Package: networkformat (via r-universe)

October 16, 2026

Type Package

Title Convert R Objects to Network Edgelists and Nodelists

Version 0.1.2

Description Network analysis and plotting packages such as 'igraph', 'tidygraph', and 'ggraph' expect the edges and nodes of a graph as flat tables. Methods are provided to build those tables from data frames (column-pair edges chosen with 'tidyselect' semantics), tree-based model objects ('randomForest', 'tree', 'rpart', 'xgboost', 'gbm'), lists (recursive parent-child edges), and atomic vectors (sequential edges). Graphs can also be constructed in one step via as.igraph() and as_tbl_graph() methods for tree models.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.6.0)

Imports rlang, tidyselect

Suggests randomForest, tree, xgboost, gbm, rpart, testthat (>= 3.0.0), covr, igraph, tidygraph, ggraph, knitr, rmarkdown

Config/testthat/edition 3

URL https://jessebrandtdata.github.io/networkformat/, https://github.com/jessebrandtdata/networkformat

BugReports https://github.com/jessebrandtdata/networkformat/issues

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Jesse Brandt [aut, cre] (ORCID: <https://orcid.org/0009-0005-7462-075X>)

Maintainer Jesse Brandt <jesse.a.brandt@gmail.com>

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

Date/Publication 2026-10-16 12:50:07 UTC

RemoteUrl https://github.com/cran/networkformat

RemoteRef HEAD

RemoteSha 3e9bcaf2d9d8892a1e44fc55f565c7b010ad5ab0

Contents

as.igraph . . . 2
as_tbl_graph . . . 3
courses . . . 4
edgelist . . . 5
edgelist.data.frame . . . 6
edgelist.default . . . 8
edgelist.gbm . . . 9
edgelist.list . . . 10
edgelist.randomForest . . . 11
edgelist.rpart . . . 12
edgelist.tree . . . 13
edgelist.xgb.Booster . . . 14
nodelist . . . 15
nodelist.data.frame . . . 16
nodelist.default . . . 17
nodelist.gbm . . . 17
nodelist.list . . . 18
nodelist.randomForest . . . 19
nodelist.rpart . . . 20
nodelist.tree . . . 21
nodelist.xgb.Booster . . . 22

Index **24**

`as.igraph` *Convert to igraph*

Description

S3 methods for converting tree-based model objects into `igraph` graph objects. Each method calls `edgelist` and `nodelist` internally and handles column reconciliation so you get a ready-to-use graph. These methods are registered against the `as.igraph` generic from **igraph** via delayed S3 registration and are available whenever **igraph** is loaded.

Usage

```
## S3 method for class 'tree'
as.igraph(x, ...)

## S3 method for class 'randomForest'
as.igraph(x, treenum = NULL, ...)

## S3 method for class 'rpart'
as.igraph(x, ...)

## S3 method for class 'xgb.Booster'
as.igraph(x, treenum = NULL, ...)

## S3 method for class 'gbm'
as.igraph(x, treenum = NULL, ...)
```

Arguments

`x`	An object to convert (`tree`, `randomForest`, `rpart`, `xgb.Booster`, or `gbm`).
`...`	Additional arguments passed to methods.
`treenum`	Integer vector of tree numbers to extract. Default `NULL` returns all trees combined into one graph with disconnected components. Pass a single integer (e.g. `1`) to extract one tree.

Value

An `igraph` object. For `randomForest` with multiple trees, the graph contains disconnected components (one per tree) and a `treenum` vertex/edge attribute.

Examples

```
if (requireNamespace("rpart", quietly = TRUE) &&
    requireNamespace("igraph", quietly = TRUE)) {
  fit <- rpart::rpart(Sepal.Length ~ ., data = iris)
  g <- igraph::as.igraph(fit)
  igraph::vcount(g)
  igraph::ecount(g)
}
```

`as_tbl_graph` *Convert to tbl_graph*

Description

S3 methods for converting tree-based model objects into `tbl_graph` objects. Each method wraps the corresponding `as.igraph` method. These methods are registered against the `as_tbl_graph` generic from **tidygraph** via delayed S3 registration and are available whenever **tidygraph** is loaded.

