Package 'ppforest2'

July 21, 2026

Type Package

Title Projection Pursuit Oblique Decision Trees and Random Forests

Version 0.1.2

Date 2026-07-20

Description Builds decision trees by splitting on linear combinations of randomly chosen variables. Projection pursuit is used to choose a projection of the variables that best separates the groups. Using linear combinations of variables to separate groups takes the correlation between variables into account, which allows the model to outperform a traditional decision tree when the separation between groups occurs in combinations of variables. Single trees can be assembled into random forests for improved accuracy. Implements projection pursuit classification trees (Lee, Cook, Park and Lee (2013) <doi:10.1214/13-EJS810>) and projection pursuit forests (da Silva, Cook and Lee (2021) <doi:10.1080/10618600.2020.1870480>), following the earlier 'PPforest' package.

License MIT + file LICENSE

Imports Rcpp (>= 1.0.11)

LinkingTo Rcpp, RcppEigen

Depends R (>= 3.5.0)

LazyData true

RoxygenNote 7.3.2

Encoding UTF-8

Suggests ggplot2, jsonlite, knitr, parsnip, patchwork, rlang, rmarkdown, rsample, testthat (>= 3.0.0), tibble, tune, vdiffr, withr, workflows, yardstick

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel false

NeedsCompilation yes

Author Andrés Vidal [aut, cre, cph],
Natalia da Silva [aut]

Maintainer Andrés Vidal <andres@andresvidal.dev>

Repository CRAN

Date/Publication 2026-07-21 08:20:02 UTC

Contents

`bag_samples` *In-bag row indices per tree.*

Description

Returns a list where element ‘i‘ is the integer vector of row indices (1-based, with replacement) drawn into the bootstrap sample of tree ‘i‘.

Usage

```
bag_samples(model)
```

Arguments

`model`	A `pprf` forest model.

Value

A list of integer vectors, one per tree.

See Also

`oob_samples`

binarize_disabled *Disabled binarization strategy (placeholder).*

Description

Placeholder binarizer for specs where binarization never fires — notably regression, where 'grouping_by_cutpoint()' always produces a 2-group partition at each node. Selecting `binarize_disabled()` documents that intent explicitly; if binarization is ever invoked at runtime with this strategy configured, training aborts with a clear error rather than silently passing through. Default for regression specs.

Usage

```
binarize_disabled()
```

Value

A `binarize_strategy` object.

See Also

`binarize_largest_gap`

Examples

```
binarize_disabled()
```

binarize_largest_gap *Largest-gap binarization strategy.*

Description

Creates a binarization strategy that reduces multiclass nodes to binary by projecting group means and splitting at the largest gap. Default for classification specs.

Usage

```
binarize_largest_gap()
```

Value

A `binarize_strategy` object.

See Also

`pptr`, `pprf`, `binarize_disabled`

Examples

```
binarize_largest_gap()
```

`california_housing` *California Housing Dataset*

Description

Median house values and demographic/geographic predictors for 20,433 California block groups, derived from the 1990 US Census. Classic regression benchmark. Originally released by Pace & Barry (1997); this version drops the categorical ‘ocean_proximity‘ column (to keep the bundled schema all-numeric, matching the rest of the package’s datasets) and removes rows with missing ‘total_bedrooms‘. The regression target, ‘median_house_value‘, is the last column.

Usage

```
data(california_housing)
```

Format

A data frame with 20,433 rows and 9 variables.

Details

longitude Longitude of the block group (degrees, negative = west).

latitude Latitude of the block group (degrees).

housing_median_age Median age of houses in the block group (years).

total_rooms Total number of rooms.

total_bedrooms Total number of bedrooms.

population Block group population.

households Number of households.

median_income Median household income (tens of thousands of USD).

median_house_value Regression target: median house value (USD).

Source

Pace, R.K. and Barry, R. (1997) Sparse spatial autoregressions. Statistics & Probability Letters, 33, 291-297.

crab *Australian Crabs Dataset*

Description

Measurements on rock crabs of the genus Leptograpsus. The dataset contains 200 observations from two species of crab (blue and orange), there are 50 specimens of each sex of each species, collected on site at Fremantle, Western Australia.

Type is the group variable and has 4 groups with the combinations of species and sex (BlueMale, BlueFemale, OrangeMale and OrangeFemale).

FL the size of the frontal lobe length, in mm

RW rear width, in mm

CL length of midline of the carapace, in mm

CW maximum width of carapace, in mm

BD depth of the body; for females, measured after displacement of the abdomen, in mm

Usage

```
data(crab)
```

Format

A data frame with 200 rows and 6 variables

Source

Campbell, N. A. & Mahon, R. J. (1974), A Multivariate Study of Variation in Two Species of Rock Crab of genus Leptograpsus, Australian Journal of Zoology 22(3), 417 - 425.

crabs *Crabs Dataset*

Description

Measurements of 200 crabs from the genus Leptograpsus, with 50 crabs for each of two colour forms and both sexes. Measurements include the frontal lobe size, rear width, carapace length, carapace width, and body depth.

Format

A data frame with 200 rows and 8 variables:

Type Species code, with 'B' indicating blue and 'O' indicating orange.

sex Sex of the crab, with 'M' indicating male and 'F' indicating female.

FL Size of the frontal lobe in mm.

RW Rear width in mm.

CL Carapace length in mm.

CW Carapace width in mm.

BD Body depth in mm.

Source

Campbell, N.A., and Mahon, R.J. (1974) A multivariate study of variation in two species of rock crab of genus Leptograpsus. Australian Journal of Zoology, 22, 417-425.

`cutpoint_mean_of_means` *Mean-of-means split cutpoint strategy.*

Description

Creates a split cutpoint strategy that splits at the midpoint between group means. This is the default (and currently only) split cutpoint.

Usage

```
cutpoint_mean_of_means()
```

Value

A `cutpoint_strategy` object.

