Package 'dials'

September 16, 2020

Version 0.0.9

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://dials.tidymodels.org, https://github.com/tidymodels/dials

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

Encoding UTF-8

LazyData true

ByteCompile true

RoxygenNote 7.1.1

VignetteBuilder knitr

Depends scales, R (>= 2.10)

Imports glue, purrr, tibble, rlang, utils, withr, DiceDesign, dplyr (>= 0.8.5), vctrs (>= 0.3.0.9000)

Suggests testthat, knitr, rpart, covr, kernlab, xml2, rmarkdown

NeedsCompilation no

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

Maintainer Max Kuhn <max@rstudio.com>

Repository CRAN

Date/Publication 2020-09-16 05:40:02 UTC

R topics documented:

`activation` *Activation functions between network layers*

Description

Activation functions between network layers

Usage

```
activation(values = values_activation)

values_activation
```

Arguments

`values`	A character string of possible values. See `values_activation` in examples below.

Format

An object of class `character` of length 4.

Details

This parameter is used in `parsnip` models for neural networks such as `parsnip:::mlp()`.

Examples

```
values_activation
activation()
```

`all_neighbors` *Parameter to determine which neighbors to use*

Description

Used in `themis::step_bsmote()`.

Usage

```
all_neighbors(values = c(TRUE, FALSE))
```

Arguments

`values`	A vector of possible values (TRUE or FALSE).

Examples

```
all_neighbors()
```

`Chicago` *Chicago Ridership Data*

Description

Chicago Ridership Data

Details

These data are from Kuhn and Johnson (2020) and contain an *abbreviated* training set for modeling the number of people (in thousands) who enter the Clark and Lake L station.

The `date` column corresponds to the current date. The columns with station names (`Austin` through `California`) are a *sample* of the columns used in the original analysis (for file size reasons). These are 14 day lag variables (i.e. date - 14 days). There are columns related to weather and sports team schedules.

The station at 35th and Archer is contained in the column `Archer_35th` to make it a valid R column name.

Value

`Chicago`	a tibble
`stations`	a vector of station names

Source

Kuhn and Johnson (2020), *Feature Engineering and Selection*, Chapman and Hall/CRC . https://bookdown.org/max/FES/ and https://github.com/topepo/FES

Examples

```
data(Chicago)
str(Chicago)
stations
```

confidence_factor *Parameters for possible engine parameters for C5.0*

Description

These parameters are auxiliary to tree-based models that use the "C5.0" engine. They correspond to tuning parameters that would be specified using `set_engine("C5.0",...)`.

Usage

```
confidence_factor(range = c(-1, 0), trans = log10_trans())

no_global_pruning(values = c(TRUE, FALSE))

predictor_winnowing(values = c(TRUE, FALSE))

fuzzy_thresholding(values = c(TRUE, FALSE))

rule_bands(range = c(2L, 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`.
values	For `no_global_pruning()`, `predictor_winnowing()`, and `fuzzy_thresholding()` either `TRUE` or `FALSE`.

Details

To use these, check `?C50::C5.0Control` to see how they are used.

Examples

```
confidence_factor()
no_global_pruning()
predictor_winnowing()
fuzzy_thresholding()
rule_bands()
```

cost *Support vector machine parameters*

Description

Parameters related to the SVM objective function(s).

Usage

```
cost(range = c(-10, 5), trans = log2_trans())

svm_margin(range = c(0, 0.2), 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.

Examples

```
cost()
svm_margin()
```

degree *Parameters for exponents*

Description

These parameters help model cases where an exponent is of interest (e.g. degree() or spline_degree()) or a product is used (e.g. prod_degree).

