Package ‘dials’

April 3, 2023

**Title**  Tools for Creating Tuning Parameter Values

**Version**  1.2.0

**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**  MIT + file LICENSE


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

**Depends**  R (>= 3.4), scales

**Imports**  cli, DiceDesign, dplyr (>= 0.8.5), glue, hardhat (>= 1.1.0), lifecycle, pillar, purrr, rlang (>= 1.1.0), tibble, utils, vctrs (>= 0.3.8), withr

**Suggests**  covr, kernlab, knitr, rmarkdown, rpart, testthat (>= 3.0.0), xml2

**VignetteBuilder**  knitr

**ByteCompile**  true

**Config/testthat/edition**  3

**Config/Needs/website**  tidyverse/tidytemplate

**Encoding**  UTF-8

**RoxygenNote**  7.2.3

**NeedsCompilation**  no

**Author**  Max Kuhn [aut],
           Hannah Frick [aut, cre],
           Posit Software, PBC [cph, fnd]

**Maintainer**  Hannah Frick <hannah@posit.co>

**Repository**  CRAN

**Date/Publication**  2023-04-03 15:30:10 UTC
### R topics documented:

- activation .................................................. 3
- adjust_deg_free ............................................. 4
- all_neighbors ................................................. 5
- bart-param ................................................... 5
- class_weights ............................................... 6
- conditional_min_criterion ................................. 6
- confidence_factor ........................................... 7
- cost ........................................................... 8
- degree ........................................................ 9
- deg_free ...................................................... 10
- dist_power ................................................... 10
- dropout ........................................................ 11
- extrapolation ................................................. 12
- finalize ....................................................... 12
- freq_cut ....................................................... 15
- grid_max_entropy ........................................... 16
- grid_regular .................................................. 19
- harmonic_frequency ......................................... 21
- Laplace ....................................................... 22
- learn_rate .................................................... 22
- max_nodes ..................................................... 23
- max_num_terms ............................................... 24
- max_times ..................................................... 24
- max_tokens .................................................... 25
- min_dist ....................................................... 26
- min_unique .................................................... 26
- mixture ......................................................... 27
- momentum ....................................................... 27
- mtry .......................................................... 28
- mtry_prop ...................................................... 29
- neighbors ..................................................... 30
- new-param ..................................................... 31
- num_breaks ................................................... 33
- num_clusters ............................................... 33
- num_comp ..................................................... 34
- num_hash ...................................................... 34
- num_knots ..................................................... 35
- num_leaves .................................................... 36
- num_runs ....................................................... 36
- num_tokens ................................................... 37
- over_ratio ..................................................... 38
- parameters .................................................... 38
- penalty ......................................................... 39
- predictor_prop ............................................... 40
- prior_slab_dispersion ....................................... 40
- prune_method ................................................... 41
### Description

Activation functions between network layers

### Usage

```r
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 5.

Details

This parameter is used in \texttt{parsnip} models for neural networks such as \texttt{parsnip::mlp()}. 

Examples

```r
values_activation
activation()
```

```
adjust_deg_free

\begin{itemize}
\item \texttt{adjust_deg_free} Parameters to adjust effective degrees of freedom
\end{itemize}
```

Description

This parameter can be used to moderate smoothness of spline or other terms used in generalized additive models.

Usage

```r
adjust_deg_free(range = c(0.25, 4), trans = NULL)
```

Arguments

- `range` A two-element vector holding the \textit{defaults} for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the \textit{transformed units}. 
- `trans` A \texttt{trans} object from the \texttt{scales} package, such as \texttt{scales::log10_trans()} or \texttt{scales::reciprocal_trans()}. If not provided, the default is used which matches the units used in \texttt{range}. If no transformation, \texttt{NULL}. 

Details

Used in \texttt{parsnip::gen_additive_mod()}. 

Examples

```r
adjust_deg_free()
```
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()

bart-param

Parameters for BART models These parameters are used for constructing Bayesian adaptive regression tree (BART) models.

Description

Parameters for BART models These parameters are used for constructing Bayesian adaptive regression tree (BART) models.

Usage

prior_terminal_node_coef(range = c(0, 1), trans = NULL)
prior_terminal_node_expo(range = c(1, 3), trans = NULL)
prior_outcome_range(range = c(0, 5), trans = NULL)

Arguments

range A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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

These parameters are often used with Bayesian adaptive regression trees (BART) via parsnip::bart().
class_weights  

Parameters for class weights for imbalanced problems

Description
This parameter can be used to moderate how much influence certain classes receive during training.

Usage