See Also

`pptr`, `pprf`, `pp_pda`, `vars_uniform`

Examples

```
cutpoint_mean_of_means()
```

fishcatch *Fish Catch Dataset*

Description

There are 159 fishes of 7 species are caught and measured. Altogether there are 7 variables. All the fishes are caught from the same lake(Laengelmavesi) near Tampere in Finland.

Type has 7 fish groups, with 35 cases of Bream, 11 cases of Parkki, 56 cases of Perch 17 cases of Pike, 20 cases of Roach, 14 cases of Smelt and 6 cases of Whitewish.

weight Weight of the fish (in grams)

length1 Length from the nose to the beginning of the tail (in cm)

length2 Length from the nose to the notch of the tail (in cm)

length3 Length from the nose to the end of the tail (in cm)

height Maximal height as % of Length3

width Maximal width as % of Length3

Usage

```
data(fishcatch)
```

Format

A data frame with 159 rows and 7 variables

Source

https://jse.amstat.org/jse_data_archive.htm

fitted.ppmodel *Fitted (in-sample) predictions from a ppforest2 model.*

Description

Returns predictions on the training data — a factor for classification, a numeric vector for regression. For forests, training predictions are optimistic; use `oob_predictions()` for an honest estimate.

Usage

```
## S3 method for class 'ppmodel'
fitted(object, ...)
```

Arguments

object	A pptr or pprf model.
...	Unused.

Value

A factor (classification) or numeric vector (regression), length equal to the number of training observations.

See Also

residuals.ppmodel, oob_predictions

---

formula.ppmodel *Formula extractor for ppforest2 models.*

Description

Formula extractor for ppforest2 models.

Usage

```
## S3 method for class 'ppmodel'
formula(x, ...)
```

Arguments

x	A pptr or pprf model.
...	Unused.

Value

The formula the model was trained with, or ‘NULL‘ for matrix-interface fits.

glass *Glass Dataset*

Description

Contains measurements 214 observations of 6 types of glass; defined in terms of their oxide content.

Type has 6 types of glasses

X1 refractive index

X2 Sodium (unit measurement: weight percent in corresponding oxide).

X3 Magnesium

X4 Aluminum

X5 Silicon

X6 Potassium

X7 Calcium

X8 Barium

X9 Iron

Usage

```
data(glass)
```

Format

A data frame with 214 rows and 10 variables

grouping_by_cutpoint *Cutpoint-based grouping strategy (regression).*

Description

Creates a grouping strategy for regression trees: observations are split by the cutpoint in projected space, then each child's observations are sorted by the continuous response and median-split into 2 new groups for the next projection-pursuit step. This is the default grouping strategy for regression.

Usage

```
grouping_by_cutpoint()
```

Value

A `grouping_strategy` object.

See Also

pptr, pprf, grouping_by_label

Examples

```
grouping_by_cutpoint()
```

---

grouping_by_label *Label-based grouping strategy.*

Description

Creates a grouping strategy that routes all observations of a group to the same child node. This is the default grouping strategy for classification.

Usage

```
grouping_by_label()
```

Value

A grouping_strategy object.

See Also

pptr, pprf

Examples

```
grouping_by_label()
```

---

image *Image Dataset*

Description

Contains 2310 observations of instances from 7 outdoor images

Type has 7 types of outdoor images, brickface, cement, foliage, grass, path, sky, and window.

X1 the column of the center pixel of the region

X2 the row of the center pixel of the region.

X3 the number of pixels in a region = 9.

X4 the results of a line extraction algorithm that counts how many lines of length 5 (any orientation) with low contrast, less than or equal to 5, go through the region.

X5 measure the contrast of horizontally adjacent pixels in the region. There are 6, the mean and standard deviation are given. This attribute is used as a vertical edge detector.

X6 X5 sd

X7 measures the contrast of vertically adjacent pixels. Used for horizontal line detection.

X8 sd X7

X9 the average over the region of (R + G + B)/3

X10 the average over the region of the R value.

X11 the average over the region of the B value.

X12 the average over the region of the G value.

X13 measure the excess red: (2R - (G + B))

X14 measure the excess blue: (2B - (G + R))

X15 measure the excess green: (2G - (R + B))

X16 3-d nonlinear transformation of RGB. (Algorithm can be found in Foley and VanDam, Fundamentals of Interactive Computer Graphics)

X17 mean of X16

X18 hue mean

Usage

```
data(image)
```

Format

A data frame contains 2310 observations and 19 variables

---

leaf_majority_vote *Majority-vote leaf strategy.*

Description

Creates a leaf strategy that assigns the majority group label as the leaf prediction. When groups are tied, the smallest label wins. This is the default leaf strategy for classification.

Usage

```
leaf_majority_vote()
```

Value

A `leaf_strategy` object.

See Also

pptr, pprf

Examples

```
leaf_majority_vote()
```

`leaf_mean_response` *Mean-response leaf strategy.*

Description

Creates a leaf strategy that predicts the mean of the continuous response values for the observations in the leaf. Used for regression trees; requires a numeric response.

Usage

```
leaf_mean_response()
```

Value

A `leaf_strategy` object.

See Also

pptr, pprf, leaf_majority_vote

Examples

```
leaf_mean_response()
```

`leukemia` *Leukemia Dataset*

Description

This dataset comes from a study of gene expression in two types of acute leukemias, acute lymphoblastic leukemia (ALL) and acute myeloid leukemia (AML). Gene expression levels were measured using Affymetrix high density oligonucleotide arrays containing 6817 human genes. A dataset containing 72 observations from 3 leukemia type groups.

Type has 3 groups with 38 cases of B-cell ALL, 25 cases of AML and 9 cases of T-cell ALL.

Gene1 to Gen 40 gene expression levels

Usage

```
data(leukemia)
```

Format

A data frame with 72 rows and 41 variables

Source

Dudoit, S., Fridlyand, J. and Speed, T. P. (2002). Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. Journal of the American statistical Association 97 77-87.

