Package 'dials'

November 11, 2026

Title Tools for Creating Tuning Parameter Values

Version 1.4.5

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

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

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

Depends R (>= 4.1), scales (>= 1.3.0)

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

Suggests covr, ggplot2, kernlab, knitr, rmarkdown, rpart, testthat (>= 3.1.9), xml2

VignetteBuilder knitr

ByteCompile true

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-23

Encoding UTF-8

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Max Kuhn [aut],
Hannah Frick [aut, cre],
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Hannah Frick <hannah@posit.co>

Repository CRAN

Date/Publication 2026-11-11 09:20:02 UTC

Contents

activation . . . 4
adjust_deg_free . . . 4
all_neighbors . . . 5
attention-param . . . 6
bart-param . . . 7
buffer . . . 8
cal_method_class . . . 8
class_weights . . . 9
conditional_min_criterion . . . 10
confidence_factor . . . 11
cost . . . 12
degree . . . 12
deg_free . . . 13
dist_power . . . 14
dropout . . . 14
extrapolation . . . 16
finalize . . . 16
freq_cut . . . 19
grid_hybrid . . . 20
grid_regular . . . 22
grid_space_filling . . . 25
harmonic_frequency . . . 29
initial_umap . . . 29
l2_leaf_reg . . . 30
Laplace . . . 32
learn_rate . . . 32
max_nodes . . . 33
max_num_terms . . . 34
max_times . . . 34
max_tokens . . . 35
min_dist . . . 36
min_unique . . . 36
mixture . . . 37
momentum . . . 38
mtry . . . 38
mtry_prop . . . 39
neighbors . . . 40
new-param . . . 41
num_breaks . . . 43
num_clusters . . . 44
num_comp . . . 44
num_hash . . . 45
num_knots . . . 46
num_leaves . . . 46
num_runs . . . 47
num_tokens . . . 47

ordinal_link . . . 48
over_ratio . . . 49
parallel_reg . . . 49
parameters . . . 50
penalty . . . 51
predictor_prop . . . 52
prior_slab_dispersion . . . 52
prop_terms . . . 53
prune_method . . . 54
range_limits . . . 54
range_validate . . . 55
rbf_sigma . . . 56
regularization_factor . . . 57
regularization_method . . . 58
resnet-param . . . 59
rln-param . . . 59
scale_pos_weight . . . 61
scheduler-param . . . 61
select_features . . . 62
shrinkage_correlation . . . 63
smoothness . . . 64
stop_iter . . . 64
summary_stat . . . 65
survival_link . . . 65
surv_dist . . . 66
tab-pfn-param . . . 66
target_weight . . . 67
threshold . . . 68
threshold_structure . . . 68
token . . . 69
trees . . . 70
trim_amount . . . 71
unknown . . . 72
update.parameters . . . 73
validation_set_prop . . . 73
value_validate . . . 74
vocabulary_size . . . 76
weight . . . 76
weight_func . . . 77
weight_scheme . . . 77
window_size . . . 78

Index **79**

activation *Activation functions between network layers*

Description

Activation functions between network layers

Usage

```
activation(values = values_activation)

activation_2(values = values_activation)

values_activation
```

Arguments

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

Details

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

Examples

```
values_activation
activation()
```

adjust_deg_free *Parameters to adjust effective degrees of freedom*

Description

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

Usage

```
adjust_deg_free(range = c(0.25, 4), 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::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Details

Used in parsnip::gen_additive_mod().

Examples

```
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()
```

attention-param *Parameters for attention-based tabular models*

Description

These parameters are used with attention-based (transformer) tabular models, such as `brulee_saint()` and `tabular_auto_int()` when fit with the `brulee` engine.

Usage

```
attention_type(values = values_attention_type)

values_attention_type

dropout_attn(range = c(0, 0.5), trans = NULL)

num_attn_heads(range = c(1L, 8L), trans = NULL)

num_attn_blocks(range = c(1L, 6L), trans = NULL)

num_attn_feat(range = c(8L, 64L), trans = NULL)

target_token(values = c(TRUE, FALSE))

normalization(values = values_normalization)

values_normalization
```

Arguments

`values`	A character string of possible values. See `values_attention_type` in examples below.
`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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Details

- `attention_type()`: The type of attention mechanism to use.
- `dropout_attn()`: The proportion of attention weights to randomly set to zero during model training.
- `num_attn_heads()`: The number of parallel attention mechanisms (heads) in the multi-head attention layer. Multiple attention heads allow the model to attend to different aspects of the input features simultaneously.

- num_attn_blocks(): The number of sequential attention blocks in the model architecture. Each attention block consists of a multi-head attention layer followed by feed-forward layers.
- num_attn_feat(): The dimensionality of the feature space used in the attention mechanism.
- target_token(): A logical to specify whether the SAINT model should add a supervised token to the embeddings.

Examples

```
values_attention_type
attention_type()
num_attn_heads()
target_token()
```

---

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::transform_log10() or scales::transform_reciprocal(). 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().

buffer *Buffer size*

Description

In equivocal zones, predictions are considered equivocal (i.e. "could go either way") if their probability falls within some distance on either side of the classification threshold. That distance is called the "buffer."

Usage

```
buffer(range = c(0, 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::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Details

A buffer of .5 is only possible if the classification threshold is .5. In that case, all probability predictions are considered equivocal, regardless of their value in [0, 1]. Otherwise, the maximum buffer is min(threshold, 1 - threshold).

