Package ‘finetune’

March 24, 2022

Title Additional Functions for Model Tuning

Version 0.2.0

Description The ability to tune models is important. 'finetune' enhances the 'tune' package by providing more specialized methods for finding reasonable values of model tuning parameters. Two racing methods described by Kuhn (2014) <arXiv:1405.6974> are included. An iterative search method using generalized simulated annealing (Bohachevsky, Johnson and Stein, 1986) <doi:10.1080/00401706.1986.10488128> is also included.

License MIT + file LICENSE

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

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

Depends R (>= 3.4), tune (>= 0.2.0)

Imports cli, dials (>= 0.1.0), dplyr, ggplot2, purrr, rlang, tibble, tidyr, tidyselect, utils, vctrs, workflows (>= 0.2.6)

Suggests BradleyTerry2, covr, discrim, klaR, lme4, modeldata, parsnip (>= 0.2.1), ranger, recipes (>= 0.2.0), rpart, rsample, spelling, testthat, yardstick

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.1.2

NeedsCompilation no

Author Max Kuhn [aut, cre] (<https://orcid.org/0000-0003-2402-136X>), RStudio [cph, fnd]

Maintainer Max Kuhn <max@rstudio.com>

Repository CRAN

Date/Publication 2022-03-24 18:30:02 UTC

R topics documented:

`control_race` *Control aspects of the grid search racing process*

Description

Control aspects of the grid search racing process

Usage

```
control_race(
  verbose = FALSE,
  verbose_elim = FALSE,
  allow_par = TRUE,
  extract = NULL,
  save_pred = FALSE,
  burn_in = 3,
  num_ties = 10,
  alpha = 0.05,
  randomize = TRUE,
  pkgs = NULL,
  save_workflow = FALSE,
  event_level = "first",
  parallel_over = "everything"
)
```

Arguments

`verbose`	A logical for logging results as they are generated. Despite this argument, warnings and errors are always shown. If using a dark IDE theme, some logging messages might be hard to see. If this is the case, try setting the `tidymodels.dark` option with `options(tidymodels.dark = TRUE)` to print lighter colors.
`verbose_elim`	A logical for whether logging of the elimination of tuning parameter combinations should occur.
`allow_par`	A logical to allow parallel processing (if a parallel backend is registered).
`extract`	An optional function with at least one argument (or `NULL`) that can be used to retain arbitrary objects from the model fit object, recipe, or other elements of the workflow.

save_pred — A logical for whether the out-of-sample predictions should be saved for each model *evaluated*.

burn_in — An integer for how many resamples should be completed for all grid combinations before parameter filtering begins.

num_ties — An integer for when tie-breaking should occur. If there are two final parameter combinations being evaluated, `num_ties` specified how many more resampling iterations should be evaluated. After `num_ties` more iterations, the parameter combination with the current best results is retained.

alpha — The alpha level for a one-sided confidence interval for each parameter combination.

randomize — Should the resamples be evaluated in a random order? By default, the resamples are evaluated in a random order so the random number seed should be control prior to calling this method (to be reproducible). For repeated cross-validation the randomization occurs within each repeat.

pkgs — An optional character string of R package names that should be loaded (by namespace) during parallel processing.

save_workflow — A logical for whether the workflow should be appended to the output as an attribute.

event_level — A single string containing either "first" or "second". This argument is passed on to `yardstick` metric functions when any type of class prediction is made, and specifies which level of the outcome is considered the "event".

parallel_over — A single string containing either `"resamples"` or `"everything"` describing how to use parallel processing. Alternatively, `NULL` is allowed, which chooses between `"resamples"` and `"everything"` automatically.

If `"resamples"`, then tuning will be performed in parallel over resamples alone. Within each resample, the preprocessor (i.e. recipe or formula) is processed once, and is then reused across all models that need to be fit.

