRUE to display a progress bar, FALSE (default) to suppress it.

Details

This function assesses the influence of each category in out_var on the indicator value by iteratively leaving out one category at a time, similar to leave-one-out cross-validation. K-fold CV works in a similar fashion but is experimental and will not be covered here.

1. **Original Sample Data**: $\mathbf{X} = \{X_{11}, X_{12}, X_{13}, \ldots, X_{sn}\}$
   - The initial set of data points, where there are s different categories in out_var and n total samples across all categories (= the sample size). n corresponds to the number of cells in a data cube or the number of rows in tabular format.
2. **Statistic of Interest**: θ
   - The parameter or statistic being estimated, such as the mean $\bar{X}$, variance σ^2, or a biodiversity indicator. Let $\hat{\theta}$ denote the estimated value of θ calculated from the complete dataset $\mathbf{X}$.
3. **Cross-Validation (CV) Sample**: $\mathbf{X}_{-s_j}$
   - The full dataset $\mathbf{X}$ excluding all samples belonging to category j. This subset is used to investigate the influence of category j on the estimated statistic $\hat{\theta}$.
4. **CV Estimate for Category j**: $\hat{\theta}_{-s_j}$
   - The value of the statistic of interest calculated from $\mathbf{X}_{-s_j}$, which excludes category j. For example, if θ is the sample mean, $\hat{\theta}_{-s_j} = \bar{X}_{-s_j}$.

5. **Error Measures**:
   - The **Error** is the difference between the statistic estimated without category j ($\hat{\theta}_{-s_j}$) and the statistic calculated on the complete dataset ($\hat{\theta}$).

$$\text{Error}_{s_j} = \hat{\theta}_{-s_j} - \hat{\theta}$$

   - The **Relative Error** is the absolute error, normalised by the true estimate $\hat{\theta}$ and a small error term $\epsilon = 10^{-8}$ to avoid division by zero.

$$\text{Rel. Error}_{s_j} = \frac{|\hat{\theta}_{-s_j} - \hat{\theta}|}{\hat{\theta} + \epsilon}$$

   - The **Percent Error** is the relative error expressed as a percentage.

$$\text{Perc. Error}_{s_j} = \text{Rel. Error}_{s_j} \times 100\%$$

6. **Summary Measures**:
   - The **Mean Relative Error (MRE)** is the average of the relative errors over all categories.

$$\text{MRE} = \frac{1}{s}\sum_{j=1}^{s}\text{Rel. Error}_{s_j}$$

   - The **Mean Squared Error (MSE)** is the average of the squared errors.

$$\text{MSE} = \frac{1}{s}\sum_{j=1}^{s}(\text{Error}_{s_j})^2$$

   - The **Root Mean Squared Error (RMSE)** is the square root of the MSE.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

Value

A dataframe containing the cross-validation results with the following columns:

- Cross-Validation id (`id_cv`)
- The grouping variable `grouping_var` (e.g., year)
- The category left out during each cross-validation iteration (specified `out_var` with suffix '_out' in lower case)
- The computed statistic values for both training (`rep_cv`) and true datasets (`est_original`)
- Error metrics: error (`error`), squared error (`sq_error`), absolute difference (`abs_error`), relative difference (`rel_error`), and percent difference (`perc_error`)
- Error metrics summarised by `grouping_var`: mean relative difference (`mre`), mean squared error (`mse`) and root mean squared error (`rmse`)

See Details section on how these error metrics are calculated.

Examples

```
## Not run:
# After processing a data cube with b3gbi::process_cube()

# Function to calculate statistic of interest
# Mean observations per year
mean_obs <- function(x) {
  out_df <- aggregate(obs ~ year, x, mean) # Calculate mean obs per year
  names(out_df) <- c("year", "diversity_val") # Rename columns
  return(out_df)
}
mean_obs(processed_cube$data)

# Perform leave-one-species-out CV
cv_mean_obs <- cross_validate_cube(
  data_cube = processed_cube,
  fun = mean_obs,
  grouping_var = "year",
  out_var = "taxonKey",
  crossv_method = "loo",
  progress = FALSE
)
head(cv_mean_obs)

## End(Not run)
```

---

diagnose_cube *Diagnose data quality of a processed data cube*

Description

Evaluates a set of diagnostic rules describing the data quality of a biodiversity occurrence cube. Each rule computes a metric on the cube and assigns a severity level indicating potential limitations of the data for exploratory analysis or indicator calculation.

