Package ‘rules’

March 9, 2023

Title Model Wrappers for Rule-Based Models

Version 1.0.2

Description Bindings for additional models for use with the 'parsnip' package. Models include prediction rule ensembles (Friedman and Popescu, 2008) <doi:10.1214/07-AOAS148>, C5.0 rules (Quinlan, 1992 ISBN: 1558602380), and Cubist (Kuhn and Johnson, 2013) <doi:10.1007/978-1-4614-6849-3>.

License MIT + file LICENSE

URL https://github.com/tidymodels/rules, https://rules.tidymodels.org/

BugReports https://github.com/tidymodels/rules/issues

Depends parsnip (>= 0.2.1.9004), R (>= 3.4)

Imports dials (>= 0.1.1.9001), dplyr, generics (>= 0.1.0), purrr, rlang, stats, stringr, tidyr, utils

Suggests C50, covr, Cubist, knitr, modeldata, recipes, rmarkdown, spelling, testthat (>= 3.0.0), tibble, xrf (>= 0.2.0)

Config/Needs/website tidyr, tidyverse/tidytemplate, recipes, xrf

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

NeedsCompilation no

Author Emil Hvitfeldt [aut, cre] (<https://orcid.org/0000-0002-0679-1945>), Max Kuhn [aut] (<https://orcid.org/0000-0003-2402-136X>), Posit Software, PBC [cph, fnd]

Maintainer Emil Hvitfeldt <emil.hvitfeldt@posit.co>

Repository CRAN

Date/Publication 2023-03-08 23:40:02 UTC

R topics documented:

`committees` *Parameter functions for Cubist models*

Description

Committee-based models enact a boosting-like procedure to produce ensembles. `committees` parameter is for the number of models in the ensembles while `max_rules` can be used to limit the number of possible rules.

Usage

```
committees(range = c(1L, 100L), trans = NULL)

max_rules(range = c(1L, 500L), trans = NULL)
```

Arguments

`range`	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively.
`trans`	A `trans` object from the `scales` package, such as `scales::log10_trans()` or `scales::reciprocal_trans()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Value

A function with classes "quant_param" and "param"

Examples

```
committees()
committees(4:5)

max_rules()
```

multi_predict._cubist   multi_predict() *methods for rule-based models*

Description

multi_predict() methods for rule-based models

Usage

```
## S3 method for class '`_cubist`'
multi_predict(object, new_data, type = NULL, neighbors = NULL, ...)

## S3 method for class '`_xrf`'
multi_predict(object, new_data, type = NULL, penalty = NULL, ...)
```

Arguments

object	A model_fit object.
new_data	A rectangular data object, such as a data frame.
type	A single character value or NULL. This argument is ignored in the method for _cubist objects and is handled internally (since type = "numeric" is always used).
neighbors	A numeric vector of neighbors values between zero and nine.
...	Not currently used.
penalty	Non-negative penalty values.

tidy.C5.0   *Turn C5.0 and rule-based models into tidy tibbles*

Description

Turn C5.0 and rule-based models into tidy tibbles

Usage

```
## S3 method for class 'C5.0'
tidy(x, trees = x$trials["Actual"], ...)

## S3 method for class 'cubist'
tidy(x, committees = x$committee, ...)

## S3 method for class 'xrf'
tidy(x, penalty = NULL, unit = c("rules", "columns"), ...)
```

Arguments

x	A Cubist, C5.0, or xrf object.
trees	The number of boosting iterations to tidy (defaults to the entire ensemble).
...	Not currently used.
committees	The number of committees to tidy (defaults to the entire ensemble).
penalty	A single numeric value for the lambda penalty value.
unit	What data should be returned? For unit = 'rules', each row corresponds to a rule. For unit = 'columns', each row is a predictor column. The latter can be helpful when determining variable importance.

Details

The outputs for these tidy functions are different since the model structures are different.

Let's look at Cubist and RuleFit first, using the Ames data, then C5.0 with a different data set.

