

Package: randomPlantedForest (via r-universe)

October 16, 2026

Title Random Planted Forest: A Directly Interpretable Tree Ensemble

Version 0.5.0.9000

Description Implements Random Planted Forest (Hiabu, Mammen and Meyer, 2020) <doi:10.48550/arXiv.2012.14563>, a directly interpretable tree ensemble for regression and classification. Trees are restricted to a maximum number of interacting predictors, so the fitted model decomposes exactly into main effects and low-order interactions, a functional ANOVA decomposition that can be extracted for interpretation. Fitting and prediction are multithreaded, and fitted models can be serialized.

License Apache License (>= 2)

URL <https://github.com/PlantedML/randomPlantedForest>,
<https://plantedml.com/randomPlantedForest/>

BugReports <https://github.com/PlantedML/randomPlantedForest/issues>

Depends R (>= 4.1.0)

Imports checkmate, cli, data.table, hardhat, methods, Rcpp, rlang,
utils

Suggests bundle, ggplot2, knitr, palmerpenguins, patchwork, recipes,
rmarkdown, testthat (>= 3.0.0)

LinkingTo Rcpp

VignetteBuilder knitr

Config/Needs/coverage covr

Config/Needs/website bench, mvtnorm

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Repository <https://plantedml.r-universe.dev>

Date/Publication 2026-10-16 15:16:44 UTC
RemoteUrl https://github.com/PlantedML/randomPlantedForest
RemoteRef HEAD
RemoteSha 6db527c76fa80b99485c3a06b5af9bfd69186187

Contents

bundle.rpf	2
plot.rpf	3
predict.rpf	4
predict_components	6
print.rpf	7
print.rpf_forest	8
purify	9
rpf	10
rpf_boxes	14
rpf_is_valid	16
rpf_marshall	16
Index	18

bundle.rpf	<i>Bundle an rpf model</i>
------------	----------------------------

Description

Method for `bundle::bundle()` wrapping `rpf_marshall()/rpf_unmarshal()`, so rpf models work with the standard tidymodels serialization workflow. Training data is not included; see `rpf_marshall()` for the implications.

Usage

```
## S3 method for class 'rpf'
bundle(x, ...)
```

Arguments

- x [rpf]: A fitted `rpf` model.
- ... Unused.

Value

An object of class `bundled_rpf` for `bundle()`, or the restored `rpf` object for `unbundle()`.

Examples

```
fit <- rpf(mpg ~ wt + cyl, data = mtcars, ntrees = 10)
b <- bundle::bundle(fit)
tmp <- tempfile(fileext = ".rds")
saveRDS(b, tmp)
restored <- bundle::unbundle(readRDS(tmp))
predict(restored, mtcars)
```

plot.rpf

Plot a tree family

Description

Illustrates how a Random Planted Forest predicts by drawing the trees of one tree family. Each tree belongs to a set of variables, and its leaves are boxes in those variables with a value each. The prediction for a point is the root value plus the values of all leaves containing it, averaged over tree families.

Usage

```
## S3 method for class 'rpf'
plot(
  x,
  family = 1L,
  ...,
  type = c("tree", "boxes"),
  max_interaction = 2L,
  trees = NULL,
  class = NULL,
  factor_levels = TRUE
)
```

Arguments

x	[rpf]: A fitted rpf model.
family	[integer(1): 1]: Which tree family to extract, between 1 and ntrees.
...	Reserved for future expansion.
type	[character(1): "tree"]: "tree" for a split diagram, "boxes" for the leaves in variable space.
max_interaction	[integer(1): 2]: Draw trees on up to this many variables, at most 2 for type = "boxes". The remaining trees are summarized.
trees	[character NULL: NULL]: Names of the trees to draw, as in the tree column of rpf_boxes() , e.g. c("x1", "x1:x2"). NULL draws all trees up to max_interaction.

