Package 'classbound'

November 6, 2026

Title Visualization for Classification Decision Boundaries

Version 1.0.1

Description Exploring, visualizing, and comparing classification decision boundaries. Provides a unified interface for fitting classifiers and rendering 2D decision boundary plots, with support for 2D slice visualization (fixing non-plotted dimensions at reference values) and projection-based visualization for high-dimensional data (including Principal Component Analysis (PCA) and tour projections from the 'tourr' package). Supports native 'R' classifiers, 'tidymodels' workflows, and custom user-supplied models via a flexible adapter system. Includes an interactive 'Shiny' application ('explorapp') for visual exploration, data simulation, drawing, model comparison, probability surfaces, and reproducible exports.

License GPL (>= 2)

URL https://github.com/natydasilva/classbound, https://natydasilva.github.io/classbound/

LazyData yes

Depends R (>= 4.3.0)

Imports class, ggplot2, rlang, ggnewscale, shiny, DT, RColorBrewer

Suggests tidymodels, workflowsets, workflows, parsnip, tourr, rpart, roxygen2 (>= 7.3.0), PPtreeViz, PPtreeExt, ppforest2, randomForest, testthat (>= 3.0.0), knitr, rmarkdown, palmerpenguins, e1071, MixSim, MASS, jsonlite, zip

Encoding UTF-8

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Vaibhav Manihar [aut, cre], Natalia da Silva [aut] (ORCID: <https://orcid.org/0000-0002-6031-7451>), Ignacio Alvarez-Castro [aut] (ORCID: <https://orcid.org/0000-0003-1633-2432>)

Maintainer Vaibhav Manihar <vaibhav.manihar@gmail.com>

Repository CRAN

Date/Publication 2026-11-06 12:40:02 UTC

Contents

as_classbound 2
boundary_compute 4
boundary_explore_nd 6
boundary_workflow_set 7
classbound 9
classbound_palette 10
data69_1 11
explorapp 13
export_data_csv 14
export_metrics_csv 14
export_models_rds 15
export_plots_pdf 15
export_plots_png 16
fit_model 16
plot.classbound 18
plot.classbound_boundary 19
plot_boundary 20
predict.classbound 23
predict_model 24
preprocess_data 25
print.classbound 25
simulate_mixsim 26
simu_n 28
summary.classbound 29

Index **30**

`as_classbound` *Convert a fitted model into a classbound object*

Description

Wraps a pre-fitted model in a `classbound` object, adding the feature metadata (names, types, ranges, imputation values) required by `boundary_compute()` and `plot_boundary()`. This is the entry point for "bring your own model" (BYO) workflows, including native `tidymodels` workflows and `parsnip` model fits.

Usage

```
as_classbound(model, data, response = NULL, ...)
```

Arguments

model	A fitted model object. For tidymodels objects, must be a trained workflow or model_fit (not a model specification).
data	A data frame of the training data. Used only to extract feature metadata; the data is not passed through preprocess_data(). Must contain all features that the model was trained on.
response	Optional string naming the response column in data. If provided, this column is excluded from feature metadata and its levels are stored as class levels. If NULL, class levels will be NULL in the returned object.
...	Additional arguments passed to methods.

Details

Unlike fit_model(), as_classbound() does **not** refit the model or call preprocess_data(). Feature metadata is extracted solely from the data argument, which should be the same data used to fit the model. If the model was fitted on scaled or transformed data, ensure data reflects that transformation.

as_classbound() dispatches on the class of model:

- **Default**: wraps any fitted model object (e.g., rpart, lda).
- workflow: requires the workflow to already be trained (via fit()). Use boundary_workflow_set() to train and wrap an entire workflow set at once.
- model_fit: wraps a fitted parsnip model.

Value

A classbound object ready for use with boundary_compute() and plot_boundary().

See Also

fit_model(), boundary_workflow_set(), boundary_compute()

Examples

```
library(palmerpenguins)
data(penguins)
peng_data <- na.omit(penguins[, c("species", "bill_length_mm", "bill_depth_mm")])

# Wrap any pre-fitted model
raw_model <- rpart::rpart(species ~ ., data = peng_data)
cb_model <- as_classbound(raw_model, data = peng_data, response = "species")

# Tidymodels: wrap a fitted workflow
# (boundary_workflow_set() does this automatically for entire workflow sets)
```

boundary_compute *Compute the classification decision boundary*

Description

Generates a 2D grid of predicted class labels (and optionally class probabilities) for a fitted classbound model. The resulting boundary data is stored in the returned object and consumed directly by plot_boundary().