Usage

```
degree(range = c(1, 3), trans = NULL)

degree_int(range = c(1L, 3L), trans = NULL)

spline_degree(range = c(1L, 10L), trans = NULL)

prod_degree(range = c(1L, 2L), 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.

Details

degree() is helpful for parameters that are real number exponents (e.g. x^degree) whereas degree_int() is for cases where the exponent should be an integer.

The difference between degree_int() and spline_degree() is the default ranges (which is based on the context of how/where they are used).

prod_degree() is used by parsnip::mars() for the number of terms in interactions (and generates an integer).

Examples

```
degree()
degree_int()
spline_degree()
prod_degree()
```

---

deg_free *Degrees of freedom (integer)*

Description

The number of degrees of freedom used for model parameters.

Usage

```
deg_free(range = c(1L, 5L), 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.

Details

One context in which this parameter is used is spline basis functions.

Examples

```
deg_free()
```

`dist_power` *Minkowski distance parameter*

Description

Used in `parsnip::nearest_neighbor()`.

Usage

```
dist_power(range = c(1, 2), 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`.

Details

This parameter controls how distances are calculated. For example, `dist_power = 1` corresponds to Manhattan distance while `dist_power = 2` is Euclidean distance.

Examples

```
dist_power()
```

`dropout` *Neural network parameters*

Description

These functions generate parameters that are useful for neural network models.

Usage

```
dropout(range = c(0, 1), trans = NULL)

epochs(range = c(10L, 1000L), trans = NULL)

hidden_units(range = c(1L, 10L), trans = NULL)

batch_size(range = c(unknown(), unknown()), trans = log2_trans())
```

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.

Details

- dropout(): The parameter dropout rate. (See parsnip::mlp()).
- epochs(): The number of iterations of training. (See parsnip::mlp()).
- hidden_units(): The number of hidden units in a network layer. (See parsnip::mlp()).
- batch_size(): The mini-batch size for neural networks.

Examples

```
dropout()
```

---

extrapolation *Parameters for possible engine parameters for Cubist*

Description

These parameters are auxiliary to models that use the "Cubist" engine. They correspond to tuning parameters that would be specified using set_engine("Cubist0",...).

Usage

```
extrapolation(range = c(1, 110), trans = NULL)

unbiased_rules(values = c(TRUE, FALSE))

max_rules(range = c(1L, 100L), 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.
values	For unbiased_rules(), either TRUE or FALSE.

Details

To use these, check ?Cubist::cubistControl to see how they are used.

Examples

```
extrapolation()
unbiased_rules()
max_rules()
```

`finalize` *Functions to finalize data-specific parameter ranges*

Description

These functions take a parameter object and modify the unknown parts of `ranges` based on a data set and 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, ...)

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

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

## Default S3 method:
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 a list of param objects.
...	Other arguments to pass to the underlying parameter finalizer functions. For example, for get_rbf_range(), the dots are passed along 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(), a vector of two fractional values are required.
seed	An integer to control the randomness of the calculations.

Details

finalize() runs the embedded finalizer function contained in the param object (object$finalize) and returns the updated version. The finalization function is one of the get_*() helpers.

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 the number of rows in the data or a fraction of the total 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 or a list of updated param objects depending on what is provided in object.

Examples

```
library(dplyr)
car_pred <- select(mtcars, -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)
```

`freq_cut` *Near-zero variance parameters*

Description

These parameters control the specificity of the filter for near-zero variance parameters in `recipes::step_nzv()`.

Usage

```
freq_cut(range = c(5, 25), trans = NULL)

unique_cut(range = c(0, 100), 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`.

Details

Smaller values of `freq_cut()` and `unique_cut()` make the filter less sensitive.

Examples

```
freq_cut()
unique_cut()
```

`grid_max_entropy` *Space-filling parameter grids*

Description

Experimental designs for computer experiments are used to construct parameter grids that try to cover the parameter space such that any portion of the space has an observed combination that is not too far from it.

Usage

```
grid_max_entropy(
  x,
  ...,
  size = 3,
  original = TRUE,
  variogram_range = 0.5,
  iter = 1000
)

## S3 method for class 'parameters'
grid_max_entropy(
  x,
  ...,
  size = 3,
  original = TRUE,
  variogram_range = 0.5,
  iter = 1000
)

