Package 'ctreeMI'

September 18, 2026

Title Conditional Inference Trees with Stacked Multiple Imputation

Version 1.1.0

Description Implements the stacked-imputation workflow for conditional inference trees ('ctree') described in Sherlock et al. (2026) <doi:10.1080/00273171.2026.2661244>. When data contain missing values, multiply imputed datasets (e.g., from 'mice') are stacked vertically and a single 'ctree' is fit on the combined data. To correct for the artificially inflated sample size introduced by stacking, every node-level test statistic is divided by the number of imputations M, the node-level p-values are recomputed from the chi-squared reference distribution 'ctree' uses (including its multiplicity adjustment across candidate splitting variables), and the tree is compressed bottom-up (the Stack/M correction). Degrees of freedom are derived for each node and each candidate variable, so univariate, bivariate and higher-dimensional outcomes are all handled, as are unordered factor predictors, whose degrees of freedom depend on how many levels remain in a node. The result is a single interpretable tree that incorporates imputation uncertainty without requiring pooling of structurally different trees. Also exports stack_imputations(), rescale_statistic(), prune_stackM(), node_table() and report_ctreeMI() as standalone utilities. The underlying 'ctree' algorithm is provided by 'partykit' (Hothorn & Zeileis, 2015; Hothorn, Hornik & Zeileis, 2006 <doi:10.1198/106186006X133933>).

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Imports partykit (>= 1.2-0), mice (>= 3.0.0), stats, methods

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Depends R (>= 4.0.0)

URL https://github.com/Phillip-Sherlock/ctreeMI

BugReports https://github.com/Phillip-Sherlock/ctreeMI/issues

NeedsCompilation no

Author Phillip Sherlock [aut, cre] (ORCID:
<https://orcid.org/0000-0003-0433-3681>)

Maintainer Phillip Sherlock <phillip.sherlock@ufl.edu>

Repository CRAN

Date/Publication 2026-09-18 19:30:02 UTC

Contents

check_stackM_extraction 2
confirm_ctreeMI 3
ctree_stacked 5
discover_confirm 10
node_table 11
print.ctreeMI 13
print.ctreeMI_nodes 13
print.ctreeMI_report 14
prune_stackM 14
prune_unconfirmed 16
report_confirm 17
report_ctreeMI 18
rescale_statistic 19
split_holdout 20
stack_imputations 21
summary.ctreeMI 22

Index **23**

`check_stackM_extraction` *Check That Node Statistics Can Be Extracted*

Description

Diagnostic. Fits small trees under both of the multiplicity settings the correction supports, and verifies that node-level statistics and degrees of freedom can be recovered from the fitted objects. Run it once after installing, or after upgrading `partykit`.

Usage

```
check_stackM_extraction(verbose = TRUE)
```

Arguments

`verbose`	Logical. Print a per-node report.

Value

Invisibly, TRUE if extraction succeeded everywhere it was tried.

Examples

```
check_stackM_extraction(verbose = FALSE)
```

`confirm_ctreeMI` *Confirm a Discovered Tree on Independent Data*

Description

Applies the terminal-node rules of a tree fitted by `ctree_stacked()` to an independently imputed confirmation sample, and tests whether the outcome differs across those subgroups using a test with valid error control under multiple imputation.

Usage

```
confirm_ctreeMI(
  tree,
  data,
  outcome_type = "auto",
  min_node = 5L,
  adjust = "holm",
  conf.level = 0.95
)
```

Arguments

`tree`	An object of class `ctreeMI`, fitted on the discovery data.
`data`	The confirmation data as a `mids` object from `mice::mice()` or a list of imputed data frames. Must have been imputed separately from the discovery data; see `split_holdout()`.
`outcome_type`	Either `"auto"` (default), in which case numeric outcomes are modelled by linear regression and two-level factors by logistic regression, or a character vector of the same length as the number of outcomes with entries `"continuous"` or `"binary"`.
`min_node`	Minimum number of confirmation observations, per imputation, that a terminal node must receive. Nodes below this are reported but excluded from the test. Default 5.
`adjust`	Method for adjusting the per-split p-values across the internal nodes of the tree, passed to `stats::p.adjust()`. Default `"holm"`. Use `"none"` to report unadjusted values.
`conf.level`	Confidence level for the pooled contrast reported with each split. Default 0.95.

Details

Why a separate confirmation step:

A tree uses the data to choose which variables to split on and where. Testing the resulting subgroups on the same data is invalid regardless of how the missing values were handled, because the partition was selected to maximise exactly the separation being tested. Independently of that, the node-level test of `ctree_stacked()` is not calibrated under outcome-conditioned imputation; see the "Node-level calibration" section of that help page.