Usage

```
## S3 method for class 'tree'
as_tbl_graph(x, ...)

## S3 method for class 'randomForest'
as_tbl_graph(x, treenum = NULL, ...)

## S3 method for class 'rpart'
as_tbl_graph(x, ...)

## S3 method for class 'xgb.Booster'
as_tbl_graph(x, treenum = NULL, ...)

## S3 method for class 'gbm'
as_tbl_graph(x, treenum = NULL, ...)
```

Arguments

`x`	An object to convert (currently `tree`, `randomForest`, `rpart`, `xgb.Booster`, or `gbm`).
`...`	Additional arguments passed to `as.igraph`.
`treenum`	Integer vector of tree numbers to extract. Default `NULL` returns all trees combined into one graph with disconnected components. Pass a single integer (e.g. `1`) to extract one tree.

Value

A `tbl_graph` object.

Examples

```
if (requireNamespace("rpart", quietly = TRUE) &&
    requireNamespace("tidygraph", quietly = TRUE) &&
    requireNamespace("igraph", quietly = TRUE)) {
  fit <- rpart::rpart(Sepal.Length ~ ., data = iris)
  tg <- tidygraph::as_tbl_graph(fit)
  tg
}
```

---

courses — *Course prerequisite and crosslisting network*

Description

A small dataset of university courses with prerequisite and crosslisting relationships, suitable for demonstrating `edgelist()` and `nodelist()`.

Usage

```
courses
```

Format

A data.frame with 13 rows and 7 columns:

dept Department code (STAT, MATH, DATA, CS)

course Course identifier, used as node ID

prereq Prerequisite course (NA if none)

prereq2 Second prerequisite course (NA if none)

crosslist Crosslisted equivalent course (NA if none)

credits Number of credit hours (integer)

level Course level: 100, 200, or 300 (integer)

Examples

```
# Prerequisite edgelist (course -> prereq)
edgelist(courses, source_cols = course, target_cols = prereq)

# Node list with course as ID column
nodelist(courses, id_col = course)
```

---

edgelist — *Extract Edgelist from Various Object Types*

Description

Generic function to extract a network edgelist from various object types. Methods exist for atomic vectors (sequential edges), data frames (column-pair edges), lists (recursive parent-child edges), and tree-based model objects (`randomForest`, `tree`, `rpart`, `xgb.Booster`, `gbm`). The specific columns returned depend on the input type.

Usage

```
edgelist(input_object, ...)
```

Arguments

`input_object`	An object to extract an edgelist from
`...`	Additional arguments passed to specific methods

Value

A data.frame representing an edgelist. The specific columns depend on the input object type.

Examples

```
# Vector --- sequential edges
edgelist(c("A", "B", "C", "D"))

# Data.frame example using bundled dataset
edgelist(courses, source_cols = course, target_cols = prereq)

# RandomForest example
if (requireNamespace("randomForest", quietly = TRUE)) {
  rf <- randomForest::randomForest(Species ~ ., data = iris, ntree = 5)
  edges_rf <- edgelist(rf)
  head(edges_rf)
}

# Tree example
if (requireNamespace("tree", quietly = TRUE)) {
  tr <- tree::tree(Species ~ ., data = iris)
  edges_tr <- edgelist(tr)
  head(edges_tr)
}
```

---

edgelist.data.frame *Extract Edgelist from Data Frame*

Description

Converts a data.frame to network edgelist format by specifying which columns represent source and target nodes. This is useful for creating edgelists from tabular data where node relationships are stored in columns.