See Also

threshold()

Examples

```
buffer()
```

cal_method_class *Methods for model calibration*

Description

Methods for model calibration

Usage

```
cal_method_class(values = values_cal_cls)

cal_method_reg(values = values_cal_reg)

values_cal_cls

values_cal_reg
```

Arguments

values	A character string of possible values. See values_cal_cls and values_cal_reg (in examples below).

Details

This parameter is used in postprocessing methods in the tailor package.

Examples

```
values_cal_cls
cal_method_class()

values_cal_reg
cal_method_reg()
```

---

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

```
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::transform_log10() or scales::transform_reciprocal(). 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

```
class_weights()
```

`conditional_min_criterion` *Parameters for possible engine parameters for partykit models*

Description

Parameters for possible engine parameters for partykit models

Usage

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

values_test_type

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::transform_log10()` or `scales::transform_reciprocal()`. 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.

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".

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 = transform_log10())

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. If a transformation is specified, these values should be in the *transformed units*.
trans	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. 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 = transform_log2())

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. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). 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. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). 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. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). 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. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. 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)

hidden_units_2(range = c(1L, 10L), trans = NULL)

batch_size(range = c(2L, 7L), trans = transform_log2())

dropout_hidden(range = c(0, 0.5), trans = NULL)

dropout_last(range = c(0, 0.5), trans = NULL)

num_embedding(range = c(8L, 64L), trans = NULL)

dropout_embedding(range = c(0, 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::transform_log10() or scales::transform_reciprocal(). 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.
- dropout_hidden(): The proportion of hidden-layer units to randomly set to zero during model training.
- dropout_last(): The proportion of final-layer units to randomly set to zero during model training.
- num_embedding(): The dimensionality of the embedding space for features.
- dropout_embedding(): The proportion of embedding values to randomly set to zero during model training.

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. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. 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), ...)
```

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. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. 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_hybrid *Hybrid space-filling/regular parameter grids*

Description

A grid can be created that is efficient for some parameter(s) but fine-grained for others. This can be especially helpful when a model has a "submodel" parameter that can be tuned with virtually no cost.

Usage

```
grid_hybrid(
  x,
  ...,
  sfd_size = 10,
  sfd_type = "any",
  reg_parameters = NULL,
  reg_levels = 20,
  original = TRUE
)

## Default S3 method:
grid_hybrid(
  x,
  ...,
  sfd_size = 10,
  sfd_type = "any",
  reg_parameters = NULL,
  reg_levels = 20,
  original = TRUE
)

## S3 method for class 'parameters'
grid_hybrid(
  x,
  ...,
  sfd_size = 10,
  sfd_type = "any",
  reg_parameters = NULL,
  reg_levels = 20,
  original = TRUE
)

## S3 method for class 'list'
grid_hybrid(
  x,
  ...,
```

```
  sfd_size = 10,
  sfd_type = "any",
  reg_parameters = NULL,
  reg_levels = 20,
  original = TRUE
)

## S3 method for class 'param'
grid_hybrid(
  x,
  ...,
  sfd_size = 10,
  sfd_type = "any",
  reg_parameters = NULL,
  reg_levels = 20,
  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.
sfd_size	An integer for the maximum size of the space-filling portion of the design.
sfd_type	The type of space-filling design. See the type argument of grid_space_filling() for possible values.
reg_parameters	<tidy-select> The parameter(s) used to make the regular portion of the grid, selected by their identifiers. If no parameters are selected, a space-filling design with sfd_size candidates is created. If all parameters are selected, a regular grid with reg_levels^p candidates is created where p is the number of selected parameters.
reg_levels	The number of values for *each* regular grid parameter.
original	A logical: should the parameters be in the original units or in the transformed space (if any)?

Details

This function first creates a space-filling design for the parameters that are not selected by reg_parameters (with sfd_size total candidates). Then a regular grid is created with reg_levels^p candidates where p is the number of parameters selected by reg_parameters. These two grids are crossed to produce the end result.

Examples

```
if (rlang::is_installed("ggplot2")) {

  library(ggplot2)
```

```
  # Most boosting methods can make many predictions across a number of trees
  # from a single model fit object. Those are nearly free (computationally) so
  # we would do many tree values for each of the other parameters.