## S3 method for class 'list'
grid_max_entropy(
  x,
  ...,
  size = 3,
  original = TRUE,
  variogram_range = 0.5,
  iter = 1000
)

## S3 method for class 'param'
grid_max_entropy(
  x,
```

```
  ...,
  size = 3,
  original = TRUE,
  variogram_range = 0.5,
  iter = 1000
)

## S3 method for class 'workflow'
grid_max_entropy(
  x,
  ...,
  size = 3,
  original = TRUE,
  variogram_range = 0.5,
  iter = 1000
)

grid_latin_hypercube(x, ..., size = 3, original = TRUE)

## S3 method for class 'parameters'
grid_latin_hypercube(x, ..., size = 3, original = TRUE)

## S3 method for class 'list'
grid_latin_hypercube(x, ..., size = 3, original = TRUE)

## S3 method for class 'param'
grid_latin_hypercube(x, ..., size = 3, original = TRUE)

## S3 method for class 'workflow'
grid_latin_hypercube(x, ..., size = 3, original = TRUE)
```

Arguments

x	A param object, list, or parameters.
...	One or more param objects (such as mtry() or penalty()). None of the objects can have unknown() values in the parameter ranges or values.
size	A single integer for the total number of parameter value combinations returned. If duplicate combinations are generated from this size, the smaller, unique set is returned.
original	A logical: should the parameters be in the original units or in the transformed space (if any)?
variogram_range	A numeric value greater than zero. Larger values reduce the likelihood of empty regions in the parameter space.
iter	An integer for the maximum number of iterations used to find a good design.

Details

The types of designs supported here are latin hypercube designs and designs that attempt to maximize the determinant of the spatial correlation matrix between coordinates. Both designs use random sampling of points in the parameter space.

Note that there may a difference in grids depending on how the function is called. If the call uses the parameter objects directly the possible ranges come from the objects in `dials`. For example:

```
cost()

## Cost  (quantitative)
## Transformer:  log-2
## Range (transformed scale): [-10, -1]

set.seed(283)
cost_grid_1 <- grid_latin_hypercube(cost(), size = 1000)
range(log2(cost_grid_1$cost))

## [1] -9.998623 -1.000423
```

However, in some cases, the `tune` package overrides the default ranges for specific models. If the grid function uses a `parameters` object created from a model or recipe, the ranges my have different defaults (specific to those models). Using the example above, the `cost` argument above is different for SVM models:

```
library(parsnip)
library(tune)

# When used in tune, the log2 range is [-10, 5]
svm_mod <-
  svm_rbf(cost = tune()) %>%
  set_engine("kernlab")

set.seed(283)
cost_grid_2 <- grid_latin_hypercube(parameters(svm_mod), size = 1000)
range(log2(cost_grid_2$cost))

## [1] -9.997704  4.999296
```

References

Sacks, Jerome & Welch, William & J. Mitchell, Toby, and Wynn, Henry. (1989). Design and analysis of computer experiments. With comments and a rejoinder by the authors. Statistical Science. 4. 10.1214/ss/1177012413.

Santner, Thomas, Williams, Brian, and Notz, William. (2003). The Design and Analysis of Computer Experiments. Springer.

Dupuy, D., Helbert, C., and Franco, J. (2015). DiceDesign and DiceEval: Two R packages for design and analysis of computer experiments. Journal of Statistical Software, 65(11)

Examples

```
grid_max_entropy(
  hidden_units(),
  penalty(),
  epochs(),
  activation(),
  learn_rate(c(0, 1), trans = scales::log_trans()),
  size = 10,
  original = FALSE)

grid_latin_hypercube(penalty(), mixture(), original = TRUE)
```

---

grid_regular *Create grids of tuning parameters*

Description

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

Usage

```
grid_regular(x, ..., levels = 3, original = TRUE, filter = NULL)

