Package: CaseBasedReasoning (via r-universe)

November 23, 2026

Type Package

Title Case Based Reasoning

Version 0.5.0

Date 2026-11-22

Description Case-based reasoning is a problem-solving methodology that involves solving a new problem by referring to the solution of a similar problem in a large set of previously solved problems. The key aspect of Case Based Reasoning is to determine the problem that ``most closely" matches the new problem at hand. This is achieved by defining a family of distance functions and using these distance functions as parameters for local averaging regression estimates of the final result. The optimal distance function is chosen based on a specific error measure used in regression estimation. This approach allows for efficient problem-solving by leveraging past experiences and adapting solutions from similar cases. The underlying concept is inspired by the work of Dippon J. et al. (2002) <doi:10.1016/S0167-9473(02)00058-0>.

URL https://github.com/sipemu/case-based-reasoning, https://sipemu.github.io/case-based-reasoning/

BugReports https://github.com/sipemu/case-based-reasoning/issues

License MIT + file LICENSE

Depends rms

Imports R6, Rcpp, RcppParallel, ranger, survival, ggplot2

Suggests testthat, knitr, rmarkdown, RcppArmadillo, patchwork

LinkingTo Rcpp, RcppArmadillo, RcppParallel

SystemRequirements GNU make

NeedsCompilation yes

ByteCompile yes

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.3

Author Simon Mueller [aut, cre], PD Dr. Juergen Dippon [ctb]

Maintainer Simon Mueller <simon.mueller@muon-stat.com>

Config/pak/sysreqs make

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

Date/Publication 2026-11-22 22:50:08 UTC

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

RemoteRef HEAD

RemoteSha 24cc35b94bccb2055e7750abe2c7d8f086d7019d

Contents

asDistObject . . . 3
CBRBase . . . 3
CoxModel . . . 7
depth_distance . . . 8
distanceRandomForest . . . 9
edges_between_terminal_nodes . . . 10
generate_grid . . . 11
LinearModel . . . 11
LogisticModel . . . 12
predict.CoxModel . . . 13
predict.LinearModel . . . 13
predict.LogisticModel . . . 14
predict.RFModel . . . 14
print.CoxModel . . . 15
print.LinearModel . . . 15
print.LogisticModel . . . 16
print.RFModel . . . 16
proximity_distance . . . 17
ranger_forests_to_matrix . . . 18
RegressionModel . . . 19
RFModel . . . 20
summary.CoxModel . . . 22
summary.LinearModel . . . 22
summary.LogisticModel . . . 23
summary.RFModel . . . 23
terminalNodes . . . 24
weightedDistance . . . 24

Index **26**

asDistObject *Converts a distance vector into an object of class* dist

Description

Converts a distance vector into an object of class dist

Usage

```
asDistObject(x, n, method)

as_dist_object(x, n, method)
```

Arguments

x	data vector
n	length of x
method	method description

CBRBase *Root class for common functionality of this package*

Description

Root class for common functionality of this package

Root class for common functionality of this package

Public fields

model the statistical model

data training data

model_fit trained object

formula Object of class formula or character describing the model fit

terms terms of the formula (term labels, e.g. "rcs(age, 3)")

variables variables used by the terms of the formula

endpoint Target variable

dist_matrix A matrix with distances

order_matrix A matrix with the order indices for similar cases search

Active bindings

`endPoint` Deprecated: use `endpoint` instead.

`distMat` Deprecated: use `dist_matrix` instead.

`orderMat` Deprecated: use `order_matrix` instead.

Methods

Public methods:

- `CBRBase$new()`
- `CBRBase$fit()`
- `CBRBase$calc_distance_matrix()`
- `CBRBase$get_similar_cases()`
- `CBRBase$predict_survival()`
- `CBRBase$evaluate_survival()`
- `CBRBase$get_counterfactual_cases()`
- `CBRBase$explain_match()`
- `CBRBase$clone()`

Method `new()`**:** Initialize object for searching similar cases

Usage:

```
CBRBase$new(formula, data)
```

Arguments:

`formula` Object of class formula or character describing the model fit

`data` Training data of class data.frame

Method `fit()`**:** Fit the Model

Usage:

```
CBRBase$fit()
```

Method `calc_distance_matrix()`**:** Calculates the distance matrix

Usage:

```
CBRBase$calc_distance_matrix(query = NULL)
```

Arguments:

`query` Query data of class data.frame

Method `get_similar_cases()`**:** Extracts similar cases

Usage:

```
CBRBase$get_similar_cases(
  query,
  k = 1,
  add_distance = TRUE,
  merge = FALSE,
  exclude_self = FALSE
)
```

Arguments:

`query` Query data of class data.frame

`k` number of similar cases

`add_distance` Add distance to result data.frame

`merge` Add query data to matched cases data.frame

`exclude_self` Exclude training cases whose row name equals the query's row name. Use this when the query rows are taken from the training data, so that a case is not returned as its own nearest neighbour.