Usage

```
## S3 method for class 'data.frame'
edgelist(
  input_object,
  source_cols = 1,
  target_cols = 2,
  attr_cols = NULL,
  na.rm = TRUE,
  symmetric_cols = NULL,
  dedupe = TRUE,
  weights = FALSE,
  ...
)
```

Arguments

`input_object`	A data.frame containing network information
`source_cols`	Column(s) for source nodes (default: 1). Accepts tidyselect expressions: bare names, strings, numeric indices, or helpers like `starts_with()`.
`target_cols`	Column(s) for target nodes (default: 2). Same syntax as `source_cols`.
`attr_cols`	Columns to carry as edge attributes (default: `NULL` keeps all columns not used as source or target). Pass an empty `c()` to keep only from, to, and metadata columns. Accepts the same tidyselect syntax as `source_cols`.
`na.rm`	Logical; if `TRUE` (the default), rows where `from` or `to` is `NA` are removed from the result. Set to `FALSE` to keep all rows including those with `NA` endpoints.
`symmetric_cols`	Target column names that represent undirected (symmetric) relationships. When non-`NULL`, a `directed` column is added: `FALSE` for edges from target columns named in `symmetric_cols`, `TRUE` otherwise. Accepts the same tidyselect syntax as `target_cols`; names must be a subset of `target_cols`.
`dedupe`	Logical; if `TRUE` (the default) and `symmetric_cols` is non-`NULL`, duplicate undirected edges are removed by keeping only rows where `from <= to` (lexicographic comparison). Set to `FALSE` to preserve both directions.
`weights`	Logical; if `TRUE`, fully identical rows (same `from`, `to`, and all attribute columns) are collapsed and a `weight` column is added with the count. Rows that share `(from, to)` but differ in any attribute are kept separate. Applied after NA removal and symmetric deduplication. Defaults to `FALSE`.
`...`	Additional arguments (currently unused)

Value

A data.frame with columns:

from Source node values

to Target node values

from_col Name of the original column each source value came from

to_col Name of the original column each target value came from

directed (only when `symmetric_cols` is provided) Logical: `FALSE` for edges from symmetric target columns, `TRUE` otherwise

weight (only when `weights = TRUE`) Integer count of how many times each unique `(from, to)` pair occurred

... Additional attribute columns selected by `attr_cols`

Examples

```
# Basic usage --- all non-source/target columns kept by default
edgelist(courses, source_cols = course, target_cols = prereq)

# Multiple target columns --- to_col identifies the relationship type
edgelist(courses, source_cols = course, target_cols = c(prereq, crosslist))
```

```
# Keep only the edgelist (no attribute columns)
edgelist(courses, source_cols = course, target_cols = prereq, attr_cols = c())

# Select specific attribute columns
edgelist(courses, source_cols = course, target_cols = prereq,
         attr_cols = c(dept, credits))

# NA removal (default) vs preservation
edgelist(courses, source_cols = course, target_cols = c(prereq, crosslist))
edgelist(courses, source_cols = course, target_cols = c(prereq, crosslist),
         na.rm = FALSE)

# Mark symmetric (undirected) target columns
edgelist(courses, source_cols = course,
         target_cols = c(prereq, crosslist),
         symmetric_cols = crosslist)
```

`edgelist.default` *Extract Edgelist from a Vector or Unsupported Object*

Description

When called on an atomic vector (character, numeric, integer, logical, factor), creates a sequential edgelist connecting each element to the next: element `i` is connected to element `i + 1`. For all other unsupported types, an informative error is raised.

Usage

```
## Default S3 method:
edgelist(input_object, weights = FALSE, ...)
```

Arguments

`input_object`	An atomic vector, or an object for which no specific edgelist method exists.
`weights`	Logical; if `TRUE`, duplicate edges are collapsed and a `weight` column is added with the count of each unique `(from, to)` pair. Defaults to `FALSE`.
`...`	Additional arguments (currently unused)

Value

A data.frame with columns `from` and `to` (and `weight` when `weights = TRUE`). For unsupported types, an error is raised.

Examples

```
# Character vector
edgelist(c("A", "B", "C", "D"))

# Numeric vector
edgelist(1:5)

# With duplicate counting
edgelist(c("A", "B", "A", "B", "C"), weights = TRUE)

# Unsupported type
try(edgelist(as.formula(y ~ x)))
```

`edgelist.gbm` *Extract Edgelist from GBM Model*

Description

Converts a gbm model object into a network edgelist representation using `pretty.gbm.tree()`. Missing-sentinel nodes (the phantom routing nodes gbm creates for NA handling) are excluded — only real `LeftNode`/`RightNode` edges are returned.