## S3 method for class 'parameters'
grid_regular(x, ..., levels = 3, original = TRUE, filter = NULL)

## S3 method for class 'list'
grid_regular(x, ..., levels = 3, original = TRUE, filter = NULL)

## S3 method for class 'param'
grid_regular(x, ..., levels = 3, original = TRUE, filter = NULL)

## S3 method for class 'workflow'
grid_regular(x, ..., levels = 3, original = TRUE, filter = NULL)

make_regular_grid(..., levels = 3, original = TRUE, filter = NULL)

grid_random(x, ..., size = 5, original = TRUE, filter = NULL)

## S3 method for class 'parameters'
grid_random(x, ..., size = 5, original = TRUE, filter = NULL)

## S3 method for class 'list'
grid_random(x, ..., size = 5, original = TRUE, filter = NULL)

## S3 method for class 'param'
grid_random(x, ..., size = 5, original = TRUE, filter = NULL)
```

```
## S3 method for class 'workflow'
grid_random(x, ..., size = 5, original = TRUE, filter = NULL)
```

Arguments

x	A param object, list, or parameters.
...	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 the number of values of each parameter to use 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 .... levels can be a named integer vector, with names that match the id values of parameters.
original	A logical: should the parameters be in the original units or in the transformed space (if any)?
filter	A logical: should the parameters be filtered prior to generating the grid. Must be a single expression referencing parameter names that evaluates to a logical vector.
size	A single integer for the total number of parameter value combinations returned for the random grid. If duplicate combinations are generated from this size, the smaller, unique set is returned.

Details

Note that there may a difference in grids depending on how the function is called. If the call uses the parameter objects directly the possible ranges come from the objects in dials. For example:

```
mixture()

## Proportion of lasso Penalty  (quantitative)
## Range: [0, 1]

set.seed(283)
mix_grid_1 <- grid_random(mixture(), size = 1000)
range(mix_grid_1$mixture)

## [1] 0.001490161 0.999741096
```

However, in some cases, the tune package overrides the default ranges for specific models. If the grid function uses a parameters object created from a model or recipe, the ranges my have different defaults (specific to those models). Using the example above, the mixture argument above is different for glmnet models:

```
library(parsnip)
library(tune)

# When used with glmnet, the range is [0.05, 1.00]
```

```
glmn_mod <-
  linear_reg(mixture = tune()) %>%
  set_engine("glmnet")

set.seed(283)
mix_grid_2 <- grid_random(parameters(glmn_mod), size = 1000)
range(mix_grid_2$mixture)

## [1] 0.05141565 0.99975404
```

Value

A tibble. There are columns for each parameter and a row for every parameter combination.

Examples

```
# filter arg will allow you to filter subsequent grid data frame based on some condition.
p <- parameters(penalty(), mixture())
grid_regular(p)
grid_regular(p, filter = penalty <= .01)

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

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

Laplace *Laplace correction parameter*

Description

Laplace correction for smoothing low-frequency counts.

Usage

```
Laplace(range = c(0, 3), 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.

Details

This parameter is often used to correct for zero-count data in tables or proportions.

Value

A function with classes "quant_param" and "param"

Examples

```
Laplace()
```

`learn_rate` *Learning rate*

Description

The parameter is used in boosting methods (`parsnip::boost_tree()`) or some types of neural network optimization methods.

Usage

```
learn_rate(range = c(-10, -1), trans = log10_trans())
```

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`.

Details

The parameter is used on the log10 scale. The units for the `range` function are on this scale.

`learn_rate()` corresponds to `eta` in **xgboost**.

Examples

```
learn_rate()
```

max_nodes *Parameters for possible engine parameters for randomForest*

Description

These parameters are auxiliary to random forest models that use the "randomForest" engine. They correspond to tuning parameters that would be specified using `set_engine("randomForest",...)`.

Usage

```
max_nodes(range = c(100L, 10000L), 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`.

Examples

```
max_nodes()
```

max_num_terms *Parameters for possible engine parameters for earth models*

Description

These parameters are auxiliary to models that use the "earth" engine. They correspond to tuning parameters that would be specified using `set_engine("earth",...)`.