Usage

```
boundary_compute(
  model,
  feature_range = NULL,
  resolution = 100,
  predfun = NULL,
  projection = NULL,
  reference = NULL,
  ...
)
```

Arguments

model	A classbound model object returned by fit_model() or as_classbound(), or a named list of classbound objects for multi-model comparison. All models in a list must share the same training features and class levels. A multi-model result of an earlier boundary_compute() call can also be passed to recompute its boundaries.
feature_range	A named list of length 2 specifying the axis ranges, e.g., list(bill_length_mm = c(30, 60), bill_depth_mm = c(10, 25)). Alternatively, a character vector of exactly two feature names (ranges computed from training data). If NULL and the model has exactly two numeric features, ranges are auto-detected. When projection is provided, this defines the limits of the 2D projected space.
resolution	An integer >= 2 specifying the number of grid points per axis. Higher values produce smoother boundaries. Default is 100; use 50 for faster interactive exploration.
predfun	An optional custom prediction function for non-standard classifiers. Must accept (model, newdata, ...) and return a vector/factor of predicted classes, or a list with $class (factor) and $probs (probability matrix or NULL).
projection	An optional named list defining a high-dimensional projection. Must contain $basis (a numeric matrix, rows = features, columns = 2 axes, must be orthonormal). Optionally contains $center and $scale (numeric vectors of length equal to the number of features) to reverse pre-projection standardization. Only supported for models trained on numeric features.

reference	An optional named list of fixed reference values for features not specified in feature_range (2D slice mode only). Names must match training feature names. If NULL, numeric features are imputed at their median and categorical features at their mode.
...	Additional arguments passed to predict_model().

Details

2D slice (two-feature visualization):

When projection is NULL, boundary_compute() generates a regular grid over the two features named in feature_range. If the model was trained on more than two features, all remaining numeric features are fixed at their training-set median and categorical features at their training-set mode, unless you supply explicit values via reference.

This is a **2D slice** of the full multivariate decision boundary. Two observations that appear in the same region may still be separated in a dimension that is held fixed. Use this mode when you want to examine how two specific features interact, or when your model was trained on exactly two features.

Projection (high-dimensional visualization):

When projection is provided, the 2D grid is generated in the projected space and then **inverse-projected** back to the original feature space before prediction. The grid points therefore lie on a two-dimensional plane in the full feature space: the plane spanned by the two basis columns and passing through center (or the origin if center is NULL). The predicted regions are a **planar section** (a zero-thickness slice) of the full decision boundary along that plane, not a projection of it. The model uses all of its training features, but the region colors are exact only for points on the plane. Observations drawn by plot_boundary() are projected orthogonally onto the plane, so an observation far from the plane may appear in a region that differs from the model's prediction for it; depth fading lowers the opacity of such points.

The projection $basis must be a numeric matrix with:

- row count equal to the number of training features, row names matching feature names
- exactly two columns (the projected axes)
- orthonormal columns (crossprod(basis) must equal diag(2))

Suitable bases can be obtained from prcomp() or the tourr package.

Multi-model comparison:

Pass a named list of classbound objects (all trained on the same features and class levels) to compute boundaries for multiple models at once. The result contains a model column, suitable for use with plot_boundary(facet_col = "model").

Value

A modified classbound object with the additional class "classbound_boundary". The boundary grid is stored in $boundary_data (columns: x, y, prediction, and per-class probability columns when available). For multi-model input, also contains a model column.

See Also

fit_model(), plot_boundary(), boundary_workflow_set()

Examples

```
library(palmerpenguins)
data(penguins)
peng_data <- na.omit(penguins[, c("species", "bill_length_mm", "bill_depth_mm")])

# Fit and compute 2D boundary with auto-detected ranges
m <- fit_model(peng_data, species ~ ., rpart::rpart)
m <- boundary_compute(m, resolution = 50)
head(m$boundary_data)

# Explicit axis ranges
m <- boundary_compute(m,
  feature_range = list(bill_length_mm = c(30, 60), bill_depth_mm = c(10, 25)),
  resolution = 100
)

# 3-feature model visualized as a 2D slice
peng3d <- na.omit(penguins[, c(
  "species", "bill_length_mm",
  "bill_depth_mm", "flipper_length_mm"
)])
m3d <- fit_model(peng3d, species ~ ., rpart::rpart)
m3d_slice <- boundary_compute(m3d,
  feature_range = list(bill_length_mm = c(30, 60), bill_depth_mm = c(10, 25)),
  resolution = 50
)

# 3-feature model visualized via PCA projection
feat_cols <- c("bill_length_mm", "bill_depth_mm", "flipper_length_mm")
pca <- prcomp(peng3d[, feat_cols], scale. = TRUE)
basis <- pca$rotation[, 1:2]
m3d_proj <- boundary_compute(m3d,
  feature_range = list(PC1 = c(-4, 4), PC2 = c(-3, 3)),
  resolution = 50,
  projection = list(basis = basis, center = pca$center, scale = pca$scale)
)
```

---

boundary_explore_nd *Explore high-dimensional classification boundaries*

Description

Generates a high-dimensional uniform domain and identifies points near the decision boundaries of multiple models.

Usage

```
boundary_explore_nd(models, data, n_points = 10000, threshold = 0.05)
```

Arguments

`models`	A named list of fitted `classbound` model objects.
`data`	A data frame representing the original training data. The bounding box of this data determines the simulation domain.
`n_points`	Integer. Number of points to simulate in the high-dimensional space (default: 10000).
`threshold`	Numeric. A percentile threshold (0 to 1) for determining boundary proximity. Points below this advantage percentile are considered boundary-near. Default: 0.05.