  # To illustrate, we'll only show four tree values:

  boost_example <-
    parameters(list(trees = trees(), learn_rate = learn_rate(), min_n = min_n()))

  boost_example |>
    grid_hybrid(reg_parameters = trees, sfd_size = 20, reg_levels = 4) |>
    ggplot(aes(learn_rate, min_n)) +
    geom_point() +
    facet_wrap(~ trees, labeller = "label_both") +
    scale_x_log10()

  # In other cases, we have 1+ parameters with very few values. We can make a
  # small regular grid over these and a much larger space-filling design

  nnet_example <-
    parameters(
      list(
        learn_rate = learn_rate(),
        dropout = dropout(),
        # Only a few values:
        activation = activation(c("relu", "tanh", "elu")),
        schedule = rate_schedule(c("none", "cyclic", "step")) # note different name
      )
    )

  nnet_example |>
    grid_hybrid(
      reg_parameters = c(activation, schedule),
      sfd_size = 20,
      reg_levels = 3
    ) |>
    ggplot(aes(learn_rate, dropout)) +
    geom_point() +
    facet_grid(activation ~ schedule, labeller = "label_both") +
    scale_x_log10()
}
```

`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)

## Default S3 method:
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)

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

## Default S3 method:
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)
```

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 `parsnip` and `recipe` packages overrides the default ranges for specific models and preprocessing steps. If the grid function uses a `parameters` 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())
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())
```

`grid_space_filling` *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 does not have an observed combination that is unnecessarily close to any other point.

Usage

```
grid_space_filling(x, ..., size = 5, type = "any", original = TRUE)

## Default S3 method:
grid_space_filling(x, ..., size = 5, type = "any", original = TRUE)

## S3 method for class 'parameters'
grid_space_filling(
  x,
  ...,
  size = 5,
  type = "any",
  variogram_range = 0.5,
  iter = 1000,
  original = TRUE
)

## S3 method for class 'list'
grid_space_filling(
  x,
  ...,
  size = 5,
  type = "any",
  variogram_range = 0.5,
  iter = 1000,
  original = TRUE
)

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

```
  ...,
  size = 5,
  variogram_range = 0.5,
  iter = 1000,
  type = "any",
  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 maximum number of parameter value combinations returned. If duplicate combinations are generated from this size, the smaller, unique set is returned.
type	A character string with possible values: "any", "audze_eglais", "max_min_l1", "max_min_l2", "uniform", "max_entropy", or "latin_hypercube". A value of "any" will choose the first design available (in the order listed above, excluding "latin_hypercube"). For a single-point design, a random grid is created.
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. Only used for type = "max_entropy".
iter	An integer for the maximum number of iterations used to find a good design. Only used for type = "max_entropy".

Details

The types of designs supported here are latin hypercube designs of different types. The simple designs produced by grid_latin_hypercube() are space-filling but don't guarantee or optimize any other properties. grid_space_filling() might be able to produce designs that discourage grid points from being close to one another. There are a lot of methods for doing this, such as maximizing the minimum distance between points (see Husslage *et al* 2001). grid_max_entropy() attempts to maximize the determinant of the spatial correlation matrix between coordinates.

Latin hypercube and maximum entropy designs use random numbers to make the designs.

By default, grid_space_filling() will try to use a pre-optimized space-filling design from https://www.spacefillingde (see Husslage *et al*, 2011) or using a uniform design. If no pre-made design is available, then a maximum entropy design is created.

Also 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_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 `parameters` 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 <-
  glmn_mod |>
  extract_parameter_set_dials() |>
  grid_latin_hypercube(size = 1000)
range(mix_grid_2$mixture)


## [1] 0.0501454 0.9999554
```

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)

Husslage, B. G., Rennen, G., Van Dam, E. R., & Den Hertog, D. (2011). Space-filling Latin hypercube designs for computer experiments. *Optimization and Engineering*, 12, 611-630.

Fang, K. T., Lin, D. K., Winker, P., & Zhang, Y. (2000). Uniform design: Theory and application. _Technometric_s, 42(3), 237-248

Examples

```
grid_space_filling(
  hidden_units(),
  penalty(),
  epochs(),
  activation(),
  learn_rate(c(0, 1), trans = scales::transform_log()),
  size = 10,
  original = FALSE
)
# ------------------------------------------------------------------------------
# comparing methods

if (rlang::is_installed("ggplot2")) {

  library(dplyr)
  library(ggplot2)

  set.seed(383)
  parameters(trees(), mixture()) |>
    grid_space_filling(size = 25, type = "latin_hypercube") |>
    ggplot(aes(trees, mixture)) +
    geom_point() +
    lims(y = 0:1, x = c(1, 2000)) +
    ggtitle("latin hypercube")

  set.seed(383)
  parameters(trees(), mixture()) |>
    grid_space_filling(size = 25, type = "max_entropy") |>
    ggplot(aes(trees, mixture)) +
    geom_point() +
    lims(y = 0:1, x = c(1, 2000)) +
    ggtitle("maximum entropy")

  parameters(trees(), mixture()) |>
    grid_space_filling(size = 25, type = "audze_eglais") |>
    ggplot(aes(trees, mixture)) +
    geom_point() +
    lims(y = 0:1, x = c(1, 2000)) +
    ggtitle("Audze-Eglais")

  parameters(trees(), mixture()) |>
    grid_space_filling(size = 25, type = "uniform") |>
    ggplot(aes(trees, mixture)) +
    geom_point() +
    lims(y = 0:1, x = c(1, 2000)) +
    ggtitle("uniform")
}
```

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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Examples

```
harmonic_frequency()
```

initial_umap *Initialization method for UMAP*

Description

This parameter is the type of initialization for the UMAP coordinates. Can be one of `"spectral"`, `"normlaplacian"`, `"random"`, `"lvrandom"`, `"laplacian"`, `"pca"`, `"spca"`, or `"agspectral"`. See `uwot::umap()` for more details.