```r
class_weights(range = c(1, 10), trans = NULL)
```

Arguments

- `range`: A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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

Used in `brulee::brulee_logistic_reg()` and `brulee::brulee_mlp()`

Examples

```r
class_weights()
```

conditional_min_criterion

Parameters for possible engine parameters for partykit models

Description
Parameters for possible engine parameters for partykit models

Usage

```r
conditional_min_criterion(
  range = c(1.386294, 15),
  trans = scales::logit_trans()
)
```

values_test_type
confidence_factor

conditional_test_type(values = values_test_type)
values_test_statistic
conditional_test_statistic(values = values_test_statistic)

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 character string of possible values.

Format

An object of class character of length 4.
An object of class character of length 2.

Details

The range of conditional_min_criterion() corresponds to roughly 0.80 to 0.99997 in the natural units. For several test types, this parameter corresponds to 1 - \{p-value\}.

Value

For the functions, they return a function with classes "param" and either "quant_param" or "qual_param".

confidenc_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())
noglobal_pruning(values = c(TRUE, FALSE))
predictor_winnowing(values = c(TRUE, FALSE))
fuzzy_thresholding(values = c(TRUE, FALSE))
rulbands(range = c(2L, 500L), trans = NULL)
Arguments
range A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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.

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
cost()
svm_margin()
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

```r
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. If a transformation is specified, these values should be in the 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

`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

```r
degree()
degree_int()
spline_degree()
prod_degree()
```
**dist_power**

### deg_free

**Description**

The number of degrees of freedom used for model parameters.

**Usage**

```r
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. If a transformation is specified, these values should be in the *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**

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

**Examples**

```r
deg_free()
```

### dist_power

**Description**

Used in `parsnip::nearest_neighbor()`.

**Usage**

```r
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. If a transformation is specified, these values should be in the *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 controls how distances are calculated. For example, dist_power = 1 corresponds to Manhattan distance while dist_power = 2 is Euclidean distance.

Examples

dist_power()

<table>
<thead>
<tr>
<th>dropout</th>
<th>Neural network parameters</th>
</tr>
</thead>
</table>

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. If a transformation is specified, these values should be in the 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

- 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("Cubist", ...).

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. If a transformation is specified, these values should be in the 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.

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.
finalize

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

```r
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
```
freq_cut

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. If a transformation is specified, these values should be in the 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
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 be 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:

```r
mixture()

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

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

## [1] 0.0001530482 0.9999530388

However, in some cases, the `parsnip` and `recipe` packages overrides the default ranges for specific models and preprocessing steps. If the grid function uses a parameter's object created from a model or recipe, the ranges may have different defaults (specific to those models). Using the example above, the `mixture` argument above is different for `glmnet` models:

```r
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_latin_hypercube(extract_parameter_set_dials(glmn_mod), size = 1000)
range(mix_grid_2$mixture)

## [1] 0.0501454 0.9999554

References


**Examples**

```r
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**

```r
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)
```

```r
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 be 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 parsnip and recipe packages overrides the default ranges for specific models and preprocessing steps. If the grid function uses a parameter's object created from a model or recipe, the ranges may 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(extract_parameter_set_dials(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())
grd_regular(p)
grd_regular(p, filter = penalty <= .01)

# Will fail due to unknowns:
# grid_regular(mtry(), min_n())
grd_regular(penalty(), mixture())
grd_regular(penalty(), mixture(), levels = 3:4)
grd_regular(penalty(), mixture(), levels = c(mixture = 4, penalty = 3))
grd_random(penalty(), mixture())

harmonic_frequency  Harmonic Frequency

Description

Used in recipes::step_harmonic().

Usage

harmonic_frequency(range = c(0.01, 1), trans = NULL)

Arguments

range  A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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.

Examples

harmonic_frequency()
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. If a transformation is specified, these values should be in the 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 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. If a transformation is specified, these values should be in the *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**

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**

```r
learn_rate()
```

---

```r
table(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**

```r
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. If a transformation is specified, these values should be in the *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*.