Details

This function follows the approach of the 'classifly' package (Wickham and Cook) and Wickham, Cook and Hofmann (2015). Points are simulated uniformly within the bounding box of the training data and classified by each model. For each point, the "advantage" is computed as the difference between the two largest class probabilities. Points whose advantage falls at or below the `threshold` quantile are treated as lying near a decision boundary. When a model does not return class probabilities, the advantage is approximated by the proportion of the 5 nearest simulated neighbors (on range-scaled numeric features) that share the point's predicted class.

Value

A data frame containing only the simulated points that are near the boundary for at least one of the provided models. It has the simulated coordinates, the predicted classes for each model (prefixed with `Y_`), and a logical `is_boundary` column (always `TRUE` in the returned rows).

References

Wickham, H., Cook, D. and Hofmann, H. (2015). Visualizing statistical models: Removing the blindfold. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 8(4), 203–225. doi:10.1002/sam.11271

Wickham, H. and Cook, D. classifly: Explore Classification Models in High Dimensions. R package. https://CRAN.R-project.org/package=classifly

---

`boundary_workflow_set` *Compute classification boundaries for a workflow_set*

Description

A dedicated helper to automatically fit and extract classification boundaries for an entire `workflow_set`. This avoids the need for manual iteration over models.

Usage

```
boundary_workflow_set(
  wf_set,
  data,
  feature_range = NULL,
  response,
  resolution = 100,
  ...
)
```

Arguments

wf_set	A workflow_set object from the workflowsets package.
data	A data frame containing the training data. This is required to extract feature metadata and to fit any workflows that are not yet trained.
feature_range	An optional named list specifying the minimum and maximum values for each feature, or a character vector of feature names. If NULL, the ranges are automatically computed from the training data (if there are exactly 2 numeric features).
response	A string specifying the name of the response column in data.
resolution	An integer specifying the number of points along each axis (default = 100).
...	Additional arguments passed to boundary_compute().

Value

A multi-model classbound object (class "classbound_boundary"), as returned by boundary_compute() for a list of models. The combined grid is in $boundary_data, with a model column indicating the wflow_id.

Examples

```
library(palmerpenguins)
library(workflowsets)
library(parsnip)

data(penguins)
peng_data <- na.omit(penguins[, c("species", "bill_length_mm", "bill_depth_mm")])

# Define multiple engines
spec_rpart <- decision_tree() |>
  set_engine("rpart") |>
  set_mode("classification")
spec_glm <- multinom_reg() |>
  set_engine("nnet") |>
  set_mode("classification")

# Create a workflow set
wf_set <- workflow_set(
  preproc = list(base = species ~ bill_length_mm + bill_depth_mm),
  models = list(tree = spec_rpart, log_reg = spec_glm)
```

```
)

# Compute 2D boundaries for all models simultaneously (auto-range)
bounds <- boundary_workflow_set(wf_set, peng_data, response = "species", resolution = 30)
```

`classbound` *Visualize a classification decision boundary*

Description

A high-level unified wrapper to fit a model, compute its 2D decision boundary, and plot the results in a single step.

Usage

```
classbound(
  data,
  formula,
  classifier,
  interface = c("formula", "matrix", "custom"),
  projection = NULL,
  fit_args = list(),
  predict_args = list(),
  predfun = NULL,
  resolution = 100,
  ...
)
```

Arguments

`data`	A data frame containing the full training dataset. The specific variables used for modeling and plotting are strictly determined by the `formula`.
`formula`	A formula specifying the response and predictors.
`classifier`	The classification function to use (e.g., `rpart::rpart`, `e1071::svm`). This works with any R package classification algorithm. If the classifier uses a non-standard API, you can adapt it via the `predfun` argument.
`interface`	A string specifying how to invoke the classifier: `"formula"`, `"matrix"`, or `"custom"`.
`projection`	An optional list (e.g., `list(basis=..., center=..., scale=...)`) specifying a 2D projection for high-dimensional data.
`fit_args`	A named list of additional arguments passed to the classifier during fitting.
`predict_args`	A named list of additional arguments passed to `predict()` during boundary computation.
`predfun`	A custom function to generate predictions for non-standard models.
`resolution`	An integer specifying the grid resolution for the decision boundary.
`...`	Additional arguments passed to `plot_boundary`.

Value

A ggplot object visualizing the 2D decision boundary and original observations.

Examples

```
library(palmerpenguins)
data(penguins)
peng_data <- na.omit(penguins[, c("species", "bill_length_mm", "bill_depth_mm")])

# Quick 2D boundary visualization for an SVM
classbound(peng_data, species ~ bill_length_mm + bill_depth_mm, e1071::svm)
```

---

classbound_palette *Generate deterministic colors for classification classes*

Description

Returns a named character vector mapping class labels to hex color codes. Colors are assigned deterministically: class labels are sorted alphabetically before assignment, so the same class always receives the same color regardless of the order the classes are supplied.

Usage

```
classbound_palette(classes)
```

Arguments

classes	A character vector or factor containing class labels. Duplicates and NA values are handled gracefully.