Usage

```
initial_umap(values = values_initial_umap)

values_initial_umap
```

Arguments

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

Details

This parameter is used in `embed::step_umap()`.

Examples

```
values_initial_umap
initial_umap()
```

`l2_leaf_reg` *Possible engine parameters for catboost*

Description

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

Usage

```
l2_leaf_reg(range = c(0, 1), trans = transform_log10())

max_leaves(range = c(10, 64), trans = NULL)

grow_policy(values = values_grow_policy)

values_grow_policy

one_hot_max_size(range = c(1L, 25L), trans = NULL)

score_randomness(range = c(0, 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::transform_log10()` or `scales::transform_reciprocal()`. 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_grow_policy` in examples below.

Details

"catboost" is an available engine in the parsnip extension package bonsai

The parameters are:

- `l2_leaf_reg()`: the L2 regularization coefficient for the leaf values.

- `max_leaves()`: the maximum number of terminal nodes in a tree.
- `grow_policy()`: the method for growing trees. `"SymmetricTree"` (the default in catboost) splits every node at a given depth using the same condition. `"Depthwise"` grows the tree level-by-level, choosing the best split for each node. `"Lossguide"` grows the tree leaf-by-leaf, splitting the leaf with the best loss improvement.
- `one_hot_max_size()`: categorical predictors with this many distinct values or fewer are one-hot encoded. Those with more values are encoded using target statistics.
- `score_randomness()`: a multiplier for the amount of Gaussian noise added to the splitting score (zero means no additional noise).

Conditional constraints:

Several catboost parameters are only used for specific values of other parameters. Using the dials parameter names:

- `max_leaves()` is only used when `grow_policy()` is `"Lossguide"`. catboost will fail with an error for other grow policies.
- `min_n()` (`min_data_in_leaf` in catboost) is only used when `grow_policy()` is `"Depthwise"` or `"Lossguide"`. For `"SymmetricTree"`, it is silently ignored.
- `tree_depth()` is used for all grow policies, but its effect differs. For `"SymmetricTree"`, each tree has exactly `2^tree_depth` terminal nodes. For `"Lossguide"`, it is a cap on the depth of any branch while `max_leaves()` controls the tree size. catboost's maximum depth is 16.
- `sample_prop()` (`subsample` in catboost) cannot be used with Bayesian bootstrapping. Bayesian bootstrapping is catboost's default for multiclass outcomes, so catboost will fail with an error in that case unless a different `bootstrap_type` is set as an engine argument.
- `one_hot_max_size()` only affects predictors that are passed to catboost as factors. If categorical predictors are converted to indicator variables beforehand (e.g., with a recipe), it has no effect.
- `mtry_prop()` (`rsm` in catboost) is only supported on the CPU, except for pairwise ranking models.

For more information, see the catboost webpage.

Examples

```
l2_leaf_reg()
max_leaves()
values_grow_policy
grow_policy()
one_hot_max_size()
score_randomness()
```

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::transform_log10() or scales::transform_reciprocal(). 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 = transform_log10())
```

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::transform_log10() or scales::transform_reciprocal(). 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. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). 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. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. 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. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). 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. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). 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 = transform_log10())
```

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::transform_log10()` or `scales::transform_reciprocal()`. 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. Used in `recipes::step_discretize()`.

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::transform_log10()` or `scales::transform_reciprocal()`. 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. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. 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()
```

momentum *Gradient descent momentum parameter*

Description

A useful parameter for neural network models using gradient descent

Usage

```
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::transform_log10() or scales::transform_reciprocal(). 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 = transform_log10())
```

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::transform_log10() or scales::transform_reciprocal(). 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

`mtry_prop()` is a variation on `mtry()` where the value is interpreted as the *proportion* of predictors that will be randomly sampled at each split rather than the *count*.

This parameter is not intended for use in accommodating engines that take in this argument as a proportion; `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 `mtry` in its sense as a proportion, use the `mtry()` parameter in `parsnip::set_model_arg()` and process the passed argument in an internal wrapping function as `mtry / number_of_predictors`. In addition, introduce a logical argument `counts` to the wrapping function, defaulting to `TRUE`, that indicates whether to interpret the supplied argument as a count rather than a proportion.

For an example implementation, see `parsnip::xgb_train()`.

See Also

mtry_prop

Examples

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

---

mtry_prop *Proportion of Randomly Selected Predictors*

Description

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

Usage

```
mtry_prop(range = c(0.1, 1), trans = NULL)
```

Arguments

range	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Value

A dials object with classes "quant_param" and "param". The range element of the object is always converted to a list with elements "lower" and "upper".