**Examples**

```r
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**

```r
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. If a transformation is specified, these values should be in the *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**

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

**Examples**

```r
max_num_terms()
```

---

**max_times**

Word frequencies for removal

---

**Description**

Used in `textrecipes::step_tokenfilter()`.

**Usage**

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

```r
min_times(range = c(0L, 1000L), trans = NULL)
```
**max_tokens**

**Arguments**

- **range**
  A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *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*.

**Examples**

```r
max_tokens()
min_tokens()
```

<table>
<thead>
<tr>
<th>max_tokens</th>
<th>Maximum number of retained tokens</th>
</tr>
</thead>
</table>

**Description**

Used in `textrecipes::step_tokenfilter()`.

**Usage**

```r
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. If a transformation is specified, these values should be in the *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*.

**Examples**

```r
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. If a transformation is specified, these values should be in the 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.

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. If a transformation is specified, these values should be in the 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.

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**

```r
mixture(range = c(0, 1), trans = NULL)
```

**Arguments**

- **range**: A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *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. 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**

```r
mixture()
```

**momentum**

*Gradient descent momentum parameter*

**Description**

A useful parameter for neural network models using gradient descent.

**Usage**

```r
momentum(range = c(0, 1), trans = NULL)
```
Arguments

range  A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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.

Examples

momentum()

---

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. If a transformation is specified, these values should be in the 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::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.
Interpretation

\texttt{mtry\_prop()} is a variation on \texttt{mtry()} where the value is interpreted as the \textit{proportion} of predictors that will be randomly sampled at each split rather than the \textit{count}.

This parameter is not intended for use in accommodating engines that take in this argument as a proportion; \texttt{mtry} is often a main model argument rather than an engine-specific argument, and thus should not have an engine-specific interface.

When wrapping modeling engines that interpret \texttt{mtry} in its sense as a proportion, use the \texttt{mtry()} parameter in \texttt{parsnip::set\_model\_arg()} and process the passed argument in an internal wrapping function as \texttt{mtry / number\_of\_predictors}. In addition, introduce a logical argument \texttt{counts} to the wrapping function, defaulting to \texttt{TRUE}, that indicates whether to interpret the supplied argument as a count rather than a proportion.

For an example implementation, see \texttt{parsnip::xgb\_train()}.

See Also

\texttt{mtry\_prop}

Examples

\begin{verbatim}
  mtry(c(1L, 10L)) # in original units
  mtry_long(c(0, 5)) # in log10 units
\end{verbatim}

---

**mtry\_prop**

\textit{Proportion of Randomly Selected Predictors}

Description

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

Usage

\begin{verbatim}
  mtry_prop(range = c(0.1, 1), trans = NULL)
\end{verbatim}

Arguments

- \texttt{range} A two-element vector holding the \textit{defaults} for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the \textit{transformed units}.

- \texttt{trans} A \texttt{trans} object from the \texttt{scales} package, such as \texttt{scales::log10\_trans()} or \texttt{scales::reciprocal\_trans()}. If not provided, the default is used which matches the units used in \texttt{range}. If no transformation, \texttt{NULL}.

Value

A \texttt{dials} object with classes "quant\_param" and "param". The \texttt{range} element of the object is always converted to a list with elements "lower" and "upper".
Interpretation

\texttt{mtry\_prop()} is a variation on \texttt{mtry()} where the value is interpreted as the \textit{proportion} of predictors that will be randomly sampled at each split rather than the \textit{count}.

This parameter is not intended for use in accommodating engines that take in this argument as a proportion; \texttt{mtry} is often a main model argument rather than an engine-specific argument, and thus should not have an engine-specific interface.

When wrapping modeling engines that interpret \texttt{mtry} in its sense as a proportion, use the \texttt{mtry()} parameter in \texttt{parsnip::set\_model\_arg()} and process the passed argument in an internal wrapping function as \texttt{mtry / number\_of\_predictors}. In addition, introduce a logical argument \texttt{counts} to the wrapping function, defaulting to \texttt{TRUE}, that indicates whether to interpret the supplied argument as a count rather than a proportion.

For an example implementation, see \texttt{parsnip::xgb\_train()}.

See Also

\texttt{mtry}, \texttt{mtry\_long}

Examples

\texttt{mtry\_prop()}

\begin{tabular}{ll}
neighbors & \textit{Number of neighbors} \\
\hline
\end{tabular}

Description

The number of neighbors is used for models (\texttt{parsnip::nearest\_neighbor()}), imputation (\texttt{recipes::step\_impute\_knn()}), and dimension reduction (\texttt{recipes::step\_isomap()}).

