Package ‘rules’

August 7, 2021

Title Model Wrappers for Rule-Based Models

Version 0.1.2

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

License MIT + file LICENSE


Depends modeldata,
   parsnip (\(\geq 0.1.4\))

Imports dials,
   dplyr,
   generics (\(\geq 0.1.0\)),
   purrr,
   rlang,
   stringr,
   tibble,
   tidyr

Suggests C50,
   covr,
   Cubist,
   knitr,
   recipes,
   rmarkdown,
   spelling,
   testthat,
   xrf (\(\geq 0.2.0\))

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1.9000

R topics documented:

C5.rules

1
C5.0 rule-based classification models

Description

C5.rules() defines a model that derives feature rules from a tree for prediction. A single tree or boosted ensemble can be used.

The engine for this model is:

- C5.0 (default)

More information on how parsnip is used for modeling is at https://www.tidymodels.org/.

Usage

C5.rules(mode = "classification", trees = NULL, min_n = NULL, engine = "C5.0")

Arguments

- **mode**: A single character string for the type of model. The only possible value for this model is "classification".
- **trees**: A non-negative integer (no greater than 100 for the number of members of the ensemble.
- **min_n**: An integer greater than one zero and nine for the minimum number of data points in a node that are required for the node to be split further.
- **engine**: A single character string specifying what computational engine to use for fitting.

Details

C5.0 is a classification model that is an extension of the C4.5 model of Quinlan (1993). It has tree- and rule-based versions that also include boosting capabilities. C5.rules() enables the version of the model that uses a series of rules (see the examples below). To make a set of rules, an initial C5.0 tree is created and flattened into rules. The rules are pruned, simplified, and ordered. Rule sets are created within each iteration of boosting.

This function only defines what type of model is being fit. Once an engine is specified, the method to fit the model is also defined.

The model is not trained or fit until the fit.model_spec() function is used with the data.
committees

References
https://www.tidymodels.org, *Tidy Models with R*

See Also
C50::C5.0(), C50::C5.0Control(), C5.0 engine details

Examples

```r
show_engines("C5_rules")

C5_rules()
```

---

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

```r
committees()
committees(4:5)
max_rules()
```
cubist_rules() defines a model that derives simple feature rules from a tree ensemble and uses creates regression models within each rule.

The engine for this model is:

- Cubist (default)

More information on how `parsnip` is used for modeling is at [https://www.tidymodels.org/](https://www.tidymodels.org/).

**Usage**

```r
cubist_rules(
    mode = "regression",
    committees = NULL,
    neighbors = NULL,
    max_rules = NULL,
    engine = "Cubist"
)
```

**Arguments**

- `mode`: A single character string for the type of model. The only possible value for this model is "regression".
- `committees`: A non-negative integer (no greater than 100 for the number of members of the ensemble.
- `neighbors`: An integer between zero and nine for the number of training set instances that are used to adjust the model-based prediction.
- `max_rules`: The largest number of rules.
- `engine`: A single character string specifying what computational engine to use for fitting.

**Details**

Cubist is a rule-based ensemble regression model. A basic model tree (Quinlan, 1992) is created that has a separate linear regression model corresponding for each terminal node. The paths along the model tree is flattened into rules these rules are simplified and pruned. The parameter `min_n` is the primary method for controlling the size of each tree while `max_rules` controls the number of rules.

Cubist ensembles are created using `committees`, which are similar to boosting. After the first model in the committee is created, the second model uses a modified version of the outcome data based on whether the previous model under- or over-predicted the outcome. For iteration $m$, the new outcome $y^*$ is computed using

$$y^*_m = y - (\hat{y}(m-1) - y)$$
If a sample is under-predicted on the previous iteration, the outcome is adjusted so that the next time it is more likely to be over-predicted to compensate. This adjustment continues for each ensemble iteration. See Kuhn and Johnson (2013) for details.

After the model is created, there is also an option for a post-hoc adjustment that uses the training set (Quinlan, 1993). When a new sample is predicted by the model, it can be modified by its nearest neighbors in the original training set. For \( K \) neighbors, the model based predicted value is adjusted by the neighbor using:

\[
\frac{1}{K} \sum_{\ell=1}^{K} w_\ell \left[ t_\ell + (\hat{y} - \hat{t}_\ell) \right]
\]

where \( t \) is the training set prediction and \( w \) is a weight that is inverse to the distance to the neighbor.

This function only defines what type of model is being fit. Once an engine is specified, the method to fit the model is also defined.

The model is not trained or fit until the `fit.model_spec()` function is used with the data.

References

https://www.tidymodels.org, Tidy Models with R


See Also

`Cubist::cubist()`, `Cubist::cubistControl()`, Cubist engine details

Examples

cubist_rules()

# ---------------------------------------------

data(car_prices, package = "modeldata")
car_rules <-
cubist_rules(committees = 1) %>%
fit(log10(Price) ~., data = car_prices)
car_rules
summary(car_rules$fit)
mtry_prop

**Proportion of Randomly Selected Predictors**

**Description**

Proportion of Randomly Selected Predictors

**Usage**

mtry_prop(range = c(0.1, 1), 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 `dials` with classes "quant_param" and "param". The `range` element of the object is always converted to a list with elements "lower" and "upper".