Usage

```
diagnose_cube(data_cube, rules = "basic", verbose = TRUE, ...)
```

Arguments

data_cube	A processed_cube object as returned by b3gbi::process_cube().
rules	Diagnostic rules to evaluate. Can be: • A character vector referring to built-in rule sets (e.g. "basic", "spatial"). • A list of rule objects. • A combination of both.
verbose	Logical indicating whether a diagnostic summary should be printed.
...	Additional arguments passed to print.cube_diagnostics() in case verbose = TRUE.

Value

An object of class `cube_diagnostics`, containing one row per metric with the following columns:

- `dimension`: Dimension of the cube being evaluated (e.g. `"spatial"`, `"temporal"`, `"taxonomical"`).
- `metric`: Name of the diagnostic metric.
- `value`: Computed metric value.
- `severity`: Severity level (`"ok"`, `"note"`, `"important"`, `"very_important"`).
- `message`: Human-readable description of the diagnostic result.

The rule objects are attached as an attribute of the diagnostics object.

See Also

Other data_exploration: `filter_cube()`

Examples

```
# Example cube
# ! Real cubes should be processed with b3gbi::process_cube()
processed_cube <- list(
  data = data.frame(
    obs = c(5, 2, 10, 1),
    year = c(2001, 2001, 2002, 2003),
    minCoordinateUncertaintyInMeters = c(50, 2000, NA, 10)
  ),
  resolutions = "10km"
)
class(processed_cube) <- "processed_cube"

# Diagnose based on default rules
diag <- diagnose_cube(processed_cube)

# Sort diagnoses
diag <- diagnose_cube(processed_cube, sort_summary = "asc")

# Only show at least important diagnoses
diag <- diagnose_cube(processed_cube, filter_summary = "important")
```

`filter_cube` *Filter a processed data cube using diagnostic rules*

Description

Filters observations from a `processed_cube` based on rule definitions. Filtering reuses the rule infrastructure used by `diagnose_cube()`, but applies row-level filtering logic through rule-specific `filter_fn()` functions.

Usage

```
filter_cube(
  data_cube,
  rules = NULL,
  diagnostics = NULL,
  ...,
  process_cube_args = list()
)
```

Arguments

data_cube	A processed_cube object as returned by b3gbi::process_cube().
rules	Character vector or list of cube rule objects. Ignored if diagnostics is supplied.
diagnostics	Optional cube_diagnostics object returned by diagnose_cube(). If provided, rules are extracted from this object.
...	Additional arguments passed to rule-specific filter_fn() functions.
process_cube_args	Named list of additional arguments passed to b3gbi::process_cube() when rebuilding the filtered cube. For example, list(cols_occurrences = "n"). The argument cube_name is automatically supplied and must not be included.

Details

The function evaluates rule-specific filter_fn() functions that return a logical vector indicating which rows should be removed. Only rules that implement a filter_fn() are applied. Rules without a filtering function are ignored.

Filtering rules operate independently from diagnostic severity levels. For example, a cube may have acceptable overall diagnostics while still containing individual observations that fail filtering criteria.

After filtering, the function attempts to rebuild the cube using b3gbi::process_cube() to ensure cube metadata remains consistent. If this function is unavailable or fails, the filtered data replaces data_cube$data directly and the original cube metadata is retained. In that case a warning is issued.

Value

A filtered processed_cube.

See Also

Other data_exploration: diagnose_cube()

Examples

```
# Example cube
# ! Real cubes should be processed with b3gbi::process_cube()
processed_cube <- list(
  data = data.frame(
```

```
    obs = c(5, 2, 10, 1),
    year = c(2001, 2001, 2002, 2003),
    minCoordinateUncertaintyInMeters = c(50, 2000, NA, 10)
  ),
  resolutions = "10km"
)
class(processed_cube) <- "processed_cube"

# Filter cube based on rule
filtered_cube1 <- filter_cube(
  processed_cube,
  rules = list(rule_spatial_miss_uncertainty())
)

# Filter cube based cube diagnostics
diag <- diagnose_cube(
  processed_cube,
  rules = list(
    rule_spatial_miss_uncertainty(),
    rule_temporal_missing_years()
  )
)

filtered_cube2 <- filter_cube(
  processed_cube,
  diagnostics = diag
)

# The results are identical
identical(filtered_cube1$data, filtered_cube2$data)
```

`norm_ci` *Calculate normal bootstrap confidence interval*

Description

This function calculates a normal confidence interval from a bootstrap sample. It is used by `calculate_bootstrap_ci()`.