Usage

\texttt{neighbors(range = c(1L, 10L), trans = NULL)}

Arguments

\begin{itemize}
  \item \texttt{range} A two-element vector holding the \textit{defaults} for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the \textit{transformed units}.
  \item \texttt{trans} A trans object from the scales package, such as \texttt{scales::log10\_trans()} or \texttt{scales::reciprocal\_trans()}. If not provided, the default is used which matches the units used in range. If no transformation, \texttt{NULL}.
\end{itemize}

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()

Description

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

Usage

```r
new_quant_param(
  type = c("double", "integer"),
  range = NULL,
  inclusive = NULL,
  default = deprecated(),
  trans = NULL,
  values = NULL,
  label = NULL,
  finalize = NULL,
  ...
  call = caller_env()
)
```

```r
new_qual_param(
  type = c("character", "logical"),
  values,
  default = deprecated(),
  label = NULL,
  finalize = NULL,
  ...
  call = caller_env()
)
```

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. If values is supplied, and range is NULL, range will be set to `range(values)`. 
inclusive  A two-element logical vector for whether the range values should be inclusive or exclusive. If values is supplied, and inclusive is NULL, inclusive will be set to c(TRUE, TRUE).

default    [Deprecated] No longer used. If a value is supplied, it will be ignored and a warning will be thrown.

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, this can be used as an alternative to range and inclusive. If set, these will be used by value_seq() and value_sample().

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).

...        These dots are for future extensions and must be empty.

call       The call passed on to rlang::abort().

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**

```r
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. If a transformation is specified, these values should be in the *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`.

**Examples**

```r
num_breaks()
```

### num_clusters

**Number of Clusters**

**Description**

Used in most `tidyclust` models.

**Usage**

```r
num_clusters(range = c(1L, 10L), trans = NULL)
```

**Arguments**

- `range`: A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *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`.

**Examples**

```r
num_clusters()
```
num_comp

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of derived predictors from models or feature engineering methods.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_comp(range = c(1L, unknown()), trans = NULL)</td>
</tr>
<tr>
<td>num_terms(range = c(1L, unknown()), trans = NULL)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
</tr>
<tr>
<td>trans</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
<tr>
<td>The difference between num_comp() and num_terms() is semantics.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_terms()</td>
</tr>
<tr>
<td>num_terms(c(2L, 10L))</td>
</tr>
</tbody>
</table>

num_hash

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used in textrecipes::step_texthash().</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_hash(range = c(8L, 12L), trans = log2_trans())</td>
</tr>
<tr>
<td>signed_hash(values = c(TRUE, FALSE))</td>
</tr>
</tbody>
</table>
Argument

range A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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.

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

Examples

num_hash()
signed_hash()

---

num_knots  Number of knots (integer)

Description

The number of knots used for spline model parameters.

Usage

num_knots(range = c(0L, 5L), trans = NULL)

Arguments

range A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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

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

Examples

num_knots()
num_leaves

**Possible engine parameters for lightgbm**

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

**Usage**

```r
num_leaves(range = c(5, 100), trans = NULL)
```

**Arguments**
- `range`: A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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**

"lightgbm" is an available engine in the parsnip extension package `bonsai`. For more information, see the lightgbm webpage.

**Examples**

```r
num_leaves()
```

---

num_runs

**Number of Computation Runs**

**Description**

Used in `recipes::step_nnmf()`.

**Usage**

```r
num_runs(range = c(1L, 10L), trans = NULL)
```
num_tokens

Arguments

range  A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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.

Examples

num_tokens()

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. If a transformation is specified, these values should be in the 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.

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. If a transformation is specified, these values should be in the 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
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, ...)

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.

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. If a transformation is specified, these values should be in the 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()
**prior_slab_dispersion**

Bayesian PCA parameters

Description

A numeric parameter function representing parameters for the spike-and-slab prior used by `embed::step_pca_sparse_bayes()`.

Usage