---

load_json *Load a model from a JSON file.*

Description

Deserializes a `pptr` or `pprf` model from a JSON file. The model can be used for prediction immediately. If variable importance metrics and OOB fields were saved, they are restored into the model's lazy-computation cache so later accessor calls (`oob_error()`, `permuted_importance()`, etc.) return the stored values without recomputation.

Usage

```
load_json(path)
```

Arguments

`path`	File path to read the JSON from.

Details

Note that `formula`, `x`, and `y` are not stored in the JSON and will be `NULL` on the loaded model. Formula-based prediction and any accessor that requires training data will error unless the user re-attaches those fields.

Value

A `pptr` or `pprf` model (with the appropriate `_classification` / `_regression` subclass).

See Also

`save_json`, `pptr`, `pprf`

Examples

```
model <- pptr(Species ~ ., data = iris, seed = 0)
path <- tempfile(fileext = ".json")
save_json(model, path)
loaded <- load_json(path)
predict(loaded, as.matrix(iris[, 1:4]))
```

`lymphoma` *Lymphoma Dataset*

Description

Gene expression in the three most prevalent adult lymphoid malignancies: B-cell chronic lymphocytic leukemia (B-CLL), follicular lymphoma (FL), and diffuse large B-cell lym- phoma (DLBCL). Gene expression levels were measured using a specialized cDNA microarray, the Lymphochip, containing genes that are preferentially expressed in lymphoid cells or that are of known immunologic or oncologic importance. This dataset contain 80 observations from 3 lymphoma types.

Type Group variable has 3 groups with 29 cases of B-cell ALL (B-CLL), 42 cases of diffuse large B-cell lymphoma (DLBCL) and 9 cases of follicular lymphoma (FL).

Gene1 to Gen 50 gene expression

Usage

```
data(lymphoma)
```

Format

A data frame with 80 rows and 51 variables

Source

Dudoit, S., Fridlyand, J. and Speed, T. P. (2002). Comparison of Discrimination Methods for the Classification of Tumors Using Gene Ex- pression Data. Journal of the American statistical Association 97 77-87.

NCI60 *NCI60 Dataset*

Description

cDNA microarrays were used to examine the variation in gene expression among the 60 cell lines. The cell lines are derived from tumors with different sites of origin. This dataset contain 61 observations and 30 feature variables from 8 different tissue types.

Usage

```
data(NCI60)
```

Format

A data frame with 61 rows and 31 variables

Details

Type has 8 different tissue types, 9 cases of breast, 5 cases of central nervous system (CNS), 7 cases pf colon, 8 cases of leukemia, 8 cases of melanoma, 9 cases of non-small-cell lung carcinoma (NSCLC), 6 cases of ovarian and 9 cases of renal.

Gene1 to Gen 30 gene expression information

Source

Dudoit, S., Fridlyand, J. and Speed, T. P. (2002). Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. Journal of the American statistical Association 97 77-87.

nobs.ppmodel *Number of observations used to fit a ppforest2 model.*

Description

Implements the standard `stats::nobs()` contract so downstream tools (`step()`, broom's `glance()`, information criteria) can ask for the training-sample size.

Usage

```
## S3 method for class 'ppmodel'
nobs(object, ...)
```

Arguments

`object`	A `pptr` or `pprf` model.
`...`	Unused.

Value

Integer scalar. Returns `NA_integer_` for models loaded from JSON without their original training data.

`olive` *Olive Dataset*

Description

Contains 572 observations and 10 variables

Region Three super-groups of Italy: North, South and the island of Sardinia

area Nine collection areas: three from North, four from South and 2 from Sardinia

palmitic fatty acids percent x 100

palmitoleic fatty acids percent x 100

stearic fatty acids percent x 100

oleic fatty acids percent x 100

linoleic fatty acids percent x 100

linolenic fatty acids percent x 100

arachidic fatty acids percent x 100

eicosenoic fatty acids percent x 100

Usage

```
data(olive)
```

Format

A data frame contains 573 observations and 10 variables

`oob_error` *Out-of-bag error for a random forest.*

Description

Computes (or returns the cached) OOB error using the training data stored on the model. For classification, this is the misclassification rate in ‘[0, 1]‘. For regression, it is the mean squared error against the continuous response.

Usage

```
oob_error(model)
```

Arguments

model	A pprf forest model.

Value

A numeric scalar in ‘[0, 1]‘ for classification or ‘[0, Inf)‘ for regression. Returns ‘NA_real_‘ when no observation has any out-of-bag tree (e.g. a degenerate forest where every tree saw every row). Callers should check with ‘is.na()‘ rather than comparing against a sentinel value; in earlier versions this condition was signalled as ‘-1‘, which was not distinguishable from a (mathematically impossible but representable) error rate.

See Also

oob_predictions, oob_samples

---

oob_predictions *Out-of-bag predictions for a random forest.*

Description

Returns predictions for each training row using only trees that did not see that row in their bootstrap sample. Observations with no OOB tree are represented as ‘NA‘ in both modes: ‘NA‘ at the factor level for classification, and ‘NA_real_‘ for regression. Filter with the standard ‘is.na()‘ idiom.

Usage

```
oob_predictions(model)
```

Arguments

model	A pprf forest model.

Value

A factor (classification) or numeric vector (regression), length ‘n‘.

See Also

oob_error, oob_samples

oob_samples *Out-of-bag row indices per tree.*

Description

Returns a list where element ‘i‘ is the integer vector of row indices (1-based) that were **not** in the bootstrap sample of tree ‘i‘.

Usage

```
oob_samples(model)
```

Arguments

model	A pprf forest model.

Value

A list of integer vectors, one per tree.

See Also

bag_samples, oob_error

parkinson *Parkinson Dataset*

Description

A dataset containing 195 observations from 2 parkinson types.

Type Group variable has 2 groups, there are 48 cases of healthy people and 147 cases with Parkinson. The feature variables are biomedical voice measures.