Both problems are avoided by treating the discovery tree as a hypothesis about subgroup structure and testing it on data the tree has not seen. This function implements that step. The subgroups are fixed by the discovery tree, so the multiplicity of the search is not carried into the test, and the confirmation test pools across imputations by Rubin's rules rather than stacking, so the between-imputation variance that miscalibrates the stacked test is properly accounted for.

What is tested:

Two families of test are run, both pooled across imputations by the multivariate Wald procedure of Li, Raghunathan and Rubin (1991), implemented in `mice::D1()`.

The **omnibus test** fits, for each outcome, a model with terminal-node membership as the only predictor, and tests whether the outcome differs across nodes at all. This establishes that the partition carries information, not that every split is real: a tree with one genuine split and one spurious one will still yield a small p value.

The **per-split tests** address that directly. For each internal node of the discovery tree, the confirmation observations falling within that node are divided by the node's own split rule, and the outcome is tested for a difference between the resulting children. Because the parent node and its rule are fixed by the discovery tree, each is a pre-specified contrast, and the test is valid. A split whose children do not differ in independent data is one the discovery tree should not have made. The per-split p values are adjusted across internal nodes by the method in `adjust`.

Each split is also reported with the pooled difference between its children and a confidence interval, on the scale of the outcome for a continuous response and as a log odds ratio for a binary one. These are the quantities to report: the p-value says whether the split survived, the contrast says how much the subgroups differ and how precisely that is known. `prune_unconfirmed()` collapses the splits that did not survive.

A split below a non-confirmed split has no clear interpretation, since the partition it refines was not itself supported; the `depth` column allows this to be read off. Pooled node-level estimates are returned so that the pattern of differences can be examined directly.

Node assignment under imputation:

Each imputed confirmation dataset is passed through the discovery tree's split rules. Because imputed predictor values vary across imputations, an observation whose imputed value straddles a split point may be assigned to different nodes in different imputations. This is not an error; it reflects imputation uncertainty, and pooling by Rubin's rules accounts for it.

Value

An object of class `"ctreeMI_confirm"`: a list with elements

`test` A data frame with one row per outcome: `outcome`, `F`, `df1`, `df2`, `p`, and `riv`, the relative increase in variance due to nonresponse.

`splits` A data frame with one row per internal node and outcome: `node_id`, `depth`, `split_var`, `rule` (the split as written), `outcome`, `contrast` with `lower` and `upper` (the pooled difference between the children and its confidence interval), `F`, `df1`, `df2`, `p`, `p_adj`, and `n_confirm` (mean observations in the parent node per imputation). `NA` where a child node fell below `min_node` in some imputation.

`nodes` A data frame of pooled per-node estimates: `node_id`, `outcome`, `estimate` (mean or proportion), `se`, `n_confirm` (mean observations per imputation), and `tested`.

`m` Number of imputations used.

`n_confirm` Number of confirmation observations.

`excluded` Terminal-node ids excluded for having fewer than `min_node` observations, if any.

References

Li, K. H., Raghunathan, T. E., and Rubin, D. B. (1991). Large-sample significance levels from multiply imputed data using moment-based statistics and an F reference distribution. *Journal of the American Statistical Association*, 86, 1065–1073.

See Also

`split_holdout()`, `discover_confirm()`, `ctree_stacked()`

Examples

```
## Not run:
library(mice)
set.seed(7)
n <- 600
d <- data.frame(x1 = rnorm(n), x2 = rnorm(n))
d$y <- rnorm(n) + 1.2 * (d$x1 > 0)
d$x1[sample(n, 90)] <- NA

parts <- split_holdout(d, seed = 7)
imp_d <- mice(parts$discover, m = 20, printFlag = FALSE, seed = 1)
imp_c <- mice(parts$confirm,  m = 20, printFlag = FALSE, seed = 2)

tree <- ctree_stacked(y ~ x1 + x2, data = imp_d, verbose = FALSE)
confirm_ctreeMI(tree, imp_c)

## End(Not run)
```

`ctree_stacked` *Conditional Inference Tree on Stacked Multiply Imputed Data*

Description

Fits a conditional inference tree (ctree) on stacked multiply imputed datasets using the Stack / M rescaling procedure described in Sherlock et al. (2026). Multiply imputed datasets are concatenated vertically ("stacked") and one tree is grown on the combined data. Each node-level chi-square statistic is then divided by the number of imputations (M) to counteract the artificially inflated sample size, the node-level p-values are recomputed, and the tree is compressed bottom-up. This yields a single, coherent, interpretable tree that incorporates imputation variability without requiring the pooling of structurally different trees.