```r
prior_slab_dispersion(range = c(-1/2, log10(3)), trans = log10_trans())
prior_mixture_threshold(range = c(0, 1), trans = NULL)
```
prune_method

Arguments

range  A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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

prior_slab_dispersion() is related to the prior for the case where a PCA loading is selected (i.e. non-zero). Smaller values result in an increase in zero coefficients.
prior_mixture_threshold() is used to threshold the prior to determine which parameters are non-zero or zero. Increasing this parameter increases the number of zero coefficients.

Examples

mixture()

---

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()
Description

Setters, getters, and validators for parameter ranges.

Usage

range_validate(object, range, ukn_ok = TRUE, ..., call = caller_env())

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.
- **...**: These dots are for future extensions and must be empty.
- **call**: The call passed on to `rlang::abort()`.
- **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

```r
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**

```r
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. If a transformation is specified, these values should be in the 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**

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

**Examples**

```r
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**

```r
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. If a transformation is specified, these values should be in the *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`.
- `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()

regularization_method(values = values_regularization_method)

Arguments

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

Format

An object of class character of length 4.

Details

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

Examples

values_regularization_method
regularization_method()
scale_pos_weight  Parameters for possible engine parameters for xgboost

Description

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

Usage

scale_pos_weight(range = c(0.8, 1.2), trans = NULL)
penalty_L2(range = c(-10, 1), trans = log10_trans())
penalty_L1(range = c(-10, 1), trans = log10_trans())

Arguments

range  A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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

For more information, see the xgboost webpage.

Examples

scale_pos_weight()
penalty_L2()
penalty_L1()

scheduler-param  Parameters for neural network learning rate schedulers These parameters are used for constructing neural network models.

Description

Parameters for neural network learning rate schedulers These parameters are used for constructing neural network models.
Usage

rate_initial(range = c(-3, -1), trans = log10_trans())

rate_largest(range = c(-1, -1/2), trans = log10_trans())

rate_reduction(range = c(1/5, 1), trans = NULL)

rate_steps(range = c(2, 10), trans = NULL)

rate_step_size(range = c(2, 20), trans = NULL)

rate_decay(range = c(0, 2), trans = NULL)

rate_schedule(values = values_scheduler)

values_scheduler

Arguments

- **range**: A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *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.

- **values**: A character string of possible values. See values_scheduler in examples below.

Format

An object of class character of length 5.

Details

These parameters are often used with neural networks via parsnip::mlp(engine = "brulee").

The details for how the *brulee* schedulers change the rates:

- **schedule_decay_time()**: $rate(epoch) = initial/(1 + decay \times epoch)$
- **schedule_decay_expo()**: $rate(epoch) = initial \exp(-decay \times epoch)$
- **schedule_step()**: $rate(epoch) = initial \times reduction^{floor(epoch/steps)}$
- **schedule_cyclic()**: $cycle = floor(1+(epoch/2/stepsize)), x = abs((epoch/stepsize)- (2 \times cycle) + 1), and rate(epoch) = initial + (largest - initial) \times max(0, 1 - x)$
### select_features

**Parameter to enable feature selection**

**Description**

Used in `parsnip::gen_additive_mod()`.

**Usage**

```r
select_features(values = c(TRUE, FALSE))
```

**Arguments**

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

**Examples**

```r
select_features()
```

---

### shrinkage_correlation

**Parameters for possible engine parameters for sda models**

**Description**

These functions can be used to optimize engine-specific parameters of `sda::sda()` via `parsnip::discrim_linear()`.

**Usage**

```r
shrinkage_correlation(range = c(0, 1), trans = NULL)
shrinkage_variance(range = c(0, 1), trans = NULL)
shrinkage_frequencies(range = c(0, 1), trans = NULL)
diagonal_covariance(values = c(TRUE, FALSE))
```

**Arguments**

- `range` A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *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`.
- `values` A vector of possible values (TRUE or FALSE).
smoothness

Details

These functions map to sda::sda() arguments via:

- shrinkage_correlation() to lambda
- shrinkage_variance() to lambda.var
- shrinkage_frequencies() to lambda.freqs
- diagonal_covariance() to diagonal

Value

For the functions, they return a function with classes "param" and either "quant_param" or "qual_param".

smoothness

<table>
<thead>
<tr>
<th>smoothness</th>
<th>Kernel Smoothness</th>
</tr>
</thead>
</table>

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. If a transformation is specified, these values should be in the 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.

Examples

smoothness()
stop_iter                   Early stopping parameter

Description

For some models, the effectiveness of the model can decrease as training iterations continue. `stop_iter()` can be used to tune how many iterations without an improvement in the objective function occur before training should be halted.

Usage

`stop_iter(range = c(3L, 20L), trans = NULL)`

Arguments

- `range`: A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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.

Examples

`stop_iter()`

summary_stat               Rolling summary statistic for moving windows

Description

This parameter is used in `recipes::step_window()`.

Usage

`summary_stat(values = values_summary_stat)`

values_summary_stat

Arguments

- `values`: A character string of possible values. See `values_summary_stat` in examples below.
survival_link

Format

An object of class character of length 8.

Examples

values_summary_stat
summary_stat()

survival_link  Survival Model Link Function

Description

Survival Model Link Function

Usage

survival_link(values = values_survival_link)

values_survival_link

Arguments

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

Format

An object of class character of length 3.

Details

This parameter is used in parsnip::set_engine('flexsurvspline').

Examples

values_survival_link
survival_link()
surv_dist  
*Parametric distributions for censored data*

**Description**

Parametric distributions for censored data

**Usage**