An example using the Ames data:

First we will fit a Cubist model and tidy it:

```
library(tidymodels)
library(rules)
library(rlang)

data(ames, package = "modeldata")

ames <- ames %>%
  mutate(Sale_Price = log10(Sale_Price)) %>%
  select(Sale_Price, Longitude, Latitude, Central_Air)

cb_fit <-
  cubist_rules(committees = 10) %>%
  set_engine("Cubist") %>%
  fit(Sale_Price ~ ., data = ames)

cb_res <- tidy(cb_fit)

cb_res

## # A tibble: 223 x 5
##    committee rule_num rule                                     estimate statistic
##        <int>    <int> <chr>                                    <list>   <list>
##  1         1        1 ( Central_Air == 'N' ) & ( Latitude <=~ <tibble> <tibble>
##  2         1        2 ( Latitude <= 41.992611 ) & ( Latitude~ <tibble> <tibble>
##  3         1        3 ( Central_Air == 'N' ) & ( Latitude > ~ <tibble> <tibble>
##  4         1        4 ( Latitude <= 42.026997 ) & ( Longitud~ <tibble> <tibble>
##  5         1        5 ( Longitude > -93.63002 ) & ( Latitude~ <tibble> <tibble>
##  6         1        6 ( Latitude <= 42.035858 ) & ( Longitud~ <tibble> <tibble>
##  7         1        7 ( Latitude <= 42.024029 ) & ( Latitude~ <tibble> <tibble>
```

```
## 8       1       8 ( Longitude > -93.602348 ) & ( Latitud~ <tibble> <tibble>
## 9       1       9 ( Latitude <= 41.991756 ) & ( Longitud~ <tibble> <tibble>
## 10      1      10 ( Latitude > 42.041813 ) & ( Longitude~ <tibble> <tibble>
## # ... with 213 more rows
```

Since Cubist fits linear regressions within the data from each rule, the coefficients are in the `estimate` column and other information are in `statistic`:

```
cb_res$estimate[[1]]
```

```
## # A tibble: 3 x 2
##   term        estimate
##   <chr>          <dbl>
## 1 (Intercept)  -509.
## 2 Longitude      -5.05
## 3 Latitude        0.99
```

```
cb_res$statistic[[1]]
```

```
## # A tibble: 1 x 6
##   num_conditions coverage  mean   min   max error
##            <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl>
## 1              3       38  4.87  4.12  5.22 0.149
```

Note that we can get the data for this rule by using `rlang::parse_expr()` with it:

```
rule_1_expr <- parse_expr(cb_res$rule[1])
rule_1_expr
```

```
## (Central_Air == "N") & (Latitude <= 42.026997) & (Longitude >
##     -93.639572)
```

then use it to get the data back:

```
filter(ames, !!rule_1_expr)
```

```
## # A tibble: 38 x 4
##    Sale_Price Longitude Latitude Central_Air
##         <dbl>     <dbl>    <dbl> <fct>
##  1       5.04     -93.6     42.0 N
##  2       4.74     -93.6     42.0 N
##  3       4.75     -93.6     42.0 N
##  4       4.54     -93.6     42.0 N
##  5       4.64     -93.6     42.0 N
##  6       5.22     -93.6     42.0 N
##  7       4.80     -93.6     42.0 N
##  8       4.99     -93.6     42.0 N
##  9       5.09     -93.6     42.0 N
## 10       4.89     -93.6     42.0 N
## # ... with 28 more rows
```

Now let's fit a RuleFit model. First, we'll use a recipe to convert the Central Air predictor to an indicator:

```
xrf_reg_mod <-
  rule_fit(trees = 3, penalty = .001) %>%
  set_engine("xrf") %>%
  set_mode("regression")
# Make dummy variables since xgboost will not

ames_rec <-
  recipe(Sale_Price ~ ., data = ames) %>%
  step_dummy(Central_Air) %>%
  step_zv(all_predictors())

ames_processed <- prep(ames_rec) %>% bake(new_data = NULL)

xrf_reg_fit <-
  xrf_reg_mod %>%
  fit(Sale_Price ~ ., data = ames_processed)

xrf_rule_res <- tidy(xrf_reg_fit, penalty = .001)

xrf_rule_res
```

```
## # A tibble: 8 x 3
##   rule_id      rule                                                   estimate
##   <chr>        <chr>                                                     <dbl>
## 1 (Intercept)   ( TRUE )                                                 16.4
## 2 Central_Air_Y ( Central_Air_Y )                                         0.0567
## 3 Latitude      ( Latitude )                                             -0.424
## 4 Longitude     ( Longitude )                                            -0.0694
## 5 r1_1         ( Longitude <  -93.6299744 )                               0.102
## 6 r2_3        ( Central_Air_Y <  0.5 ) & ( Latitude <  42.0460129 )    -0.136
## 7 r2_5        ( Latitude >= 42.0460129 ) & ( Longitude <  -93.650901~   0.302
## 8 r2_6       ( Latitude >= 42.0460129 ) & ( Longitude >= -93.650901~  0.0853
```