Usage

```
max_num_terms(range = c(20L, 200L), 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`.

Details

To use these, check `?earth::earth` to see how they are used.

Examples

```
max_num_terms()
```

`max_times` *Word frequencies for removal*

Description

Used in `textrecipes::step_tokenfilter()`.

Usage

```
max_times(range = c(1L, as.integer(10^5)), trans = NULL)

min_times(range = c(0L, 1000L), 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`.

Examples

```
max_times()
min_times()
```

`max_tokens` *Maximum number of retained tokens*

Description

Used in `textrecipes::step_tokenfilter()`.

Usage

```
max_tokens(range = c(0L, as.integer(10^3)), 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.

Examples

```
max_tokens()
```

---

min_dist *Parameter for the effective minimum distance between embedded points*

Description

Used in embed::step_umap().

Usage

```
min_dist(range = c(-4, 0), trans = log10_trans())
```

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.

Examples

```
min_dist()
```

min_unique *Number of unique values for pre-processing*

Description

Some pre-processing parameters require a minimum number of unique data points to proceed.

Usage

```
min_unique(range = c(5L, 15L), 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.

Examples

```
min_unique()
```

mixture *Mixture of penalization terms*

Description

A numeric parameter function representing the relative amount of penalties (e.g. L1, L2, etc) in regularized models.

Usage

```
mixture(range = c(0, 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.

Details

This parameter is used for regularized or penalized models such as `parsnip::linear_reg()`, `parsnip::logistic_reg()`, and others. It is formulated as the proportion of L1 regularization (i.e. lasso) in the model. In the `glmnet` model, `mixture = 1` is a pure lasso model while `mixture = 0` indicates that ridge regression is being used.

Examples

```
mixture()
```

---

mtry *Number of randomly sampled predictors*

Description

The number of predictors that will be randomly sampled at each split when creating tree models.

Usage

```
mtry(range = c(1L, unknown()), trans = NULL)

mtry_long(range = c(0L, unknown()), trans = log10_trans())
```

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`.

Details

This parameter is used for regularized or penalized models such as `parsnip::rand_forest()` and others. `mtry_long()` has the values on the log10 scale and is helpful when the data contain a large number of predictors.

Since the scale of the parameter depends on the number of columns in the data set, the upper bound is set to unknown but can be filled in via the `finalize()` method.

Examples

```
mtry(c(1L, 10L))   # in original units
mtry_long(c(0, 5)) # in log10 units
```

neighbors *Number of neighbors*

Description

The number of neighbors is used for models (`parsnip::nearest_neighbor()`), imputation (`recipes::step_knnimpute()`) and dimension reduction (`recipes::step_isomap()`).

Usage

```
neighbors(range = c(1L, 10L), 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`.

Details

A static range is used but a broader range should be used if the data set is large or more neighbors are required.

Examples

```
neighbors()
```

new-param *Tools for creating new parameter objects*

Description

These functions are used to construct new parameter objects. Generally, these functions are called from higher level parameter generating functions like `mtry()`.

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 vector with the smallest 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 range values should be inclusive or exclusive.
default	A single value with 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(). Create custom transforms with scales::trans_new().
values	A vector of possible values that is required when type is "character" or "logical" but optional otherwise. For quantitative parameters, these override the range when generating sequences if set.
label	An optional named character string that can be used for printing and plotting. The name should match 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

```
# Create a function that generates a quantitative parameter
# corresponding to the number of subgroups.
num_subgroups <- function(range = c(1L, 20L), trans = NULL) {
  new_quant_param(
    type = "integer",
    range = range,
    inclusive = c(TRUE, TRUE),
```

```
    trans = trans,
    label = c(num_subgroups = "# Subgroups"),
    finalize = NULL
  )
}

num_subgroups()

num_subgroups(range = c(3L, 5L))

# Custom parameters instantly have access
# to sequence generating functions
value_seq(num_subgroups(), 5)
```

`num_breaks` *Number of cut-points for binning*

Description

This parameter controls how many bins are used when discretizing predictors.