Interpretation

mtry_prop() is a variation on mtry() where the value is interpreted as the *proportion* of predictors that will be randomly sampled at each split rather than the *count*.

This parameter is not intended for use in accommodating engines that take in this argument as a proportion; 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 mtry in its sense as a proportion, use the mtry() parameter in parsnip::set_model_arg() and process the passed argument in an internal wrapping function as mtry / number_of_predictors. In addition, introduce a logical argument counts to the wrapping function, defaulting to TRUE, that indicates whether to interpret the supplied argument as a count rather than a proportion.

For an example implementation, see parsnip::xgb_train().

See Also

mtry, mtry_long

Examples

```
mtry_prop()
```

---

neighbors *Number of neighbors*

Description

The number of neighbors is used for models (parsnip::nearest_neighbor()), imputation (recipes::step_impute_knn()
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. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). 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 = NULL,
  inclusive = NULL,
  default = deprecated(),
  trans = NULL,
  values = NULL,
  label = NULL,
  finalize = NULL,
  ...,
  call = caller_env()
)

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::transform_log() or scales::transform_reciprocal(). Create custom transforms with scales::new_transform().
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 of the label should match the object name (e.g., "mtry", "neighbors", etc.). If NULL, the parameter will be labeled with "Unlabeled parameter".
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 cli::cli_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. Used in `recipes::step_discretize()` and `embed::step_discretize_xgb()`.

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. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Examples

```
num_breaks()
```

num_clusters *Number of Clusters*

Description

Used in most `tidyclust` models.

Usage

```
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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Examples

```
num_clusters()
```

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. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. 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() and textrecipes::step_dummy_hash().

Usage

```
num_hash(range = c(8L, 12L), trans = transform_log2())

signed_hash(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::transform_log10() or scales::transform_reciprocal(). 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::transform_log10() or scales::transform_reciprocal(). 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 lightbgm*

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

```
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::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Details

"lightbgm" is an available engine in the parsnip extension package bonsai

For more information, see the lightgbm webpage.

Examples

```
num_leaves()
```

---

num_runs *Number of Computation Runs*

Description

Used in `recipes::step_nnmf()`.

Usage

```
num_runs(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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Examples

```
num_runs()
```

---

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. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Examples

```
num_tokens()
```

---

ordinal_link *Ordinal regression link functions (character)*

Description

The ordinal and odds link functions of an ordinal regression model.

Usage

```
ordinal_link(values = values_ordinal_link)

values_ordinal_link

odds_link(values = values_odds_link)

values_odds_link
```

Arguments

values	For *_link(), a character string of possible values. See values_*_link in examples below.

Details

These parameters are used by ordinal regression models specified by parsnip::ordinal_reg(), for example parsnip::set_engine('polr'). The nomenclature is taken from Wurm et al (2021), who characterize the pair of functions as a composite link function. Note that different engines support different subsets of link functions.

References

Wurm, Michael J., Rathouz, Paul J., & Hanlon, Bret M. (2021). Regularized Ordinal Regression and the ordinalNet R Package. *Journal of Statistical Software*, 99(6), 1–42. 10.18637/jss.v099.i06

Examples

```
values_ordinal_link
ordinal_link()
values_odds_link
odds_link()
```

---

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. Used in `themis::step_rose()`, `themis::step_smotenc()`, `themis::step_bsmote()`, `themis::step_upsample()`, `themis::step_downsample()`, and `themis::step_nearmiss()`.

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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Examples

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

---

parallel_reg *Parallel regression assumption (logical)*

Description

Whether the parallel regression assumption is imposed in an ordinal regression model.

Usage

```
parallel_reg(values = values_parallel_reg)

values_parallel_reg
```

Arguments

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

Details

This parameter is used by ordinal regression models specified by parsnip::ordinal_reg(), for example with the "clm", "vglm", or "ordinalNet" engines. It controls whether the regression coefficients are shared across all ordinal thresholds (e.g. proportional odds in logit-linked models) or allowed to vary (generalized ordered logit). TRUE makes all effects parallel, FALSE makes all effects nominal or category-specific.

Examples

```
values_parallel_reg
parallel_reg()
```

---

parameters *Create a parameter set*

Description

Create a parameter set

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.

Value

A parameter set of class parameters in the form of a data frame where each row represents one tunable parameter.