If `"everything"`, then tuning will be performed in parallel at two levels. An outer parallel loop will iterate over resamples. Additionally, an inner parallel loop will iterate over all unique combinations of preprocessor and model tuning parameters for that specific resample. This will result in the preprocessor being re-processed multiple times, but can be faster if that processing is extremely fast.

If `NULL`, chooses `"resamples"` if there are more than one resample, otherwise chooses `"everything"` to attempt to maximize core utilization.

Value

An object of class `control_race` that echos the argument values.

Examples

```
control_race()
```

`control_sim_anneal` *Control aspects of the simulated annealing search process*

Description

Control aspects of the simulated annealing search process

Usage

```
control_sim_anneal(
  verbose = TRUE,
  no_improve = Inf,
  restart = 8L,
  radius = c(0.05, 0.15),
  flip = 3/4,
  cooling_coef = 0.02,
  extract = NULL,
  save_pred = FALSE,
  time_limit = NA,
  pkgs = NULL,
  save_workflow = FALSE,
  save_history = FALSE,
  event_level = "first",
  parallel_over = NULL
)
```

Arguments

`verbose`	A logical for logging results as they are generated. Despite this argument, warnings and errors are always shown. If using a dark IDE theme, some logging messages might be hard to see. If this is the case, try setting the `tidymodels.dark` option with `options(tidymodels.dark = TRUE)` to print lighter colors.
`no_improve`	The integer cutoff for the number of iterations without better results.
`restart`	The number of iterations with no improvement before new tuning parameter candidates are generated from the last, overall best conditions.
`radius`	Two real numbers on (0, 1) describing what a value "in the neighborhood" of the current result should be. If all numeric parameters were scaled to be on the [0, 1] scale, these values set the min. and max. of a radius of a circle used to generate new numeric parameter values.
`flip`	A real number between [0, 1] for the probability of changing any non-numeric parameter values at each iteration.
`cooling_coef`	A real, positive number to influence the cooling schedule. Larger values decrease the probability of accepting a sub-optimal parameter setting.
`extract`	An optional function with at least one argument (or `NULL`) that can be used to retain arbitrary objects from the model fit object, recipe, or other elements of the workflow.

- `save_pred` A logical for whether the out-of-sample predictions should be saved for each model *evaluated*.
- `time_limit` A number for the minimum number of *minutes* (elapsed) that the function should execute. The elapsed time is evaluated at internal checkpoints and, if over time, the results at that time are returned (with a warning). This means that the `time_limit` is not an exact limit, but a minimum time limit.
- `pkgs` An optional character string of R package names that should be loaded (by namespace) during parallel processing.
- `save_workflow` A logical for whether the workflow should be appended to the output as an attribute.
- `save_history` A logical to save the iteration details of the search. These are saved to `tempdir()` named `sa_history.RData`. These results are deleted when the R session ends. This option is only useful for teaching purposes.
- `event_level` A single string containing either "first" or "second". This argument is passed on to `yardstick` metric functions when any type of class prediction is made, and specifies which level of the outcome is considered the "event".
- `parallel_over` A single string containing either `"resamples"` or `"everything"` describing how to use parallel processing. Alternatively, `NULL` is allowed, which chooses between `"resamples"` and `"everything"` automatically.

  If `"resamples"`, then tuning will be performed in parallel over resamples alone. Within each resample, the preprocessor (i.e. recipe or formula) is processed once, and is then reused across all models that need to be fit.

  If `"everything"`, then tuning will be performed in parallel at two levels. An outer parallel loop will iterate over resamples. Additionally, an inner parallel loop will iterate over all unique combinations of preprocessor and model tuning parameters for that specific resample. This will result in the preprocessor being re-processed multiple times, but can be faster if that processing is extremely fast.

  If `NULL`, chooses `"resamples"` if there are more than one resample, otherwise chooses `"everything"` to attempt to maximize core utilization.

Value

An object of class `control_sim_anneal` that echos the argument values.