<code>class</code>	[character(1) NULL: NULL]: For multiclass classification, the outcome level whose values to plot. NULL uses the first level.
<code>factor_levels</code>	[logical(1): TRUE]: Label factors with their levels instead of integer codes. Levels are ordered by their association with the outcome, as used in the fit.

Details

- `type = "tree"` draws a split diagram in the style of Figure 1b of Hiabu et al. (2023), with the trees in rows by number of variables, each leaf filled by its value. The forest stores only its final leaves, not the order of splits or which leaf a tree grew from, so the splits within a tree are reconstructed: the diagram shows a split structure consistent with the leaves, and dashed lines link trees to the trees they could have grown from. Leaves that overlap other leaves of the same tree hang from dotted lines. Readable for small fits, e.g. `ntrees = 1` and `splits = 10`.
- `type = "boxes"` draws one panel per tree in variable space: leaves of trees on one variable as bars from zero to their value, leaves of trees on two variables as rectangles filled by their value. Overlapping leaves add up, which the translucent drawing shows.

Trees on more than `max_interaction` variables, and trees not in `trees`, are not drawn but summarized as a remainder, similar to the remainder term of `predict_components()`.

After `purify()`, the trees are the components of the functional decomposition and only `type = "boxes"` applies. To plot components averaged over all tree families, use `predict_components()`, e.g. with the plotting functions in `glex`.

Value

A `ggplot` for `type = "tree"`, a patchwork for `type = "boxes"`.

References

Hiabu, M., Mammen, E., Meyer, J. T. (2023). Random Planted Forest: a directly interpretable tree ensemble. arXiv:2012.14563.

Examples

```
rpfit <- rpf(mpg ~ wt + hp, data = mtcars, ntrees = 1, splits = 10)
plot(rpfit)
purify(rpfit)
plot(rpfit, type = "boxes")
```

predict.rpf

Random Planted Forest Predictions

Description

Random Planted Forest Predictions

Usage

```
## S3 method for class 'rpf'
predict(
  object,
  new_data,
  type = ifelse(object$mode == "regression", "numeric", "prob"),
  nthreads = NULL,
  ...
)
```

Arguments

<code>object</code>	[<code>rpf</code>]: A fitted <code>rpf</code> model.
<code>new_data</code>	[<code>data.frame</code> <code>matrix</code>]: New observations to predict.
<code>type</code>	[<code>character(1)</code>]: "numeric" for regression outcomes, "class" for class predictions or "prob" for probability predictions. Defaults to "numeric" for regression and "prob" for classification. For classification and <code>loss = "L1"</code> or <code>"L2"</code> , "numeric" yields raw predictions which are not guaranteed to be valid probabilities in $[0, 1]$. For <code>type = "prob"</code> , these are truncated to ensure this property. If <code>loss</code> is "logit" or "exponential", <code>type = "link"</code> is an alias for <code>type = "numeric"</code> , as in this case the raw predictions have the additional interpretation similar to the linear predictor in a <code>glm</code> .
<code>nthreads</code>	[<code>integer(1)</code> <code>NULL</code> : <code>NULL</code>]: Number of threads. <code>NULL</code> uses the <code>nthreads</code> the forest was fitted with, capped at the available cores.
<code>...</code>	Not currently used, but required for extensibility. Unknown arguments are an error.

Value

For regression: A `tbl` with column `.pred` with the same number of rows as `new_data`.

For classification: A `tbl` with one column `.pred_<level>` for each level in `y` containing class probabilities if `type = "prob"`. For `type = "class"`, one column `.pred_class` with class predictions is returned. For `type = "numeric"` or `"link"`, raw predictions are returned: one column `.pred` for binary outcomes, and one column `.pred_<level>` per level for multiclass outcomes. With `loss = "logit"`, the first level is the reference class and has no column, so $K - 1$ columns are returned for K levels.

Examples

```
# Regression with L2 loss
rpf_fit <- rpf(y = mtcars$mpg, x = mtcars[, c("cyl", "wt")])
predict(rpf_fit, mtcars[, c("cyl", "wt")])
```

predict_components	<i>Extract predicted components from a Random Planted Forest</i>
--------------------	--

Description

Prediction components are a functional decomposition of the model prediction. The sum of all components equals the overall predicted value for an observation.

