Package 'parttree'

January 15, 2026

Title Visualize Simple 2-D Decision Tree Partitions

Version 0.1.2

Date 2026-10-14

Description Visualize the partitions of simple decision trees, involving one or
two predictors, on the scale of the original data. Provides an intuitive
alternative to traditional tree diagrams, by visualizing how a decision tree
divides the predictor space in a simple 2D plot alongside the original data.
The 'parttree' package supports both classification and regression trees
from 'rpart' and 'partykit', as well as trees produced by popular frontend
systems like 'tidymodels' and 'mlr3'. Visualization methods are provided for
both base R graphics and 'ggplot2'.

License MIT + file LICENSE

Encoding UTF-8

URL <https://grantmcdermott.com/parttree/>

BugReports <https://github.com/grantmcdermott/parttree/issues>

Imports graphics, stats, data.table, partykit, rlang, rpart, tinyplot
(>= 0.3.0)

Enhances ggplot2 (>= 3.4.0)

Suggests tinytest, tinysnapshot (>= 0.0.3), fontquiver, rsvg, svglite
(>= 2.2.0), palmerpenguins, titanic, mlr3, parsnip, workflows,
magick, imager, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.3.3

NeedsCompilation no

Author Grant McDermott [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7883-8573>),
Achim Zeileis [ctb] (ORCID: <https://orcid.org/0000-0003-0918-3766>),
Brian Heseung Kim [ctb] (ORCID:
<https://orcid.org/0000-0002-7988-1802>),
Julia Silge [ctb] (ORCID: <https://orcid.org/0000-0002-3671-836X>)

Maintainer Grant McDermott <gmcd@amazon.com>

Depends R (>= 4.1.0)

Repository CRAN

Date/Publication 2026-01-15 09:20:02 UTC

Contents

---

geom_parttree	*Visualize tree partitions with ggplot2*

Description

geom_parttree() is a simple wrapper around parttree() that takes a tree model object and then converts into an amenable data frame that ggplot2 knows how to plot. Please note that ggplot2 is not a hard dependency of parttree and must thus be installed separately on the user's system before calling geom_parttree.

Usage

```
geom_parttree(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  flip = FALSE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	An rpart::rpart.object or an object of compatible type (e.g. a decision tree constructed via the partykit, tidymodels, or mlr3 front-ends).

stat   The statistical transformation to use on the data for this layer. When using a
                    geom_*() function to construct a layer, the stat argument can be used to over-
                    ride the default coupling between geoms and stats. The stat argument accepts
                    the following:

- A Stat ggproto subclass, for example StatCount.
- A string naming the stat. To give the stat as a string, strip the function name
  of the stat_ prefix. For example, to use stat_count(), give the stat as
  "count".
- For more information and other ways to specify the stat, see the layer stat
  documentation.

position   A position adjustment to use on the data for this layer. This can be used in
                    various ways, including to prevent overplotting and improving the display. The
                    position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This
  method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string,
  strip the function name of the position_ prefix. For example, to use
  position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer
  position documentation.

linejoin   Line join style (round, mitre, bevel).

na.rm   If FALSE, the default, missing values are removed with a warning. If TRUE,
                    missing values are silently removed.

show.legend   logical. Should this layer be included in the legends? NA, the default, includes if
                    any aesthetics are mapped. FALSE never includes, and TRUE always includes. It
                    can also be a named logical vector to finely select the aesthetics to display. To
                    include legend keys for all levels, even when no data exists, use TRUE. If NA, all
                    levels are shown in legend, but unobserved levels are omitted.

inherit.aes   If FALSE, overrides the default aesthetics, rather than combining with them.
                    This is most useful for helper functions that define both data and aesthetics and
                    shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

flip   Logical. By default, the "x" and "y" axes variables for plotting are determined by
                    the first split in the tree. This can cause plot orientation mismatches depending
                    on how users specify the other layers of their plot. Setting to TRUE will flip the
                    "x" and "y" variables for the geom_parttree layer.

...   Other arguments passed on to layer()'s params argument. These arguments
                    broadly fall into one of 4 categories below. Notably, further arguments to the
                    position argument, or aesthetics that are required can *not* be passed through
                    .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and
  apply to the layer as a whole. For example, colour = "red" or linewidth
  = 3. The geom's documentation has an **Aesthetics** section that lists the
  available options. The 'required' aesthetics cannot be passed on to the
  params. Please note that while passing unmapped aesthetics as vectors is
  technically possible, the order and required length is not guaranteed to be
  parallel to the input data.

- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of [layer()](#) may also be passed on through .... This can be one of the functions described as [key glyphs](#), to change the display of the layer in the legend.

Details

Because of the way that ggplot2 validates inputs and assembles plot layers, note that the data input for geom_parttree() (i.e. decision tree object) must assigned in the layer itself; not in the initialising [ggplot2::ggplot()](#) call. See Examples.

Value

A [ggplot](#) layer.

Aesthetics

geom_parttree() aims to "work-out-of-the-box" with minimal input from the user's side, apart from specifying the data object. This includes taking care of the data transformation in a way that, generally, produces optimal corner coordinates for each partition (i.e. xmin, xmax, ymin, and ymax). However, it also understands the following aesthetics that users may choose to specify manually:

- fill (particularly encouraged, since this will provide a visual cue regarding the prediction in each partition region)
- colour
- alpha
- linetype
- size

See Also

[plot.parttree()](#), which provides an alternative plotting method using base R graphics.

Examples