Method `predict_survival()`**:** Predict survival curves from the `k` most similar training cases: a Kaplan-Meier estimate over the retrieved cases of each query (Lowsky et al., 2013). Only for survival endpoints (`Surv(time, status)`).

Usage:

```
CBRBase$predict_survival(query, k = 10, times = NULL, weighted = FALSE)
```

Arguments:

`query` Query data of class data.frame

`k` number of similar cases

`times` time points; default: the event times of the training data

`weighted` If `TRUE`, cases are weighted with a tricube kernel of their distance to the query.

Returns: A matrix of survival probabilities with one row per query and one column per time point (attribute `times`).

Method `evaluate_survival()`**:** Evaluate survival predictions from retrieved cases on test data: Uno's concordance index and the IPCW (integrated) Brier score. Use it to choose `k` and the distance method on held-out data.

Usage:

```
CBRBase$evaluate_survival(
  test,
  k = 10,
  times = NULL,
  tau = NULL,
  weighted = FALSE
)
```

Arguments:

`test` Test data with the endpoint columns

`k` number of similar cases

`times` time points for the Brier score; default: 20 points between the 10% and 90% quantile of the training event times

`tau` time horizon for the concordance index (risk score $1 - S(\tau)$); default: the largest of `times`

`weighted` passed to `predict_survival()`

Returns: A list with `c_index` (Uno's C), `ibs` (integrated Brier score) and `brier` (data.frame with the Brier score at each time).

Method `get_counterfactual_cases()`**:** Counterfactual cases: the nearest training cases whose outcome differs from the query's outcome ("native counterfactuals", Keane & Smyth, 2020).

The query's outcome is taken from its endpoint columns if present, otherwise it is predicted from its `k_outcome` most similar cases (majority class, mean, or majority event status). A training case is a counterfactual if

- classification endpoint: its class differs;
- numeric endpoint: its outcome differs by at least `min_diff` (default: standard deviation of the training outcome);
- survival endpoint: its event status at `horizon` differs (event before `horizon` vs. followed beyond `horizon`; cases censored before `horizon` are not used).

Usage:

```
CBRBase$get_counterfactual_cases(
  query,
  k = 1,
  horizon = NULL,
  min_diff = NULL,
  k_outcome = 10
)
```

Arguments:

`query` Query data of class data.frame

`k` number of counterfactual cases per query

`horizon` time horizon for survival endpoints (default: median training event time)

`min_diff` minimum outcome difference for numeric endpoints

`k_outcome` number of similar cases used to predict the query's outcome if it is not given

Returns: A data.frame like `get_similar_cases()` with the additional column `queryOutcome`.

Method `explain_match()`**:** Explain why a training case is similar to a query. For regression models: the contribution of each term to the linear predictor of query and case, and its share of the distance. For random forests: in the trees where query and case end in different terminal nodes, the split variable at which their paths diverge.

Usage:

```
CBRBase$explain_match(query, case)
```

Arguments:

`query` One query case (data.frame with one row)

`case` One case (data.frame with one row), e.g. a row returned by `get_similar_cases()`

Returns: A data.frame with one row per term (regression) or variable (random forest).

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CBRBase$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Lowsky, D. J. et al. (2013). A K-nearest neighbors survival probability prediction method. Statistics in Medicine, 32(12), 2062–2069. doi:10.1002/sim.5673

Uno, H. et al. (2011). On the C-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. Statistics in Medicine, 30(10), 1105–1117. doi:10.1002/sim.4154

Graf, E. et al. (1999). Assessment and comparison of prognostic classification schemes for survival data. Statistics in Medicine, 18(17-18), 2529–2545.