Usage

```
predict_components(object, new_data, max_interaction = NULL, predictors = NULL)
```

Arguments

object	[rpf]: A fitted rpf model.
new_data	[data.frame matrix]: New observations to predict.
max_interaction	[integer(1) NULL: NULL]: Maximum degree of interactions to consider. NULL uses the max_interaction parameter from the rpf object. Must be between 1 (main effects only) and the max_interaction of the rpf object.
predictors	[character NULL: NULL]: Vector of one or more column names of predictor variables in new_data to extract components for. If NULL, all variables and their interactions are returned.

Details

Extracts all possible components up to max_interaction degrees, up to the value set when calling [rpf\(\)](#). The intercept is always included. Optionally predictors can be specified to only include components including the given variables. If max_interaction is greater than length(predictors), the max_interaction will be lowered accordingly.

The components are only unique for a purified forest. An unpurified forest is purified in place on first use with the defaults of [purify\(\)](#). For other settings, such as more threads, call [purify\(\)](#) before [predict_components\(\)](#).

Value

A list with elements:

- m ([data.table](#)): Components for each main effect and interaction term, representing the functional decomposition of the prediction. All components together with the intercept sum up to the prediction. For multiclass classification, the number of output columns is multiplied by the number of levels in the outcome.
- intercept (numeric): Expected value of the prediction. For multiclass classification, one value per target level, named by target_levels.
- x ([data.table](#)): Copy of new_data containing predictors selected by predictors.
- target_levels (character): For multiclass classification only: Vector of target levels which can be used to disassemble m, as names include both term and target level.

Note

Depending on the number of predictors and max_interaction, the number of components will increase drastically to `sum(choose(ncol(new_data), seq_len(max_interaction)))`.

Examples

```
# Regression task, only some predictors
train <- mtcars[1:20, 1:4]
test <- mtcars[21:32, 1:4]

set.seed(23)
rpfit <- rpf(mpg ~ ., data = train, max_interaction = 3, ntrees = 30)

# Extract all components, including main effects and interaction terms up to `max_interaction`
(components <- predict_components(rpfit, test))

# sums to prediction
cbind(
  m_sum = rowSums(components$m) + components$intercept,
  prediction = predict(rpfit, test)
)

# Only get components with interactions of a lower degree, ignoring 3-way interactions
predict_components(rpfit, test, max_interaction = 2)

# Only retrieve main effects
(main_effects <- predict_components(rpfit, test, max_interaction = 1))

# The difference is the combined contribution of interaction effects
cbind(
  m_sum = rowSums(main_effects$m) + main_effects$intercept,
  prediction = predict(rpfit, test)
)
```

print.rpf*Print an rpf*

Description

Print an rpf

Usage

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

Arguments

x [rpf]: A fitted [rpf](#) model.
 ... Further arguments passed to or from other methods.

Value

Invisibly: x.

See Also

[rpf](#).

Examples

```
rpf(mpg ~ cyl + wt + drat, data = mtcars, max_interaction = 2, ntrees = 10)
```

print.rpf_forest	<i>Compact printing of forest structures</i>
------------------	--

Description

These methods are provided to avoid flooding the console with long nested lists containing tree structures.

Usage

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

```
## S3 method for class 'rpf_forest'
str(object, ...)
```

Arguments

x [rpf_forest]: Flattened forest, as in \$forest of an [rpf](#).
 ... Further arguments passed to or from other methods.
 object [rpf_forest]: Flattened forest.

See Also

[rpf](#)

Examples

```
rpf_fit <- rpf(mpg ~ cyl + wt, data = mtcars, ntrees = 10)
print(rpf_fit$forest)
str(rpf_fit$forest)
```

purify

*Purify a Random Planted Forest***Description**

Purification makes the components of the forest's functional decomposition unique, which `predict_components()` relies on. Unless `rpf()` was called with `purify = TRUE`, `predict_components()` purifies the forest on first use.