Usage

```
norm_ci(
  t0,
  t,
  conf = 0.95,
  h = function(t) t,
  hinv = function(t) t,
  no_bias = FALSE
)
```

Arguments

`t0`	Original statistic.
`t`	Numeric vector of bootstrap replicates.
`conf`	A numeric value specifying the confidence level of the interval. Default is `0.95` (95 % confidence level).
`h`	A function defining a transformation. The intervals are calculated on the scale of `h(t)` and the inverse function `hinv` applied to the resulting intervals. It must be a function of one variable only. The default is the identity function.
`hinv`	A function, like `h`, which returns the inverse of `h`. It is used to transform the intervals calculated on the scale of `h(t)` back to the original scale. The default is the identity function. If `h` is supplied but `hinv` is not, then the intervals returned will be on the transformed scale.
`no_bias`	Logical. If `TRUE` intervals are centered around the original estimates (bias is ignored). Default is `FALSE`.

Details

$$CI_{norm} = \left[\hat{\theta} - \text{Bias}_{\text{boot}} - \text{SE}_{\text{boot}} \times z_{1-\alpha/2}, \hat{\theta} - \text{Bias}_{\text{boot}} + \text{SE}_{\text{boot}} \times z_{1-\alpha/2}\right]$$

where $z_{1-\alpha/2}$ is the $1 - \alpha/2$ quantile of the standard normal distribution.

Value

A matrix with four columns:

- `conf`: confidence level
- `ll`: lower confidence limit
- `ul`: lower confidence limit

Note

This function is adapted from the function `norm.ci()` in the **boot** package (Canty & Ripley, 1999).

References

Canty, A., & Ripley, B. (1999). boot: Bootstrap Functions (Originally by Angelo Canty for S) [Computer software]. https://CRAN.R-project.org/package=boot

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

See Also

Other interval_calculation: `basic_ci()`, `bca_ci()`, `perc_ci()`

Examples

```
set.seed(123)
boot_reps <- rnorm(1000)
t0 <- mean(boot_reps)

# Normal-based CI
norm_ci(t0, boot_reps, conf = 0.90)

# Without bias correction
norm_ci(t0, boot_reps, conf = 0.90, no_bias = TRUE)
```

`perc_ci` *Calculate percentile bootstrap confidence interval*

Description

This function calculates a percentile confidence interval from a bootstrap sample. It is used by `calculate_bootstrap_ci()`.

Usage

```
perc_ci(t, conf = 0.95, h = function(t) t, hinv = function(t) t)
```

Arguments

`t`	Numeric vector of bootstrap replicates.
`conf`	A numeric value specifying the confidence level of the interval. Default is `0.95` (95 % confidence level).
`h`	A function defining a transformation. The intervals are calculated on the scale of `h(t)` and the inverse function `hinv` applied to the resulting intervals. It must be a function of one variable only. The default is the identity function.
`hinv`	A function, like `h`, which returns the inverse of `h`. It is used to transform the intervals calculated on the scale of `h(t)` back to the original scale. The default is the identity function. If `h` is supplied but `hinv` is not, then the intervals returned will be on the transformed scale.

Details

$$CI_{perc} = \left[\hat{\theta}^*_{(\alpha/2)}, \hat{\theta}^*_{(1-\alpha/2)}\right]$$

where $\hat{\theta}^*_{(\alpha/2)}$ and $\hat{\theta}^*_{(1-\alpha/2)}$ are the $\alpha/2$ and $1-\alpha/2$ percentiles of the bootstrap distribution, respectively.

Value

A matrix with four columns:

- `conf`: confidence level
- `rk_lower`: rank of lower endpoint (interpolated)
- `rk_upper`: rank of upper endpoint (interpolated)
- `ll`: lower confidence limit
- `ul`: lower confidence limit

Note

This function is adapted from the internal function `perc.ci()` in the **boot** package (Canty & Ripley, 1999).