Keane, M. T. and Smyth, B. (2020). Good Counterfactuals and Where to Find Them: A Case-Based Technique for Generating Counterfactuals for Explainable AI. ICCBR 2020, LNCS 12311, 163–178. doi:10.1007/9783030583422_11

`CoxModel` *Cox-Beta Model for Case-Based-Reasoning*

Description

Cox-Beta Model for Case-Based-Reasoning

Cox-Beta Model for Case-Based-Reasoning

Details

Regression beta coefficients obtained from a CPH regression model fitted on the training data are used for building a weighted distance measure between train and test data. Afterwards, we will use these weights for calculating a (n x m)-distance matrix, where n is the number of observations in the training data, and m is the number of observations of the test data. The user can use this distance matrix for further cluster analysis or for extracting for each test observation k (= 1,...,l) similar cases from the train data. We use the rms-package for model fitting, variable selection, and checking model assumptions. If the user omits the test data, this functions returns a n x n-distance matrix.

Super classes

`CaseBasedReasoning::CBRBase` -> `CaseBasedReasoning::RegressionModel` -> `CoxModel`

Public fields

`model` the statistical model

`model_params` rms arguments

Methods

Public methods:

- CoxModel$check_ph()
- CoxModel$clone()

Method `check_ph()`**:** Check proportional hazard assumption graphically

Usage:

```
CoxModel$check_ph()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CoxModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`depth_distance` *Depth Distance*

Description

This function returns for each observation the pairwise sum of edges between the corresponding terminal nodes over each tree in the random forest.

Usage

```
depth_distance(x, y = NULL, rfObject)
```

Arguments

`x`	A data.frame with the same columns as in the training data of the RandomForest model
`y`	A data.frame with the same columns as in the training data of the RandomForest model
`rfObject`	`ranger` object

Examples

```
library(ranger)
rf <- ranger(Species ~ ., data = iris, num.trees = 5, write.forest = TRUE)
depth_distance(x = iris[, -5], rfObject = rf)
```

distanceRandomForest *Distance calculation based on RandomForest Proximity or Depth*

Description

Distance calculation based on RandomForest Proximity or Depth

Usage

```
distanceRandomForest(
  x,
  y = NULL,
  rfObject,
  method = "Proximity",
  threads = NULL
)

distance_random_forest(
  x,
  y = NULL,
  rfObject,
  method = "Proximity",
  threads = NULL,
  proximity_type = c("original", "oob", "rfgap")
)
```

Arguments

x	a data.frame
y	a second data.frame
rfObject	ranger object
method	distance calculation method, Proximity (Default) or Depth.
threads	number of threads to use. The previous RcppParallel thread setting is restored on exit.
proximity_type	type of proximity for method = "Proximity", see proximity_distance.

Value

a dist or a matrix object with pairwise distance of observations in x vs y (if not null)

Examples

```
library(ranger)
# proximity pairwise distances
rf.fit <- ranger(Species ~ ., data = iris, num.trees = 500, write.forest = TRUE)
distance_random_forest(x = iris[, -5], rfObject = rf.fit, method = "Proximity", threads = 1)
```

```
# depth distance for train versus test subset
set.seed(1234L)
learn <- sample(1:150, 100)
test <- (1:150)[-learn]
rf.fit <- ranger(Species ~ ., data = iris[learn, ], num.trees = 500, write.forest = TRUE)
distance_random_forest(x = iris[learn, -5], y = iris[test, -5], rfObject = rf.fit, method = "Depth")
```

`edges_between_terminal_nodes` *Number of Edges between Terminal Nodes*

Description

first two columns are terminal node IDs; If an ID pair do not appear in a tree -1 is inserted

Usage

```
edges_between_terminal_nodes(rfObject)
```

Arguments

`rfObject`	`ranger` object

Value

a `matrix` object with pairwise terminal node edge length

Examples

```
library(ranger)
rf.fit <- ranger(Species ~ ., data = iris, num.trees = 5, write.forest = TRUE)
edges_between_terminal_nodes(rf.fit)
```

generate_grid *Generate Grid*

Description

Generates a uniform grid over the distribution of the time2event variable, calculates closest point and returns this point for each input time2event element. Memory consumption will increase when performing the randomForest model with many unique time2event values. Therefore, we offer a reduction of the time2event values by choosing closest elements in a grid.

Usage