See Also

update.parameters(), finalize(), hardhat::extract_parameter_set_dials()

Examples

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

---

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 = transform_log10())
```

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::transform_log10() or scales::transform_reciprocal(). 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. If a transformation is specified, these values should be in the *transformed units*.
trans	A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). 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()
```

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_baye

Usage

```
prior_slab_dispersion(range = c(-1/2, log10(3)), trans = transform_log10())

prior_mixture_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::transform_log10() or scales::transform_reciprocal(). 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()
```

---

prop_terms *Proportion of top predictors*

Description

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

Usage

```
prop_terms(range = c(0.05, 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::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Details

prop_terms() is used in recipe stes in the **important** package.

Examples

```
prop_terms()
```

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.

Details

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

Examples

```
values_prune_method
prune_method()
```

range_limits *Limits for the range of predictions*

Description

Range limits truncate model predictions to a specific range of values, typically to avoid extreme or unrealistic predictions.

Usage

```
lower_limit(range = c(-Inf, Inf), trans = NULL)

upper_limit(range = c(-Inf, Inf), 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::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Examples

```
lower_limit()
upper_limit()
```

---

range_validate *Tools for working with parameter ranges*

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, call = caller_env())
```

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 cli::cli_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

```
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 = transform_log10())

scale_factor(range = c(-10, -1), trans = transform_log10())

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::transform_log10() or scales::transform_reciprocal(). 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 = transform_log10())

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

splitting_rule(values = ranger_split_rules)

ranger_class_rules

ranger_reg_rules

ranger_survival_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::transform_log10()` or `scales::transform_reciprocal()`. 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`, `ranger_reg_rules`, and `ranger_survival_rules` for appropriate values. For `regularize_depth()`, either `TRUE` or `FALSE`.

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 *Estimation methods for regularized models*

Description

Estimation methods for regularized models

Usage

```
regularization_method(values = values_regularization_method)

values_regularization_method
```

Arguments

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

Details

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

Examples

```
values_regularization_method
regularization_method()
```

resnet-param *Parameters for residual networks*

Description

These parameters are used with tabular residual networks, such as `tabular_resnet()` when fit with the `brulee` engine.

Usage

```
bottleneck_units(range = c(2L, 25L), trans = NULL)

resid_at(range = c(2L, unknown()), 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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Details

- `bottleneck_units()`: The number of units used in ResNet layers within a residual block.
- `resid_at()`: The layer indices at which residual connections are added. The upper bound depends on the number of hidden layers in the network and so is left as `unknown()` by default.

Examples

```
bottleneck_units()
resid_at()
```

rln-param *Parameters for regularization learning networks*

Description

These parameters are used with regularization learning networks (RLN), such as `tabular_rln()` when fit with the `brulee` engine.

Usage

```
penalty_average(range = c(-15, -5), trans = scales::transform_log10())

step_rate(range = c(0, 8), trans = scales::transform_log10())

penalty_type(values = values_penalty_type)

values_penalty_type
```

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::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.
values	A character vector of possible values.

Details

- penalty_average(): The target geometric mean of the per-weight regularization coefficients (Theta in Shavitt and Segal (2018)). Best tuned on the log10 scale.
- step_rate(): The step size used to update the per-weight regularization coefficients (nu in Shavitt and Segal (2018)). Best tuned on the log10 scale.
- penalty_type(): The type of regularization norm applied to per-weight coefficients. L1 is recommended by the original paper.

For penalty_average() and step_rate(), the value is passed to brulee::brulee_rln() on the natural scale.

References

Shavitt, I., & Segal, E. (2018). Regularization learning networks: Deep learning for tabular datasets. *Advances in Neural Information Processing Systems*, 31, 1379-1389.

Examples

```
penalty_average()
step_rate()
values_penalty_type
penalty_type()
```

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 = transform_log10())

penalty_L1(range = c(-10, 1), trans = transform_log10())
```

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::transform_log10()` or `scales::transform_reciprocal()`. 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*

Description

These parameters are used for constructing neural network models.

Usage

```
rate_initial(range = c(-3, -1), trans = transform_log10())

rate_largest(range = c(-1, -1/2), trans = transform_log10())

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::transform_log10() or scales::transform_reciprocal(). 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.

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(): $cycle = floor(1+(epoch/2/stepsize))$, $x = abs((epoch/stepsize) - (2 * cycle) + 1)$, and $rate(epoch) = initial + (largest - initial) * \max(0, 1 - x)$

---

select_features *Parameter to enable feature selection*

Description

Used in parsnip::gen_additive_mod().

Usage

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

Arguments

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

Examples

```
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

```
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::transform_log10()` or `scales::transform_reciprocal()`. 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 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` *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. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. 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::transform_log10()` or `scales::transform_reciprocal()`. 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.

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.

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)

values_surv_dist
```

Arguments

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

Details

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

Examples

```
values_surv_dist
surv_dist()
```

`tab-pfn-param` *Parameters for TabPFN models*

Description

These parameters are used for constructing Prior data fitted network (TabPFN) models.

Usage

```
num_estimators(range = c(1, 25), trans = NULL)

softmax_temperature(range = c(0, 10), trans = NULL)

balance_probabilities(values = c(TRUE, FALSE))

average_before_softmax(values = c(TRUE, FALSE))

training_set_limit(range = c(2L, 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::transform_log10() or scales::transform_reciprocal(). 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 parameters are often used with TabPFN models via parsnip::tab_pfn().

---

target_weight *Amount of supervision parameter*

Description

For uwot::umap() and embed::step_umap(), this is a weighting factor between data topology and target topology. A value of 0.0 weights entirely on data, a value of 1.0 weights entirely on target. The default of 0.5 balances the weighting equally between data and target.

Usage

```
target_weight(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::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Details

This parameter is used in recipes via embed::step_umap().