X1 Average vocal fundamental frequency

X2 Maximum vocal fundamental frequency

X3 Minimum vocal fundamental frequency

X4 MDVP:Jitter(%) measures of variation in fundamental frequency

X5 MDVP:Jitter(Abs) measures of variation in fundamental frequency

X6 MDVP:RAP measures of variation in fundamental frequency

X7 MDVP:PPQ measures of variation in fundamental frequency

X8 Jitter:DDP measures of variation in fundamental frequency

X9 MDVP:Shimmer measures of variation in amplitude

X10 MDVP:Shimmer(dB) measures of variation in amplitude

X11 Shimmer:APQ3 measures of variation in amplitude

X12 Shimmer:APQ5 measures of variation in amplitude

X13 MDVP:APQ measures of variation in amplitude

X14 Shimmer:DDA measures of variation in amplitude

X15 NHR measures of ratio of noise to tonal components in the voice

X16 HNR measures of ratio of noise to tonal components in the voice

X17 RPDE nonlinear dynamical complexity measures

X18 D2 nonlinear dynamical complexity measures

X19 DFA - Signal fractal scaling exponent

X20 spread1 Nonlinear measures of fundamental frequency variation

X21 spread2 Nonlinear measures of fundamental frequency variation

X22 PPE Nonlinear measures of fundamental frequency variation

Usage

```
data(parkinson)
```

Format

A data frame with 195 rows and 23 variables

Source

https://archive.ics.uci.edu/ml/datasets/Parkinsons

---

permuted_importance *Permuted variable importance for a random forest.*

Description

For each feature, measures the drop in OOB accuracy (classification) or the increase in normalised MSE (regression) after randomly permuting that feature across the OOB observations. Computed lazily from the training data stored on the model; the result is cached.

Usage

```
permuted_importance(model)
```

Arguments

`model`	A `pprf` forest model.

Details

Sign semantics. Entries may be **negative**. That is not an error and not a sentinel: it means permuting the feature did not degrade OOB fit on average — the feature's signal sits at or below the noise floor of the permutation procedure. Interpret negative or near-zero entries as "no evidence of importance"; rely on the ranking rather than clipping at zero or normalizing. The scale is already comparable within a fitted model.

Value

A numeric vector, one entry per feature. Negative values are meaningful (see Sign semantics above).

`plot.pprf` *Plot a pprf model.*

Description

Visualizes a pprf model. By default, shows variable importance with one plot per metric side by side. Use `metric` to show a single importance metric.

Usage

```
## S3 method for class 'pprf'
plot(x, type = "importance", metric = NULL, tree_index = 1L, node = 1L, ...)
```

Arguments

`x`	A pprf model.
`type`	Character string specifying the plot type. `"importance"` (default) shows variable importance, `"structure"` shows a specific tree with embedded histograms, `"projection"` shows projected data at a node, `"boundaries"` shows decision boundaries of a specific tree.
`metric`	Character string selecting a single importance metric to plot: `"projections"`, `"weighted"`, or `"permuted"`. `NULL` (default) shows all available metrics side by side in separate panels. Only used when `type = "importance"`.
`tree_index`	Integer index of the tree to plot (1-based). Only used when `type = "structure"`, `type = "projection"`, or `type = "boundaries"`. Defaults to 1.
`node`	Integer index of the node for projection plots. Defaults to 1 (root). Only used when `type = "projection"`.
`...`	Additional arguments passed to the internal plotting function.

Value

A ggplot2-compatible object (invisibly). The importance grid returns a patchwork object that works with `ggplot2::ggsave()`.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE)) {
  forest <- pprf(Species ~ ., data = iris, size = 10)
  plot(forest)                                    # all metrics side by side
  plot(forest, metric = "permuted")               # single metric
  plot(forest, type = "structure", tree_index = 1)
  plot(forest, type = "projection", tree_index = 1)
}
```

`plot.pptr` *Plot a pptr model.*

Description

Visualizes a pptr model. By default, shows a mosaic overview with tree structure, decision boundaries, and variable importance. Use `type` to show individual plots.

Usage

```
## S3 method for class 'pptr'
plot(x, type = NULL, metric = NULL, node = 1L, ...)
```

Arguments

`x`	A pptr model.
`type`	Character string specifying the plot type. `NULL` (default) shows a mosaic overview. Other options: `"structure"` for tree with embedded histograms, `"importance"` for variable importance, `"projection"` for projected data at a node, `"boundaries"` for decision boundaries in feature space.
`metric`	Character string selecting a single importance metric to plot: `"projections"`, `"weighted"`, or `"permuted"` (availability depends on the model). `NULL` (default) shows all available metrics together. Only used when `type = "importance"`.
`node`	Integer index of the node for projection plots (1-based, breadth-first order). Defaults to 1 (root node). Only used when `type = "projection"`.
`...`	Additional arguments passed to the internal plotting function.

Value

A ggplot2-compatible object (invisibly). The mosaic layout returns a patchwork object that works with `ggplot2::ggsave()`.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE)) {
  model <- pptr(Species ~ ., data = iris)
  plot(model)                          # mosaic overview
  plot(model, type = "structure")      # tree structure only
  plot(model, type = "importance")     # variable importance
  plot(model, type = "projection")     # projection histogram
  plot(model, type = "boundaries")     # decision boundaries
}
```

`pprf` *Trains a Random Forest of Projection-Pursuit oblique decision trees.*

Description

This function trains a Random Forest of Projection-Pursuit oblique decision tree using either a formula and data frame interface or a matrix-based interface. When using the formula interface, specify the model formula and the data frame containing the variables. For the matrix-based interface, provide matrices for the features and labels directly. The number of trees is controlled by the `size` parameter. Each tree is trained on a stratified bootstrap sample drawn from the data. The number of variables to consider at each split is controlled by the `n_vars` parameter. If `lambda = 0`, the model is trained using Linear Discriminant Analysis (LDA). If `lambda > 0`, the model is trained using Penalized Discriminant Analysis (PDA).