Examples

```
control_sim_anneal()
```

`plot_race` *Plot racing results*

Description

Plot the model results over stages of the racing results. A line is given for each submodel that was tested.

Usage

```
plot_race(x)
```

Arguments

x	A object with class `tune_results`

Value

A ggplot object.

`tune_race_anova` *Efficient grid search via racing with ANOVA models*

Description

`tune_race_anova()` computes a set of performance metrics (e.g. accuracy or RMSE) for a pre-defined set of tuning parameters that correspond to a model or recipe across one or more resamples of the data. After an initial number of resamples have been evaluated, the process eliminates tuning parameter combinations that are unlikely to be the best results using a repeated measure ANOVA model.

Usage

```
tune_race_anova(object, ...)

## S3 method for class 'model_spec'
tune_race_anova(
  object,
  preprocessor,
  resamples,
  ...,
  param_info = NULL,
  grid = 10,
  metrics = NULL,
  control = control_race()
)

## S3 method for class 'workflow'
tune_race_anova(
  object,
  resamples,
  ...,
  param_info = NULL,
  grid = 10,
  metrics = NULL,
  control = control_race()
)
```

Arguments

object	A parsnip model specification or a workflows::workflow().
...	Not currently used.
preprocessor	A traditional model formula or a recipe created using recipes::recipe(). This is only required when object is not a workflow.
resamples	An rset() object that has multiple resamples (i.e., is not a validation set).
param_info	A dials::parameters() object or NULL. If none is given, a parameters set is derived from other arguments. Passing this argument can be useful when parameter ranges need to be customized.
grid	A data frame of tuning combinations or a positive integer. The data frame should have columns for each parameter being tuned and rows for tuning parameter candidates. An integer denotes the number of candidate parameter sets to be created automatically.
metrics	A yardstick::metric_set() or NULL.
control	An object used to modify the tuning process. See control_race() for more details.

Details

The technical details of this method are described in Kuhn (2014).

Racing methods are efficient approaches to grid search. Initially, the function evaluates all tuning parameters on a small initial set of resamples. The burn_in argument of control_race() sets the number of initial resamples.

The performance statistics from these resamples are analyzed to determine which tuning parameters are *not* statistically different from the current best setting. If a parameter is statistically different, it is excluded from further resampling.

The next resample is used with the remaining parameter combinations and the statistical analysis is updated. More candidate parameters may be excluded with each new resample that is processed.

This function determines statistical significance using a repeated measures ANOVA model where the performance statistic (e.g., RMSE, accuracy, etc.) is the outcome data and the random effect is due to resamples. The control_race() function contains are parameter for the significance cutoff applied to the ANOVA results as well as other relevant arguments.

There is benefit to using racing methods in conjunction with parallel processing. The following section shows a benchmark of results for one dataset and model.

Benchmarking results:

To demonstrate, we use a SVM model with the kernlab package.

```
library(kernlab)
library(tidymodels)
library(finetune)
library(doParallel)

## ---------------------------------------------------------------------------
```

```
data(cells, package = "modeldata")
cells <- cells %>% select(-case)

## -------------------------------------------------------------------------------

set.seed(6376)
rs <- bootstraps(cells, times = 25)
```

We'll only tune the model parameters (i.e., not recipe tuning):

```
## -------------------------------------------------------------------------------

svm_spec <-
  svm_rbf(cost = tune(), rbf_sigma = tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

svm_rec <-
  recipe(class ~ ., data = cells) %>%
  step_YeoJohnson(all_predictors()) %>%
  step_normalize(all_predictors())

svm_wflow <-
  workflow() %>%
  add_model(svm_spec) %>%
  add_recipe(svm_rec)

set.seed(1)
svm_grid <-
  svm_spec %>%
  parameters() %>%
  grid_latin_hypercube(size = 25)
```