Usage

```
## S3 method for class 'gbm'
edgelist(input_object, treenum = NULL, ...)
```

Arguments

`input_object`	A gbm model object from the **gbm** package
`treenum`	Integer vector of 1-based tree numbers to extract (default: `NULL` extracts all trees). For multinomial models, this indexes physical trees (not boosting iterations).
`...`	Additional arguments (currently unused)

Details

Node IDs are 0-based integers per tree. For models with `num.classes` > `1` (multinomial), physical trees are stored as `n.trees` * `num.classes` entries; `treenum` indexes into this flat sequence.

Value

A data.frame with the following columns:

from Parent node ID (0-based integer)

to Child node ID (0-based integer)

split_var 0-based variable index

split_point Split threshold for continuous variables, or `c.splits` index for categorical variables

prediction Prediction value at the child node

treenum 1-based tree number

split_var_name Human-readable variable name

Examples

```
if (requireNamespace("gbm", quietly = TRUE)) {
  set.seed(1)
  fit <- gbm::gbm(mpg ~ ., data = mtcars,
                  distribution = "gaussian", n.trees = 5,
                  interaction.depth = 3, n.minobsinnode = 3)
  el <- edgelist(fit)
  head(el)

  # Single tree
  el1 <- edgelist(fit, treenum = 1)
}
```

`edgelist.list` *Extract Edgelist from a List*

Description

Recursively traverses a nested list structure, producing a parent-child edgelist. Each element of the list becomes a node; nested lists create deeper edges. Path-style IDs ensure unique node names even when element names repeat at different levels.

Usage

```
## S3 method for class 'list'
edgelist(input_object, name_root = "root", max_depth = NULL, ...)
```

Arguments

`input_object`	A list (or S3 object falling through to this method).
`name_root`	Character; label for the root node. Defaults to `"root"`.
`max_depth`	Integer or `NULL`; maximum node depth to include (root is depth 0, its children are depth 1). `NULL` (the default) means unlimited. `max_depth = 0` returns an empty edgelist (root only).
`...`	Additional arguments (currently unused).

Details

When called on an S3 object that has no dedicated `edgelist` method (e.g. an `lm` object), the object is treated as a plain list and a diagnostic message is emitted. To force structural decomposition of an object that has its own method (e.g. a `tree`), use `edgelist(unclass(x))`.

Value

A data.frame with columns `from`, `to`, and `depth` (integer depth of the child node, root children are depth 1). An empty list returns a zero-row data.frame with the same columns.

Examples

```
edgelist(list(a = 1, b = list(c = 2, d = 3)))

# Unnamed elements use positional indices
edgelist(list(1, 2, list(3, 4)))

# Limit depth (root = 0, children = 1, ...)
edgelist(list(a = list(b = list(c = 1))), max_depth = 2)
```

`edgelist.randomForest` *Extract Edgelist from RandomForest Model*

Description

Converts a randomForest model object into a network edgelist representation by extracting parent-child relationships from one or more trees in the forest. Each edge represents a split in the decision tree, with additional attributes including split variable, split point, and prediction values.

Usage

```
## S3 method for class 'randomForest'
edgelist(input_object, treenum = NULL, ...)
```

Arguments

`input_object`	A randomForest model object from the randomForest package
`treenum`	Integer vector of tree numbers to extract (default: `NULL` extracts all trees). Values must be between 1 and `input_object$ntree`.
`...`	Additional arguments (currently unused)

Value

A data.frame with the following columns:

from Parent node index within the tree

to Child node index (left or right daughter)

split_var Numeric index of the variable used for splitting

split_point Threshold value for the split

prediction Prediction value at the child node

direction Branch direction: `"left"` or `"right"`

treenum Tree number within the forest

split_var_name Character vector with human-readable variable names

Examples

```
if (requireNamespace("randomForest", quietly = TRUE)) {
  set.seed(12)
  rf_model <- randomForest::randomForest(
    Species ~ .,
    data = iris,
    ntree = 5,
    maxnodes = 10
  )

  # Extract all trees
  rf_edges <- edgelist(rf_model)
  head(rf_edges)

  # Extract specific trees
  rf_edges_1 <- edgelist(rf_model, treenum = 1)
  rf_edges_13 <- edgelist(rf_model, treenum = c(1, 3))
}
```

---

edgelist.rpart *Extract Edgelist from rpart Model*

Description

Converts an rpart model object into a network edgelist representation. The rpart package uses binary heap numbering for node IDs (root = 1, left child of k = 2k, right child of k = 2k + 1), so parent-child edges are derived directly from the node IDs in `rownames(input_object$frame)`.