```
surv_dist(values = 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::survival_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. If a transformation is specified, these values should be in the *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.

### Examples

threshold()

---

<table>
<thead>
<tr>
<th>token</th>
<th>Token types</th>
</tr>
</thead>
<tbody>
<tr>
<td>token</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Token types

**Usage**

```r
token(values = 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**

```r
values_token
token()
```
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. If a transformation is specified, these values should be in the 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`.
- `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()`).
trim_amount

- 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. (Seeparsnip::boost_tree()). This corresponds to gamma in xgboost.
- tree_depth(): The maximum depth of the tree (i.e. number of splits). (Seeparsnip::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
update.parameters

Update a single parameter in a parameter set

Description

Update a single parameter in a parameter set

Usage

```r
## 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

```r
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)))
```

validation_set_prop

Proportion of data used for validation

Description

Used in embed::step_discretize_xgb().

Usage

```r
validation_set_prop(range = c(0.05, 0.7), trans = NULL)
```
Arguments

range A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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.

Examples

validation_set_prop()

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, ..., call = caller_env())
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.
... These dots are for future extensions and must be empty.
call The call passed on to rlang::abort().
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.

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

```r
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)
```
vocabulary_size

Description

Used in textrecipes::step_tokenize_sentencepiece() and textrecipes::step_tokenize_bpe().

Usage

vocabulary_size(range = c(1000L, 32000L), trans = NULL)

Arguments

range A two-element vector holding the defaults for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the 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.

Examples

vocabulary_size()

weight

Description

Parameter for "double normalization" when creating token counts

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. If a transformation is specified, these values should be in the 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.

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
window_size

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. If a transformation is specified, these values should be in the 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.

Examples

window_size()
Index

* datasets
  activation, 3
  conditional_min_criterion, 6
  prune_method, 41
  regularization_factor, 44
  regularization_method, 45
  scheduler-param, 46
  summary_stat, 50
  surv_dist, 52
  survival_link, 51
  token, 53
  weight_func, 61
  weight_scheme, 61

activation, 3
adjust_deg_free, 4
all_neighbors, 5

bart-param, 5
batch_size (dropout), 11

class_weights, 6
conditional_min_criterion, 6
conditional_test_statistic
  (conditional_min_criterion), 6
  conditional_test_type
    (conditional_min_criterion), 6
  confidence_factor, 7
  cost, 8
  cost_complexity (trees), 54

deg_free, 10
degree, 9
degree_int (degree), 9
diagonal_covariance
  (shrinkage_correlation), 48
dist_power, 10
dropout, 11
epochs (dropout), 11
extrapolation, 12
finalize, 12
freq_cut, 15
fuzzy_thresholding (confidence_factor), 7
get_batch_sizes (finalize), 12
get_log_p (finalize), 12
get_n (finalize), 12
get_n_frac (finalize), 12
get_n_frac_range (finalize), 12
get_p (finalize), 12
get_rbf_range (finalize), 12
grid_latin_hypercube
  (grid_max_entropy), 16
  grid_max_entropy, 16
  grid_random (grid_regular), 19
  grid_regular, 19
harmonic_frequency, 21
has_unknowns (unknown), 56
hidden_units (dropout), 11
is_unknown (unknown), 56
kernel_offset (rbf_sigma), 43
kernlab::sigest(), 13, 14
Laplace, 22
learn_rate, 22
loss_reduction (trees), 54
lower_quantile (regularization_factor), 44
max_nodes, 23
max_num_terms, 24
max_rules (extrapolation), 12
max_times, 24
max_tokens, 25
min_dist, 26
min_n (trees), 54
min_times (max_times), 24
INDEX