References

Canty, A., & Ripley, B. (1999). boot: Bootstrap Functions (Originally by Angelo Canty for S) [Computer software]. https://CRAN.R-project.org/package=boot

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap Methods and their Application (1st ed.). Cambridge University Press. doi:10.1017/CBO9780511802843

See Also

Other interval_calculation: `basic_ci()`, `bca_ci()`, `norm_ci()`

Examples

```
set.seed(123)
boot_reps <- rnorm(1000)      # bootstrap replicates
t0 <- mean(boot_reps)         # observed statistic

# Percentile CI
perc_ci(boot_reps, conf = 0.95)
```

---

plot.cube_diagnostics *Plot cube diagnostics*

Description

Visualises diagnostic results returned by `diagnose_cube()`. The plot summarises the number of diagnostics per severity level and cube dimension.

Usage

```
## S3 method for class 'cube_diagnostics'
plot(x, type = "severity", ...)
```

Arguments

x	A cube_diagnostics object returned by diagnose_cube().
type	Type of plot. Options are "severity" (default), "dimension", or "rule".
...	Additional arguments passed to other methods (currently unused).

Details

Three visualisations are supported:

- "severity": Number of diagnostics per severity level.
- "dimension": Diagnostics grouped by cube dimension.
- "rule": Severity levels per diagnostic rule and dimension.

Value

A ggplot object.

See Also

Other diagnostic_methods: print.cube_diagnostics(), summary.cube_diagnostics()

---

print.cube_diagnostics

Print cube diagnostics

Description

Displays a human-readable summary of data cube diagnostics produced by diagnose_cube(). Each diagnostic metric is shown with a severity flag, the metric name, and a short explanatory message.

Usage

```
## S3 method for class 'cube_diagnostics'
print(x, filter_summary = "ok", sort_summary = NA, ...)
```

Arguments

x	A cube_diagnostics object returned by diagnose_cube().
filter_summary	Filter the summary output based on a minimum severity level. Default, all levels are shown: filter_summary = "ok".
sort_summary	Sort the summary output based on severity level. Options are descending ("desc"), ascending ("asc") or no sorting (NA, default).
...	Additional arguments passed to other methods (currently unused).

Details

Severity levels are indicated using coloured symbols:

- green ball: ok
- yellow ball: note
- orange ball: important
- red ball: very important

Value

The input object `x`, returned invisibly.

See Also

Other diagnostic_methods: `plot.cube_diagnostics()`, `summary.cube_diagnostics()`

`resolve_bootstrap_method` *Resolve bootstrap method including use of the boot package*

Description

Resolves the effective bootstrap method to be used by `bootstrap_cube()`, combining:

Usage

```
resolve_bootstrap_method(
  df,
  fun,
  ...,
  cat_var,
  ref_group = NA,
  method = "smart"
)
```

Arguments

`df`	A dataframe.
`fun`	A function which, when applied to `df`, returns the statistic(s) of interest. This function must return a dataframe with a column `diversity_val`.
`...`	Additional arguments passed to `fun`.
`cat_var`	A character vector specifying the grouping variable(s) used by `fun`.
`ref_group`	A value indicating the reference group. If `NA` (default), bootstrapping may be delegated to the **boot** package.
`method`	Character string specifying the bootstrap method. One of `"whole_cube"`, `"group_specific"`, `"boot_whole_cube"`, `"boot_group_specific"`, or `"smart"` (default).

Details

- the scope of the indicator (group-specific vs whole-cube), and
- whether a reference group is used.

When `method = "smart"`, the scope of the indicator is inferred using `derive_bootstrap_method()`. If no reference group is specified (`ref_group = NA`) and exactly one grouping variable is used (`length(cat_var) == 1`), the corresponding `boot_*` method is selected.

The resolution follows these rules:

1. If `method` is not `"smart"`, it is returned unchanged.
2. If `method = "smart"`, the indicator scope is inferred using `derive_bootstrap_method()`.
3. If more than one grouping variable is specified (`length(cat_var) > 1`), bootstrapping via the **boot** package is disabled and the inferred non-boot method is returned.
4. If exactly one grouping variable is used and `ref_group = NA`, the resolved method is prefixed with `"boot_"`, resulting in `"boot_group_specific"` or `"boot_whole_cube"`.
5. If a reference group is specified, the non-boot variants `"group_specific"` or `"whole_cube"` are returned.