Usage

```
ctree_stacked(
  formula,
  data,
  m = NULL,
  alpha = 0.05,
  scale_minsize = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`formula`	A model formula, passed to `partykit::ctree()`.
`data`	A `mids` object from `mice::mice()`, a list of imputed data frames, or a single complete data frame. If a single complete data frame is supplied (no missing data handling needed), the function falls back to a standard `partykit::ctree()` call with a warning.
`m`	Integer. Number of imputations to use. If `data` is a `mids` object or a list, defaults to the number of datasets available. Ignored when `data` is a plain data frame.
`alpha`	Numeric. The nominal significance threshold for node-level splitting (default 0.05). It is applied to p-values recomputed from node statistics that have been divided by `m`; `alpha` itself is not rescaled. Must be strictly between 0 and 1.
`scale_minsize`	Logical. If `TRUE` (default), `minsplit` and `minbucket` are multiplied by `m` so they refer to original rather than stacked observations. The `partykit` defaults would otherwise permit terminal nodes holding fewer than one original observation.
`verbose`	Logical. If `TRUE` (default), prints a message summarizing the stacking and correction applied.
`...`	Additional arguments passed to `partykit::ctree_control()`. Note: `alpha` in `...` is ignored in favor of the `alpha` argument above.

Details

Methodological background:

When data contain missing values, a common and principled approach is multiple imputation (Rubin, 1987), wherein M completed datasets are generated by drawing from the posterior predictive

distribution of the missing values. For most statistical models, results from M datasets are pooled using Rubin's rules.

Conditional inference trees (ctree; Hothorn, Hornik & Zeileis, 2006) are not straightforward to pool across imputations because the tree structure itself – which variables are split, at what values, and in what order – can differ across imputations. Pooling structurally different trees produces inconsistent subgroup definitions and uninterpretable results.

Rodgers et al. (2021) proposed stacking the M imputed datasets vertically and fitting a single tree to the combined data. This circumvents the structural-mismatch problem and produces one interpretable tree. However, stacking inflates the nominal sample size by a factor of M, causing node-level test statistics to be similarly inflated and the tree to split more aggressively than warranted.

Sherlock et al. (2026) proposed and validated the **Stack / M** correction: each node-level test statistic computed on the stacked data is divided by M, the p-value is recomputed from the rescaled statistic, the Bonferroni adjustment for the number of candidate splitting variables is reapplied, and nodes that no longer meet `alpha` are pruned. See "Scope" and "Node-level calibration" below for the conditions under which the procedure has been examined and for the behavior of its node-level test.

Correction applied to the statistic, not to alpha:

Rescaling the statistic is **not** equivalent to dividing the significance threshold by M. Writing `q(p, df)` for the chi-square quantile function, the two rules are:

- statistic rescaling (correct): reject when `X > M * q(1 - alpha, df)`
- threshold rescaling (incorrect): reject when `X > q(1 - alpha / M, df)`

These coincide only at `M = 1`. At `df = 1`, `alpha = 0.05`, `M = 30` the first requires `X > 115.2` and the second only `X > 9.9`, so threshold rescaling under-corrects by an order of magnitude and grows substantially larger trees than the published method.

Versions 0.1.0 and 0.2.0 of this package implemented threshold rescaling. Trees fitted with those versions are under-corrected and should be refitted.

When to use this procedure:

The problem it solves is specific. Multiple imputation produces M completed datasets, and for most models their results are combined by Rubin's rules; a tree offers no estimand that can be averaged, since the M trees may split on different variables in a different order. Stacking avoids that, and the correction addresses the inflated sample size that stacking introduces. If a single interpretable tree is wanted from multiply imputed data, that is what this package is for.

In the simulations archived at the DOI given below, the procedure recovered known structure more often than listwise deletion, surrogate splits, missingness incorporated in attributes, or single imputation, and produced the most stable partitions across independent sets of imputations. Its accuracy held across missingness rates of 15\ while each of those four declined, listwise deletion most steeply. Tree sizes tracked the true size closely wherever real structure was present, and were stable from M = 5 to M = 50.

Two settings matter in practice. The outcome should be included in the imputation model, which `mice()` does by default: omitting it attenuates the associations a tree is meant to find, at a cost described under "Node-level calibration" below. And M should follow the usual guidance for the fraction of missing information; M = 30 was used throughout these simulations.

The procedure suits least the exploratory case in which no structure may be present, since the miscalibration described below is concentrated under a true null. A single unreplicated subgroup

found in data with no prior reason to expect one warrants the checks given below rather than the node-level p-value.

Scope:

The procedure assumes the data are missing at random in the sense of Rubin (1976), as does the multiple imputation it is built on. In the simulations archived at the DOI given below it was examined under MCAR and MAR across a range of missingness rates, sample sizes and outcome types, and it is intended for use where missingness is plausibly at random. Under MNAR, recovery of the true structure degrades for every imputation-based approach, this one included, and a tree fitted in that setting should be treated as provisional. The mechanism is rarely known in applied work, and MNAR cannot be distinguished from MAR using the observed data, so this is a statement about where the assumption is defensible rather than a condition that can be checked.

Node-level calibration:

The node-level test is not calibrated to its nominal level when the imputation model conditions on the outcome. That is recommended practice, and it is what `mice()` does by default when run on a data frame containing the outcome. Under a true null the test rejects more often than `alpha` implies, increasingly so as the missingness rate rises. Omitting the outcome from the imputation model reverses this into conservatism rather than restoring calibration, and is not recommended, since it attenuates the associations a tree is meant to detect.

Node-level p-values should therefore be read as approximate rather than nominal, and should not be reported as error rates. The recommended remedy is to treat the tree as discovery and test its partition on independent data with `confirm_ctreeMI()`, which pools across imputations by Rubin's rules and so has valid error control; `discover_confirm()` performs the whole sequence. Two further checks help in interpreting a fitted tree. `node_table()` reports the effective sample size behind each terminal node, in original rather than stacked observations, so a node resting on few original cases can be identified. And refitting on an independent set of imputations, then comparing the resulting partitions, indicates whether the structure is stable.

Simulations characterizing this behavior are archived at doi:10.5281/zenodo.21939940.

Documentation prior to version 1.0.1 described the correction as sub-nominal under MCAR. That characterization comes from the simulations in Sherlock et al. (2026), which used a marginal imputation model conditioning on neither the outcome nor the remaining predictors. It does not hold under outcome-conditioned imputation.

Usage with `mice`:

```
library(mice)
imp <- mice(my_data, m = 30, printFlag = FALSE)
fit <- ctree_stacked(outcome ~ ., data = imp, alpha = 0.05)
print(fit)
plot(fit)
```

Usage with a list of data frames:

```
imp_list <- lapply(1:30, function(i) complete(imp, i))
fit <- ctree_stacked(outcome ~ ., data = imp_list, alpha = 0.05)
```

Value

An object of class `c("ctreeMI", "constparty", "party")`. This inherits all methods from `partykit::ctree()` output and additionally carries a `ctreeMI_info` attribute with the following elements:

`m` Number of imputations used.

`n_original` Number of rows in a single imputed dataset.

`n_stacked` Total rows in the stacked dataset.

`alpha` The significance threshold applied to the rescaled p-values.

`correction` Character, `"statistic/M"`.

`node_stats` Data frame of per-node raw statistics, degrees of freedom, rescaled statistics, p-values, and retention.

`n_splits_before`, `n_splits_after` Splits before and after the correction was applied.

`outcome_dim` Rank of the influence function of the response, i.e. the degrees of freedom carried by a numeric or ordered predictor.

`minsplit`, `minbucket` Minimum node sizes actually used, in stacked rows.

`formula` The model formula.

`call` The matched call.

References

Sherlock, P., Mansolf, M., Hofheimer, J., Hockett, C. W., O'Connor, T. G., Roubinov, D., Graff, J. C., Lai, J.-S., Bush, N. R., Wright, R. J., & Chiu, Y.-H. M. (2026). Beyond linear risk: A machine learning approach to understanding perinatal depression in context. *Multivariate Behavioral Research*, 1-16. doi:10.1080/00273171.2026.2661244

Hothorn, T., Hornik, K., & Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3), 651-674. doi:10.1198/106186006X133933

Hothorn, T., & Zeileis, A. (2015). partykit: A modular toolkit for recursive partitioning in R. *Journal of Machine Learning Research*, 16, 3905-3909.

Rodgers, D. M., Jacobucci, R., & Grimm, K. J. (2021). A multiple imputation approach for handling missing data in classification and regression trees. *Journal of Behavioral Data Science*, 1(1), 127-153. doi:10.35566/jbds/v1n1/p6

Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys.* Wiley.

See Also

`partykit::ctree()`, `partykit::ctree_control()`, `mice::mice()`, `stack_imputations()`, `rescale_statistic()`, `prune_stackM()`, `node_table()`

Examples

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

# Introduce missingness into the airquality dataset
```

```
set.seed(42)
aq <- airquality
aq$Ozone[sample(nrow(aq), 20)] <- NA
aq$Solar.R[sample(nrow(aq), 15)] <- NA

# Impute
imp <- mice(aq, m = 10, printFlag = FALSE)

# Fit ctree with Stack/M correction
fit <- ctree_stacked(Ozone ~ Solar.R + Wind + Temp + Month,
                     data = imp,
                     alpha = 0.05)
print(fit)
plot(fit)

## End(Not run)
```