Usage

```
pprf(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
  mode = NULL,
  size = 100,
  lambda = 0.5,
  n_vars = NULL,
  p_vars = NULL,
  seed = NULL,
  max_retries = 3L,
  threads = NULL,
  pp = NULL,
  vars = NULL,
  cutpoint = NULL,
  stop = NULL,
  binarize = NULL,
  grouping = NULL,
```

```
  leaf = NULL
)
```

Arguments

formula	A formula of the form y ~ x1 + x2 + ..., where y is a vector of labels and x1, x2, ... are the features.
data	A data frame containing the variables in the formula.
x	A matrix containing the features for each observation.
y	A matrix containing the labels for each observation.
mode	Training mode: either "classification" or "regression". When NULL (default), mode is auto-detected from y's type — factor or character vectors trigger classification, numeric vectors trigger regression. Setting it explicitly is useful for the binary-integer-labels case (mode = "classification" with integer 0/1 labels) and for failing fast on a type mismatch (mode = "regression" with a factor y errors immediately).
size	The number of trees in the forest (default: 100).
lambda	A regularization parameter (default: 0.5). If lambda = 0, the model is trained using Linear Discriminant Analysis (LDA). If lambda > 0, the model is trained using Penalized Discriminant Analysis (PDA). The default uses PDA because pure LDA (lambda = 0) is ill-conditioned when there are more variables than effective observations (see the "Known limitations" section of the README). Cannot be used together with pp.
n_vars	The number of variables to consider at each split (integer). These are chosen uniformly in each split. By default, half of the variables are used (p_vars = 0.5). Cannot be used together with p_vars or dr.
p_vars	The proportion of variables to consider at each split (number between 0 and 1, exclusive). For example, p_vars = 0.5 uses half the features. Cannot be used together with n_vars or dr.
seed	An optional integer seed for reproducibility. If NULL (default), a seed is drawn from R's RNG, so set.seed() controls reproducibility. If an integer is provided, that value is used directly. The same seed is used for training and for computing permuted variable importance.
max_retries	Maximum number of retries for degenerate trees (default: 3). When a bootstrap sample yields a singular covariance matrix, the tree is retrained with a different seed up to this many times.
threads	The number of threads to use. The default is the number of cores available.
pp	A projection pursuit strategy object created by pp_pda. Cannot be used together with lambda.
vars	A variable selection strategy object created by vars_uniform or vars_all. Cannot be used together with n_vars or p_vars.
cutpoint	A split cutpoint strategy object created by cutpoint_mean_of_means (default).
stop	A stopping rule object. Default depends on mode: stop_pure_node() for classification, and stop_any(stop_min_size(5), stop_min_variance(0.01)) for regression.

binarize	A binarization strategy object. Default depends on mode: binarize_largest_gap() for classification, and binarize_disabled() for regression (regression's default grouping always yields a 2-group partition, so no binarization is needed).
grouping	A grouping strategy object. Default depends on mode: grouping_by_label() for classification, and grouping_by_cutpoint() for regression.
leaf	A leaf strategy object. Default depends on mode: leaf_majority_vote() for classification, and leaf_mean_response() for regression.

Details

Mode is taken from the mode argument when explicit, and otherwise auto-detected from 'y' (factor/character → classification, numeric → regression). Pass mode = "classification" to force classification on integer labels (e.g. binary 0/1), or mode = "regression" to assert intent on numeric responses.

OOB error, OOB predictions, permuted variable importance, and weighted variable importance are computed lazily on first access via the accessor functions ('oob_error()', 'oob_predictions()', 'permuted_importance()', 'weighted_importance()'). Training itself is fast because these OOB-based computations are deferred.

Value

A pprf model. Its S3 class vector is c("pprf_classification", "pprf", "ppmodel") or c("pprf_regression", "pprf", "ppmodel") depending on the mode.

See Also

predict.pprf_classification, predict.pprf_regression, formula.ppmodel, oob_error, save_json, load_json, pp_rand_forest for parsnip integration, vignette("introduction") for a tutorial

Examples

```
# Example 1: formula interface with the `iris` dataset
pprf(Species ~ ., data = iris)

# Example 2: formula interface with the `iris` dataset with regularization
pprf(Species ~ ., data = iris, lambda = 0.5)

# Example 3: matrix interface with the `iris` dataset
pprf(x = iris[, 1:4], y = iris[, 5])

# Example 4: matrix interface with the `iris` dataset with regularization
pprf(x = iris[, 1:4], y = iris[, 5], lambda = 0.5)

# Example 5: formula interface with the `crabs` dataset
pprf(Type ~ ., data = crabs)

# Example 6: formula interface with the `crabs` dataset with regularization
pprf(Type ~ ., data = crabs, lambda = 0.5)
```

pptr *Trains a Projection-Pursuit oblique decision tree.*

Description

This function trains a Projection-Pursuit oblique decision tree using either a formula and data frame interface or a matrix-based interface. When using the formula interface, specify the model formula and the data frame containing the variables. For the matrix-based interface, provide matrices for the features and labels directly. If `lambda = 0`, the model is trained using Linear Discriminant Analysis (LDA). If `lambda > 0`, the model is trained using Penalized Discriminant Analysis (PDA).

Usage

```
pptr(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
  mode = NULL,
  lambda = 0.5,
  seed = NULL,
  pp = NULL,
  cutpoint = NULL,
  stop = NULL,
  binarize = NULL,
  grouping = NULL,
  leaf = NULL
)
```

Arguments

`formula`	A formula of the form `y ~ x1 + x2 + ...`, where `y` is a vector of labels and `x1`, `x2`, ... are the features.
`data`	A data frame containing the variables in the formula.
`x`	A matrix containing the features for each observation.
`y`	A matrix containing the labels for each observation.
`mode`	Training mode: either `"classification"` or `"regression"`. When `NULL` (default), mode is auto-detected from `y`'s type — factor or character vectors trigger classification, numeric vectors trigger regression. Setting it explicitly is useful for the binary-integer-labels case (`mode = "classification"` with integer 0/1 labels) and for failing fast on a type mismatch (`mode = "regression"` with a factor `y` errors immediately).
`lambda`	A regularization parameter (default: 0.5). If `lambda = 0`, the model is trained using Linear Discriminant Analysis (LDA). If `lambda > 0`, the model is trained using Penalized Discriminant Analysis (PDA). The default uses PDA because

pure LDA (`lambda = 0`) is ill-conditioned when there are more variables than effective observations (see the "Known limitations" section of the README). Cannot be used together with `pp`.

