Package 'dials'

December 9, 2018

Version 0.0.2

Title Tools for Creating Tuning Parameter Values

Description Many models contain tuning parameters (i.e. parameters that cannot be directly estimated from the data). These tools can be used to define objects for creating, simulating, or validating values for such parameters.

License GPL-2

URL https://tidymodels.github.io/dials

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

Encoding UTF-8

LazyData true

ByteCompile true

RoxygenNote 6.1.1

VignetteBuilder knitr

Depends scales

Imports glue, purrr, rlang, tibble, dplyr, utils, withr

Suggests testthat, knitr, rpart, covr, kernlab

NeedsCompilation no

Author Max Kuhn [aut, cre],
RStudio [cph]

Maintainer Max Kuhn <max@rstudio.com>

Repository CRAN

Date/Publication 2018-12-09 14:50:22 UTC

R topics documented:

---

dropout *Parameter objects related to parametric models.*

Description

These are objects that can be used for modeling, especially in conjunction with the **parsnip** package.

Usage

```
dropout

epochs

activation

mixture

penalty

rbf_sigma

prod_degree

num_terms

num_comp

cost

scale_factor

margin

degree

deg_free

hidden_units
```

batch_size

Format

An object of class quant_param (inherits from param) of length 7.

Details

These objects are pre-made parameter sets that are useful when the model is based on some type of slope/intercept model.

- penalty: The total amount of regularization used. This is used by parsnip::linear_reg() and parsnip::logistic_reg() with glmnet models.
- mixture: the proportion of L1 regularization in the model. (parsnip::linear_reg() and parsnip::logistic_reg())
- dropout: the parameter dropout rate. (parsnip:::mlp())
- epochs: the number of iterations of training. (parsnip:::mlp())
- activation: the type of activation function between network layers. (parsnip:::mlp())
- hidden_units: the number of hidden units in a network layer. (parsnip:::mlp())
- batch_size: the mini-batch size for neural networks.
- rbf_sigma: the sigma parameters of a radial basis function.
- cost: a cost value for SVM models.
- scale_factor: the polynomial and hyperbolic tangent kernel scaling factor.
- margin: the SVM margin parameter (e.g. epsilon in the insensitive-loss function for regression).
- degree: the polynomial degree.
- prod_degree: the number of terms to combine into interactions. A value of 1 implies an additive model. Useful for MARS models.
- num_terms: a nonspecific parameter for the number of terms in a model. This can be used with models that include feature selection, such as MARS.
- num_comp: the number of components in a model (e.g. PCA or PLS components).
- deg_free: a parameter for the degrees of freedom.

Value

Each object is generated by either new_quant_param or new_qual_param.

finalize *Functions to finalize data-specific parameter ranges*

Description

These functions take a parameter object and modify the unknown parts of ranges based on simple heuristics.

Usage

```
finalize(object, ...)

## S3 method for class 'list'
finalize(object, x, force = TRUE, ...)

## S3 method for class 'param'
finalize(object, x, force = TRUE, ...)

get_p(object, x, log_vals = FALSE, ...)

get_log_p(object, x, ...)

get_n_frac(object, x, log_vals = FALSE, frac = 1/3, ...)

get_n_frac_range(object, x, log_vals = FALSE, frac = c(1/10, 5/10),
  ...)

get_n(object, x, log_vals = FALSE, ...)

get_rbf_range(object, x, seed = sample.int(10^5, 1), ...)

get_batch_sizes(object, x, frac = c(1/10, 1/3), ...)
```

Arguments

object	A param object or list.
...	Other arguments to pass to kernlab::sigest.
x	The predictor data. In some cases (see below) this should only include numeric data.
force	A single logical that indicates that, even if the parameter object is complete, should it update the ranges anyway?
log_vals	A logical: should the ranges be set on the log10 scale?
frac	A double for the fraction of the data to be used for the upper bound. For get_n_frac_range and get_batch_sizes, two fractional values are required.
seed	An integer to control the randomness of the calculations.