min_unique, 26
mixture, 27
momentum, 27
mtry, 28
mtry(), 17, 20, 29–31
mtry_long (mtry), 28
mtry_prop, 29
mtry_prop(), 29, 30

neighbors, 30
new-param, 31
new_qual_param (new-param), 31
new_quant_param (new-param), 31
no_global_pruning (confidence_factor), 7
num_breaks, 33
num_clusters, 33
num_comp, 34
num_hash, 34
num_knots, 35
num_leaves, 36
num_random_splits
  (regularization_factor), 44
num_runs, 36
num_terms (num_comp), 34
num_tokens, 37

over_ratio, 38
parameters, 38
penalty, 39
penalty(), 17, 20
penalty_L1 (scale_pos_weight), 46
penalty_L2 (scale_pos_weight), 46
predictor_prop, 40
predictor_winnowing
  (confidence_factor), 7
prior_mixture_threshold
  (prior_slab_dispersion), 40
prior_outcome_range (bart-param), 5
prior_slab_dispersion, 40
prior_terminal_node_coeff (bart-param), 5
prior_terminal_node_exp (bart-param), 5
prod_degree (degree), 9
prune (trees), 54
prune_method, 41

range_get (range_validate), 42
range_set (range_validate), 42
range_validate, 42
ranger_class_rules
  (regularization_factor), 44
ranger_reg_rules
  (regularization_factor), 44
ranger_split_rules
  (regularization_factor), 44
rate_decay (scheduler-param), 46
rate_initial (scheduler-param), 46
rate_largest (scheduler-param), 46
rate_reduction (scheduler-param), 46
rate_schedule (scheduler-param), 46
rate_step_size (scheduler-param), 46
rate_steps (scheduler-param), 46
rbf_sigma, 43
regularization_factor, 44
regularization_method, 45
regularize_depth
  (regularization_factor), 44
rlang::abort(), 32, 42, 58
rule_bands (confidence_factor), 7
sample_prop (trees), 54
sample_size (trees), 54
scale_factor (rbf_sigma), 43
scale_pos_weight, 46
scales::log10_trans(), 32
scales::reciprocal_trans(), 32
scales::trans_new(), 32
scheduler-param, 46
select_features, 48
shrinkage_correlation, 48
shrinkage_frequencies
  (shrinkage_correlation), 48
shrinkage_variance
  (shrinkage_correlation), 48
signed_hash (num_hash), 34
significance_threshold
  (regularization_factor), 44
smoothness, 49
spline_degree (degree), 9
splitting_rule (regularization_factor), 44
stop_iter, 50
summary_stat, 50
surv_dist, 52
survival_link, 51
svm_margin (cost), 8
threshold, 52

ranger_class_rules
  (regularization_factor), 44
ranger_reg_rules
  (regularization_factor), 44
ranger_split_rules
  (regularization_factor), 44
rate_decay (scheduler-param), 46
rate_initial (scheduler-param), 46
rate_largest (scheduler-param), 46
rate_reduction (scheduler-param), 46
rate_schedule (scheduler-param), 46
rate_step_size (scheduler-param), 46
rate_steps (scheduler-param), 46
rbf_sigma, 43
regularization_factor, 44
regularization_method, 45
regularize_depth
  (regularization_factor), 44
rlang::abort(), 32, 42, 58
rule_bands (confidence_factor), 7
sample_prop (trees), 54
sample_size (trees), 54
scale_factor (rbf_sigma), 43
scale_pos_weight, 46
scales::log10_trans(), 32
scales::reciprocal_trans(), 32
scales::trans_new(), 32
scheduler-param, 46
select_features, 48
shrinkage_correlation, 48
shrinkage_frequencies
  (shrinkage_correlation), 48
shrinkage_variance
  (shrinkage_correlation), 48
signed_hash (num_hash), 34
significance_threshold
  (regularization_factor), 44
smoothness, 49
spline_degree (degree), 9
splitting_rule (regularization_factor), 44
stop_iter, 50
summary_stat, 50
surv_dist, 52
survival_link, 51
svm_margin (cost), 8
threshold, 52