`seed` An optional integer seed for reproducibility. If `NULL` (default), a seed is drawn from R's RNG, so `set.seed()` controls reproducibility. If an integer is provided, that value is used directly.

`pp` A projection pursuit strategy object created by `pp_pda`. Cannot be used together with `lambda`.

`cutpoint` A split cutpoint strategy object created by `cutpoint_mean_of_means` (default).

`stop` A stopping rule object. Default depends on mode: `stop_pure_node()` for classification, and `stop_any(stop_min_size(5), stop_min_variance(0.01))` for regression.

`binarize` A binarization strategy object. Default depends on mode: `binarize_largest_gap()` for classification, and `binarize_disabled()` for regression (regression's default grouping always yields a 2-group partition, so no binarization is needed).

`grouping` A grouping strategy object. Default depends on mode: `grouping_by_label()` for classification, and `grouping_by_cutpoint()` for regression.

`leaf` A leaf strategy object. Default depends on mode: `leaf_majority_vote()` for classification, and `leaf_mean_response()` for regression.

Details

Mode is taken from the `mode` argument when explicit, and otherwise auto-detected from 'y' (factor/character → classification, numeric → regression). Pass `mode = "classification"` to force classification on integer labels (e.g. binary 0/1), or `mode = "regression"` to assert intent on numeric responses.

Value

A `pptr` model. Its S3 class vector is `c("pptr_classification", "pptr", "ppmodel")` or `c("pptr_regression", "pptr", "ppmodel")` depending on the mode.

See Also

`predict.pptr_classification`, `predict.pptr_regression`, `formula.ppmodel`, `print.pptr`, `save_json`, `load_json`, `pp_tree` for parsnip integration

Examples

```
# Example 1: formula interface with the `iris` dataset
pptr(Species ~ ., data = iris)

# Example 2: formula interface with the `iris` dataset with regularization
pptr(Species ~ ., data = iris, lambda = 0.5)

# Example 3: matrix interface with the `iris` dataset
pptr(x = iris[, 1:4], y = iris[, 5])
```

pp_pda *PDA projection pursuit strategy.*

Description

Creates a Penalized Discriminant Analysis (PDA) projection pursuit strategy for use with `pptr` or `pprf`.

Usage

```
pp_pda(lambda = 0)
```

Arguments

`lambda`	A regularization parameter between 0 and 1. If `lambda = 0`, the model uses Linear Discriminant Analysis (LDA). If `lambda > 0`, the model uses Penalized Discriminant Analysis (PDA).

Value

A `pp_strategy` object.

See Also

`pptr`, `pprf`, `vars_uniform`, `vars_all`, `cutpoint_mean_of_means`

Examples

```
# PDA with lambda = 0.5
pp_pda(0.5)

# Use with pptr
pptr(Species ~ ., data = iris, pp = pp_pda(0.5))

# Use with pprf
pprf(Species ~ ., data = iris, pp = pp_pda(0.5), vars = vars_uniform(n_vars = 2))
```

pp_rand_forest *Parsnip model specification for pprf.*

Description

Creates a model specification for a Projection Pursuit random forest. Use `set_engine("ppforest2")` to select the ppforest2 engine.

Usage

```
pp_rand_forest(
  mode = "classification",
  trees = NULL,
  mtry = NULL,
  mtry_prop = NULL,
  penalty = NULL
)
```

Arguments

mode	A character string for the model type. Either "classification" or "regression".
trees	The number of trees in the forest (maps to size).
mtry	The number of variables to consider at each split (maps to n_vars).
mtry_prop	The proportion of variables to consider at each split (maps to p_vars). An alternative to mtry that expresses the feature subsample as a fraction in (0, 1], tunable via the dials mtry_prop() parameter. Supply mtry or mtry_prop, not both.
penalty	The regularization parameter (maps to lambda).

Value

A parsnip model specification.

See Also

pprf for the underlying training function, pp_tree for single trees

Examples

```
if (requireNamespace("parsnip", quietly = TRUE)) {
  library(parsnip)
  spec <- pp_rand_forest(trees = 50, mtry = 2) %>% set_engine("ppforest2")
  fit <- spec %>% fit(Species ~ ., data = iris)
  predict(fit, iris)
  predict(fit, iris, type = "prob")
}
```

---

pp_tree *Parsnip model specification for pptr.*

Description

Creates a model specification for a single Projection Pursuit decision tree. Use set_engine("ppforest2") to select the ppforest2 engine.

Usage

```
pp_tree(mode = "classification", penalty = NULL)
```

Arguments

mode	A character string for the model type. Either "classification" or "regression".
penalty	The regularization parameter (maps to lambda).

Value

A parsnip model specification.

See Also

pptr for the underlying training function, pp_rand_forest for forests

Examples

```
if (requireNamespace("parsnip", quietly = TRUE)) {
  library(parsnip)
  spec <- pp_tree(penalty = 0) %>% set_engine("ppforest2")
  fit <- spec %>% fit(Species ~ ., data = iris)
  predict(fit, iris)
}
```

---

predict.pprf_classification

Predicts labels or vote proportions from a pprf model (classification mode).

Description

Predicts labels or vote proportions from a pprf model (classification mode).

Usage

```
## S3 method for class 'pprf_classification'
predict(object, new_data = NULL, type = NULL, ...)
```

Arguments

object	A pprf_classification model.
new_data	A data frame or matrix of new observations. If NULL, the first positional argument in ... is used for backward compatibility.
type	The type of prediction: "class" (default) returns a factor of predicted labels, "prob" returns a data frame of vote proportions.
...	For backward compatibility, the first positional argument is treated as new_data when new_data is NULL.

