# dials

This package contains tools to create and manage values of tuning parameters and is designed to integrate well with the [parsnip](https://github.com/topepo/parsnip) package.

The name reflects the idea that tuning predictive models can be like turning a set of dials on a complex machine under duress.

To install:

install.packages("dials")

## for development version:

require("devtools")

[install\_github](http://www.rdocumentation.org/packages/devtools/topics/install_github)("tidymodels/dials")

# Working with Tuning Parameters

## Tuning Parameters

Some statistical and machine learning models contain tuning parameters (also known as hyperparameters), which are parameters that cannot be directly estimated by the model. An example would be the number of neighbors in a K-nearest neighbors model. To determine reasonable values of these elements, some indirect method is used such as resampling or profile likelihood. Search methods, such as genetic algorithms or Bayesian search can also be used to [determine good values](https://github.com/topepo/Optimization-Methods-for-Tuning-Predictive-Models).

In any case, some information is needed to create a grid or to validate whether a candidate value is appropriate (i.e. neighbors should be a positive integer). dials is designed to:

* Create an easy to use framework for describing and querying tuning parameters. this can include getting sequences or random tuning values, validating current values, transforming parameters, and other tasks.
* Standardize the names of different parameters. Different packages in R use different argument names for the same quantities. dials proposes some standardized names so that the user doesn’t need to memorize the syntactical minutiae of every package.
* Work with the [parsnip](https://topepo.github.io/parsnip/) package which is a modern attempt to standardize the interfaces for specific models across R packages and computational engines.
* Adhere to [tidy principals](https://cran.r-project.org/package=tidyverse/vignettes/manifesto.html).

The main type of objects in dials have class param.

## param Objects

param objects contain information about possible values, ranges, types, and other aspects. There are two main subclasses related to the type of variable. Double and integer valued data have the subclass “quant\_param” while character and logicals have “qual\_param.” There are some common elements for each:

* Labels are strings that describe the parameter (e.g. “Number of Components”)
* Defaults are optional single values that can be set when one non-random value is requested.

Otherwise, the information contained in param objects are different for different data types.

## Numeric Parameters

An example of a numeric tuning parameter is the cost-complexity parameter of CART trees, otherwise known as Cp. The parameter object in dials is:

library(dials)

Cp

#> Cost-Complexity Parameter (quantitative)

#> Transformer: log-10

#> Range (transformed scale): [-10, -1]

Note that this parameter is handled in log units and the default range of values is between 10^-10 and 0.1. The range of possible values can be returned and changed based on some utility functions. We’ll use the pipe operator here:

library(dplyr)

Cp %>% [range\_get](https://tidymodels.github.io/dials/reference/range_validate.html)()

#> $lower

#> [1] 1e-10

#>

#> $upper

#> [1] 0.1

Cp %>% [range\_set](https://tidymodels.github.io/dials/reference/range_validate.html)(c(-5, 1))

#> Cost-Complexity Parameter (quantitative)

#> Transformer: log-10

#> Range (transformed scale): [-5, 1]

Values for this parameter can be obtained in a few different ways. To get a sequence of values that span the range:

# Natural units:

Cp %>% [value\_seq](https://tidymodels.github.io/dials/reference/value_validate.html)(n = 4)

#> [1] 1e-10 1e-07 1e-04 1e-01

# Stay in the transformed space:

Cp %>% [value\_seq](https://tidymodels.github.io/dials/reference/value_validate.html)(n = 4, original = FALSE)

#> [1] -10 -7 -4 -1

Random values can be sampled too. A random uniform distribution is used (between the range values). Since this parameter has a transformation associated with it, the values are simulated in the transformed scale and then returned in the natural units (although another original argument can be used here):

set.seed(5473)

Cp %>% [value\_sample](https://tidymodels.github.io/dials/reference/value_validate.html)(n = 4)

#> [1] 6.91e-09 8.46e-04 3.45e-06 5.90e-10

For CART trees, there is a discrete set of values that exist for a given data set. It may be a good idea to assign these possible values to the object. We can get them by fitting an initial rpart model and then adding the values to the object. For mtcars, there are only three values:

1 0.357 0.669 0.168

#> 3 0.000001 library(rpart)

cart\_mod <- [rpart](http://www.rdocumentation.org/packages/rpart/topics/rpart)(mpg ~ ., data = mtcars, control = [rpart.control](http://www.rdocumentation.org/packages/rpart/topics/rpart.control)(cp = 0.000001))

cart\_mod$cptable

#> CP nsplit rel error xerror xstd

#> 1 0.643125 0 1.000 1.083 0.259

#> 2 0.097484 2 0.259 0.574 0.125

cp\_vals <- cart\_mod$cptable[, "CP"]

# We should only keep values associated with at least one split:

cp\_vals <- cp\_vals[ cart\_mod$cptable[, "nsplit"] > 0 ]

# Here the specific Cp values, on their natural scale, are added:

mtcars\_cp <- Cp %>% [value\_set](https://tidymodels.github.io/dials/reference/value_validate.html)(cp\_vals)

#> Error in new\_quant\_param(type = object$type, range = object$range, inclusive = object$inclusive, : Some values are not valid: 0.09748...

The error occurs because the values are not in the transformed scale:

mtcars\_cp <- Cp %>% [value\_set](https://tidymodels.github.io/dials/reference/value_validate.html)(log10(cp\_vals))

mtcars\_cp

#> Cost-Complexity Parameter (quantitative)

#> Transformer: log-10

#> Range (transformed scale): [-10, -1]

Now, if sequence or random sample is requested, it uses the set values:

mtcars\_cp %>% [value\_seq](https://tidymodels.github.io/dials/reference/value_validate.html)(2)

#> [1] 0.097484 0.000001

# Sampling specific values is done with replacement

mtcars\_cp %>%

[value\_sample](https://tidymodels.github.io/dials/reference/value_validate.html)(20) %>%

table()

#> .

#> 1e-06 0.0974840733898344

#> 8 12

## Discrete Parameters

In this case there is no notion or a range scale. The parameter objects are defined by their values. For example, consider a parameter for the types of kernel functions that is used with distance functions:

weight\_func

#> Distance Weighting Function (qualitative)

#> 10 possible value include:

#> 'rectangular', 'triangular', 'epanechnikov', 'biweight', 'triweight', 'c...

The helper functions are analogues to the quantitative parameters:

# redefine values

weight\_func %>% [value\_set](https://tidymodels.github.io/dials/reference/value_validate.html)(c("rectangular", "triangular"))

#> Distance Weighting Function (qualitative)

#> 2 possible value include:

#> 'rectangular' and 'triangular'

weight\_func %>% [value\_sample](https://tidymodels.github.io/dials/reference/value_validate.html)(3)

#> [1] "triangular" "biweight" "cos"

# the sequence is returned in the order of the levels

weight\_func %>% [value\_seq](https://tidymodels.github.io/dials/reference/value_validate.html)(3)

#> [1] "rectangular" "triangular" "epanechnikov"

## Creating Novel Parameters

The package contains two constructors that can be used to create new quantitative and qualitative parameters. [This file](https://github.com/tidymodels/dials/blob/master/R/parameters.R)contains the code to create the parameters contained in the package.

## Unknown Values

There are some cases where the range of parameter values are data dependent. For example, the upper bound on the number of neighbors cannot be known if the number of data points in the training set is not known. For that reason, some parameters have unknown placeholder:

mtry

#> # Randomly Selected Predictors (quantitative)

#> Range: [1, ?]

neighbors

#> # Nearest Neighbors (quantitative)

#> Range: [1, ?]

min\_n

#> Minimal Node Size (quantitative)

#> Range: [2, ?]

sample\_size

#> # Observations Sampled (quantitative)

#> Range: [?, ?]

num\_terms

#> # Model Terms (quantitative)

#> Range: [1, ?]

num\_comp

#> # Components (quantitative)

#> Range: [1, ?]

# and so on

These values must be initialized prior to generating parameter values.

## Parameter Grids

Sets or combinations or parameters can be created for use in grid search. grid\_regular and grid\_random take any number of param objects.