Usage

```
purify(x, ..., maxp_interaction = NULL, mode = 2L, nthreads = NULL)
```

```
is_purified(x)
```

Arguments

<code>x</code>	[<code>rpf</code>]: A fitted <code>rpf</code> model.
<code>...</code>	Reserved for future expansion.
<code>maxp_interaction</code>	[<code>integer(1) NULL: NULL</code>]: Highest interaction order to purify. Higher-order components are set to zero, but still influence lower orders during purification, and <code>predict_components()</code> then returns zero for them. <code>NULL</code> purifies all orders.
<code>mode</code>	[<code>integer(1): 2</code>]: Purification algorithm: 2 is the fast exact KD-tree based algorithm, 1 the original grid-based one.
<code>nthreads</code>	[<code>integer(1) NULL: NULL</code>]: Number of threads. <code>NULL</code> uses the <code>nthreads</code> the forest was fitted with, capped at the available cores.

Details

The forest is modified in place: `x` and every copy of it are purified, whether or not the result is assigned. `purify()` is idempotent, meaning if the forest is already purified it just returns it unmodified.

Value

`purify()` returns `x` invisibly. `is_purified()` returns `TRUE` or `FALSE`.

Examples

```
rpf_fit <- rpf(mpg ~ ., data = mtcars, max_interaction = 2, ntrees = 10)
is_purified(rpf_fit)
purify(rpf_fit)
is_purified(rpf_fit)
```

rpf

*Random Planted Forest***Description**

Fits a random planted forest (Hiabu, Mammen and Meyer, 2020) for regression or classification. The model is a sum of trees that each split on at most `max_interaction` predictors, so it decomposes into main effects and interaction components; see [predict_components\(\)](#).

Usage

```
rpf(x, ...)  
  
## S3 method for class 'formula'  
rpf(  
  formula,  
  data,  
  max_interaction = 2,  
  ntrees = 50,  
  splits = 30,  
  split_structure = "leaves",  
  split_try = 10,  
  t_try = 0.4,  
  max_candidates = 50,  
  split_decay_rate = 0.1,  
  delete_leaves = TRUE,  
  loss = NULL,  
  delta = 0.001,  
  epsilon = 0.1,  
  purify = FALSE,  
  nthreads = 1,  
  export_forest = FALSE,  
  deterministic = FALSE,  
  ...  
)  
  
## S3 method for class 'data.frame'  
rpf(  
  x,  
  y,  
  max_interaction = 2,  
  ntrees = 50,  
  splits = 30,  
  split_structure = "leaves",  
  split_try = 10,  
  t_try = 0.4,  
  max_candidates = 50,
```

```
split_decay_rate = 0.1,
delete_leaves = TRUE,
loss = NULL,
delta = 0.001,
epsilon = 0.1,
purify = FALSE,
nthreads = 1,
export_forest = FALSE,
deterministic = FALSE,
...
)

## S3 method for class 'matrix'
rpf(
  x,
  y,
  max_interaction = 2,
  ntrees = 50,
  splits = 30,
  split_structure = "leaves",
  split_try = 10,
  t_try = 0.4,
  max_candidates = 50,
  split_decay_rate = 0.1,
  delete_leaves = TRUE,
  loss = NULL,
  delta = 0.001,
  epsilon = 0.1,
  purify = FALSE,
  nthreads = 1,
  export_forest = FALSE,
  deterministic = FALSE,
  ...
)

## S3 method for class 'recipe'
rpf(
  x,
  data,
  max_interaction = 2,
  ntrees = 50,
  splits = 30,
  split_structure = "leaves",
  split_try = 10,
  t_try = 0.4,
  max_candidates = 50,
  split_decay_rate = 0.1,
  delete_leaves = TRUE,
```

```

    loss = NULL,
    delta = 0.001,
    epsilon = 0.1,
    purify = FALSE,
    nthreads = 1,
    export_forest = FALSE,
    deterministic = FALSE,
    ...
)