```
generate_grid(t2e, grid_length = 250)
```

Arguments

t2e	numeric vector with time2event values
grid_length	number of grid elements

Value

a list with new_t2e and grid_error

LinearModel *Linear Regression Model for Case-Based-Reasoning*

Description

Linear Regression Model for Case-Based-Reasoning

Linear Regression Model for Case-Based-Reasoning

Super classes

CaseBasedReasoning::CBRBase -> CaseBasedReasoning::RegressionModel -> LinearModel

Public fields

model the statistical model

Methods

Public methods:

- `LinearModel$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LinearModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`LogisticModel` *Logistic Regression Model for Case-Based-Reasoning*

Description

Logistic Regression Model for Case-Based-Reasoning

Logistic Regression Model for Case-Based-Reasoning

Super classes

`CaseBasedReasoning::CBRBase` -> `CaseBasedReasoning::RegressionModel` -> `LogisticModel`

Public fields

`model` the statistical model

Methods

Public methods:

- `LogisticModel$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LogisticModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

predict.CoxModel *Predict method for CoxModel*

Description

Predict method for CoxModel

Usage

```
## S3 method for class 'CoxModel'
predict(object, newdata, k = 1, ...)
```

Arguments

object	A CoxModel R6 object
newdata	Query data of class data.frame
k	Number of similar cases to return
...	Additional arguments (currently unused)

Value

A data.frame of similar cases

predict.LinearModel *Predict method for LinearModel*

Description

Predict method for LinearModel

Usage

```
## S3 method for class 'LinearModel'
predict(object, newdata, k = 1, ...)
```

Arguments

object	A LinearModel R6 object
newdata	Query data of class data.frame
k	Number of similar cases to return
...	Additional arguments (currently unused)

Value

A data.frame of similar cases

predict.LogisticModel *Predict method for LogisticModel*

Description

Predict method for LogisticModel

Usage

```
## S3 method for class 'LogisticModel'
predict(object, newdata, k = 1, ...)
```

Arguments

object	A LogisticModel R6 object
newdata	Query data of class data.frame
k	Number of similar cases to return
...	Additional arguments (currently unused)

Value

A data.frame of similar cases

predict.RFModel *Predict method for RFModel*

Description

Predict method for RFModel

Usage

```
## S3 method for class 'RFModel'
predict(object, newdata, k = 1, ...)
```

Arguments

object	An RFModel R6 object
newdata	Query data of class data.frame
k	Number of similar cases to return
...	Additional arguments (currently unused)

Value

A data.frame of similar cases

print.CoxModel *Print method for CoxModel*

Description

Print method for CoxModel

Usage

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

Arguments

x	A CoxModel R6 object
...	Additional arguments (currently unused)

print.LinearModel *Print method for LinearModel*

Description

Print method for LinearModel

Usage

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

Arguments

x	A LinearModel R6 object
...	Additional arguments (currently unused)

print.LogisticModel *Print method for LogisticModel*

Description

Print method for LogisticModel

Usage

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

Arguments

x	A LogisticModel R6 object
...	Additional arguments (currently unused)

print.RFModel *Print method for RFModel*

Description

Print method for RFModel

Usage

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

Arguments

x	An RFModel R6 object
...	Additional arguments (currently unused)

`proximity_distance` *Get proximity matrix of an ranger object*

Description

Random forest proximities between the observations in `x` and `y`, converted to the distance $\sqrt{1-p}$ by default.

Usage

```
proximity_distance(
  x,
  y = NULL,
  rfObject,
  as_dist = TRUE,
  type = c("original", "oob", "rfgap")
)
```

Arguments

`x`	a new dataset. For `type = "oob"` and `"rfgap"`: the training data of `rfObject`.
`y`	a second new dataset (Default: NULL)
`rfObject`	`ranger` object
`as_dist`	Bool, convert proximity p to the distance $\sqrt{1-p}$.
`type`	proximity type: `"original"` (default), `"oob"` or `"rfgap"`.

Details

Three types of proximity are available:

`"original"` Breiman's proximity: the share of trees in which two observations share a terminal node. In-sample proximities are inflated because observations are counted in the trees they were trained on.

`"oob"` Only trees in which the training observation (and, for in-sample queries, the query) is out-of-bag are used.

`"rfgap"` RF-GAP proximities (Rhodes, Cutler & Moon, 2023): for query q and training observation j, $p(q,j) = \frac{1}{|S_q|}\sum_{t\in S_q} c_j(t)1[j \in leaf_t(q)]/|M_q(t)|$, where S_q are the trees in which q is out-of-bag (all trees for new data), $c_j(t)$ is the in-bag count of j and $|M_q(t)|$ the total in-bag count in q's leaf. For each query the proximities sum to 1, and the proximity-weighted average of the training outcomes reproduces the forest's (out-of-bag) prediction for regression forests.

`"oob"` and `"rfgap"` need the in-bag counts: fit the forest with `keep.inbag = TRUE` and pass the forest's training data as `x` (same rows, same order). If `y` is `NULL`, the training data is also used as query (in-sample proximities).

Value

For type = "original": a dist object (if y is NULL) or a matrix with pairwise proximities (or distances, see as_dist) of observations in x (rows) vs y (columns). For the other types always a matrix; RF-GAP proximities are not symmetric.

References

Rhodes, J. S., Cutler, A. and Moon, K. R. (2023). Geometry- and Accuracy-Preserving Random Forest Proximities. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(9), 10947–10959. doi:10.1109/TPAMI.2023.3263774

Examples