Usage

```
num_breaks(range = c(2L, 10L), 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`.

Examples

```
num_breaks()
```

num_comp *Number of new features*

Description

The number of derived predictors from models or feature engineering methods.

Usage

```
num_comp(range = c(1L, unknown()), trans = NULL)

num_terms(range = c(1L, unknown()), 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.

Details

Since the scale of these parameters often depends on the number of columns in the data set, the upper bound is set to unknown. For example, the number of PCA components is limited by the number of columns and so on.

The difference between num_comp() and num_terms() is semantics.

Examples

```
num_terms()
num_terms(c(2L, 10L))
```

num_hash *Text hashing parameters*

Description

Used in textrecipes::step_texthash().

Usage

```
num_hash(range = c(8L, 12L), trans = log2_trans())

signed_hash(values = c(TRUE, FALSE))
```

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.
values	A vector of possible values (TRUE or FALSE).

Examples

```
num_hash()
signed_hash()
```

---

num_tokens *Parameter to determine number of tokens in ngram*

Description

Used in textrecipes::step_ngram().

Usage

```
num_tokens(range = c(1, 3), 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.

Examples

```
num_tokens()
```

over_ratio *Parameters for class-imbalance sampling*

Description

For up- and down-sampling methods, these parameters control how much data are added or removed from the training set.

Usage

```
over_ratio(range = c(0.8, 1.2), trans = NULL)

under_ratio(range = c(0.8, 1.2), 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.

Details

See recipes::step_upsample() and recipes::step_downsample() for the interpretation of these parameters.

Examples

```
under_ratio()
over_ratio()
```

parameters *Information on tuning parameters within an object*

Description

Information on tuning parameters within an object

Usage

```
parameters(x, ...)

## Default S3 method:
parameters(x, ...)

## S3 method for class 'param'
parameters(x, ...)

## S3 method for class 'list'
parameters(x, ...)

param_set(x, ...)
```

Arguments

x	An object, such as a list of param objects or an actual param object.
...	Only used for the param method so that multiple param objects can be passed to the function.

---

penalty *Amount of regularization/penalization*

Description

A numeric parameter function representing the amount of penalties (e.g. L1, L2, etc) in regularized models.

Usage

```
penalty(range = c(-10, 0), trans = log10_trans())
```

Arguments

range	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. Note that these are in transformed units.
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.

Details

This parameter is used for regularized or penalized models such as parsnip::linear_reg(), parsnip::logistic_reg(), and others.

Examples

```
penalty()
```

predictor_prop *Proportion of predictors*

Description

The parameter is used in models where a parameter is the proportion of predictor variables.

Usage

```
predictor_prop(range = c(0, 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.

Details

predictor_prop() is used in step_pls().

Examples

```
predictor_prop()
```

prune_method *MARS pruning methods*

Description

MARS pruning methods

Usage

```
prune_method(values = values_prune_method)

values_prune_method
```

Arguments

values	A character string of possible values. See values_prune_method in examples below.

Format

An object of class `character` of length 6.

Details

This parameter is used in `parsnip:::mars()`.

Examples

```
values_prune_method
prune_method()
```

`pull_dials_object` *Return a dials parameter object associated with parameters*

Description

`pull_dials_object()` can extract a single `dials` parameter object from different types of objects (e.g. parameter sets, recipes, etc.).

Usage

```
pull_dials_object(x, id, ...)

## S3 method for class 'parameters'
pull_dials_object(x, id, ...)

## S3 method for class 'recipe'
pull_dials_object(x, id, ...)

## S3 method for class 'model_spec'
pull_dials_object(x, id, ...)

## S3 method for class 'workflow'
pull_dials_object(x, id, ...)
```

Arguments

`x`	The results of a call to `parameters()`, a recipe, model specification, or workflow.
`id`	A single string for the `id` of the parameter.
`...`	Not currently used.

Value

A `dials` parameter object.

Examples

```
glmn_param <- parameters(lambda = penalty(), mixture())
pull_dials_object(glmn_param, "lambda")