Here, the focus is on the model coefficients produced by glmnet. We can also break down the results and sort them by the original predictor columns:

```
tidy(xrf_reg_fit, penalty = .001, unit = "columns")
```

```
## # A tibble: 11 x 3
##    rule_id       term          estimate
##    <chr>         <chr>            <dbl>
##  1 r1_1          Longitude       0.102
##  2 r2_3          Latitude       -0.136
##  3 r2_5          Latitude        0.302
##  4 r2_6          Latitude        0.0853
##  5 r2_3          Central_Air_Y  -0.136
##  6 r2_5          Longitude       0.302
##  7 r2_6          Longitude       0.0853
##  8 (Intercept)   (Intercept)    16.4
##  9 Longitude     Longitude      -0.0694
```

```
## 10 Latitude      Latitude      -0.424
## 11 Central_Air_Y Central_Air_Y   0.0567
```

C5.0 classification models:

Here, we'll use the Palmer penguin data:

```
data(penguins, package = "modeldata")

penguins <- drop_na(penguins)
```

First, let's fit a boosted rule-based model and tidy:

```
rule_model <-
  C5_rules(trees = 3) %>%
  fit(island ~ ., data = penguins)

rule_info <- tidy(rule_model)

rule_info
```

```
## # A tibble: 25 x 4
##    trial rule_num rule                                                   statistic
##    <int>    <int> <chr>                                                     <list>
##  1     1        1 ( bill_length_mm > 37.5 )                              <tibble>
##  2     1        2 ( species == 'Chinstrap' )                             <tibble>
##  3     1        3 ( body_mass_g > 3200 ) & ( body_mass_g < 3700 ) & (~ <tibble>
##  4     1        4 ( flipper_length_mm < 193 )                            <tibble>
##  5     1        5 ( species == 'Adelie' ) & ( bill_length_mm > 38.299~ <tibble>
##  6     1        6 ( bill_length_mm < 40.799999 ) & ( bill_depth_mm > ~ <tibble>
##  7     1        7 ( species == 'Adelie' ) & ( bill_length_mm > 41.599~ <tibble>
##  8     1        8 ( species == 'Adelie' ) & ( bill_depth_mm > 18.9 ) ~ <tibble>
##  9     2        1 ( species == 'Gentoo' )                                <tibble>
## 10     2        2 ( body_mass_g > 3700 ) & ( sex == 'female' )           <tibble>
## # ... with 15 more rows
```

```
# The statistic column has the pre-computed data about the
# data covered by the rule:
rule_info$statistic[[1]]
```

```
## # A tibble: 1 x 4
##   num_conditions coverage  lift class
##            <dbl>    <dbl> <dbl> <chr>
## 1              1      286  1.10 Biscoe
```

Tree-based models can also be tidied. Rather than saving the results in a recursive tree structure, we can show the paths to each of the terminal nodes (which is just a rule).

Let's fit a model and tidy:

```
tree_model <-
  boost_tree(trees = 3) %>%
  set_engine("C5.0") %>%
```

```
  set_mode("classification") %>%
  fit(island ~ ., data = penguins)

tree_info <- tidy(tree_model)

tree_info
```

```
## # A tibble: 34 x 4
##    trial  node rule                                                    statistic
##    <int> <int> <chr>                                                      <list>
##  1     1     1 "( species %in% c(\"Adelie\") ) & ( sex == \"female\" ~ <tibble>
##  2     1     2 "( species %in% c(\"Adelie\") ) & ( sex == \"female\" ~ <tibble>
##  3     1     3 "( species %in% c(\"Adelie\") ) & ( sex == \"female\" ~ <tibble>
##  4     1     4 "( species %in% c(\"Adelie\") ) & ( sex == \"female\" ~ <tibble>
##  5     1     5 "( species %in% c(\"Adelie\") ) & ( sex == \"female\" ~ <tibble>
##  6     1     6 "( species %in% c(\"Adelie\") ) & ( sex == \"female\" ~ <tibble>
##  7     1     7 "( species %in% c(\"Adelie\") ) & ( sex == \"female\" ~ <tibble>
##  8     1     8 "( species %in% c(\"Adelie\") ) & ( sex == \"male\" ) ~ <tibble>
##  9     1     9 "( species %in% c(\"Adelie\") ) & ( sex == \"male\" ) ~ <tibble>
## 10     1    10 "( species %in% c(\"Adelie\") ) & ( sex == \"male\" ) ~ <tibble>
## # ... with 24 more rows
```

```
# The statistic column has the class breakdown:
tree_info$statistic[[1]]
```

```
## # A tibble: 3 x 2
##   value     count
##   <chr>     <dbl>
## 1 Biscoe        3
## 2 Dream         1
## 3 Torgersen     0
```

Note that C5.0 models can have fractional estimates of counts in the terminal nodes.

Index