Value

A single character string giving the resolved bootstrap method:

- `"whole_cube"`
- `"group_specific"`
- `"boot_whole_cube"`
- `"boot_group_specific"`

See Also

Other indicator_uncertainty_helper: `boot_list_to_dataframe()`, `bootstrap_cube_raw()`, `calculate_acceleration()`, `calculate_boot_ci_from_boot()`

Examples

```
# Example 1: Group-specific indicator without a reference group
# Mean sepal length per species (calculated independently per group)
mean_sepal_length <- function(x) {
  out_df <- aggregate(Sepal.Length ~ Species, x, mean)
  names(out_df) <- c("Species", "diversity_val")
  out_df
}

resolve_bootstrap_method(
  df = iris,
  fun = mean_sepal_length,
  cat_var = "Species",
  ref_group = NA,
  method = "smart"
```

```
)

# Example 2: Group-specific indicator with a reference group
resolve_bootstrap_method(
  df = iris,
  fun = mean_sepal_length,
  cat_var = "Species",
  ref_group = "setosa",
  method = "smart"
)

# Example 3: Indicator that depends on the whole cube
# The statistic per species depends on all species together
scaled_sepal_length <- function(x) {
  out_df <- aggregate(Sepal.Length ~ Species, x, mean)
  out_df$Sepal.Length <- out_df$Sepal.Length / nrow(out_df)
  names(out_df) <- c("Species", "diversity_val")
  out_df
}

resolve_bootstrap_method(
  df = iris,
  fun = scaled_sepal_length,
  cat_var = "Species",
  ref_group = NA,
  method = "smart"
)
```

`rule_obs_min_records` *Minimum number of records diagnostic rule*

Description

Creates a diagnostic rule that evaluates whether a data cube contains a sufficient number of observation records (rows). The rule counts the number of records present in the cube and compares it to a threshold to determine the severity level.

Usage

```
rule_obs_min_records(
  thresholds = c(ok = 40, note = 30, important = 20, very_important = 0)
)
```

Arguments

`thresholds`	Named numeric vector with severity thresholds: ok, note, important, very_important. Defaults are used if not provided.

Value

An object of class cube_rule.

See Also

Other diagnostic_rules: rule_obs_min_total(), rule_spatial_max_uncertainty(), rule_spatial_min_cells(), rule_spatial_miss_uncertainty(), rule_taxon_min_taxa(), rule_temporal_min_years(), rule_temporal_missing_years()

---

rule_obs_min_total *Minimum total number of observations diagnostic rule*

Description

Creates a diagnostic rule that evaluates whether a data cube contains a sufficient number of total observations, using a named vector of thresholds for severity classification.

Usage

```
rule_obs_min_total(
  thresholds = c(ok = 40, note = 30, important = 20, very_important = 0)
)
```

Arguments

thresholds	Named numeric vector with severity thresholds: ok, note, important, very_important. Defaults are used if not provided.

Value

An object of class cube_rule.

See Also

Other diagnostic_rules: rule_obs_min_records(), rule_spatial_max_uncertainty(), rule_spatial_min_cells(), rule_spatial_miss_uncertainty(), rule_taxon_min_taxa(), rule_temporal_min_years(), rule_temporal_missing_years()

rule_spatial_max_uncertainty

Maximal coordinate uncertainty diagnostic rule

Description

Creates a diagnostic rule that evaluates whether a data cube contains a records with high coordinate uncertainty. The rule counts the number of records (rows) in the cube where the minimal coordinate uncertainty is larger than the resolution of the grid, and compares it to a threshold to determine the severity level.

Usage

```
rule_spatial_max_uncertainty(
  thresholds = c(ok = 0, note = 1, important = 3, very_important = 5)
)
```

Arguments

thresholds	Named numeric vector with severity thresholds: ok, note, important, very_important. Defaults are used if not provided.

Value

An object of class cube_rule.