`discover_confirm` *Discover a Tree and Confirm It on Held-Out Data*

Description

A single call that splits the data, imputes each half separately, fits the Stack/M tree on the discovery half, and tests the resulting partition on the confirmation half. Equivalent to calling `split_holdout()`, `mice::mice()` twice, `ctree_stacked()` and `confirm_ctreeMI()` in sequence.

Usage

```
discover_confirm(
  formula,
  data,
  prop = 0.5,
  m = 30L,
  strata = NULL,
  seed = NULL,
  alpha = 0.05,
  mice_args = list(),
  ...
)
```

Arguments

`formula`	Model formula, as for `ctree_stacked()`.
`data`	A data frame containing missing values.
`prop`	Proportion assigned to discovery. Default 0.5.
`m`	Number of imputations for each half. Default 30.

strata	Optional stratification column for the split.
seed	Optional integer seed. The confirmation imputation uses seed + 1L so that the two halves are imputed with different streams.
alpha	Nominal level for the discovery tree. Default 0.05.
mice_args	A list of additional arguments passed to mice::mice() for both halves.
...	Further arguments passed to ctree_stacked().

Details

Imputing the halves separately is deliberate. A single imputation of the full data lets each half's outcomes inform the other's imputed predictors, and the confirmation test then no longer sees independent data. The cost is that each imputation model is fitted to half the observations. For most designs this is a reasonable trade; for small samples it may not be, and the user should judge.

Value

An object of class "ctreeMI_dc" containing tree (the ctreeMI fit), confirmation (a ctreeMI_confirm object), split (the ctreeMI_split object), and the two mids objects as imp_discover and imp_confirm.

See Also

split_holdout(), confirm_ctreeMI(), ctree_stacked()

Examples

```
## Not run:
set.seed(11)
n <- 800
d <- data.frame(x1 = rnorm(n), x2 = factor(sample(letters[1:3], n, TRUE)))
d$y <- rnorm(n) + 1.5 * (d$x1 > 0.3) + (d$x2 == "a")
d$x1[sample(n, 120)] <- NA
d$x2[sample(n, 80)]  <- NA

res <- discover_confirm(y ~ x1 + x2, data = d, m = 20, seed = 11)
res$tree
res$confirmation

## End(Not run)
```

---

node_table *Terminal-Node Summary With Effective Sample Sizes*

Description

Summarizes the nodes of a fitted tree: the split path leading to each node, its size in the stacked data, its effective size in original observations (stacked size divided by M), and node-level outcome summaries.

Usage

```
node_table(object, terminal_only = TRUE, max_levels = NULL, digits = 3)
```

Arguments

object	A ctreeMI object from ctree_stacked(), or any party object (in which case M is taken to be 1).
terminal_only	Logical. Report terminal nodes only (default), or every node including the root.
max_levels	Integer or NULL. Truncate factor level sets longer than this in the printed split path.
digits	Number of digits for the outcome summaries.

Details

Terminal-node sample sizes printed by partykit refer to the stacked data, in which every observation appears M times. effective_n divides by M to give the number of original observations behind each node. Node-level means and proportions need no such adjustment: they are already averages over the imputations.

Value

A data frame of class "ctreeMI_nodes" with one row per node: node_id, depth, n_stacked, effective_n, any outcome summaries, path, and a conditions list column holding the split conditions individually.

See Also

ctree_stacked(), report_ctreeMI()

Examples

```
set.seed(1)
imps <- lapply(1:5, function(i) {
  x <- stats::rnorm(200)
  data.frame(x = x, y = stats::rnorm(200) + 1.5 * (x > 0))
})
fit <- ctree_stacked(y ~ x, data = imps, verbose = FALSE)
node_table(fit)
```

print.ctreeMI *Print Method for ctreeMI Objects*

Description

Prints a summary of the stacked-imputation settings used to fit the tree, followed by the standard `partykit::ctree()` output.

Usage

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

Arguments

`x`	An object of class `"ctreeMI"`, as returned by `ctree_stacked()`.
`...`	Further arguments passed to the partykit print method.

Value

`x`, invisibly.

print.ctreeMI_nodes *Print a Node Table*

Description

Print a Node Table

Usage

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

Arguments

`x`	A `"ctreeMI_nodes"` object from `node_table()`.
`...`	Passed to `print.data.frame()`.