Details

The first 20 classes are assigned colors from a curated palette of visually distinct hues designed for use on both light and dark backgrounds. For more than 20 classes, additional colors are generated using the golden-angle method (hue = (index * phi) %% 1 where phi = 0.618...), which produces perceptually distinct colors with uniform spacing in HCL space.

plot_boundary() uses this palette by default. If an RColorBrewer palette is requested via palette = but cannot accommodate the number of classes (e.g., "Dark2" supports a maximum of 8), it automatically falls back to classbound_palette().

Value

A named character vector where names are the unique sorted class labels and values are hex color codes (e.g., "#E6194B").

See Also

plot_boundary()

Examples

```
# 3-class palette
classbound_palette(c("Adelie", "Chinstrap", "Gentoo"))

# Colors are alphabetically ordered, so order of input doesn't matter
identical(
  classbound_palette(c("B", "A", "C")),
  classbound_palette(c("C", "A", "B"))
)

# Use in plot_boundary() via the palette argument

library(palmerpenguins)
data(penguins)
peng_data <- na.omit(penguins[, c("species", "bill_length_mm", "bill_depth_mm")])
m <- fit_model(peng_data, species ~ ., rpart::rpart)
m <- boundary_compute(m, resolution = 50)
plot_boundary(m,
  obs_data = peng_data, x_col = "bill_length_mm",
  y_col = "bill_depth_mm", true_label = "species"
)
```

---

data69_1 *Waveform Dataset (UCI)*

Description

The UCI Waveform Database Generator dataset (Version 1). Contains all 5000 observations with 21 numeric features and a 3-class response variable. All 21 features contribute to class membership, making this a useful dataset for demonstrating high-dimensional boundary visualization, including 2D slice and projection-based approaches.

Usage

```
data(data69_1)
```

Format

A data frame (tibble) with 5000 rows and 22 columns:

Y Class label: numeric, with values `0`, `1`, and `2`

V1 Numeric feature variable

V2 Numeric feature variable

V3 Numeric feature variable

V4 Numeric feature variable

V5 Numeric feature variable

V6 Numeric feature variable

V7 Numeric feature variable

V8 Numeric feature variable

V9 Numeric feature variable

V10 Numeric feature variable

V11 Numeric feature variable

V12 Numeric feature variable

V13 Numeric feature variable

V14 Numeric feature variable

V15 Numeric feature variable

V16 Numeric feature variable

V17 Numeric feature variable

V18 Numeric feature variable

V19 Numeric feature variable

V20 Numeric feature variable

V21 Numeric feature variable

Details

Each observation is generated from one of three waveform classes. All features are numeric. The response variable `Y` is stored as a numeric column with values `0`, `1`, and `2`; convert it with `as.factor()` before fitting a classifier (for example `data69_1$Y <- as.factor(data69_1$Y)`). Load the dataset with `data(data69_1)`.

This dataset is included primarily to demonstrate high-dimensional boundary visualization with tools like PCA projection and `tourr`. For introductory 2D examples, the `palmerpenguins::penguins` dataset provides a more accessible alternative.

Source

https://archive.ics.uci.edu/dataset/107/waveform+database+generator+version+1

Examples

```
data(data69_1)
dim(data69_1) # 5000 x 22
table(data69_1$Y) # classes 0, 1, 2
head(data69_1[, 1:5])
```

`explorapp` *Shiny app to visually explore and compare classification boundaries for built-in, custom, and tidymodels algorithms in 2D*

Description

Shiny app to visually explore and compare classification boundaries for built-in, custom, and tidymodels algorithms in 2D

Usage

```
explorapp(data = NULL, target_col = NULL, custom_models = list())
```

Arguments

`data`	Optional data frame to import directly into the app.
`target_col`	Optional string specifying the column in `data` that contains the true class labels.
`custom_models`	A list of custom models to inject into the app's comparison UI. Each element should be a named list containing at least `fn` (the model fitting function). Optionally, it can contain `args` (a list of arguments to pass to `fn`) and `predict_args` (a function returning a list of arguments for prediction).

Value

No return value, called for side effects. Shinyapp is launched.

Examples

```
if (interactive()) {
  # Launch with default models
  explorapp()

  # Launch with a custom SVM model
  explorapp(custom_models = list(
    "SVM" = list(
      fn = e1071::svm,
      args = list(kernel = "linear")
    )
  ))
}
```

export_data_csv *Export data to CSV*

Description

Export data to CSV

Usage

```
export_data_csv(data, file)
```

Arguments

data	A data frame to export.
file	The file path to write to.

Value

The file path (invisibly).

export_metrics_csv *Export comparison metrics to CSV*

Description

Export comparison metrics to CSV

Usage

```
export_metrics_csv(metrics, file)
```

Arguments

metrics	A data frame of metrics.
file	The file path to write to.

Value

The file path (invisibly).

export_models_rds *Export fitted models to RDS*

Description

Export fitted models to RDS

Usage

```
export_models_rds(models, file)
```

Arguments

models	A list of fitted model objects.
file	The file path to write to.

Value

The file path (invisibly).

export_plots_pdf *Export ggplot objects to a multi-page PDF*

Description

Export ggplot objects to a multi-page PDF

Usage

```
export_plots_pdf(plots, file, width = 8, height = 8)
```

Arguments

plots	A list of ggplot objects.
file	The file path to write to.
width	Width of the PDF in inches.
height	Height of the PDF in inches.