We'll get the times for grid search and ANOVA racing with and without parallel processing:

```
## -------------------------------------------------------------------------------
## Regular grid search

system.time({
  set.seed(2)
  svm_wflow %>% tune_grid(resamples = rs, grid = svm_grid)
})

##    user  system elapsed
## 741.660  19.654 761.357

## -------------------------------------------------------------------------------
## With racing

system.time({
  set.seed(2)
```

```
  svm_wflow %>% tune_race_anova(resamples = rs, grid = svm_grid)
})
```

```
##    user  system elapsed
## 133.143   3.675 136.822
```

Speed-up of 5.56-fold for racing.

```
## ---------------------------------------------------------------------------
## Parallel processing setup

cores <- parallel::detectCores(logical = FALSE)
cores
```

```
## [1] 10
```

```
cl <- makePSOCKcluster(cores)
registerDoParallel(cl)
```

```
## ---------------------------------------------------------------------------
## Parallel grid search

system.time({
  set.seed(2)
  svm_wflow %>% tune_grid(resamples = rs, grid = svm_grid)
})
```

```
##  user  system elapsed
## 1.112   0.190 126.650
```

Parallel processing with grid search was 6.01-fold faster than sequential grid search.

```
## ---------------------------------------------------------------------------
## Parallel racing

system.time({
  set.seed(2)
  svm_wflow %>% tune_race_anova(resamples = rs, grid = svm_grid)
})
```

```
##  user  system elapsed
## 1.908   0.261  21.442
```

Parallel processing with racing was 35.51-fold faster than sequential grid search.

There is a compounding effect of racing and parallel processing but its magnitude depends on the type of model, number of resamples, number of tuning parameters, and so on.

Value

An object with primary class `tune_race` in the same standard format as objects produced by `tune::tune_grid()`.

References

Kuhn, M 2014. "Futility Analysis in the Cross-Validation of Machine Learning Models." https://arxiv.org/abs/1405.6974.

See Also

tune::tune_grid(), control_race(), tune_race_win_loss()

Examples

```
library(parsnip)
library(rsample)
library(dials)

## -----------------------------------------------------------------------------

if (rlang::is_installed(c("discrim", "lme4", "modeldata"))) {
  library(discrim)
  data(two_class_dat, package = "modeldata")

  set.seed(6376)
  rs <- bootstraps(two_class_dat, times = 10)

  ## -----------------------------------------------------------------------------

  # optimize an regularized discriminant analysis model
  rda_spec <-
    discrim_regularized(frac_common_cov = tune(), frac_identity = tune()) %>%
    set_engine("klaR")

  ## -----------------------------------------------------------------------------

  ctrl <- control_race(verbose_elim = TRUE)
  set.seed(11)
  grid_anova <-
    rda_spec %>%
    tune_race_anova(Class ~ ., resamples = rs, grid = 10, control = ctrl)

  # Shows only the fully resampled parameters
  show_best(grid_anova, metric = "roc_auc", n = 2)

  plot_race(grid_anova)
}
```

---

tune_race_win_loss *Efficient grid search via racing with win/loss statistics*

Description

tune_race_win_loss() computes a set of performance metrics (e.g. accuracy or RMSE) for a pre-defined set of tuning parameters that correspond to a model or recipe across one or more resamples of the data. After an initial number of resamples have been evaluated, the process eliminates tuning parameter combinations that are unlikely to be the best results using a statistical model. For each pairwise combinations of tuning parameters, win/loss statistics are calculated and a logistic regression model is used to measure how likely each combination is to win overall.

Usage

```
tune_race_win_loss(object, ...)

## S3 method for class 'model_spec'
tune_race_win_loss(
  object,
  preprocessor,
  resamples,
  ...,
  param_info = NULL,
  grid = 10,
  metrics = NULL,
  control = control_race()
)

## S3 method for class 'workflow'
tune_race_win_loss(
  object,
  resamples,
  ...,
  param_info = NULL,
  grid = 10,
  metrics = NULL,
  control = control_race()
)
```

Arguments

`object` A `parsnip` model specification or a `workflows::workflow()`.