```

Arguments

<code>x, y</code>	[data.frame matrix recipe], [numeric factor]: Predictors and outcome for the x/y interface. <code>x</code> holds the predictors, or is a recipe (then supply data instead of <code>y</code>). <code>y</code> is the outcome vector: numeric for regression, factor for classification.
<code>...</code>	Not currently used, but required for extensibility. Unknown (e.g. misspelled) arguments are an error.
<code>formula</code>	[formula]: A formula with the outcome on the left-hand side and the predictors on the right-hand side, e.g. <code>y ~ x1 + x2</code> .
<code>data</code>	[data.frame]: Data containing the predictors and the outcome, used with formula or a recipe <code>x</code> .
<code>max_interaction</code>	[integer(1): 2]: Maximum number of predictors a single tree may split on. 1 fits main effects only (an additive model), 2 adds pairwise interactions, and so on. 0 uses all predictors and values above the number of predictors are reduced to it.
<code>ntrees</code>	[integer(1): 50]: Number of tree families, i.e. the size of the forest. Each family is grown on a bootstrap sample and predictions are averaged.
<code>splits</code>	[integer(1): 30]: Number of splits per tree family. The main tuning parameter: more splits fit more complex functions.
<code>split_structure</code>	[character(1): "leaves"]: How split candidates are formed and sampled; one of "leaves", "hist", "cur_trees_1", "cur_trees_2", or "res_trees". See Details.
<code>split_try</code>	[integer(1): 10]: Number of thresholds evaluated per split candidate.
<code>t_try</code>	[numeric(1): 0.4]: Proportion in (0, 1] of all possible splits sampled as candidates in each round, capped at <code>max_candidates</code> .
<code>max_candidates</code>	[integer(1): 50]: Maximum number of split candidates per round.
<code>split_decay_rate</code>	[numeric(1): 0.1]: Down-weights possible splits that were sampled as candidates but not chosen. Each such round ages a split by one, choosing it resets its age to zero, and splits are sampled with weight $\exp(-\text{split_decay_rate} * \text{age})$. 0 samples uniformly.
<code>delete_leaves</code>	[logical(1): TRUE]: Whether to delete a parent leaf when splitting along an existing dimension.

loss	[character(1) NULL: NULL]: Loss function. NULL uses "L2" for regression, the only supported loss, and "exponential" for classification. Classification also supports "logit", which gives similar results but is slower, and "L1" and "L2", which fit class indicators directly and do not yield proper probability estimates.
delta	[numeric(1): 0.001]: Only used if loss is "logit" or "exponential". Class proportions are truncated to [delta, 1 - delta] when computing the loss of a split. Should be positive for "logit": with delta = 0, nodes containing a single class have infinite loss and their splits are always rejected.
epsilon	[numeric(1): 0.1]: Only used if loss is "logit" or "exponential". Class proportions are truncated to [epsilon, 1 - epsilon] when computing the fit in a leaf. Unlike delta, this caps the size of individual leaf updates and acts as regularization: smaller values permit larger jumps on the link scale.
purify	[logical(1): FALSE]: Whether to purify the forest after fitting, which <code>predict_components()</code> requires, with the defaults of <code>purify()</code> . For other settings, call <code>purify()</code> after fitting instead.
nthreads	[integer(1): 1]: Number of threads for fitting. Also the default for <code>predict()</code> and <code>purify()</code> .
export_forest	[logical(1): FALSE]: Whether to store the flattened forest in the returned object as \$forest. If FALSE, \$forest is NULL, which saves memory.
deterministic	[logical(1): FALSE]: Whether to fit without randomness: no bootstrap, the first max_candidates possible splits as candidates (t_try is ignored) and evenly spaced thresholds. The result does not depend on the seed, and all tree families are identical, so use ntrees = 1. Mainly useful for testing and debugging.