See Also

Other diagnostic_rules: rule_obs_min_records(), rule_obs_min_total(), rule_spatial_min_cells(), rule_spatial_miss_uncertainty(), rule_taxon_min_taxa(), rule_temporal_min_years(), rule_temporal_missing_years()

rule_spatial_min_cells

Spatial minimum grid cells diagnostic rule

Description

Creates a diagnostic rule that evaluates whether a data cube contains a sufficient number of spatial observations (grid cells). The rule counts the number of unique grid cells present in the cube and compares it to a threshold to determine the severity level.

Usage

```
rule_spatial_min_cells(
  thresholds = c(ok = 5, note = 3, important = 0, very_important = NULL)
)
```

Arguments

thresholds	Named numeric vector with severity thresholds: ok, note, important, very_important. Defaults are used if not provided.

Value

An object of class cube_rule.

See Also

Other diagnostic_rules: rule_obs_min_records(), rule_obs_min_total(), rule_spatial_max_uncertainty(), rule_spatial_miss_uncertainty(), rule_taxon_min_taxa(), rule_temporal_min_years(), rule_temporal_missing_years()

---

rule_spatial_miss_uncertainty

Missing coordinate uncertainty diagnostic rule

Description

Creates a diagnostic rule that evaluates whether a data cube contains a records with missing coordinate uncertainty. The rule counts the number of records (rows) with missing coordinate uncertainty and compares it to a threshold to determine the severity level.

Usage

```
rule_spatial_miss_uncertainty(
  thresholds = c(ok = 0, note = 1, important = 3, very_important = 5)
)
```

Arguments

thresholds	Named numeric vector with severity thresholds: ok, note, important, very_important. Defaults are used if not provided.

Value

An object of class cube_rule.

See Also

Other diagnostic_rules: rule_obs_min_records(), rule_obs_min_total(), rule_spatial_max_uncertainty(), rule_spatial_min_cells(), rule_taxon_min_taxa(), rule_temporal_min_years(), rule_temporal_missing_years

rule_taxon_min_taxa *Taxonomic minimum taxa diagnostic rule*

Description

Creates a diagnostic rule that evaluates whether a data cube contains a sufficient number of taxonomical observations (taxa). The rule counts the number of unique taxa present in the cube and compares it to a threshold to determine the severity level.

Usage

```
rule_taxon_min_taxa(
  thresholds = c(ok = 5, note = 3, important = 0, very_important = NULL)
)
```

Arguments

thresholds	Named numeric vector with severity thresholds: ok, note, important, very_important. Defaults are used if not provided.

Value

An object of class cube_rule.

See Also

Other diagnostic_rules: rule_obs_min_records(), rule_obs_min_total(), rule_spatial_max_uncertainty(), rule_spatial_min_cells(), rule_spatial_miss_uncertainty(), rule_temporal_min_years(), rule_temporal_missing_years()

rule_temporal_min_years *Temporal minimum years diagnostic rule*

Description

Creates a diagnostic rule that evaluates whether a data cube contains a sufficient number of temporal observations (years). The rule counts the number of unique years present in the cube and compares it to a threshold to determine the severity level.

Usage

```
rule_temporal_min_years(
  thresholds = c(ok = 5, note = 3, important = 0, very_important = NULL)
)
```

Arguments

thresholds	Named numeric vector with severity thresholds: ok, note, important, very_important. Defaults are used if not provided.

Value

An object of class cube_rule.

See Also

Other diagnostic_rules: rule_obs_min_records(), rule_obs_min_total(), rule_spatial_max_uncertainty(), rule_spatial_min_cells(), rule_spatial_miss_uncertainty(), rule_taxon_min_taxa(), rule_temporal_missing_years()

---

rule_temporal_missing_years

Temporal gaps diagnostic rule

Description

Creates a diagnostic rule that evaluates whether a data cube contains missing years. The rule counts the number of missing years present in the cube and compares it to a threshold to determine the severity level.

Usage

```
rule_temporal_missing_years(
  thresholds = c(ok = 0, note = 1, important = 3, very_important = NULL)
)
```

Arguments

thresholds	Named numeric vector with severity thresholds: ok, note, important, very_important. Defaults are used if not provided.

Value

An object of class cube_rule.