`...` Not currently used. The technical details of this method are described in Kuhn (2014).

Racing methods are efficient approaches to grid search. Initially, the function evaluates all tuning parameters on a small initial set of resamples. The `burn_in` argument of `control_race()` sets the number of initial resamples.

The performance statistics from the current set of resamples are converted to win/loss/tie results. For example, for two parameters (`j` and `k`) in a classification model that have each been resampled three times:

```
            |  area under the ROC curve |
```

```
             ----------------------------
resample | parameter j | parameter k | winner
---------------------------------------------
    1    |    0.81     |     0.92    |   k
    2    |    0.95     |     0.94    |   j
    3    |    0.79     |     0.81    |   k
---------------------------------------------
```

After the third resample, parameter k has a 2:1 win/loss ratio versus j. Parameters with equal results are treated as a half-win for each setting. These statistics are determined for all pairwise combinations of the parameters and a Bradley-Terry model is used to model these win/loss/tie statistics. This model can compute the ability of a parameter combination to win overall. A confidence interval for the winning ability is computed and any settings whose interval includes zero are retained for future resamples (since it is not statistically different form the best results).

The next resample is used with the remaining parameter combinations and the statistical analysis is updated. More candidate parameters may be excluded with each new resample that is processed.

The control_race() function contains are parameter for the significance cutoff applied to the Bradley-Terry model results as well as other relevant arguments.

preprocessor	A traditional model formula or a recipe created using recipes::recipe().
resamples	An rset() object that has multiple resamples (i.e., is not a validation set).
param_info	A dials::parameters() object or NULL. If none is given, a parameters set is derived from other arguments. Passing this argument can be useful when parameter ranges need to be customized.
grid	A data frame of tuning combinations or a positive integer. The data frame should have columns for each parameter being tuned and rows for tuning parameter candidates. An integer denotes the number of candidate parameter sets to be created automatically.
metrics	A yardstick::metric_set() or NULL.
control	An object used to modify the tuning process.

Value

An object with primary class tune_race in the same standard format as objects produced by tune::tune_grid().

References

Kuhn, M 2014. "Futility Analysis in the Cross-Validation of Machine Learning Models." https://arxiv.org/abs/1405.6974.

See Also

tune::tune_grid(), control_race(), tune_race_anova()

Examples

```
library(parsnip)
library(rsample)
library(dials)

## -----------------------------------------------------------------------------

if (rlang::is_installed(c("discrim", "modeldata"))) {
  library(discrim)
  data(two_class_dat, package = "modeldata")

  set.seed(6376)
  rs <- bootstraps(two_class_dat, times = 10)

  ## -----------------------------------------------------------------------------

  # optimize an regularized discriminant analysis model
  rda_spec <-
    discrim_regularized(frac_common_cov = tune(), frac_identity = tune()) %>%
    set_engine("klaR")

  ## -----------------------------------------------------------------------------

  ctrl <- control_race(verbose_elim = TRUE)

  set.seed(11)
  grid_wl <-
    rda_spec %>%
    tune_race_win_loss(Class ~ ., resamples = rs, grid = 10, control = ctrl)

  # Shows only the fully resampled parameters
  show_best(grid_wl, metric = "roc_auc")

  plot_race(grid_wl)
}
```

---

tune_sim_anneal — *Optimization of model parameters via simulated annealing*

Description

tune_sim_anneal() uses an iterative search procedure to generate new candidate tuning parameter combinations based on previous results. It uses the generalized simulated annealing method of Bohachevsky, Johnson, and Stein (1986).