```
library(ranger)
rf <- ranger(Species ~ ., data = iris, num.trees = 5, write.forest = TRUE)
proximity_distance(x = iris[, -5], rfObject = rf)

set.seed(1234L)
learn <- sample(1:150, 100)
test <- (1:150)[-learn]
rf <- ranger(Species ~ ., data = iris[learn, ], num.trees = 500,
             write.forest = TRUE, keep.inbag = TRUE)
proximity_distance(x = iris[learn, -5], y = iris[test, -5], rfObject = rf)
proximity_distance(x = iris[learn, -5], y = iris[test, -5], rfObject = rf,
                   type = "rfgap")
```

---

ranger_forests_to_matrix

Forest2Matrix

Description

Transform trees of a ranger-object to a matrix

Usage

```
ranger_forests_to_matrix(rfObject)
```

Arguments

rfObject	ranger object

Value

a matrix object with Column 1: tree ID Column 2: node ID Column 3: child node ID 1 Column 4: child node ID 2

Examples

```
library(ranger)
rf.fit <- ranger(Species ~ ., data = iris, num.trees = 5, write.forest = TRUE)
forest_matrix <- ranger_forests_to_matrix(rf.fit)
```

`RegressionModel` *Root class for Regression Models, e.g., CPH, logistic, and linear regression*

Description

Root class for Regression Models, e.g., CPH, logistic, and linear regression

Root class for Regression Models, e.g., CPH, logistic, and linear regression

Super class

`CaseBasedReasoning::CBRBase` -> `RegressionModel`

Public fields

`model_params` rms arguments

`weights` Weights for distance calculation

Methods

Public methods:

- `RegressionModel$print()`
- `RegressionModel$variable_selection()`
- `RegressionModel$fit()`
- `RegressionModel$clone()`

Method `print()`**:** Prints information of the initialized object

Usage:

```
RegressionModel$print()
```

Method `variable_selection()`**:** Fast backward variable selection with penalization (`rms::fastbw`)

Usage:

```
RegressionModel$variable_selection(update = FALSE, ...)
```

Arguments:

`update` If `TRUE`, restrict the model's formula and terms to the selected variables. The model must then be fitted again with `$fit()`.

`...` Further arguments passed to `rms::fastbw`, e.g. `rule` or `sls`.

Returns: Character vector of the selected (retained) variables.

Method `fit()`**:** Fit the regression model

Usage:

```
RegressionModel$fit()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RegressionModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`RFModel` *RandomForest Model for Searching Similar Cases*

Description

RandomForest Model for Searching Similar Cases

RandomForest Model for Searching Similar Cases

Details

This class uses the proximity or depth matrix of the RandomForest algorithm as a similarity matrix of training and query observations. By default all cases with at least one missing values are dropped from learning, calculating the distance matrix and searching for similar cases.

Super class

`CaseBasedReasoning::CBRBase` -> `RFModel`

Public fields

`model` the statistical model

`model_params` model arguments

`dist_method` Distance method

`proximity_type` Proximity type for `dist_method = "Proximity"`: `"original"`, `"oob"` or `"rfgap"` (see `proximity_distance`).

Methods

Public methods:

- RFModel$print()
- RFModel$new()
- RFModel$fit()
- RFModel$set_distance_method()
- RFModel$clone()

Method `print()`**:** Prints information of the initialized object

Usage:

```
RFModel$print()
```

Method `new()`**:** Initialize a RandomForest object for searching similar cases.

Usage:

```
RFModel$new(formula, data, ...)
```

Arguments:

`formula` Object of class formula or character describing the model fit.

`data` Training data of class data.frame

`...` ranger RandomForest arguments. `write.forest` is always set to `TRUE`; `verbose` defaults to `FALSE`.