Details

`finalize` runes the embedded finalization code contained in the `param` object and returns the updated version.

The "get" helper functions are designed to be used with the pipe and update the parameter object in-place.

`get_p` and `get_log_p` set the upper value of the range to be the number of columns in the data (on the natural and log10 scale, respectively). `get_n` and `get_n_frac` set the upper value to be a value that uses the number of rows.

`get_rbf_range` sets both bounds based on the heuristic defined in kernlab::sigest. It requires that all columns in `x` be numeric.

Value

An updated `param` object.

Examples

```
library(dplyr)
car_pred <- mtcars %>% select(-mpg)

# Needs an upper bound
mtry
finalize(mtry, car_pred)

# Nothing to do here since no unknowns
penalty
finalize(penalty, car_pred)

library(kernlab)
library(tibble)
library(purrr)

params <-
  tribble(
     ~parameter,   ~object,
         "mtry",      mtry,
    "num_terms", num_terms,
    "rbf_sigma", rbf_sigma
  )
params

# Note that `rbf_sigma` has a default range that does not need to be
# finalized but will be changed if used in the function:
complete_params <-
  params %>%
  mutate(object = map(object, finalize, car_pred))
complete_params

params %>% dplyr::filter(parameter == "rbf_sigma") %>% pull(object)
complete_params %>% dplyr::filter(parameter == "rbf_sigma") %>% pull(object)
```

grid_regular *Create grids of tuning parameters*

Description

Random and regular grids can be created for any number of parameter objects.

Usage

```
grid_regular(..., levels = 3, original = TRUE)

grid_random(..., size = 5, original = TRUE)
```

Arguments

...	One or more param objects (such as mtry() or penalty()). None of the objects can have unknown() values in the parameter ranges or values.
levels	An integer for how many value of each parameter will be used to make the regular grid? levels can be a single integer or a vector of integers that is the same length as the number of parameters in ....
original	A logical: should the parameters be in the original units or in the transformed space (if any)?
size	A single integer for the total number of parameter values returned for the random grid.

Value

A tibble with an additional class for the type of type of grid ("grid_regular" or "grid_random"). There are columns for each parameter and a row for every parameter or parameter combination.

Examples

```
# Will fail due to unknowns:
# grid_regular(mtry, min_n)

grid_regular(penalty, mixture)
grid_regular(penalty, mixture, levels = c(3, 4))
grid_random(penalty, mixture)
```

merge.model_spec *Merge parameter grid values into a parsnip object*

Description

parsnip contains model objects that have consistent names with **dials**. `merge` can be used to easily update any of the main parameters in a **parsnip** model.

Usage

```
## S3 method for class 'model_spec'
merge(x, y, ...)

## S3 method for class 'param_grid'
merge(x, y, ...)
```

Arguments

`x, y`	A combination of one **parsnip** model object (that has class `model_spec`) and one parameter grid resulting from `grid_regular` or `grid_random`. As long as this combination is present, the assignment to `x` and `y` isn't restricted.
`...`	Not currently used.

Value

A list containing updated model objects.

mtry *Parameter objects related to tree- and rule-based models.*

Description

These are objects that can be used for modeling, especially in conjunction with the **parsnip** package.

Usage

```
mtry

mtry_long

trees

min_n

sample_size
```

learn_rate

loss_reduction

tree_depth

prune

Cp

Format

An object of class quant_param (inherits from param) of length 7.

Details

These objects are pre-made parameter sets that are useful when the model is based on trees or rules.