Usage

```
## S3 method for class 'rpart'
edgelist(input_object, ...)
```

Arguments

`input_object`	An rpart model object from the rpart package
`...`	Additional arguments (currently unused)

Details

Edge labels come from `labels(input_object, collapse = TRUE)`, which correctly handles the `ncat` sign that controls split direction (left child does not always get "<").

Value

A data.frame with the following columns:

from Parent node ID (binary heap index)

to Child node ID (binary heap index)

label Split condition label from `labels()`

split_var Variable name used for the split

split_op Operator: `"<"` or `">="` for numeric splits, `NA` for categorical splits

split_point Numeric threshold for the split (`NA` for categorical splits)

Examples

```
if (requireNamespace("rpart", quietly = TRUE)) {
  fit <- rpart::rpart(Species ~ ., data = iris)
  el <- edgelist(fit)
  head(el)

  # Parsed split components
  el[, c("split_var", "split_op", "split_point")]
}
```

---

edgelist.tree *Extract Edgelist from Tree Model*

Description

Converts a tree model object (from the tree package) into a network edgelist representation. The tree package uses binary heap numbering for node IDs (root = 1, left child of k = 2k, right child of k = 2k + 1), so parent-child edges are derived directly from the node IDs in `rownames(input_object$frame)`.

Usage

```
## S3 method for class 'tree'
edgelist(input_object, ...)
```

Arguments

`input_object`	A tree model object from the tree package
`...`	Additional arguments (currently unused)

Value

A data.frame with the following columns:

from Parent node ID (binary heap index)

to Child node ID (binary heap index)

label Split condition label (variable and threshold)

split_var Variable name used for the split

split_op Operator: "<" or ">" for numeric splits, NA for categorical splits

split_point Numeric threshold for the split (NA for categorical splits)

Examples

```
if (requireNamespace("tree", quietly = TRUE)) {
  # Fit a classification tree
  tree_model <- tree::tree(Species ~ ., data = iris)

  # Extract edgelist
  tree_edges <- edgelist(tree_model)
  head(tree_edges)

  # Parsed split components
  tree_edges[, c("split_var", "split_op", "split_point")]
}
```

---

edgelist.xgb.Booster *Extract Edgelist from XGBoost Model*

Description

Converts an xgboost model object into a network edgelist representation using `xgb.model.dt.tree()`. Each edge connects a split node to one of its two children (yes/no branches). Node IDs are globally unique strings in "Tree-Node" format (e.g., "0-3").

Usage

```
## S3 method for class 'xgb.Booster'
edgelist(input_object, treenum = NULL, ...)
```

Arguments

input_object	An xgboost model object (xgb.Booster)
treenum	Integer vector of 1-based tree numbers to extract (default: NULL extracts all trees).
...	Additional arguments (currently unused)

Value

A data.frame with the following columns:

from Parent node ID string (`"Tree-Node"` format)

to Child node ID string

feature Name of the split variable (or 0-based index string when feature names are absent)

split Numeric split threshold

quality Information gain at the split

cover Number of observations covered

treenum 1-based tree number

Examples

```
if (requireNamespace("xgboost", quietly = TRUE)) {
  data(agaricus.train, package = "xgboost")
  bst <- xgboost::xgboost(
    x = agaricus.train$data,
    y = factor(agaricus.train$label),
    max_depth = 2, nrounds = 2, nthreads = 1
  )
  el <- edgelist(bst)
  head(el)
}
```

`nodelist` *Extract Node List from Various Objects*

Description

Generic function to extract a node list (with attributes) from various object types including atomic vectors (unique values with frequencies), data frames, lists (recursive node decomposition), and tree models (`randomForest`, `tree`, `rpart`, `xgb.Booster`, `gbm`). The node list provides metadata about each node in the network.