Value

The file path (invisibly).

export_plots_png *Export ggplot objects to individual PNG files in a directory*

Description

Export ggplot objects to individual PNG files in a directory

Usage

```
export_plots_png(plots, dir, dpi = 300, width = 8, height = 8)
```

Arguments

plots	A list of ggplot objects (named or unnamed).
dir	The directory path to write the PNGs to.
dpi	Resolution of the PNG files.
width	Width of each PNG in inches.
height	Height of each PNG in inches.

Value

The directory path (invisibly).

fit_model *Fit a machine learning model*

Description

Fits a classification model and wraps it in a `classbound` object, which carries the feature metadata needed by `boundary_compute()` and `plot_boundary()`.

Usage

```
fit_model(data, formula, classifier, ...)

## Default S3 method:
fit_model(
  data,
  formula,
  classifier,
  interface = c("formula", "matrix", "custom"),
  fit_args = list(),
  ...
)
```

```
## S3 method for class '`function`'
fit_model(
  data,
  formula,
  classifier,
  interface = c("formula", "matrix", "custom"),
  fit_args = list(),
  ...
)

## S3 method for class 'character'
fit_model(
  data,
  formula,
  classifier,
  interface = c("formula", "matrix", "custom"),
  fit_args = list(),
  ...
)
```

Arguments

`data`	A data frame containing the training features and response variable. All columns referenced in `formula` must be present.
`formula`	A formula specifying the response and predictors, e.g., `species ~ bill_length_mm + bill_depth_mm` or `species ~ .` to use all columns.
`classifier`	The classification function or model specification to use. Pass a function (e.g., `rpart::rpart`), a string name (e.g., `"rpart::rpart"`), or a `parsnip` model spec / fitted `workflow`.
`...`	Additional arguments passed to methods.
`interface`	A string specifying how to invoke the classifier: `"formula"` (default), `"matrix"`, or `"custom"`. See Details.
`fit_args`	A named list of additional arguments forwarded to the classifier during fitting (e.g., `list(cp = 0.01)` for `rpart`).

Details

Interface modes:

`fit_model()` supports three calling conventions via the `interface` argument:

- `"formula"` (default): passes `formula` and `data` directly to the classifier. Works for the vast majority of R classifiers (e.g., `rpart::rpart`, `e1071::svm`, `stats::lda`).
- `"matrix"`: constructs a predictor matrix `x` and response vector `y` from the formula, then calls `classifier(x, y, ...)`. Required for classifiers whose primary interface is matrix-based (e.g., `randomForest::randomForest`).
- `"custom"`: passes only `fit_args` to the classifier, giving you full control over the call. Use this for non-standard APIs.

Tidymodels support:

If `classifier` is a `parsnip` model specification (`model_spec`), a fitted `model_fit`, or a `tidymodels` `workflow`, `fit_model()` dispatches to the appropriate method automatically. The `interface` argument is not needed for these objects.

Preprocessing:

`fit_model()` calls `preprocess_data()` internally to coerce response labels to a factor, handle missing values, and extract feature metadata. Do not call `preprocess_data()` manually before calling `fit_model()`; this will corrupt the stored metadata.

Value

A `classbound` object (a list of class `"classbound"`) containing:

- `$fit`: the raw fitted model object returned by the classifier
- `$metadata`: a list with `$features` (names, types, ranges, imputation values) and `$class_levels` (sorted character vector of class labels)
- `$boundary_data`: `NULL` until `boundary_compute()` is called

See Also

`boundary_compute()`, `predict_model()`, `classbound()`

Examples

```
library(palmerpenguins)
data(penguins)
peng_data <- na.omit(penguins[, c("species", "bill_length_mm", "bill_depth_mm")])

# Formula interface (most classifiers)
m_rpart <- fit_model(peng_data, species ~ ., rpart::rpart)

# Matrix interface (randomForest)
m_rf <- fit_model(peng_data, species ~ ., randomForest::randomForest,
  interface = "matrix"
)

# Additional fitting arguments via fit_args
m_rpart_cp <- fit_model(peng_data, species ~ ., rpart::rpart,
  fit_args = list(control = rpart::rpart.control(cp = 0.001))
)
```

---

plot.classbound *Plot a classbound model boundary*

Description

A standard S3 plot method that delegates to `plot_boundary()`.

Usage

```
## S3 method for class 'classbound'
plot(x, ...)
```

Arguments

x	A classbound model object.
...	Additional arguments passed to plot_boundary().

Value

A ggplot2 object.

---

plot.classbound_boundary

Plot a computed boundary

Description

An S3 plot method that delegates to plot_boundary() for objects that have had boundaries computed via boundary_compute().

Usage

```
## S3 method for class 'classbound_boundary'
plot(x, ...)
```

Arguments

x	A classbound_boundary object.
...	Additional arguments passed to plot_boundary().

Value

A ggplot2 object.

plot_boundary *Visualize the classification decision boundary*

Description

Renders a 2D ggplot of a classification decision boundary, optionally overlaying observed training data. Input must be a classbound object that has already been passed through boundary_compute().