Value

`x`, invisibly.

print.ctreeMI_report *Print a Methods Paragraph*

Description

Print a Methods Paragraph

Usage

```
## S3 method for class 'ctreeMI_report'
print(x, width = 76, ...)
```

Arguments

x	A "ctreeMI_report" object from report_ctreeMI().
width	Wrapping width in characters.
...	Currently unused.

Value

x, invisibly.

prune_stackM *Prune a Stacked Tree Using the Stack / M Correction*

Description

Applies the Stack / M correction of Sherlock et al. (2026) to a conditional inference tree fitted on stacked multiply imputed data. Every node-level test statistic is divided by M, the p-values are recomputed from the chi-square reference distribution ctree uses, the multiplicity adjustment over candidate splitting variables is reapplied, and the tree is compressed bottom-up.

Usage

```
prune_stackM(tree, m, alpha = 0.05, verbose = TRUE)
```

Arguments

tree	A party/constparty object fitted by partykit::ctree() on stacked data, with teststat = "quadratic" and testtype either "Univariate" or "Bonferroni". Both are handled: whether the stored p-values already carry partykit's multiplicity adjustment is read from the fitted control rather than assumed.
m	Integer. Number of imputations.
alpha	Numeric. Nominal significance threshold.
verbose	Logical. Report how many splits were retained.

Value

A list with the pruned `tree` and a `node_stats` data frame with one row per internal node of the unpruned tree, recording the variable split on, its raw and rescaled statistics, the degrees of freedom and how they were obtained, the p-values before and after correction, the smallest corrected p-value over all candidates at that node, and whether the split was retained. The full per-candidate table is attached as `attr(node_stats, "candidates")`.

What is tested

`ctree` splits a node when *any* candidate variable meets `alpha`, so that is the rule reapplied here: all candidates tested at the node are rescaled and the smallest corrected p-value decides. Testing only the variable that was split on would not be equivalent, because candidates carry different degrees of freedom and rescaling can reorder them.

How the tree is compressed

Pruning is bottom-up, as in the analysis for the paper. An internal node is collapsed only once all of its own internal descendants have been collapsed, so a node whose descendant survives the correction keeps its split.

Degrees of freedom

With `teststat = "quadratic"` the node-level statistic is chi-square with `df` equal to the rank of the covariance matrix of the linear statistic: the outcome dimension for a numeric or ordered predictor (1 for a univariate outcome, 2 for a bivariate outcome, and so on), and `(L - 1)` times the outcome dimension for an unordered factor with `L` levels present in the node. These are derived for every node and every candidate variable, by inverting the (statistic, p-value) pairs `partykit` stored and falling back on the structural rule where the stored p-value has underflowed to zero, which happens routinely on stacked data.

References

Sherlock, P., et al. (2026). Beyond linear risk: A machine learning approach to understanding perinatal depression in context. *Multivariate Behavioral Research*, 1-16. doi:10.1080/00273171.2026.2661244

Examples

```
set.seed(1)
n <- 300
d <- data.frame(x = stats::rnorm(n))
d$y1 <- stats::rnorm(n) + 1.2 * (d$x > 0)
d$y2 <- stats::rnorm(n) + 0.9 * (d$x > 0)
grown <- partykit::ctree(y1 + y2 ~ x, data = d)
out <- prune_stackM(grown, m = 5, verbose = FALSE)
out$node_stats
```

prune_unconfirmed *Collapse Splits That Did Not Confirm*

Description

Takes a discovery tree and its confirmation, and returns the tree with every split that failed to confirm collapsed into a terminal node.

Usage

```
prune_unconfirmed(
  tree,
  confirmation,
  alpha = 0.05,
  which = c("adjusted", "raw"),
  outcome = "any"
)
```

Arguments

tree	The ctreeMI object that was confirmed.
confirmation	A ctreeMI_confirm object from confirm_ctreeMI().
alpha	Level at which a split counts as confirmed. Default 0.05.
which	Which p-value to use: "adjusted" (default) or "raw".
outcome	For a multivariate outcome, the name of the outcome whose tests govern pruning, or "any" (default) to retain a split confirmed for any outcome, or "all" to require every outcome.

Details

Pruning is bottom-up and a split is collapsed only once all of its own internal descendants have been collapsed, matching prune_stackM(). A split whose p-value could not be computed, because a child node fell below min_node in some imputation, is treated as unconfirmed; the tree offers no evidence for it on the confirmation data.

The result is the partition the confirmation data supports. Its terminal nodes can be reported without the caveat that attaches to the discovery tree, since the splits defining them were tested on data the tree had not seen.