Value

If `type = "class"`, a factor of predicted labels. If `type = "prob"`, a data frame with one column per group, each row summing to 1.

See Also

`pprf`, `predict.pprf_regression`

Examples

```
model <- pprf(Species ~ ., data = iris)
predict(model, iris)
predict(model, iris, type = "prob")
```

---

predict.pprf_regression

Predicts numeric responses from a pprf model (regression mode).

Description

Predicts numeric responses from a pprf model (regression mode).

Usage

```
## S3 method for class 'pprf_regression'
predict(object, new_data = NULL, type = NULL, ...)
```

Arguments

`object`	A `pprf_regression` model.
`new_data`	A data frame or matrix of new observations.
`type`	Must be `"response"` (default).
`...`	For backward compatibility, the first positional argument is treated as `new_data` when `new_data` is `NULL`.

Value

A numeric vector of mean predictions across the forest's trees.

See Also

`pprf`, `predict.pprf_classification`

`predict.pptr_classification` *Predicts labels or per-group one-hot proportions from a pptr model (classification mode).*

Description

Predicts labels or per-group one-hot proportions from a pptr model (classification mode).

Usage

```
## S3 method for class 'pptr_classification'
predict(object, new_data = NULL, type = NULL, ...)
```

Arguments

`object`	A `pptr_classification` model.
`new_data`	A data frame or matrix of new observations. If `NULL`, the first positional argument in `...` is used for backward compatibility.
`type`	`"class"` (default) returns a factor of predicted labels; `"prob"` returns a data frame with 1.0 for the predicted group and 0.0 elsewhere.
`...`	Backward-compat positional ‘new_data‘.

Value

A factor or data frame.

See Also

`pptr`, `predict.pptr_regression`

`predict.pptr_regression` *Predicts numeric responses from a pptr model (regression mode).*

Description

Predicts numeric responses from a pptr model (regression mode).

Usage

```
## S3 method for class 'pptr_regression'
predict(object, new_data = NULL, type = NULL, ...)
```

Arguments

object	A pptr_regression model.
new_data	A data frame or matrix of new observations.
type	Must be "response" (default).
...	Backward-compat positional ‘new_data‘.

Value

A numeric vector.

See Also

pptr, predict.pptr_classification

---

print.pprf *Prints a compact summary of a pprf forest.*

Description

Prints a compact summary of a pprf forest.

Usage

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

Arguments

x	A pprf model.
...	Unused.

Value

Invisibly returns the input pprf model x (unchanged). Called for its side effect of printing a compact description of the forest (call, number of trees, mode, group names, formula) to the console.

See Also

summary.pprf

print.pptr *Prints the structure of a pptr tree.*

Description

Prints the structure of a pptr tree.

Usage

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

Arguments

x	A pptr model.
...	Unused.

Value

Invisibly returns the input pptr model x (unchanged). Called for its side effect of printing the tree structure – the oblique split rules and leaf predictions – to the console.

projection_importance *Projection-coefficient variable importance.*

Description

The projection-based importance (VI2): each split's scaled absolute projection coefficients (‘|a_j| * sigma_j‘) aggregated into a per-feature score, averaged over the non-degenerate trees of a forest. Unlike permuted_importance and weighted_importance, this measure is not OOB-based — it depends only on the fitted projector geometry, is computed eagerly at fit time (cheap), and is available for both single trees (pptr) and forests (pprf).

Usage

```
projection_importance(model)
```

Arguments

model	A pptr or pprf model.

Details

Sign semantics. Entries are non-negative by construction (absolute coefficients scaled by each feature's standard deviation). Rely on the ranking rather than re-normalizing.

Value

A non-negative numeric vector, one entry per feature.

See Also

permuted_importance, weighted_importance

`residuals.ppmodel` *Residuals from a regression ppforest2 model.*

Description

Returns `y - fitted(model)`. Only defined for regression models — classification residuals have no canonical scalar form, so this method errors on classification models rather than inventing a convention.

Usage

```
## S3 method for class 'ppmodel'
residuals(object, ...)
```

Arguments

`object`	A `pprf` or `pptr` regression model.
`...`	Unused.

Value

A numeric vector of length `nobs(object)`.

See Also

fitted.ppmodel

save_json *Save a model to a JSON file.*

Description

Serializes a pptr or pprf model to JSON format compatible with the C++ CLI. The JSON includes the model structure, group labels, training parameters, and optionally variable importance metrics.

Usage

```
save_json(model, path, ...)

## S3 method for class 'ppmodel'
save_json(model, path, include_metrics = TRUE, ...)
```

Arguments

model	A pptr or pprf model (inheriting from ppmodel).
path	File path to write the JSON to.
...	Additional arguments (currently unused).
include_metrics	If TRUE (default), include variable importance and (for forests) OOB error in the output. Set to FALSE to save only the model structure and metadata.

Value

Invisibly returns the file path that was written (a length-one character string). Called for its side effect of serializing model to a JSON file at path.

See Also

load_json, pptr, pprf

Examples

```
model <- pptr(Species ~ ., data = iris, seed = 0)
path <- tempfile(fileext = ".json")
save_json(model, path)
```

stop_any *Composite stopping rule (logical OR).*

Description

Creates a composite stopping rule that fires when any of the child rules fires. Useful for combining multiple criteria, e.g. `stop_any(stop_min_size(5),stop_min_variance(0.01))` for regression.

Usage

```
stop_any(...)
```

Arguments

`...`	Two or more `stop_strategy` objects to combine.

Value

A `stop_strategy` object.

See Also

`stop_min_size`, `stop_min_variance`, `stop_pure_node`

Examples

```
stop_any(stop_min_size(5), stop_min_variance(0.01))
```

stop_max_depth *Maximum-depth stopping rule.*

Description

Creates a stopping rule that stops splitting when a node's depth reaches `max_depth`. Depth is zero-based at the root, so `max_depth(k)` allows at most `k + 1` levels. Mode-agnostic: useful for bounding tree complexity in both classification and regression trees.