Usage

```
nodelist(input_object, ...)
```

Arguments

`input_object`	An object containing node information (vector, data.frame, tree model, etc.)
`...`	Additional arguments passed to specific methods

Value

A data.frame where the first column is the node ID and subsequent columns contain node attributes

Examples

```
# Vector --- unique values with frequency counts
nodelist(c("A", "B", "A", "C"))

# Node list with course as ID (column 2)
nodelist(courses, id_col = 2)
```

---

nodelist.data.frame *Extract Node List from Data Frame*

Description

Reorders a data.frame to place the ID column first, creating a proper node list format. All other columns are treated as node attributes.

Usage

```
## S3 method for class 'data.frame'
nodelist(input_object, id_col = 1, ...)
```

Arguments

`input_object`	A data.frame containing node information
`id_col`	Column for the node ID (default: 1). Accepts tidyselect expressions: a bare name, string, or numeric index.
`...`	Additional arguments (currently unused)

Value

A data.frame with the ID column first, followed by all attribute columns

Examples

```
# Default: first column (dept) is ID
nodelist(courses)

# Use bare column name
nodelist(courses, id_col = course)

# Numeric index still works
nodelist(courses, id_col = 2)
```

nodelist.default *Extract Node List from a Vector or Unsupported Object*

Description

When called on an atomic vector (character, numeric, integer, logical, factor), returns the unique values as a node list with a `name` column (preserving order of first appearance) and an `n` column giving the frequency of each value. For all other unsupported types, an informative error is raised.

Usage

```
## Default S3 method:
nodelist(input_object, ...)
```

Arguments

`input_object`	An atomic vector, or an object for which no specific nodelist method exists.
`...`	Additional arguments (currently unused)

Value

A data.frame with columns `name` (unique values in order of first appearance) and `n` (frequency count). For unsupported types, an error is raised.

Examples

```
# Character vector
nodelist(c("A", "B", "C", "A", "B"))

# Numeric vector
nodelist(c(1, 2, 3, 2, 1))

# Unsupported type
try(nodelist(as.formula(y ~ x)))
```

nodelist.gbm *Extract Node List from GBM Model*

Description

Extracts node-level attributes from a gbm model via `pretty.gbm.tree()`. Missing-sentinel nodes are excluded by keeping only nodes reachable through `LeftNode`/`RightNode` edges from the root.

Usage

```
## S3 method for class 'gbm'
nodelist(input_object, treenum = NULL, ...)
```

Arguments

`input_object`	A gbm model object from the **gbm** package
`treenum`	Integer vector of 1-based tree numbers to extract (default: `NULL` extracts all trees).
`...`	Additional arguments (currently unused)

Value

A data.frame with one row per real node and the following columns:

name 0-based node ID within the tree

is_leaf Logical: `TRUE` for terminal nodes

split_var 0-based variable index (`NA` for leaves)

split_var_name Variable name (`NA` for leaves)

split_point Split threshold (`NA` for leaves)

prediction Prediction value at the node

error_reduction Impurity reduction from the split (0 for leaves); per-node variable importance from `ErrorReduction`

weight Number of training observations reaching the node, from `Weight`

treenum 1-based tree number

label Display label: `"<var>\n< <threshold>"` for splits, rounded prediction for leaves

Examples

```
if (requireNamespace("gbm", quietly = TRUE)) {
  set.seed(1)
  fit <- gbm::gbm(mpg ~ ., data = mtcars,
                  distribution = "gaussian", n.trees = 5,
                  interaction.depth = 3, n.minobsinnode = 3)
  nl <- nodelist(fit)
  head(nl)
}
```

`nodelist.list` *Extract Node List from a List*

Description

Recursively traverses a nested list structure, producing one row per node with metadata about each element.