Value

A ctreeMI object with unconfirmed splits collapsed. Its ctreeMI_info attribute gains confirmed, a data frame recording which splits were retained and why.

See Also

confirm_ctreeMI(), discover_confirm()

Examples

```
## Not run:
res <- discover_confirm(y ~ ., data = d, m = 30, seed = 1)
pruned <- prune_unconfirmed(res$tree, res$confirmation)
plot(pruned)

## End(Not run)
```

`report_confirm` *A Methods Paragraph For a Discover-Then-Confirm Analysis*

Description

Generates a paragraph describing a discover-then-confirm analysis in the form usually required by a methods section: how the sample was divided, how each half was imputed, the tree that was discovered, and which of its splits survived testing on the confirmation data.

Usage

```
report_confirm(object, tree = NULL, digits = 3)
```

Arguments

`object`	A `ctreeMI_dc` object from `discover_confirm()`, or a `ctreeMI_confirm` object from `confirm_ctreeMI()`.
`tree`	The discovery tree, required when `object` is a `ctreeMI_confirm` object and ignored otherwise.
`digits`	Number of digits used in the reported quantities.

Value

An object of class `"ctreeMI_report"`, as returned by `report_ctreeMI()`: a list whose `text` element is the paragraph.

See Also

`discover_confirm()`, `report_ctreeMI()`

Examples

```
## Not run:
res <- discover_confirm(y ~ ., data = d, m = 30, seed = 1)
report_confirm(res)

## End(Not run)
```

`report_ctreeMI` *A Methods Paragraph For a ctreeMI Analysis*

Description

Generates a paragraph describing the analysis in the form usually required by a methods section: the number of imputations, the size of the stacked dataset, the model, the Stack / M correction and the threshold applied, how many candidate splits survived it, and the shape of the resulting tree.

Usage

```
report_ctreeMI(object, digits = 3)
```

Arguments

`object`	A `ctreeMI` object from `ctree_stacked()`.
`digits`	Number of digits used in the node summaries.

Value

An object of class `"ctreeMI_report"`: a list whose `text` element is the paragraph, alongside the quantities it reports.

See Also

`ctree_stacked()`, `node_table()`

Examples

```
set.seed(1)
imps <- lapply(1:5, function(i) {
  x <- stats::rnorm(200)
  data.frame(x = x, y = stats::rnorm(200) + 1.5 * (x > 0))
})
fit <- ctree_stacked(y ~ x, data = imps, verbose = FALSE)
report_ctreeMI(fit)
```

`rescale_statistic` *Rescale a Node-Level Test Statistic for Stacking*

Description

Divides a node-level chi-square test statistic by the number of imputations `M` and recomputes the corresponding p-value. This is the Stack / M correction of Sherlock et al. (2026).

Usage

```
rescale_statistic(statistic, m, df)
```

Arguments

`statistic`	Numeric. The raw test statistic computed on the stacked data.
`m`	Integer. Number of imputations.
`df`	Numeric. Degrees of freedom of the reference chi-square distribution: the outcome dimension for a numeric or ordered predictor, `(L - 1)` times the outcome dimension for an unordered factor with `L` levels. See `prune_stackM()`, which derives this for you.

Details

Stacking `M` imputed datasets produces `M * n` rows, and a chi-square statistic computed on the stacked data scales approximately linearly with `M`. Dividing the statistic by `M` returns it to the scale of a single imputed dataset before the p-value is computed.

This is **not** the same as dividing the significance threshold by `M`. The two rules coincide only at `M = 1`:

- Statistic rescaling rejects when `X > M * qchisq(1 - alpha, df)`.
- Threshold rescaling rejects when `X > qchisq(1 - alpha / M, df)`.

For `df = 1`, `alpha = 0.05` and `M = 30`, the first requires `X > 115.2` and the second only `X > 9.9`. Versions 0.1.0 and 0.2.0 implemented the second rule.

Value

A list with `statistic_rescaled` and `p_value`.

References

Sherlock, P., Mansolf, M., Hofheimer, J., Hockett, C. W., O'Connor, T. G., Roubinov, D., Graff, J. C., Lai, J.-S., Bush, N. R., Wright, R. J., & Chiu, Y.-H. M. (2026). Beyond linear risk: A machine learning approach to understanding perinatal depression in context. *Multivariate Behavioral Research*, 1-16. doi:10.1080/00273171.2026.2661244

Examples

```
rescale_statistic(115.2, m = 30, df = 1)
```

`split_holdout` *Split Data Into Discovery and Confirmation Sets*

Description

Partitions a data frame into two disjoint subsets of original observations, before imputation. Each subset should then be imputed separately, so that neither half's outcomes inform the other's imputed predictor values.