Usage

```
plot_boundary(
  model,
  obs_data = NULL,
  x_col = NULL,
  y_col = NULL,
  true_label = NULL,
  facet_col = NULL,
  type = "2D",
  show_gradient = FALSE,
  agree_color = "#006666",
  disagree_color = "#FF8000",
  obs_alpha = 1,
  obs_size = 2.5,
  render = c("raster", "tile"),
  colors = NULL,
  palette = NULL,
  highlight_outliers = FALSE,
  xlim = NULL,
  ylim = NULL,
  ...
)
```

Arguments

model	A classbound object returned by boundary_compute(). Must contain boundary data in $boundary_data.
obs_data	An optional data frame of observations to overlay on the boundary plot. Typically the training data. If provided, x_col, y_col, and true_label are required.
x_col	Column name in obs_data for the x-axis feature. Used as the x-axis label. Required when obs_data is provided and the model has no projection.
y_col	Column name in obs_data for the y-axis feature. Used as the y-axis label. Required when obs_data is provided and the model has no projection.
true_label	Column name in obs_data containing the true class labels. Required when obs_data is provided.

facet_col	Optional string naming a column in the boundary data to facet the plot by. Set to "model" automatically when the input is a multi-model comparison object. Also compatible with any column from boundary_workflow_set() output.
type	The visualization type. "2D" (default) renders the predicted class regions. "disagreement" renders a binary map showing where models agree vs. disagree; requires a multi-model input.
show_gradient	Logical. If TRUE, decision regions are shaded by the predicted class probability (requires a classifier that provides probabilities). Defaults to FALSE.
agree_color	Color for regions where all models agree (only for type = "disagreement").
disagree_color	Color for regions where models disagree (only for type = "disagreement").
obs_alpha	Numeric transparency for overlaid observation points (0.0–1.0). When a projection is active, this is treated as the maximum opacity; actual alpha varies by depth-fading.
obs_size	Numeric point size for overlaid observations.
render	Rendering method for decision regions: "raster" (default, fast, not plotly-compatible) or "tile" (slower, compatible with plotly::ggplotly()).
colors	Optional named character vector mapping class labels to colors, e.g. c("Adelie" = "#E6194B", "Chinstrap" = "#3CB44B", "Gentoo" = "#4363D8"). Overrides both palette and the default classbound_palette().
palette	Optional RColorBrewer palette name (e.g., "Dark2", "Set1") to override the default colors. Falls back to classbound_palette() if the palette cannot support the number of classes.
highlight_outliers	Logical. If TRUE, observations with is_outlier == TRUE in obs_data are rendered as diamonds (shape 23) instead of circles.
xlim	Optional length-2 numeric vector to set x-axis limits.
ylim	Optional length-2 numeric vector to set y-axis limits.
...	Additional arguments (currently unused).

Details

Probability surface (gradient):

When show_gradient = TRUE, decision regions are shaded by the predicted class probability: deep, saturated regions indicate high model confidence, while faded regions indicate uncertainty near the boundary. Probability shading is only possible when the underlying classifier returns class probabilities. Classifiers that return only class labels (e.g., standard SVMs or PPtree models) produce a flat boundary regardless of show_gradient.

High-dimensional projections and depth fading:

When the classbound object contains a projection (from boundary_compute(..., projection = ...)), plot_boundary() automatically forward-projects any obs_data observations onto the 2D plane. It also computes each point's orthogonal distance from the projection plane and maps this distance to opacity: points lying exactly on the plane are fully opaque (alpha = 1.0), while points further away in the original feature space gradually fade toward alpha = 0.2. This depth-fading provides visual cues about how faithfully each point's position is captured by the current projection.

Rendering backend and plotly compatibility:

The default `render = "raster"` uses `ggplot2::geom_raster()`, which is fast and produces high-quality static output. However, `geom_raster()` is not supported by `plotly::ggplotly()` and will produce a blank interactive plot. To convert a boundary plot to an interactive plotly figure, use `render = "tile"` instead:

```
p <- plot_boundary(model, render = "tile")
plotly::ggplotly(p)
```

Colors and palettes:

By default, `plot_boundary()` uses `classbound_palette()`, a curated 20-color palette with deterministic (alphabetical) class-to-color assignment, ensuring consistent colors across multiple plots. Supply a `palette` name (e.g., `"Dark2"`) to use an RColorBrewer palette, or supply `colors` as a named vector for explicit control. If an RColorBrewer palette cannot accommodate the number of classes, it falls back to `classbound_palette()`.

Value

A `ggplot2` object.