Usage

```
## S3 method for class 'list'
nodelist(input_object, name_root = "root", max_depth = NULL, ...)
```

Arguments

`input_object`	A list (or S3 object falling through to this method).
`name_root`	Character; label for the root node. Defaults to `"root"`.
`max_depth`	Integer or `NULL`; maximum node depth to include (root is depth 0, its children are depth 1). `NULL` (the default) means unlimited. `max_depth = 0` returns the root node only.
`...`	Additional arguments (currently unused).

Details

When called on an S3 object that has no dedicated `nodelist` method, the object is treated as a plain list and a diagnostic message is emitted. To force structural decomposition of an object that has its own method (e.g. a `tree`), use `nodelist(unclass(x))`.

Value

A data.frame with columns `name` (path-style ID), `depth` (integer), `type` (character class of the element), `n_children` (integer, 0 for leaves), and `label` (element name or positional index). An empty list returns a one-row data.frame for the root node only.

Examples

```
nodelist(list(a = 1, b = list(c = 2, d = 3)))

# Unnamed elements
nodelist(list(1, 2, list(3, 4)))
```

`nodelist.randomForest` *Extract Node List from RandomForest Model*

Description

Extracts node-level attributes from every tree in a `randomForest` model. Node IDs match the `from`/`to` indices produced by `edgelist.randomForest`, so the two outputs can be passed directly to `igraph::graph_from_data_frame()` (after filtering to a single `treenum`).

Usage

```
## S3 method for class 'randomForest'
nodelist(input_object, treenum = NULL, ...)
```

Arguments

`input_object`	A randomForest model object from the **randomForest** package
`treenum`	Integer vector of tree numbers to extract (default: `NULL` extracts all trees). Values must be between 1 and `input_object$ntree`.
`...`	Additional arguments (currently unused)

Value

A data.frame with one row per node (across all trees) and the following columns:

name Integer node ID within the tree (matches edgelist from/to)

is_leaf Logical: TRUE for terminal nodes

split_var Numeric index of the split variable (NA for leaves)

split_var_name Name of the split variable (NA for leaves)

split_point Split threshold (NA for leaves)

prediction Predicted value (numeric for regression, integer class index for classification)

treenum Integer identifying which tree the node belongs to

label Display label: "<var>\n< <threshold>" for internal nodes, predicted class name (classification) or numeric value (regression) for leaves

Examples

```
if (requireNamespace("randomForest", quietly = TRUE)) {
  set.seed(12)
  rf <- randomForest::randomForest(Species ~ ., data = iris, ntree = 3)
  nodes <- nodelist(rf)
  head(nodes)

  # Extract a single tree
  nodes1 <- nodelist(rf, treenum = 1)
}
```

---

nodelist.rpart *Extract Node List from rpart Model*

Description

Extracts node-level attributes from an rpart model object. Node IDs are the binary heap indices from rownames(input_object$frame) and match the from/to values produced by edgelist.rpart, so the two outputs can be passed directly to igraph::graph_from_data_frame().

Usage

```
## S3 method for class 'rpart'
nodelist(input_object, ...)
```

Arguments

input_object	An rpart model object from the **rpart** package
...	Additional arguments (currently unused)

Value

A data.frame with one row per node and the following columns:

name Integer node ID (binary heap index, matches edgelist from/to)

var Split variable name, or "<leaf>" for terminal nodes

n Number of observations routed to this node

dev Deviance (impurity) at this node

yval Predicted value (numeric for regression, character class label for classification)

is_leaf Logical: TRUE for terminal nodes

depth Integer tree depth (root = 0)

wt Weighted observation count

complexity CP pruning parameter at this node

ncompete Number of competing splits considered

nsurrogate Number of surrogate splits used

dev_improvement Numeric deviance reduction from this node's split (NA for leaves)

n_* (Classification only) One column per class with the count of observations of that class at the node, named n_<classname>. Class names are sanitized via make.names() and lowercased.

prob_* (Classification only) One column per class with the class probability at that node, named prob_<classname>. Class names are sanitized via make.names() and lowercased.

nodeprob (Classification only) Proportion of training data reaching this node

label Display label: "<var>\nn=<n>" for internal nodes, "<yval>\nn=<n>" for leaves

Examples

```
if (requireNamespace("rpart", quietly = TRUE)) {
  fit <- rpart::rpart(Species ~ Sepal.Length + Sepal.Width, data = iris)
  nodelist(fit)

  # Labels ready for plotting
  nodelist(fit)$label
}
```

---

nodelist.tree *Extract Node List from Tree Model*

Description

Extracts node-level attributes from a tree model object. Node IDs are the binary heap indices from rownames(input_object$frame) and match the from/to values produced by edgelist.tree, so the two outputs can be passed directly to igraph::graph_from_data_frame().