Details

Choosing parameters:

Start with max_interaction and splits. max_interaction sets which effects the model can represent: 1 for an additive model, 2 (the default) to add pairwise interactions, which can still be plotted as heatmaps. Higher values are more flexible, but slower and the components are harder to interpret. splits sets how closely the model follows the data and is the parameter to tune. With a higher max_interaction, splits are spread over more possible components, so tune both together.

ntrees averages over bootstrap samples to reduce variance and fitting and prediction time grow linearly with it. The default is lower than the usual 500 trees of a random forest because each tree family is itself a complete model, a sum of trees covering all components, rather than a single tree. Averaging therefore stops paying off sooner, usually after a few dozen families. The split search parameters (split_structure, split_try, t_try, max_candidates, split_decay_rate) trade speed for a more thorough search.

For classification, "exponential" (the default) and "logit" fit on the link scale and are transformed to probabilities, "exponential" being faster. "L1" and "L2" fit class indicators directly and their probabilities are truncated to [0, 1].

split_structure:

In each round, a t_try fraction of all possible splits (capped at max_candidates) is drawn as candidates with weights $\exp(-\text{split_decay_rate} * \text{age})$. split_structure defines what a candidate is and how its thresholds are evaluated.

- **leaves:** Split candidates are (leaf, split-dimension) pairs. For each sampled candidate, `split_try` thresholds are drawn uniformly from the valid range within that leaf and evaluated to choose the best split.
- **hist:** As leaves, but thresholds are drawn from boundaries of quantile bins computed once per predictor, which makes evaluating them faster on large data.
- **cur_trees_1:** Split candidates are (current-tree, split-dimension) pairs. For each sampled candidate, perform `split_try` evaluations. Each evaluation samples a leaf from the set of valid current trees (with probability proportional to its number of available thresholds) and then uniformly samples a single threshold within that leaf.
- **cur_trees_2:** Split candidates are (current-tree, split-dimension) pairs. For each sampled candidate, iterate through every valid leaf. Within each leaf, sample `split_try` thresholds uniformly and evaluate them.
- **res_trees:** Split candidates are resulting trees. For each sampled candidate, run `split_try` evaluations by sampling a (split-dimension, leaf) pair from all valid pairs (with probability proportional to its number of available thresholds), then uniformly sampling one threshold within that pair.

Value

Object of class "rpf" with model object contained in `$fit`.

References

Hiabu, M., Mammen, E., & Meyer, J. T. (2020). Random Planted Forest: a directly interpretable tree ensemble. [doi:10.48550/arXiv.2012.14563](https://doi.org/10.48550/arXiv.2012.14563)

Examples

```
# Regression with formula
rpf_fit <- rpf(mpg ~ cyl + wt, data = mtcars)

# Regression with x and y, allowing pairwise interactions
rpf_fit <- rpf(x = mtcars[, c("cyl", "wt", "hp")], y = mtcars$mpg, max_interaction = 2)

# Classification
rpf_fit <- rpf(Species ~ ., data = iris, loss = "exponential")
```

rpf_boxes

Extract the boxes of a tree family

Description

Every tree in a Random Planted Forest is a set of boxes (leaves) with a value each, and the prediction is the sum of the values of all boxes containing the point, averaged over tree families. `rpf_boxes()` returns the boxes of one tree family as a data frame.

Usage

```
rpf_boxes(x, family = 1L, ...)
```

Arguments

x	[rpf]: A fitted rpf model.
family	[integer(1): 1]: Which tree family to extract, between 1 and ntrees.
...	Reserved for future expansion.

Details

Before purification, these are the leaves as grown: boxes in one tree may overlap, and the trees are not an identified decomposition. After [purify\(\)](#), they are the cells of the purified grid, one non-overlapping set per component of the functional decomposition. As [purify\(\)](#) modifies the forest in place, extract the unpurified boxes before purifying.

Factor predictors are encoded as integer codes in the order of `x$factor_levels`, with a box covering the codes `c` with lower $\leq c <$ upper.