- mtry and mtry_long: The number of predictors that will be randomly sampled at each split when creating the tree models. The latter uses a log transformation and is helpful when the data set has a large number of columns. mtry is used by **parsnip**'s parsnip::rand_forest() function.
- trees: The number of trees contained in a random forest or boosted ensemble. In the latter case, this is equal to the number of boosting iterations. (see parsnip::rand_forest() and parsnip::boost_tree()) functions.
- min_n: The minimum number of data points in a node that are required for the node to be split further. (parsnip::rand_forest() and parsnip::boost_tree())
- sample_size: the size of the data set used for modeling within an iteration of the modeling algorithm, such as stochastic gradient boosting. (parsnip::boost_tree())
- learn_rate: the rate at which the boosting algorithm adapts from iteration-to-iteration. (parsnip::boost_tree())
- loss_reduction: The reduction in the loss function required to split further. (parsnip::boost_tree())
- tree_depth: The maximum depth of the tree (i.e. number of splits). (parsnip::boost_tree())
- prune: a logical for whether a tree or set of rules should be pruned.
- Cp: The cost-complexity parameter in classical CART models.

Value

Each object is generated by either new_quant_param or new_qual_param.

new_quant_param *Tools for creating new parameter objects*

Description

These functions are used to construct new parameter objects.

Usage

```
new_quant_param(type = c("double", "integer"), range, inclusive,
  default = unknown(), trans = NULL, values = NULL, label = NULL,
  finalize = NULL)

new_qual_param(type = c("character", "logical"), values,
  default = unknown(), label = NULL, finalize = NULL)
```

Arguments

type	A single character value. For quantitative parameters, valid choices are "double" and "integer" while for qualitative factors they are "character" and "logical".
range	A two-element list of vector with the lowest or largest possible values, respectively. If these cannot be set when the parameter is defined, the unknown() function can be used. If a transformation is specified, these values should be in the *transformed units*.
inclusive	A two-element logical vector for whether the the range values should be inclusive or exclusive.
default	A single value the same class as type for the default parameter value. unknown() can also be used here.
trans	A trans object from the **scales** package, such as scales::log10_trans() or scales::reciprocal_trans().
values	A vector of possible values that is required when type is "character" or "logical" but optional otherwise.
label	An optional named character string that can be used for printing and plotting. The named should reflect the object name (e.g. "mtry", "neighbors", etc.)
finalize	A function that can be used to set the data-specific values of a parameter (such as the range).

Value

An object of class "param" with the primary class being either "quant_param" or "qual_param". The range element of the object is always converted to a list with elements "lower" and "upper". ’

Examples

```
num_subgroups <-
  new_quant_param(
    type = "integer",
    range = c(1L, 20L),
    inclusive = c(TRUE, TRUE),
    trans = NULL,
    label = c(num_subgroups = "# Subgroups"),
    finalize = NULL
  )
```

---

range_validate *Tools for working with parameter ranges*

Description

Tools for working with parameter ranges

Usage

```
range_validate(object, range, ukn_ok = TRUE)

range_get(object, original = TRUE)

range_set(object, range)
```

Arguments

object	An object with class quant_param.
range	A two-element numeric vector or list (including Inf). Values can include unknown() when ukn_ok = TRUE
ukn_ok	A single logical for wether unknown() is an acceptable value.
original	A single logical: should the range values be in the natural units (TRUE) or in the transformed space (FALSE, if applicable).

Value

range_validate returns the range if it passes the validation process (and throws an error otherwise). range_get returns also returns the range of the object. range_set returns an updated version of the parameter object.

Examples

```
library(dplyr)
my_lambda <-
  penalty %>%
  value_set(-4:-1)
try(my_lambda %>% range_validate(c(-10, NA)), silent = TRUE) %>% print()

range_get(my_lambda)

range_set(my_lambda, c(-10, 2)) %>% range_get()
```

---

unknown *Placeholder for unknown parameter values*

Description

This creates a simple expression used to signify that the value will be specified at a later time.

Usage

```
unknown()

is_unknown(x)

has_unknowns(object)
```

Arguments

x	An object or vector or objects.
object	An object of class param

Value

unknown returns expression value for unknown() and logicals for is_unknown() and has_unknowns().