Examples

```
target_weight()
```

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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Examples

```
threshold()
```

threshold_structure *Threshold structure (character)*

Description

The threshold structure for the cutpoints of an ordinal regression model.

Usage

```
threshold_structure(values = values_threshold_structure)

values_threshold_structure
```

Arguments

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

Details

This parameter is used by ordinal regression models specified by `parsnip::ordinal_reg()`, for example with the `"clm"` or `"vglm"` engines. It controls how thresholds/cutpoints are constrained.

- `"flexible"` (default): No constraints.
- `"symmetric_median"`: Symmetry about the median threshold.
- `"symmetric_zero"`: Symmetry about zero.
- `"equidistant"`: Equal differences between consecutive thresholds.

See engine documentation for more detail and supported structures.

Examples

```
values_threshold_structure
threshold_structure()
```

---

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.

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 = transform_log10())

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

prune(values = c(TRUE, FALSE))

cost_complexity(range = c(-10, -1), trans = transform_log10())
```

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::transform_log10() or scales::transform_reciprocal(). 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()
```

`trim_amount` *Amount of Trimming*

Description

Used in `recipes::step_impute_mean()`.

Usage

```
trim_amount(range = c(0, 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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Examples

```
trim_amount()
```

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)))
```

`validation_set_prop` *Proportion of data used for validation*

Description

Used in `embed::step_discretize_xgb()`.

Usage

```
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::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Examples

```
validation_set_prop()
```

---

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, ..., 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 cli::cli_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

```
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 *Number of tokens in vocabulary*

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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Examples

```
vocabulary_size()
```

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

Description

Used in `textrecipes::step_tf()`.

Usage

```
weight(range = c(-10, 0), trans = transform_log10())
```

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::transform_log10()` or `scales::transform_reciprocal()`. 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.

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.

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()` and `recipes::step_impute_roll()`.

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::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Examples

```
window_size()
```

Index

activation, 4
activation_2 *(activation)*, 4
adjust_deg_free, 4
all_neighbors, 5
attention-param, 6
attention_type *(attention-param)*, 6
average_before_softmax *(tab-pfn-param)*, 66

balance_probabilities *(tab-pfn-param)*, 66
bart-param, 7
batch_size *(dropout)*, 14
bottleneck_units *(resnet-param)*, 59
buffer, 8

cal_method_class, 8
cal_method_reg *(cal_method_class)*, 8
class_weights, 9
cli::cli_abort(), *42*, *55*, *74*
conditional_min_criterion, 10
conditional_test_statistic *(conditional_min_criterion)*, 10
conditional_test_type *(conditional_min_criterion)*, 10
confidence_factor, 11
cost, 12
cost_complexity *(trees)*, 70

deg_free, 13
degree, 12
degree_int *(degree)*, 12
diagonal_covariance *(shrinkage_correlation)*, 63
dist_power, 14
dropout, 14
dropout_attn *(attention-param)*, 6
dropout_embedding *(dropout)*, 14
dropout_hidden *(dropout)*, 14
dropout_last *(dropout)*, 14

epochs *(dropout)*, 14
extrapolation, 16

finalize, 16
finalize(), *51*
freq_cut, 19
fuzzy_thresholding *(confidence_factor)*, 11

get_log_p *(finalize)*, 16
get_n *(finalize)*, 16
get_n_frac *(finalize)*, 16
get_n_frac_range *(finalize)*, 16
get_p *(finalize)*, 16
get_rbf_range *(finalize)*, 16
grid_hybrid, 20
grid_latin_hypercube(), *26*
grid_max_entropy(), *26*
grid_random *(grid_regular)*, 22
grid_regular, 22
grid_space_filling, 25
grid_space_filling(), *21*, *26*
grow_policy *(l2_leaf_reg)*, 30

hardhat::extract_parameter_set_dials(), *51*
harmonic_frequency, 29
has_unknowns *(unknown)*, 72
hidden_units *(dropout)*, 14
hidden_units_2 *(dropout)*, 14

initial_umap, 29
is_unknown *(unknown)*, 72

kernel_offset *(rbf_sigma)*, 56
kernlab::sigest(), *17*, *18*

l2_leaf_reg, 30
Laplace, 32
learn_rate, 32
loss_reduction *(trees)*, 70

lower_limit (range_limits), 54
lower_quantile (regularization_factor), 57

max_leaves (l2_leaf_reg), 30
max_nodes, 33
max_num_terms, 34
max_rules (extrapolation), 16
max_times, 34
max_tokens, 35
min_dist, 36
min_n (trees), 70
min_times (max_times), 34
min_unique, 36
mixture, 37
momentum, 38
mtry, 38
mtry(), *21*, *23*, *26*, *39–41*
mtry_long (mtry), 38
mtry_prop, 39
mtry_prop(), *39, 40*