---

multi_predict.C5.rules

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

**Description**

`multi_predict()` methods for rule-based models

**Usage**

```r
## S3 method for class 'C5_rules'
multi_predict(object, new_data, type = NULL, trees = NULL, ...)

## 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**: An object of class `model_fit`
- **new.data**: A rectangular data object, such as a data frame.
- **type**: A single character value or `NULL`. Possible values are `class` and `"prob"`.
- **trees**: An numeric vector of `trees` between one and 100.
- ...: Not currently used.
- **neighbors**: An numeric vector of neighbors values between zero and nine.
- **penalty**: Non-negative penalty values.

## Details

For C5.0 rule-based models, the model fit may contain less boosting iterations than the number requested. Printing the object will show how many were used due to early stopping. This can be change using an option in `C50::C5.0Control()`. Beware that the number of iterations requested.

## Value

A tibble with one row for each row of `new.data`. Multiple predictions are contained in a list column called `.pred`. That column has the standard `parsnip` prediction column names as well as the column with the tuning parameter values.

## Usage

```r
## S3 method for class 'C5_rules'
update(
  object,
  parameters = NULL,
  trees = NULL,
  min_n = NULL,
  fresh = FALSE,
  ...
)
```

```r
## S3 method for class 'cubist_rules'
update(
  object,
  parameters = NULL,
  committees = NULL,
  neighbors = NULL,
  max_rules = NULL,
  fresh = FALSE,
)```


## S3 method for class 'rule_fit'
update(
  object,
  parameters = NULL,
  mtry = NULL,
  trees = NULL,
  min_n = NULL,
  tree_depth = NULL,
  learn_rate = NULL,
  loss_reduction = NULL,
  sample_size = NULL,
  penalty = NULL,
  fresh = FALSE,
  ...
)

### Arguments

- **object**
  A `rule_fit` model specification.

- **parameters**
  A 1-row tibble or named list with *main* parameters to update. If the individual arguments are used, these will supersede the values in parameters. Also, using engine arguments in this object will result in an error.

- **trees**
  A non-negative integer (no greater than 100 for the number of members of the ensemble).

- **min_n**
  An integer greater than one and nine for the minimum number of data points in a node that are required for the node to be split further.

- **fresh**
  A logical for whether the arguments should be modified in-place or replaced wholesale.

- **committees**
  A non-negative integer (no greater than 100 for the number of members of the ensemble).

- **neighbors**
  An integer between zero and nine for the number of training set instances that are used to adjust the model-based prediction.

- **max_rules**
  The largest number of rules.

- **mtry**
  An number for the number (or proportion) of predictors that will be randomly sampled at each split when creating the tree models.

- **tree_depth**
  An integer for the maximum depth of the tree (i.e. number of splits).

- **learn_rate**
  A number for the rate at which the boosting algorithm adapts from iteration-to-iteration.

- **loss_reduction**
  A number for the reduction in the loss function required to split further.

- **sample_size**
  An number for the number (or proportion) of data that is exposed to the fitting routine.

- **penalty**
  L1 regularization parameter.
Examples

```r
# ------------------------------------------------------------------------------
model <- C5_rules(trees = 10, min_n = 2)
model
update(model, trees = 1)
update(model, trees = 1, fresh = TRUE)
# ------------------------------------------------------------------------------
model <- cubist_rules(committees = 10, neighbors = 2)
model
update(model, committees = 1)
update(model, committees = 1, fresh = TRUE)
# ------------------------------------------------------------------------------
model <- rule_fit(trees = 10, min_n = 2)
model
update(model, trees = 1)
update(model, trees = 1, fresh = TRUE)
```

---

**rule_fit**  
*RuleFit models*

**Description**

`rule_fit()` defines a model that derives simple feature rules from a tree ensemble and uses them as features to a regularized model.

The engine for this model is:

- `xrf` (default)

More information on how `parsnip` is used for modeling is at [https://www.tidymodels.org/](https://www.tidymodels.org/).

**Usage**

```r
rule_fit(
  mode = "unknown",
  mtry = NULL,
  trees = NULL,
  min_n = NULL,
  tree_depth = NULL,
  learn_rate = NULL,
  loss_reduction = NULL,
  sample_size = NULL,
  penalty = NULL,
  engine = "xrf"
)
```
Arguments

- **mode**: A single character string for the type of model. Possible values for this model are "unknown", "regression", or "classification".
- **mtry**: An number for the number (or proportion) of predictors that will be randomly sampled at each split when creating the tree models.
- **trees**: An integer for the number of trees contained in the ensemble.
- **min_n**: An integer for the minimum number of data points in a node that are required for the node to be split further.
- **tree_depth**: An integer for the maximum depth of the tree (i.e. number of splits).
- **learn_rate**: A number for the rate at which the boosting algorithm adapts from iteration-to-iteration.
- **loss_reduction**: A number for the reduction in the loss function required to split further.
- **sample_size**: An number for the number (or proportion) of data that is exposed to the fitting routine.
- **penalty**: L1 regularization parameter.
- **engine**: A single character string specifying what computational engine to use for fitting.