See Also

`boundary_compute()`, `classbound()`, `classbound_palette()`

Examples

```
library(palmerpenguins)
data(penguins)
peng_data <- na.omit(penguins[, c("species", "bill_length_mm", "bill_depth_mm")])

m <- fit_model(peng_data, species ~ ., rpart::rpart)
m <- boundary_compute(m, resolution = 50)

# Basic boundary plot with observations
plot_boundary(m,
  obs_data   = peng_data,
  x_col      = "bill_length_mm",
  y_col      = "bill_depth_mm",
  true_label = "species"
)

# Probability gradient (rpart supports probabilities)
plot_boundary(m,
  obs_data      = peng_data,
  x_col         = "bill_length_mm",
  y_col         = "bill_depth_mm",
  true_label    = "species",
  show_gradient = TRUE
)

# Plotly-compatible rendering
p <- plot_boundary(m,
```

```
  obs_data   = peng_data,
  x_col      = "bill_length_mm",
  y_col      = "bill_depth_mm",
  true_label = "species",
  render     = "tile"
)
# plotly::ggplotly(p)  # uncomment to convert to interactive
```

`predict.classbound` *Predict using a classbound model*

Description

Generates predictions using a unified interface across all classifiers. Dispatches natively on objects of class `"classbound"`. Attempting to call `predict()` directly on a `"classbound_multi"` object will result in an error, as multi-model boundaries are evaluated internally by `boundary_compute()`.

Usage

```
## S3 method for class 'classbound'
predict(object, newdata, predict_args = list(), predfun = NULL, ...)

## S3 method for class 'classbound_multi'
predict(object, newdata, predict_args = list(), predfun = NULL, ...)
```

Arguments

`object`	A fitted classbound model. This is the object returned by `fit_model()` or `classbound()`.
`newdata`	A data frame of new observations to predict on.
`predict_args`	A named list of additional arguments passed to `predict_adapter`.
`predfun`	A custom function to generate predictions for non-standard models. The function must accept at least two arguments: `model` (the fitted native model) and `newdata` (a data frame of new observations). It should return either a vector/factor of predicted classes, or a list containing `class` (predicted labels) and `probs` (a probability matrix).
`...`	Additional arguments passed to the specific model adapter.

Value

A list containing `class` (a factor of predicted labels) and `probs` (a probability matrix, or strictly `NULL` if the classifier lacks probability support). Downstream functions like `boundary_compute()` are designed to handle `probs = NULL` gracefully.

predict_model *Predict using a fitted classbound model*

Description

Generates predictions using a unified interface across all classifiers. This function is a compatibility wrapper around the standard `predict()` method for "classbound" objects.

Usage

```
predict_model(model, newdata, predict_args = list(), predfun = NULL, ...)
```

Arguments

`model`	A fitted classbound model. This corresponds to the `object` argument used by the standard R `predict()` generic. This wrapper calls `predict(model, ...)`.
`newdata`	A data frame of new observations to predict on.
`predict_args`	A named list of additional arguments passed to `predict_adapter`.
`predfun`	A custom function to generate predictions for non-standard models. The function must accept at least two arguments: `model` (the fitted native model) and `newdata` (a data frame of new observations). It should return either a vector/factor of predicted classes, or a list containing `class` (predicted labels) and `probs` (a probability matrix).
`...`	Additional arguments passed to the specific model adapter.

Value

A list containing `class` (a factor of predicted labels) and `probs` (a probability matrix, or strictly `NULL` if the classifier lacks probability support).

Examples

```
library(palmerpenguins)
data(penguins)
peng_data <- na.omit(penguins[, c("species", "bill_length_mm", "bill_depth_mm")])

m_rpart <- fit_model(peng_data, species ~ ., rpart::rpart)
preds <- predict_model(m_rpart, newdata = peng_data[1:5, ])
```

preprocess_data *Preprocess data before model fitting*

Description

Applies standard preprocessing steps to a training dataset: validates structure, coerces character columns to factors, drops unused factor levels, rejects missing and infinite values, and converts response labels to a factor.

Usage

```
preprocess_data(data, labels = NULL, ...)
```

Arguments

data	A data frame of raw training data.
labels	Optional vector of target labels. Converted to a factor with unused levels dropped.
...	Additional arguments (currently unused).

Details

This function is called automatically by `fit_model()`. **Do not call it manually before** `fit_model()`, because doing so will result in feature metadata being extracted from the pre-processed data rather than the original data, which corrupts the imputation values used by `boundary_compute()` for 2D slicing.

It is exported for use in custom workflows and testing, but it is not typically needed in the standard `fit_model()` → `boundary_compute()` pipeline.

Value

A list with two elements:

- `$data`: the preprocessed data frame
- `$labels`: the processed factor labels (or `NULL` if not supplied)

print.classbound *Print a classbound model*

Description

Prints a clean summary of a classbound model object, hiding the internal wrapper structure. For single models, it delegates to the native model's print method. For multi-model objects (`classbound_multi`), it prints a comparison summary of the included models.

Usage

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

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

Arguments

x	A classbound or classbound_multi object.
...	Additional arguments passed to the native model's print method.

Value

The object, invisibly.

---

simulate_mixsim *Simulate data using Gaussian mixture models*

Description

Generates synthetic classification data using Gaussian mixture models constructed by the MixSim package. Useful for creating controlled datasets to explore decision boundary behavior when no real dataset is available.