Usage

```
stop_max_depth(max_depth)
```

Arguments

`max_depth`	Maximum depth (non-negative integer; 0 produces a root-only stump).

Value

A stop_strategy object.

See Also

stop_min_size, stop_any

Examples

```
stop_max_depth(5)
```

---

stop_min_size *Minimum-size stopping rule.*

Description

Creates a stopping rule that stops splitting when a node has fewer than min_size observations. Used primarily for regression trees.

Usage

```
stop_min_size(min_size = 5L)
```

Arguments

min_size	Minimum node size to allow a split (default: 5).

Value

A stop_strategy object.

See Also

stop_min_variance, stop_any

Examples

```
stop_min_size(5)
```

stop_min_variance *Minimum-variance stopping rule.*

Description

Creates a stopping rule that stops splitting when the within-node response variance falls below `threshold`. Used primarily for regression trees; requires a continuous response.

Usage

```
stop_min_variance(threshold = 0.01)
```

Arguments

`threshold`	Variance threshold below which to stop splitting (default: 0.01).

Value

A `stop_strategy` object.

See Also

`stop_min_size`, `stop_any`

Examples

```
stop_min_variance(0.01)
```

stop_pure_node *Pure-node stopping rule.*

Description

Creates a stopping rule that stops splitting when a node contains only one group. This is the default stopping rule for classification.

Usage

```
stop_pure_node()
```

Value

A `stop_strategy` object.

See Also

pptr, pprf

Examples

```
stop_pure_node()
```

---

summary.pprf *Summary of a pprf forest (shared header + VI).*

Description

Summary of a pprf forest (shared header + VI).

Usage

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

Arguments

object	A pprf model.
...	Unused.

Value

Invisibly returns the input pprf model object (unchanged). Called for its side effect of printing a detailed summary – the training specification, data summary, and variable-importance table (plus, for classification, the training/OOB confusion matrices) – to the console.

---

update.pp_rand_forest *Update a* pp_rand_forest *model specification.*

Description

Implements parsnip's update protocol so that tune::tune_grid() (and any other caller that finalises a spec via update()) can fill in the tuned values for trees, mtry, and penalty. Without this method, update() falls back to stats::update.default and fails with "need an object with call component".

Usage

```
## S3 method for class 'pp_rand_forest'
update(
  object,
  parameters = NULL,
  trees = NULL,
  mtry = NULL,
  mtry_prop = NULL,
  penalty = NULL,
  fresh = FALSE,
  ...
)
```

Arguments

object	A pp_rand_forest model specification.
parameters	A named list of parameters to update (alternative to passing them as args).
trees, mtry, mtry_prop, penalty	New values for the corresponding parameters.
fresh	If TRUE, drop any previously-set arg whose new value is not provided.
...	Engine-specific arguments to update.

Value

An updated pp_rand_forest model specification.

---

update.pp_tree *Update a* pp_tree *model specification.*

Description

Companion to update.pp_rand_forest — exists for the same reason (tune::tune_grid() relies on update() to finalise the spec at each grid point).

Usage

```
## S3 method for class 'pp_tree'
update(object, parameters = NULL, penalty = NULL, fresh = FALSE, ...)
```

Arguments

object	A pp_tree model specification.
parameters	A named list of parameters to update (alternative to passing them as args).
penalty	A new value for the regularisation parameter.
fresh	If TRUE, drop any previously-set arg whose new value is not provided.
...	Engine-specific arguments to update.

Value

An updated pp_tree model specification.

---

vars_all *All-variables selection strategy.*

Description

Creates a variable selection strategy that uses all variables at each split. This is the default for single trees (pptr).

Usage

```
vars_all()
```

Value

A vars_strategy object.

See Also

pptr, vars_uniform, pp_pda, cutpoint_mean_of_means

Examples

```
vars_all()
```

---

vars_uniform *Uniform random variable selection strategy.*

Description

Creates a variable selection strategy that randomly selects a subset of variables at each split. Used with pprf for random forests.

Usage

```
vars_uniform(n_vars = NULL, p_vars = NULL)
```

Arguments

n_vars	The number of variables to consider at each split (integer). Cannot be used together with p_vars.
p_vars	The proportion of variables to consider at each split (number between 0 and 1, exclusive). Resolved to an integer count when the number of features is known. Cannot be used together with n_vars.

Details

Exactly one of `n_vars` or `p_vars` may be specified. When `p_vars` is used, it is stored as-is and resolved to an integer count later by `pprf` once the number of features is known.

Value

A `vars_strategy` object.

See Also

`pprf`, `vars_all`, `pp_pda`, `cutpoint_mean_of_means`

Examples

```
# Select 2 variables at each split
vars_uniform(n_vars = 2)

# Select half the variables at each split
vars_uniform(p_vars = 0.5)

# Use with pprf
pprf(Species ~ ., data = iris, vars = vars_uniform(n_vars = 2))
```

`weighted_importance` *Weighted projection variable importance for a random forest.*

Description

Weights each tree's projection-based importance by a per-tree OOB quality score — '1 - error_rate' in '[0, 1]' for classification, and 'max(0, 1 - NMSE)' in '[0, 1]' for regression — then aggregates 'I_s × |a_j|' over splits. Computed lazily from the training data stored on the model; the result is cached.

Usage

```
weighted_importance(model)
```

Arguments

`model`	A `pprf` forest model.

Details

Sign semantics. Entries are non-negative by construction (weights and per-split contributions are both non-negative). A zero entry means "this feature never appeared in a weighted OOB-contributing split," not "within noise." Contrast with `permuted_importance`, where negative values are meaningful. Do not re-normalize — rely on the ranking.

Value

A non-negative numeric vector, one entry per feature.

`wine` *Wine Dataset*

Description

A dataset containing 178 observations from 3 wine grown cultivares in Italy.

Usage

```
data(wine)
```

Format

A data frame with 178 rows and 14 variables

Details

Type Group variable has 3 groups that are 3 different wine grown cultivars in Italy.

X1 to X13 Check vbles

Index