For example, for a glmnet model, a regular grid might be:

[grid\_regular](https://tidymodels.github.io/dials/reference/grid_regular.html)(

mixture,

regularization,

levels = 3 # or c(3, 4), etc

)

#> # A tibble: 9 x 2

#> mixture regularization

#> <dbl> <dbl>

#> 1 0 0.0000000001

#> 2 0.5 0.0000000001

#> 3 1 0.0000000001

#> 4 0 0.00001

#> 5 0.5 0.00001

#> 6 1 0.00001

#> 7 0 1

#> 8 0.5 1

#> 9 1 1

and, similarly, a random grid is created using

set.seed(1041)

[grid\_random](https://tidymodels.github.io/dials/reference/grid_regular.html)(

mixture,

regularization,

size = 6

)

#> # A tibble: 6 x 2

#> mixture regularization

#> <dbl> <dbl>

#> 1 0.200 0.0176

#> 2 0.750 0.000388

#> 3 0.191 0.000000159

#> 4 0.929 0.00000176

#> 5 0.143 0.0442

#> 6 0.973 0.0110

# Using Dials with Parsnip

[parsnip](https://tidymodels.github.io/parsnip) is a package in development that provides more unified interfaces to model functions. It has functions to create a model specification that can be used to fit a particular model using different R packages (or by other means). these model specifications have main arguments for important tuning parameters. For example, a minimal model specification is:

library(parsnip)

[boost\_tree](http://www.rdocumentation.org/packages/parsnip/topics/boost_tree)(mode = "classification")

#> Boosted Tree Model Specification (classification)

This particular model has a number of different arguments for tuning parameters:

str(boost\_tree)

#> function (mode = "unknown", mtry = NULL, trees = NULL, min\_n = NULL,

#> tree\_depth = NULL, learn\_rate = NULL, loss\_reduction = NULL, sample\_size = NULL,

#> others = list(), ...)

If we know exactly what specific value of a parameter should be, it can be specified:

[boost\_tree](http://www.rdocumentation.org/packages/parsnip/topics/boost_tree)(mode = "classification", trees = 50, min\_n = 5, sample\_size = 3/4)

#> Boosted Tree Model Specification (classification)

#>

#> Main Arguments:

#> trees = 50

#> min\_n = 5

#> sample\_size = 0.75

Note that :

* These parameter names have identically named parameter objects in dials.
* Like other parsnip functions, boost\_tree can use different R packages to fit this model, in this case, xgboost and C50. Not all parameters to boost\_tree are relevant to each of these specific models.
* Any parameters not specified in this call will use their model-specific defaults.

What happens if you know that you want to optimize the value of a parameter but don’t know what the value will be? In this case, the parsnip function [varying()](http://www.rdocumentation.org/packages/parsnip/topics/varying) can be used as a placeholder. For example, min\_n is conditional on the sample size of the training set, so we may not know a feasible value until be have the exact training or analysis set:

mod\_obj <- [boost\_tree](http://www.rdocumentation.org/packages/parsnip/topics/boost_tree)(mode = "classification", trees = 50, min\_n = [varying](http://www.rdocumentation.org/packages/parsnip/topics/varying)(), sample\_size = 3/4)

mod\_obj

#> Boosted Tree Model Specification (classification)

#>

#> Main Arguments:

#> trees = 50

#> min\_n = varying()

#> sample\_size = 0.75

If some type of grid search is used, there is a simple function in dials can be used to update this parameter specification with candidate values. Let’s create a small, random grid for these parameters. We will set the ranges for these parameters in-line when creating the grid:

library(tidymodels)

library(dials)

set.seed(1263)

bst\_grid <- [grid\_random](https://tidymodels.github.io/dials/reference/grid_regular.html)(

trees %>% [range\_set](https://tidymodels.github.io/dials/reference/range_validate.html)(c( 1, 50)),

min\_n %>% [range\_set](https://tidymodels.github.io/dials/reference/range_validate.html)(c( 2, 30)),

sample\_size %>% [range\_set](https://tidymodels.github.io/dials/reference/range_validate.html)(c(20, 130)),

size = 3

)

bst\_grid

#> # A tibble: 3 x 3

#> trees min\_n sample\_size

#> <int> <int> <int>

#> 1 35 27 59

#> 2 5 21 103

#> 3 41 5 21

We can use the merge function to combine these parameters with the model specification:

for(i in 1:nrow(bst\_grid)) {

print(merge(mod\_obj, bst\_grid[i, ]))

}

#> Boosted Tree Model Specification (classification)

#>

#> Main Arguments:

#> trees = 35

#> min\_n = 27

#> sample\_size = 59

#>

#> Boosted Tree Model Specification (classification)

#>

#> Main Arguments:

#> trees = 5

#> min\_n = 21

#> sample\_size = 103

#>

#> Boosted Tree Model Specification (classification)

#>

#> Main Arguments:

#> trees = 41

#> min\_n = 5

#> sample\_size = 21