# fails:
# pull_dials_object(glmn_param, "penalty")
```

`range_validate` *Tools for working with parameter ranges*

Description

Setters, getters, and validators for 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 whether `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 new range if it passes the validation process (and throws an error otherwise).

`range_get()` returns the current range of the object.

`range_set()` returns an updated version of the parameter object with a new range.

Examples

```
library(dplyr)

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

try(
```

```
  range_validate(my_lambda, c(-10, NA)), silent = TRUE
) %>%
  print()

range_get(my_lambda)

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

`rbf_sigma` *Kernel parameters*

Description

Parameters related to the radial basis or other kernel functions.

Usage

```
rbf_sigma(range = c(-10, 0), trans = log10_trans())

scale_factor(range = c(-10, -1), trans = log10_trans())

kernel_offset(range = c(0, 2), 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`.

Details

`degree()` can also be used in kernel functions.

Examples

```
rbf_sigma()
scale_factor()
kernel_offset()
```

regularization_factor *Parameters for possible engine parameters for ranger*

Description

These parameters are auxiliary to random forest models that use the "ranger" engine. They correspond to tuning parameters that would be specified using `set_engine("ranger",...)`.

Usage

```
regularization_factor(range = c(0, 1), trans = NULL)

regularize_depth(values = c(TRUE, FALSE))

significance_threshold(range = c(-10, 0), trans = log10_trans())

lower_quantile(range = c(0, 1), trans = NULL)

splitting_rule(values = ranger_split_rules)

ranger_class_rules

ranger_reg_rules

ranger_split_rules

num_random_splits(range = c(1L, 15L), 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`.
values	For `splitting_rule()`, a character string of possible values. See `ranger_split_rules`, `ranger_class_rules`, and `ranger_reg_rules` for appropriate values. For `regularize_depth()`, either `TRUE` or `FALSE`.

Format

An object of class `character` of length 4.

An object of class `character` of length 3.

An object of class `character` of length 7.

Details

To use these, check `?ranger::ranger` to see how they are used. Some are conditional on others. For example, `significance_threshold()`, `num_random_splits()`, and others are only used when `splitting_rule = "extratrees"`.

Examples

```
regularization_factor()
regularize_depth()
```

---

smoothness *Kernel Smoothness*

Description

Used in `discrim::naive_Bayes()`.

Usage

```
smoothness(range = c(0.5, 1.5), 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`.

Examples

```
smoothness()
```

---

surv_dist *Parametric distributions for censored data*

Description

Parametric distributions for censored data

Usage

```
surv_dist(values = values_surv_dist)

values_surv_dist
```

Arguments

values	A character string of possible values. See values_surv_dist in examples below.

Format

An object of class character of length 6.

Details

This parameter is used in parsnip:::surv_reg().

Examples

```
values_surv_dist
surv_dist()
```

---

threshold *General thresholding parameter*

Description

In a number of cases, there are arguments that are threshold values for data falling between zero and one. For example, recipes::step_other() and so on.

Usage

```
threshold(range = c(0, 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.

Examples

```
threshold()
```

token *Token types*

Description

Token types

Usage

```
token(values = values_token)

values_token
```

Arguments

values	A character string of possible values. See values_token in examples below.

Format

An object of class character of length 12.

Details

This parameter is used in textrecipes::step_tokenize().

Examples

```
values_token
token()
```

trees *Parameter functions related to tree- and rule-based models.*

Description

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

Usage

```
trees(range = c(1L, 2000L), trans = NULL)

min_n(range = c(2L, 40L), trans = NULL)

sample_size(range = c(unknown(), unknown()), trans = NULL)

sample_prop(range = c(1/10, 1), trans = NULL)

loss_reduction(range = c(-10, 1.5), trans = log10_trans())

tree_depth(range = c(1L, 15L), trans = NULL)

prune(values = c(TRUE, FALSE))

cost_complexity(range = c(-10, -1), trans = log10_trans())
```

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.
values	A vector of possible values (TRUE or FALSE).