Method `fit()`**:** Fit the RandomForest

Usage:

```
RFModel$fit()
```

Method `set_distance_method()`**:** Set the distance method. Available are Proximity and Depth

Usage:

```
RFModel$set_distance_method(method = "Depth", proximity_type = "original")
```

Arguments:

`method` Distance calculation method (default: Depth)

`proximity_type` Proximity type for `method = "Proximity"`: `"original"` (default), `"oob"` or `"rfgap"`. The latter two use the forest's in-bag counts; for a query given to `get_similar_cases()` or `calc_distance_matrix()` the query is treated as new data, without a query (`calc_distance_matrix()`) the in-sample, out-of-bag version is computed.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RFModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Englund, C. and Verikas, A. (2012). A novel approach to estimate proximity in a random forest: An exploratory study. Expert Systems with Applications, 39(17), 13046–13050. doi:10.1016/j.eswa.2012.05.094

summary.CoxModel *Summary method for CoxModel*

Description

Summary method for CoxModel

Usage

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

Arguments

object	A CoxModel R6 object
...	Additional arguments (currently unused)

summary.LinearModel *Summary method for LinearModel*

Description

Summary method for LinearModel

Usage

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

Arguments

object	A LinearModel R6 object
...	Additional arguments (currently unused)

summary.LogisticModel *Summary method for LogisticModel*

Description

Summary method for LogisticModel

Usage

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

Arguments

object	A LogisticModel R6 object
...	Additional arguments (currently unused)

summary.RFModel *Summary method for RFModel*

Description

Summary method for RFModel

Usage

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

Arguments

object	An RFModel R6 object
...	Additional arguments (currently unused)

terminalNodes *Get the terminal node id of a RandomForest Object*

Description

Extracts for each observation and for each tree in the forest the terminal node id. The index of terminal nodes are starting with 1, e.g., the root node has id 1

Usage

```
terminalNodes(x, rfObject)

terminal_nodes(x, rfObject)
```

Arguments

x	a data.frame
rfObject	ranger object

Value

Matrix with terminal node IDs for all observations in x (rows) and trees (columns)

Examples

```
library(ranger)
rf.fit <- ranger(Species ~ ., data = iris, num.trees = 5, write.forest = TRUE)
dfNodes <- terminal_nodes(iris[, -5], rf.fit)
```

weightedDistance *Weighted Distance calculation*

Description

Computes $d(x_i, y_j) = \sum_k |w_k(x_{ik} - y_{jk})|$.

Usage

```
weightedDistance(x, y = NULL, weights = NULL)

weighted_distance(x, y = NULL, weights = NULL)
```

Arguments

x	a numeric matrix
y	a second numeric matrix with the same columns as x (default: NULL, pairwise distances within x)
weights	a vector of weights, one per column (default: all 1)

Value

a dist object (if y is NULL) or a nrow(x) x nrow(y) matrix

Examples

```
x <- as.matrix(iris[1:5, 1:4])
weighted_distance(x)
weighted_distance(x, as.matrix(iris[6:7, 1:4]), weights = c(1, 1, 2, 2))
```

Index

∗ **data-preparation**
RegressionModel, 19
∗ **datapreparation**
CBRBase, 3

as_dist_object *(*asDistObject*)*, 3
asDistObject, 3

CaseBasedReasoning::CBRBase, *7*, *11, 12*, *19, 20*
CaseBasedReasoning::RegressionModel, *7*, *11, 12*
CBRBase, 3
CoxModel, 7

depth_distance, 8
distance_random_forest *(*distanceRandomForest*)*, 9
distanceRandomForest, 9

edges_between_terminal_nodes, 10

generate_grid, 11

LinearModel, 11
LogisticModel, 12

predict.CoxModel, 13
predict.LinearModel, 13
predict.LogisticModel, 14
predict.RFModel, 14
print.CoxModel, 15
print.LinearModel, 15
print.LogisticModel, 16
print.RFModel, 16
proximity_distance, *9*, 17, *20*

ranger_forests_to_matrix, 18
RegressionModel, 19
RFModel, 20

summary.CoxModel, 22
summary.LinearModel, 22
summary.LogisticModel, 23
summary.RFModel, 23

terminal_nodes *(*terminalNodes*)*, 24
terminalNodes, 24

weighted_distance *(*weightedDistance*)*, 24
weightedDistance, 24