Usage

```
simulate_mixsim(
  n,
  K,
  p,
  MaxOmega,
  class_names = NULL,
  seed = NULL,
  noise_ratio = 0,
  test_ratio = 0
)
```

Arguments

n	Number of training observations to generate.
K	Number of classes.
p	Number of numeric features (dimensions).
MaxOmega	Maximum pairwise overlap between mixture components (0 to 1). Smaller values create better-separated classes.

class_names	Optional character vector of class labels (length K). Defaults to "Class 1", "Class 2", etc.
seed	Optional integer for reproducibility. The global random seed is restored after the call.
noise_ratio	Numeric in [0, 1). Proportion of n to add as uniform background noise (randomly labeled). Use this to test classifier robustness to contamination.
test_ratio	Numeric in [0, 1). If greater than 0, generates an additional independent test dataset of size round(n * test_ratio) using the same mixture parameters. This is not a split of the training data.

Details

Class distributions are randomly generated subject to the MaxOmega overlap constraint. The simulation is fully reproducible when seed is supplied.

Test data:

When test_ratio > 0, an additional independent dataset is generated using the **same mixture parameters** but a fresh random draw. This is not a split of the training data: the training set has n observations and the test set has round(n * test_ratio) independently generated observations. The two sets are statistically independent, so the test set is a fair evaluation sample.

Value

If test_ratio == 0 (default): a data frame with a Sim class column and p feature columns (X1, X2, ...). If test_ratio > 0: a list with $train and $test data frames.

See Also

simu_n()

Examples

```
# Generate a 3-class, 2-dimensional training dataset
train_df <- simulate_mixsim(n = 200, K = 3, p = 2, MaxOmega = 0.05, seed = 42)
head(train_df)

# Generate training and independent test data
sim <- simulate_mixsim(
  n = 200, K = 3, p = 2, MaxOmega = 0.05,
  seed = 42, test_ratio = 0.3
)
nrow(sim$train) # 200
nrow(sim$test) # 60 (independently generated, not split from train)
```

simu_n *Simulate multivariate normal data for multiple classes*

Description

Generates synthetic classification data by drawing independent multivariate normal samples for each class. Useful when you want explicit control over each class's mean and covariance structure.

Usage

```
simu_n(
  means,
  covs,
  ns,
  class_names = NULL,
  seed = NULL,
  noise_ratio = 0,
  test_ratio = 0
)
```

Arguments

means	A list of numeric vectors, one per class, specifying the class means. Each vector must have length equal to the number of features.
covs	A list of covariance matrices, one per class.
ns	A numeric vector of sample sizes, one per class.
class_names	Optional character vector of class labels (length equal to length(ns)). Defaults to "Class 1", "Class 2", etc.
seed	Optional integer for reproducibility. The global random seed is restored after the call.
noise_ratio	Numeric in [0, 1). Proportion of sum(ns) to add as uniform background noise (randomly labeled).
test_ratio	Numeric in [0, 1). If greater than 0, generates an additional independent test dataset. This is not a split of the training data.

Details

Test data:
When test_ratio > 0, an additional independent test dataset is generated by drawing fresh samples of size round(ns * test_ratio) for each class using the same means and covs. This is not a split of the training data. The training set has sum(ns) observations; the test set has sum(round(ns * test_ratio)) independently generated observations.

Value

If test_ratio == 0 (default): a data frame with a Sim class column and feature columns (X1, X2, ...). If test_ratio > 0: a list with $train and $test data frames.

See Also

simulate_mixsim()

Examples

```
means <- list(c(0, 0), c(3, 3), c(0, 5))
covs <- list(diag(2), diag(2), diag(2))
ns <- c(60, 60, 60)

train_df <- simu_n(means, covs, ns, seed = 1)
head(train_df)

# With independent test data
sim <- simu_n(means, covs, ns, seed = 1, test_ratio = 0.3)
nrow(sim$train) # 180
nrow(sim$test) # 54 (independently generated, not split from train)
```

---

summary.classbound *Summarize a classbound model*

Description

For single models, delegates the summary function directly to the native model wrapped inside the classbound object. For multi-model objects (classbound_multi), it returns a summary of the contained models.

Usage

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

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

Arguments

object	A classbound or classbound_multi object.
...	Additional arguments passed to the native model's summary method.

Value

The summary of the native model.

Index

∗ **datasets**
    data69_1, 11

as_classbound, 2

boundary_compute, 4
boundary_compute(), *3*, *18*, *22*
boundary_explore_nd, 6
boundary_workflow_set, 7
boundary_workflow_set(), *3*, *5*

classbound, 9
classbound(), *18*, *22*
classbound_palette, 10
classbound_palette(), *22*

data69_1, 11

explorapp, 13
export_data_csv, 14
export_metrics_csv, 14
export_models_rds, 15
export_plots_pdf, 15
export_plots_png, 16

fit_model, 16
fit_model(), *3*, *5*

plot.classbound, 18
plot.classbound_boundary, 19
plot_boundary, *9*, 20
plot_boundary(), *5*, *11*
predict.classbound, 23
predict.classbound_multi
        *(*predict.classbound*)*, 23
predict_model, 24
predict_model(), *18*
preprocess_data, 25
print.classbound, 25
print.classbound_multi
        *(*print.classbound*)*, 25

simu_n, 28
simu_n(), *27*
simulate_mixsim, 26
simulate_mixsim(), *29*
summary.classbound, 29
summary.classbound_multi
        *(*summary.classbound*)*, 29