neighbors, 40
new-param, 41
new_qual_param (new-param), 41
new_quant_param (new-param), 41
no_global_pruning (confidence_factor), 11
normalization (attention-param), 6
num_attn_blocks (attention-param), 6
num_attn_feat (attention-param), 6
num_attn_heads (attention-param), 6
num_breaks, 43
num_clusters, 44
num_comp, 44
num_embedding (dropout), 14
num_estimators (tab-pfn-param), 66
num_hash, 45
num_knots, 46
num_leaves, 46
num_random_splits (regularization_factor), 57
num_runs, 47
num_terms (num_comp), 44
num_tokens, 47

odds_link (ordinal_link), 48
one_hot_max_size (l2_leaf_reg), 30
ordinal_link, 48
over_ratio, 49

parallel_reg, 49
parameters, 50
penalty, 51
penalty(), *21*, *23*, *26*
penalty_average (rln-param), 59
penalty_L1 (scale_pos_weight), 61
penalty_L2 (scale_pos_weight), 61
penalty_type (rln-param), 59
predictor_prop, 52
predictor_winnowing (confidence_factor), 11
prior_mixture_threshold (prior_slab_dispersion), 52
prior_outcome_range (bart-param), 7
prior_slab_dispersion, 52
prior_terminal_node_coef (bart-param), 7
prior_terminal_node_expo (bart-param), 7
prod_degree (degree), 12
prop_terms, 53
prune (trees), 70
prune_method, 54

range_get (range_validate), 55
range_limits, 54
range_set (range_validate), 55
range_validate, 55
ranger_class_rules (regularization_factor), 57
ranger_reg_rules (regularization_factor), 57
ranger_split_rules (regularization_factor), 57
ranger_survival_rules (regularization_factor), 57
rate_decay (scheduler-param), 61
rate_initial (scheduler-param), 61
rate_largest (scheduler-param), 61
rate_reduction (scheduler-param), 61
rate_schedule (scheduler-param), 61
rate_step_size (scheduler-param), 61
rate_steps (scheduler-param), 61
rbf_sigma, 56
regularization_factor, 57
regularization_method, 58
regularize_depth (regularization_factor), 57
resid_at (resnet-param), 59

resnet-param, 59
rln-param, 59
rule_bands (confidence_factor), 11

sample_prop (trees), 70
sample_size (trees), 70
scale_factor (rbf_sigma), 56
scale_pos_weight, 61
scales::new_transform(), *42*
scales::transform_log(), *42*
scales::transform_reciprocal(), *42*
scheduler-param, 61
score_randomness (l2_leaf_reg), 30
select_features, 62
shrinkage_correlation, 63
shrinkage_frequencies
        (shrinkage_correlation), 63
shrinkage_variance
        (shrinkage_correlation), 63
signed_hash (num_hash), 45
significance_threshold
        (regularization_factor), 57
smoothness, 64
softmax_temperature (tab-pfn-param), 66
spline_degree (degree), 12
splitting_rule (regularization_factor),
        57
step_rate (rln-param), 59
stop_iter, 64
summary_stat, 65
surv_dist, 66
survival_link, 65
svm_margin (cost), 12

tab-pfn-param, 66
target_token (attention-param), 6
target_weight, 67
threshold, 68
threshold(), *8*
threshold_structure, 68
token, 69
training_set_limit (tab-pfn-param), 66
tree_depth (trees), 70
trees, 70
trim_amount, 71

unbiased_rules (extrapolation), 16
under_ratio (over_ratio), 49
unique_cut (freq_cut), 19
unknown, 72
update.parameters, 73
update.parameters(), *51*
upper_limit (range_limits), 54

validation_set_prop, 73
value_inverse (value_validate), 74
value_sample (value_validate), 74
value_sample(), *42*
value_seq (value_validate), 74
value_seq(), *42*
value_set (value_validate), 74
value_transform (value_validate), 74
value_validate, 74
values_activation (activation), 4
values_attention_type
        (attention-param), 6
values_cal_cls (cal_method_class), 8
values_cal_reg (cal_method_class), 8
values_grow_policy (l2_leaf_reg), 30
values_initial_umap (initial_umap), 29
values_normalization (attention-param),
        6
values_odds_link (ordinal_link), 48
values_ordinal_link (ordinal_link), 48
values_parallel_reg (parallel_reg), 49
values_penalty_type (rln-param), 59
values_prune_method (prune_method), 54
values_regularization_method
        (regularization_method), 58
values_scheduler (scheduler-param), 61
values_summary_stat (summary_stat), 65
values_surv_dist (surv_dist), 66
values_survival_link (survival_link), 65
values_test_statistic
        (conditional_min_criterion), 10
values_test_type
        (conditional_min_criterion), 10
values_threshold_structure
        (threshold_structure), 68
values_token (token), 69
values_weight_func (weight_func), 77
values_weight_scheme (weight_scheme), 77
vocabulary_size, 76

weight, 76
weight_func, 77
weight_scheme, 77
window_size, 78