```
# install.packages("ggplot2")
library(ggplot2)  # ggplot2 must be installed/loaded separately

library(parttree) # this package
library(rpart)    # decision trees


#
```

```
## Simple decision tree (max of two predictor variables)

iris_tree = rpart(Species ~ Petal.Length + Petal.Width, data=iris)

# Plot with original iris data only
p = ggplot(data = iris, aes(x = Petal.Length, y = Petal.Width)) +
  geom_point(aes(col = Species))

# Add tree partitions to the plot (borders only)
p + geom_parttree(data = iris_tree)

# Better to use fill and highlight predictions
p + geom_parttree(data = iris_tree, aes(fill = Species), alpha=0.1)

# To drop the black border lines (i.e. fill only)
p + geom_parttree(data = iris_tree, aes(fill = Species), col = NA, alpha = 0.1)

#
## Example with plot orientation mismatch

p2 = ggplot(iris, aes(x=Petal.Width, y=Petal.Length)) +
  geom_point(aes(col=Species))

# Oops
p2 + geom_parttree(data = iris_tree, aes(fill=Species), alpha = 0.1)

# Fix with 'flip = TRUE'
p2 + geom_parttree(data = iris_tree, aes(fill=Species), alpha = 0.1, flip = TRUE)

#
## Various front-end frameworks are also supported, e.g.:

# install.packages("parsnip")
library(parsnip)

iris_tree_parsnip = decision_tree() |>
  set_engine("rpart") |>
  set_mode("classification") |>
  fit(Species ~ Petal.Length + Petal.Width, data=iris)

p + geom_parttree(data = iris_tree_parsnip, aes(fill=Species), alpha = 0.1)

#
## Trees with continuous independent variables are also supported.

# Note: you may need to adjust (or switch off) the fill legend to match the
# original data, e.g.:

iris_tree_cont = rpart(Petal.Length ~ Sepal.Length + Petal.Width, data=iris)
p3 = ggplot(data = iris, aes(x = Petal.Width, y = Sepal.Length)) +
 geom_parttree(
   data = iris_tree_cont,
   aes(fill = Petal.Length), alpha=0.5
```

```
  ) +
geom_point(aes(col = Petal.Length)) +
theme_minimal()

# Legend scales don't quite match here:
p3

# Better to scale fill to the original data
p3 + scale_fill_continuous(limits = range(iris$Petal.Length))
```

---

parttree   *Convert a decision tree into a data frame of partition coordinates*

Description

Extracts the terminal leaf nodes of a decision tree that contains no more that two numeric predictor variables. These leaf nodes are then converted into a data frame, where each row represents a partition (or leaf or terminal node) that can easily be plotted in 2-D coordinate space.

Usage

```
parttree(tree, keep_as_dt = FALSE, flip = FALSE)
```

Arguments

tree	An [rpart.object](#) or alike. This includes compatible classes from the mlr3 and tidymodels frontends, or the constparty class inheriting from [party](#).
keep_as_dt	Logical. The function relies on data.table for internal data manipulation. But it will coerce the final return object into a regular data frame (default behavior) unless the user specifies TRUE.
flip	Logical. Should we flip the "x" and "y" variables in the return data frame? The default behaviour is for the first split variable in the tree to take the "y" slot, and any second split variable to take the "x" slot. Setting to TRUE switches these around.
	Note: This argument is primarily useful when it passed via [geom_parttree](#) to ensure correct axes orientation as part of a ggplot2 visualization (see [geom_parttree](#) Examples). We do not expect users to call parttree(..., flip = TRUE) directly. Similarly, to switch axes orientation for the native (base graphics) [plot.parttree](#) method, we recommend calling plot(..., flip = TRUE) rather than flipping the underlying parttree object.

Value

A data frame comprising seven columns: the leaf node, its path, a set of rectangle limits (i.e., xmin, xmax, ymin, ymax), and a final column corresponding to the predicted value for that leaf.

See Also

Examples

```
library("parttree")

#
## rpart trees

library("rpart")
rp = rpart(Kyphosis ~ Start + Age, data = kyphosis)