Value

A [base::data.frame](#) with one row per box and columns

- tree: The tree, named by its sorted variables as in [predict_components\(\)](#), e.g. "x1:x2", or "(Intercept)".
- order: The number of variables of the tree, 0 for the intercept.
- box: The index of the box within its tree.
- value: The value of the box. For multiclass classification, one column `value_<class>` per outcome level instead.
- `<variable>_lower`, `<variable>_upper` for every predictor: The bounds of the box, NA for variables the tree does not use.

See Also

[plot.rpf\(\)](#) to plot the boxes.

Examples

```
rpf_fit <- rpf(mpg ~ wt + hp, data = mtcars, ntrees = 5, splits = 10)
head(rpf_boxes(rpf_fit))
```

rpf_is_valid	<i>Check whether an rpf object's C++ forest is still alive</i>
--------------	--

Description

The C++ forest does not survive `saveRDS()`; an rpf object restored via `readRDS()` without `rpf_marshal()/rpf_unmarshal()` is unusable.

Usage

```
rpf_is_valid(x)
```

Arguments

x [rpf]: A fitted `rpf` model, possibly restored from disk.

Value

TRUE if the underlying model can be used, FALSE otherwise.

rpf_marshal	<i>Serialize and restore Random Planted Forests</i>
-------------	---

Description

`rpf_marshal()` converts an `rpf` object into a plain R list safe for `saveRDS()`; `rpf_unmarshal()` reverses it. The C++ forest behind an rpf object does not survive R serialization, so this explicit round-trip is required to store or transfer fitted models.

Usage

```
rpf_marshal(x, include_data = FALSE)
```

```
rpf_unmarshal(blob)
```

Arguments

x [rpf]: A fitted `rpf` model.
include_data [logical(1): FALSE]: Store training data in the blob, enabling `purify()` after restoring.
blob [rpf_marshaled]: Object created by `rpf_marshal()`.

Details

Training data is only included with `include_data = TRUE`; without it, a restored forest can predict (including purified prediction if the forest was purified before marshaling) but can never be purified afterwards.

Value

`rpf_marshal()` returns a list of class `rpf_marshaled`; `rpf_unmarshal()` returns an object of class `rpf`.

Examples

```
fit <- rpf(mpg ~ wt + cyl, data = mtcars, ntrees = 10)
blob <- rpf_marshal(fit)
tmp <- tempfile(fileext = ".rds")
saveRDS(blob, tmp)
restored <- rpf_unmarshal(readRDS(tmp))
all.equal(predict(fit, mtcars), predict(restored, mtcars))
```

Index

`base::data.frame`, [15](#)
`bundle.rpf`, [2](#)
`bundle::bundle()`, [2](#)

`data.table`, [6](#)

`glm`, [5](#)

`is_purified(purify)`, [9](#)

`plot.rpf`, [3](#)
`plot.rpf()`, [15](#)
`predict()`, [13](#)
`predict.rpf`, [4](#)
`predict_components`, [6](#)
`predict_components()`, [4](#), [9](#), [10](#), [13](#), [15](#)
`print.rpf`, [7](#)
`print.rpf_forest`, [8](#)
`purify`, [9](#)
`purify()`, [4](#), [6](#), [13](#), [15](#), [16](#)

`readRDS()`, [16](#)
`recipe`, [12](#)
`rpf`, [2](#), [3](#), [5](#), [6](#), [8](#), [9](#), [10](#), [15–17](#)
`rpf()`, [6](#), [9](#)
`rpf_boxes`, [14](#)
`rpf_boxes()`, [3](#)
`rpf_is_valid`, [16](#)
`rpf_marshal`, [16](#)
`rpf_marshal()`, [2](#), [16](#)
`rpf_unmarshal(rpf_marshal)`, [16](#)
`rpf_unmarshal()`, [2](#), [16](#)

`saveRDS()`, [16](#)
`str.rpf_forest(print.rpf_forest)`, [8](#)

`tbl`, [5](#)