Usage

```
## S3 method for class 'tree'
nodelist(input_object, ...)
```

Arguments

`input_object`	A tree model object from the **tree** package
`...`	Additional arguments (currently unused)

Value

A data.frame with one row per node and the following columns:

name Integer node ID (binary heap index, matches edgelist from/to)

var Split variable name, or `"<leaf>"` for terminal nodes

n Number of observations routed to this node

dev Deviance (impurity) at this node

yval Predicted value (numeric for regression, character for classification)

is_leaf Logical: `TRUE` for terminal nodes

depth Integer tree depth (root = 0)

dev_improvement Numeric deviance reduction from this node's split (`NA` for leaves)

prob_* (Classification only) One column per class with the class probability at that node, named `prob_<classname>`. Class names are sanitized via `make.names()` and lowercased.

label Display label: `"<var>\nn=<n>"` for internal nodes, `"<yval>\nn=<n>"` for leaves

Examples

```
if (requireNamespace("tree", quietly = TRUE)) {
  tr <- tree::tree(Species ~ Sepal.Length + Sepal.Width, data = iris)
  nodelist(tr)

  # Labels ready for plotting
  nodelist(tr)$label
}
```

`nodelist.xgb.Booster` *Extract Node List from XGBoost Model*

Description

Extracts node-level attributes from an xgboost model via `xgb.model.dt.tree()`. Node IDs are globally unique strings in `"Tree-Node"` format and match the `from`/`to` columns produced by `edgelist.xgb.Booster`.

Usage

```
## S3 method for class 'xgb.Booster'
nodelist(input_object, treenum = NULL, ...)
```

Arguments

`input_object`	An xgboost model object (`xgb.Booster`)
`treenum`	Integer vector of 1-based tree numbers to extract (default: `NULL` extracts all trees).
`...`	Additional arguments (currently unused)

Value

A data.frame with one row per node and the following columns:

name Node ID string (`"Tree-Node"` format, matches edgelist from/to)

is_leaf Logical: `TRUE` for leaf nodes

feature Split variable name (`NA` for leaves)

split Split threshold (`NA` for leaves)

quality Information gain for splits, leaf score for leaves

cover Cover (sum of second-order gradient)

missing Node ID (`"Tree-Node"` format) where observations with missing values are routed; `NA` for leaves

treenum 1-based tree number

label Display label: `"<feature>\n< <threshold>"` for splits, rounded leaf score for leaves

Examples

```
if (requireNamespace("xgboost", quietly = TRUE)) {
  data(agaricus.train, package = "xgboost")
  bst <- xgboost::xgboost(
    x = agaricus.train$data,
    y = factor(agaricus.train$label),
    max_depth = 2, nrounds = 2, nthreads = 1
  )
  nl <- nodelist(bst)
  head(nl)
}
```

Index

∗ **datasets**
courses, 4

as.igraph, 2, *2–4*
as_tbl_graph, 3, *3*

courses, 4

edgelist, *2*, 5
edgelist.data.frame, 6
edgelist.default, 8
edgelist.gbm, 9
edgelist.list, 10
edgelist.randomForest, 11, *19*
edgelist.rpart, 12, *20*
edgelist.tree, 13, *21*
edgelist.xgb.Booster, 14, *22*

igraph, *2, 3*

nodelist, *2*, 15
nodelist.data.frame, 16
nodelist.default, 17
nodelist.gbm, 17
nodelist.list, 18
nodelist.randomForest, 19
nodelist.rpart, 20
nodelist.tree, 21
nodelist.xgb.Booster, 22

tbl_graph, *3, 4*