# A parttree object is just a data frame with additional attributes
(rp_pt = parttree(rp))
attr(rp_pt, "parttree")

# simple plot
plot(rp_pt)

# removing the (recursive) partition borders helps to emphasise overall fit
plot(rp_pt, border = NA)

# customize further by passing extra options to (tiny)plot
plot(
    rp_pt,
    border  = NA,                              # no partition borders
    pch     = 16,                              # filled points
    alpha   = 0.6,                             # point transparency
    grid    = TRUE,                            # background grid
    palette = "classic",                       # new colour palette
    xlab    = "Topmost vertebra operated on",  # custom x title
    ylab    = "Patient age (months)",          # custom y title
    main    = "Tree predictions: Kyphosis recurrence" # custom title
)

#
## conditional inference trees from partyit

library("partykit")
ct = ctree(Species ~ Petal.Length + Petal.Width, data = iris)
ct_pt = parttree(ct)
plot(ct_pt, pch = 19, palette = "okabe", main = "ctree predictions: iris species")

## rpart via partykit
rp2 = as.party(rp)
parttree(rp2)

#
## various front-end frameworks are also supported, e.g.

# tidymodels
```

```
# install.packages("parsnip")
library(parsnip)

decision_tree() |>
  set_engine("rpart") |>
  set_mode("classification") |>
  fit(Species ~ Petal.Length + Petal.Width, data=iris) |>
  parttree() |>
  plot(main = "This time brought to you via parsnip...")

# mlr3 (NB: use `keep_model = TRUE` for mlr3 learners)

# install.packages("mlr3")
library(mlr3)

task_iris = TaskClassif$new("iris", iris, target = "Species")
task_iris$formula(rhs = "Petal.Length + Petal.Width")
fit_iris = lrn("classif.rpart", keep_model = TRUE) # NB!
fit_iris$train(task_iris)
plot(parttree(fit_iris), main = "... and now mlr3")
```

---

plot.parttree   *Plot decision tree partitions*

Description

Provides a plot method for parttree objects.

Usage

```
## S3 method for class 'parttree'
plot(
  x,
  raw = TRUE,
  border = "black",
  fill_alpha = 0.3,
  expand = TRUE,
  jitter = FALSE,
  add = FALSE,
  ...
)
```

Arguments

x   A [parttree](#) data frame.

raw   Logical. Should the raw (original) data points be plotted too? Default is TRUE.

border	Colour of the partition borders (edges). Default is "black". To remove the borders altogether, specify as NA.
fill_alpha	Numeric in the range [0,1]. Alpha transparency of the filled partition rectangles. Default is 0.3.
expand	Logical. Should the partition limits be expanded to to meet the edge of the plot axes? Default is TRUE. If FALSE, then the partition limits will extend only until the range of the raw data.
jitter	Logical. Should the raw points be jittered? Default is FALSE. Only evaluated if raw = TRUE.
add	Logical. Add to an existing plot? Default is FALSE.
...	Additional arguments passed down to tinyplot.

Value

No return value, called for side effect of producing a plot.

No return value; called for its side effect of producing a plot.

Examples

```
library("parttree")

#
## rpart trees

library("rpart")
rp = rpart(Kyphosis ~ Start + Age, data = kyphosis)

# A parttree object is just a data frame with additional attributes
(rp_pt = parttree(rp))
attr(rp_pt, "parttree")

# simple plot
plot(rp_pt)

# removing the (recursive) partition borders helps to emphasise overall fit
plot(rp_pt, border = NA)

# customize further by passing extra options to (tiny)plot
plot(
  rp_pt,
  border  = NA,                              # no partition borders
  pch     = 16,                              # filled points
  alpha   = 0.6,                             # point transparency
  grid    = TRUE,                            # background grid
  palette = "classic",                       # new colour palette
  xlab    = "Topmost vertebra operated on",  # custom x title
  ylab    = "Patient age (months)",          # custom y title
  main    = "Tree predictions: Kyphosis recurrence" # custom title
)
```

```
#
## conditional inference trees from partyit

library("partykit")
ct = ctree(Species ~ Petal.Length + Petal.Width, data = iris)
ct_pt = parttree(ct)
plot(ct_pt, pch = 19, palette = "okabe", main = "ctree predictions: iris species")

## rpart via partykit
rp2 = as.party(rp)
parttree(rp2)

#
## various front-end frameworks are also supported, e.g.

# tidymodels

# install.packages("parsnip")
library(parsnip)

decision_tree() |>
  set_engine("rpart") |>
  set_mode("classification") |>
  fit(Species ~ Petal.Length + Petal.Width, data=iris) |>
  parttree() |>
  plot(main = "This time brought to you via parsnip...")

# mlr3 (NB: use `keep_model = TRUE` for mlr3 learners)

# install.packages("mlr3")
library(mlr3)

task_iris = TaskClassif$new("iris", iris, target = "Species")
task_iris$formula(rhs = "Petal.Length + Petal.Width")
fit_iris = lrn("classif.rpart", keep_model = TRUE) # NB!
fit_iris$train(task_iris)
plot(parttree(fit_iris), main = "... and now mlr3")
```

Index