Details

The RuleFit model creates a regression model of rules in two stages. The first stage uses a tree-based model that is used to generate a set of rules that can be filtered, modified, and simplified. These rules are then added as predictors to a regularized generalized linear model that can also conduct feature selection during model training.

This function only defines what type of model is being fit. Once an engine is specified, the method to fit the model is also defined.

The model is not trained or fit until the `fit.model_spec()` function is used with the data.

References


[https://www.tidymodels.org](https://www.tidymodels.org), *Tidy Models with R*

See Also

`xrf::xrf.formula()`, `xrf engine details`

Examples

- `show_engines("rule_fit")`
- `rule_fit()`
tidy.cubist  

Turn regression rule models into tidy tibbles

Description

Turn regression rule models into tidy tibbles

Usage

```r
## S3 method for class 'cubist'
tidy(x, ...)
```

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

Arguments

- `x`: A Cubist or xrf object.
- `...`: Not currently used.
- `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

An example:

```r
library(dplyr)

data(ames, package = "modeldata")

ames <-
  ames %>%
  mutate(Sale_Price = log10(ames$Sale_Price),
         Gr_Liv_Area = log10(ames$Gr_Liv_Area))

# ------------------------------------------------------------------------------

cb_fit <-
  cubist_rules(committees = 10) %>%
  set_engine("Cubist") %>%
  fit(Sale_Price ~ Neighborhood + Longitude + Latitude + Gr_Liv_Area + Central_Air,
      data = ames)

cb_res <- tidy(cb_fit)
cb_res

## A tibble: 157 x 5
## committee rule_num rule estimate statistic
## <int> <int> <chr> <list> <list>
```
cb_res$estimate[[1]]
## A tibble: 4 × 2
##  term estimate
## <chr>  <dbl>
## 1 (Intercept) -408.
## 2 Longitude  -1.43
## 3 Latitude    6.6
## 4 Gr_Liv_Area  0.7

cb_res$statistic[[1]]
## A tibble: 1 × 6
##  num_conditions coverage mean min max error
##    <dbl>      <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     2     154 4.94  4.11  5.31 0.0956

xrf_rule_res <- tidy(xrf_reg_fit)
xrf_rule_res$rule[nrow(xrf_rule_res)] %>% rlang::parse_expr()
## (Gr_Liv_Area < 3.30210185) & (Gr_Liv_Area < 3.38872266) & (Gr_Liv_Area >= 2.94571471) & (Gr_Liv_Area >= 3.24870872) & (Latitude < 42.0271072) & (Neighborhood_Old_Town >= -9.53674316e-07)

xrf_col_res <- tidy(xrf_reg_fit, unit = "columns")
xrf_col_res
## A tibble: 149 × 3
##  rule_id term estimate
##    <chr> <chr>  <dbl>
## 1  r0_1   Gr_Liv_Area   -1.27e- 2
## 2  r2_4   Gr_Liv_Area   -3.70e-10
## 3  r2_2   Gr_Liv_Area    7.59e- 3
## 4  r2_4   Central_Air_Y  -3.70e-10
## 5  r3_5   Longitude  1.06e- 1
## 6  r3_6   Longitude  2.65e- 2
## 7  r3_5   Latitude  1.06e- 1
## 8  r3_6   Latitude  2.65e- 2
## 9  r3_5   Longitude  1.06e- 1
##10  r3_6   Longitude  2.65e- 2
## # ... with 139 more rows
Value

The Cubist method has columns committee, rule_num, rule, estimate, and statistics. The latter two are nested tibbles. estimate contains the parameter estimates for each term in the regression model and statistics has statistics about the data selected by the rules and the model fit.

The xrf results has columns rule_id, rule, and estimate. The rule_id column has the rule identifier (e.g., "r0_21") or the feature column name when the column is added directly into the model. For multiclass models, a class column is included.

In each case, the rule column has a character string with the rule conditions. These can be converted to an R expression using rlang::parse_expr().
Index

C5.0, 2
C5.0 engine details, 9
C50::C5.0(), 3
C50::C5.0Control(), 3, 7
C5_rules, 2
committees, 3
Cubist, 4
Cubist engine details, 5
Cubist::cubist(), 5
Cubist::cubistControl(), 5
cubist_rules, 4

fit.model_spec(), 2, 5, 10
max_rules (committees), 3
mtry_prop, 6
multi_predict._C5_rules, 6
multi_predict._cubist
   (multi_predict._C5_rules), 6
multi_predict._xrf
   (multi_predict._C5_rules), 6

rlang::parse_expr(), 13
rule_fit, 9
rules_update, 7

tidy.cubist, 11
tidy.xrf (tidy.cubist), 11

update.C5_rules (rules_update), 7
update.cubist_rules (rules_update), 7
update.rule_fit (rules_update), 7

xrf, 9
xrf engine details, 10
xrf::xrf.formula(), 10