Details

These functions generate parameters that are useful when the model is based on trees or rules.

- 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()).
- min_n(): The minimum number of data points in a node that is required for the node to be split further. (See 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. (See parsnip::boost_tree()).
- sample_prop(): The same as sample_size() but as a proportion of the total sample.
- loss_reduction(): The reduction in the loss function required to split further. (See parsnip::boost_tree()). This corresponds to gamma in **xgboost**.
- tree_depth(): The maximum depth of the tree (i.e. number of splits). (See parsnip::boost_tree()).
- prune(): A logical for whether a tree or set of rules should be pruned.
- cost_complexity(): The cost-complexity parameter in classical CART models.

Examples

```
trees()
min_n()
sample_size()
loss_reduction()
tree_depth()
prune()
cost_complexity()
```

---

unknown *Placeholder for unknown parameter values*

Description

`unknown()` creates an 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 to test for unknown-ness.
`object`	An object of class `param`.

Value

`unknown()` returns expression value for `unknown()`.

`is_unknown()` returns a vector of logicals as long as `x` that are `TRUE` is the element of `x` is unknown, and `FALSE` otherwise.

`has_unknowns()` returns a single logical indicating if the `range` of a `param` object has any unknown values.

Examples

```
# Just returns an expression
unknown()

# Of course, true!
is_unknown(unknown())

# Create a range with a minimum of 1
```

```
# and an unknown maximum
range <- c(1, unknown())

range

# The first value is known, the
# second is not
is_unknown(range)

# mtry()'s maximum value is not known at
# creation time
has_unknowns(mtry())
```

`update.parameters` *Update a single parameter in a parameter set*

Description

Update a single parameter in a parameter set

Usage

```
## S3 method for class 'parameters'
update(object, ...)
```

Arguments

`object`	A parameter set.
`...`	One or more unquoted named values separated by commas. The names should correspond to the `id` values in the parameter set. The values should be parameter objects or `NA` values.

Value

The modified parameter set.

Examples

```
params <- list(lambda = penalty(), alpha = mixture(), `rand forest` = mtry())
pset <- parameters(params)
pset

update(pset, `rand forest` = finalize(mtry(), mtcars), alpha = mixture(c(.1, .2)))
```

value_validate *Tools for working with parameter values*

Description

Setters and validators for parameter values. Additionally, tools for creating sequences of parameter values and for transforming parameter values are provided.

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.out = 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 no 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 of the values is conducted with replacement. For qualitative values, a random uniform distribution is used.

Value

value_validate() throws an error or silently returns values if they are contained in the values of the object.

value_transform() and value_inverse() return a *vector* of numeric values.

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
cost_complexity()
cost_complexity() %>% value_seq(4)
cost_complexity() %>% value_seq(4, original = FALSE)

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

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

---

weight *Parameter for "*double normalization*" when creating token counts*

Description

Used in textrecipes::step_tf().

Usage

```
weight(range = c(-10, 0), trans = log10_trans())
```

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.

Examples

```
weight()
```

---

weight_func *Kernel functions for distance weighting*

Description

Kernel functions for distance weighting

Usage

```
weight_func(values = values_weight_func)

values_weight_func
```

Arguments

values	A character string of possible values. See values_weight_func in examples below.

Format

An object of class character of length 10.

Details

This parameter is used in parsnip::nearest_neighbors().

Examples

```
values_weight_func
weight_func()
```

weight_scheme *Term frequency weighting methods*

Description

Term frequency weighting methods

Usage

```
weight_scheme(values = values_weight_scheme)

values_weight_scheme
```

Arguments

values	A character string of possible values. See values_weight_scheme in examples below.

Format

An object of class character of length 5.

Details

This parameter is used in textrecipes::step_tf().

Examples

```
values_weight_scheme
weight_scheme()
```

window_size *Parameter for the moving window size*

Description

Used in recipes::step_window().

Usage

```
window_size(range = c(3L, 11L), 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.

Examples

```
window_size()
```

Index