Usage

```
split_holdout(data, prop = 0.5, strata = NULL, seed = NULL)
```

Arguments

`data`	A data frame, typically containing missing values.
`prop`	Proportion of observations assigned to the discovery set. Default 0.5.
`strata`	Optional name of a column on which to stratify the split, so that its distribution is preserved in both halves. Ordinarily the outcome, if categorical. Rows with a missing value in this column are allocated at random.
`seed`	Optional integer seed.

Details

The split is performed on original observations, not on stacked rows, and before any imputation. This matters. Imputing the full data once and then splitting leaves each half's imputed values informed by the other half's outcomes, which compromises the independence the confirmation step relies on. Impute the two halves separately.

Value

A list of class `"ctreeMI_split"` with elements `discover` and `confirm`, each a data frame, and `index`, the row indices of `data` assigned to the discovery set.

See Also

`confirm_ctreeMI()`, `discover_confirm()`

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(200), y = rnorm(200))
d$x[sample(200, 30)] <- NA
parts <- split_holdout(d, prop = 0.5, seed = 1)
nrow(parts$discover); nrow(parts$confirm)
```

`stack_imputations` *Stack Multiply Imputed Datasets*

Description

Concatenates a list of imputed data frames into a single "stacked" data frame. An imputation index column (`.imp`) is added to identify which imputed dataset each row originated from. This is the first step of the stacked-imputation workflow described in Rodgers et al. (2021) and applied to conditional inference trees in Sherlock et al. (2026).

Usage

```
stack_imputations(data_list, imp_col = ".imp")
```

Arguments

`data_list`	A list of data frames, each the same dimensions, representing M imputed versions of the same dataset.
`imp_col`	Character string. Name of the imputation-index column added to the stacked data (default `".imp"`). Set to `NULL` to suppress this column.

Value

A single data frame with `M * n` rows, where `n` is the number of rows in each imputed dataset.

References

Sherlock, P., et al. (2026). Beyond linear risk: A machine learning approach to understanding perinatal depression in context. *Multivariate Behavioral Research*, 1-16. doi:10.1080/00273171.2026.2661244

Rodgers, D. M., Jacobucci, R., & Grimm, K. J. (2021). A multiple imputation approach for handling missing data in classification and regression trees. *Journal of Behavioral Data Science*, 1(1), 127-153. doi:10.35566/jbds/v1n1/p6

Examples

```
df1 <- data.frame(x = 1:5, y = c(2, 4, 6, 8, 10))
df2 <- data.frame(x = 1:5, y = c(2, 3, 6, 9, 10))
stacked <- stack_imputations(list(df1, df2))
nrow(stacked) # 10
table(stacked$.imp)
```

summary.ctreeMI *Summary Method for ctreeMI Objects*

Description

Returns a named list with details about the stacked-imputation fit and the resulting tree structure (number of terminal nodes, tree depth).

Usage

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

Arguments

object	An object of class "ctreeMI", as returned by ctree_stacked().
...	Currently unused.

Value

A list (invisibly) with components:

ctreeMI_info Stacking metadata from ctree_stacked().

n_terminal_nodes Number of terminal nodes in the fitted tree.

depth Maximum depth of the fitted tree.

Index

check_stackM_extraction, 2
confirm_ctreeMI, 3
confirm_ctreeMI(), *8*, *10, 11*, *16, 17*, *20*
ctree_stacked, 5
ctree_stacked(), *3–5*, *10–13*, *18*, *22*

discover_confirm, 10
discover_confirm(), *5*, *8*, *16, 17*, *20*

mice::D1(), *4*
mice::mice(), *3*, *6*, *9–11*

node_table, 11
node_table(), *8, 9*, *13*, *18*

partykit::ctree(), *6*, *9*, *13, 14*
partykit::ctree_control(), *6*, *9*
print.ctreeMI, 13
print.ctreeMI_nodes, 13
print.ctreeMI_report, 14
print.data.frame(), *13*
prune_stackM, 14
prune_stackM(), *9*, *16*, *19*
prune_unconfirmed, 16
prune_unconfirmed(), *4*

report_confirm, 17
report_ctreeMI, 18
report_ctreeMI(), *12*, *14*, *17*
rescale_statistic, 19
rescale_statistic(), *9*

split_holdout, 20
split_holdout(), *3*, *5*, *10, 11*
stack_imputations, 21
stack_imputations(), *9*
stats::p.adjust(), *3*
summary.ctreeMI, 22