See Also

Other diagnostic_rules: rule_obs_min_records(), rule_obs_min_total(), rule_spatial_max_uncertainty(), rule_spatial_min_cells(), rule_spatial_miss_uncertainty(), rule_taxon_min_taxa(), rule_temporal_min_years()

summary.cube_diagnostics *Summarise cube diagnostics*

Description

Provides a summary of diagnostic results returned by `diagnose_cube()`. The summary reports the number of evaluated rules, counts per severity level, and the number of diagnostics per cube dimension.

Usage

```
## S3 method for class 'cube_diagnostics'
summary(object, ...)
```

Arguments

`object`	A `cube_diagnostics` object returned by `diagnose_cube()`.
`...`	Additional arguments passed to other methods (currently unused).

Value

An object of class `summary_cube_diagnostics`, containing aggregated diagnostic information.

See Also

Other diagnostic_methods: `plot.cube_diagnostics()`, `print.cube_diagnostics()`

Index

∗ **data_exploration**
diagnose_cube, 29
filter_cube, 30
∗ **data_sensitivity**
cross_validate_cube, 26
∗ **diagnostic_methods**
plot.cube_diagnostics, 35
print.cube_diagnostics, 36
summary.cube_diagnostics, 45
∗ **diagnostic_rule_group**
basic_cube_rules, 6
∗ **diagnostic_rules**
rule_obs_min_records, 39
rule_obs_min_total, 40
rule_spatial_max_uncertainty, 41
rule_spatial_min_cells, 41
rule_spatial_miss_uncertainty, 42
rule_taxon_min_taxa, 43
rule_temporal_min_years, 43
rule_temporal_missing_years, 44
∗ **indicator_interpretation**
add_effect_classification, 3
∗ **indicator_uncertainty_helper**
boot_list_to_dataframe, 9
bootstrap_cube_raw, 14
calculate_acceleration, 17
calculate_boot_ci_from_boot, 20
resolve_bootstrap_method, 37
∗ **indicator_uncertainty**
bootstrap_cube, 10
calculate_bootstrap_ci, 22
∗ **interval_calculation**
basic_ci, 4
bca_ci, 7
norm_ci, 32
perc_ci, 34

add_effect_classification, 3

basic_ci, 4
basic_ci(), *8*, *33*, *35*
basic_cube_rules, 6
bca_ci, 7
bca_ci(), *5*, *33*, *35*
boot::boot(), *9*, *11*, *13*
boot_list_to_dataframe, 9
boot_list_to_dataframe(), *16*, *19*, *21*, *38*
bootstrap_cube, 10
bootstrap_cube(), *9*, *25*, *37*
bootstrap_cube_raw, 14
bootstrap_cube_raw(), *9*, *19*, *21*, *38*

calculate_acceleration, 17
calculate_acceleration(), *9*, *16*, *21*, *38*
calculate_boot_ci_from_boot, 20
calculate_boot_ci_from_boot(), *9*, *16*, *19*, *38*
calculate_bootstrap_ci, 22
calculate_bootstrap_ci(), *13*
cross_validate_cube, 26

derive_bootstrap_method(), *38*
diagnose_cube, 29
diagnose_cube(), *30, 31*, *35*, *45*

filter_cube, 30
filter_cube(), *30*

norm_ci, 32
norm_ci(), *5*, *8*, *35*

perc_ci, 34
perc_ci(), *5*, *8*, *33*
plot.cube_diagnostics, 35
plot.cube_diagnostics(), *37*, *45*
print.cube_diagnostics, 36
print.cube_diagnostics(), *36*, *45*

resolve_bootstrap_method, 37
resolve_bootstrap_method(), *9*, *16*, *19*, *21*
rule_obs_min_records, 39

rule_obs_min_records(), *40–44*
rule_obs_min_total, 40
rule_obs_min_total(), *40–44*
rule_spatial_max_uncertainty, 41
rule_spatial_max_uncertainty(), *40*, *42–44*
rule_spatial_min_cells, 41
rule_spatial_min_cells(), *40–44*
rule_spatial_miss_uncertainty, 42
rule_spatial_miss_uncertainty(), *40–44*
rule_taxon_min_taxa, 43
rule_taxon_min_taxa(), *40–42*, *44*
rule_temporal_min_years, 43
rule_temporal_min_years(), *40–44*
rule_temporal_missing_years, 44
rule_temporal_missing_years(), *40–44*

summary.cube_diagnostics, 45
summary.cube_diagnostics(), *36*, *37*