---

value_validate *Tools for working with parameter values*

Description

Tools for working with parameter values

Usage

```
value_validate(object, values)

value_seq(object, n, original = TRUE)

value_sample(object, n, original = TRUE)

value_transform(object, values)

value_inverse(object, values)

value_set(object, values)
```

Arguments

object	An object with class quant_param.
values	A numeric vector or list (including Inf). Values *cannot* include unknown(). For value_validate, the units should be consistent with the parameter object's definition.
n	An integer for the (maximum) number of values to return. In some cases where a sequence is requested, the result might have less than n values. See Details.
original	A single logical: should the range values be in the natural units (TRUE) or in the transformed space (FALSE, if applicable).

Details

For sequences of integers, the code uses unique(floor(seq(min, max, length = n))) and this may generate an uneven set of values shorter than n. This also means that if n is larger than the range of the integers, a smaller set will be generated. For qualitative parameters, the first n values are returned.

If a single value sequence is requested, the default value is returned (if any). If not default is specified, the regular algorithm is used.

For quantitative parameters, any values contained in the object are sampled with replacement. Otherwise, a sequence of values between the range values is returned. It is possible that less than n values are returned.

For qualitative parameters, sampling is conducted with replacement. For qualitative values, a random uniform distribution is used.

Value

value_validate() throws an error or silently returns the values. value_transform and value_inverse return a *vector* of numeric values while value_seq and value_sample return a vector of values consistent with the type field of object.

Examples

```
library(dplyr)

penalty %>% value_set(-4:-1)

# Is a specific value valid?
penalty
penalty %>% range_get()
value_validate(penalty, 17)

# get a sequence of values
Cp
Cp %>% value_seq(4)
Cp %>% value_seq(4, original = FALSE)

on_log_scale <- Cp %>% value_seq(4, original = FALSE)
nat_units <- value_inverse(Cp, on_log_scale)
nat_units
value_transform(Cp, nat_units)

# random values in the range
set.seed(3666)
Cp %>% value_sample(2)
```

`weight` *Parameter objects related to text analysis.*

Description

These are objects that can be used for modeling, especially in conjunction with the **textrecipes** package.

Usage

```
weight

weight_scheme

token

max_times

min_times

max_tokens
```

Format

An object of class quant_param (inherits from param) of length 7.

Details

These objects are pre-made parameter sets that are useful in a variety of models.

- min_times, max_times: frequency of word occurances for removal. See ?step_tokenfilter.
- max_tokens: the number of tokens that will be retained. See ?step_tokenfilter.
- weight: A parameter for "double normalization" when creating token counts. See ?step_tf.
- weight_scheme: the method for term frequency calculations. Possible values are: "binary", "raw count", "term frequency", "log normalization", or "double normalization". See ?step_tf.
- token: the type of token with possible values: "characters", "character_shingle", "lines", "ngrams", "paragraphs", "ptb", "regex", "sentences", "skip_ngrams", "tweets", "words", "word_stems". See ?step_tokenize

Value

Each object is generated by either new_quant_param or new_qual_param.

---

weight_func *Parameter objects related to miscellaneous models.*

Description

These are objects that can be used for modeling, especially in conjunction with the **parsnip** package.

Usage

```
weight_func

surv_dist

Laplace

neighbors

dist_power

threshold
```

Format

An object of class qual_param (inherits from param) of length 5.

Details

These objects are pre-made parameter sets that are useful in a variety of models.

- `weight_func`: The type of kernel function that weights the distances between samples (e.g. in a K-nearest neighbors model).
- `surv_dist`: the statistical distribution of the data in a survival analysis model (e.g. `parsnip::surv_reg()`).
- `Laplace`: the Laplace correction used to smooth low-frequency counts.
- `neighbors`: a parameter for the number of neighbors used in a prototype model.
- `dist_power`: The order parameter used in calculating a Minkowski distance.
- `threshold`: A general thresholding parameter for values between `[0, 1]`.

Value

Each object is generated by either `new_quant_param` or `new_qual_param`.

Index