Usage

```
tune_sim_anneal(object, ...)

## S3 method for class 'model_spec'
tune_sim_anneal(
  object,
  preprocessor,
  resamples,
  ...,
  iter = 10,
  param_info = NULL,
  metrics = NULL,
  initial = 1,
  control = control_sim_anneal()
)

## S3 method for class 'workflow'
tune_sim_anneal(
  object,
  resamples,
  ...,
  iter = 10,
  param_info = NULL,
  metrics = NULL,
  initial = 1,
  control = control_sim_anneal()
)
```

Arguments

`object`	A `parsnip` model specification or a `workflows::workflow()`.
`...`	Not currently used.
`preprocessor`	A traditional model formula or a recipe created using `recipes::recipe()`. This is only required when `object` is not a workflow.
`resamples`	An `rset()` object.
`iter`	The maximum number of search iterations.
`param_info`	A `dials::parameters()` object or `NULL`. If none is given, a parameter set is derived from other arguments. Passing this argument can be useful when parameter ranges need to be customized.
`metrics`	A `yardstick::metric_set()` object containing information on how models will be evaluated for performance. The first metric in `metrics` is the one that will be optimized.
`initial`	An initial set of results in a tidy format (as would the result of `tune_grid()`, `tune_bayes()`, `tune_race_win_loss()`, or `tune_race_anova()`) or a positive integer. If the initial object was a sequential search method, the simulated annealing iterations start after the last iteration of the initial results.
`control`	The results of `control_sim_anneal()`.

Details

Simulated annealing is a global optimization method. For model tuning, it can be used to iteratively search the parameter space for optimal tuning parameter combinations. At each iteration, a new parameter combination is created by perturbing the current parameters in some small way so that they are within a small neighborhood. This new parameter combination is used to fit a model and that model's performance is measured using resampling (or a simple validation set).

If the new settings have better results than the current settings, they are accepted and the process continues.

If the new settings has worse performance, a probability threshold is computed for accepting these sub-optimal values. The probability is a function of *how* sub-optimal the results are as well as how many iterations have elapsed. This is referred to as the "cooling schedule" for the algorithm. If the sub-optimal results are accepted, the next iterations settings are based on these inferior results. Otherwise, new parameter values are generated from the previous iteration's settings.

This process continues for a pre-defined number of iterations and the overall best settings are recommended for use. The `control_sim_anneal()` function can specify the number of iterations without improvement for early stopping. Also, that function can be used to specify a *restart* threshold; if no globally best results have not be discovered within a certain number if iterations, the process can restart using the last known settings that globally best.

Creating new settings:

For each numeric parameter, the range of possible values is known as well as any transformations. The current values are transformed and scaled to have values between zero and one (based on the possible range of values). A candidate set of values that are on a sphere with random radii between `rmin` and `rmax` are generated. Infeasible values are removed and one value is chosen at random. This value is back transformed to the original units and scale and are used as the new settings. The argument `radius` of `control_sim_anneal()` controls the range neighborhood sizes.

For categorical and integer parameters, each is changes with a pre-defined probability. The `flip` argument of `control_sim_anneal()` can be used to specify this probability. For integer parameters, a nearby integer value is used.

Simulated annealing search may not be the preferred method when many of the parameters are non-numeric or integers with few unique values. In these cases, it is likely that the same candidate set may be tested more than once.

Cooling schedule:

To determine the probability of accepting a new value, the percent difference in performance is calculated. If the performance metric is to be maximized, this would be `d = (new-old)/old*100`. The probability is calculated as `p = exp(d * coef * iter)` were `coef` is a user-defined constant that can be used to increase or decrease the probabilities.

The `cooling_coef` of `control_sim_anneal()` can be used for this purpose.

Termination criterion:

The restart counter is reset when a new global best results is found.

The termination counter resets when a new global best is located or when a suboptimal result is improved.

Parallelism:

The `tune` and `finetune` packages currently parallelize over resamples. Specifying a parallel back-end will improve the generation of the initial set of sub-models (if any). Each iteration of the search are also run in parallel if a parallel backend is registered.

Value

A tibble of results that mirror those generated by `tune_grid()`. However, these results contain an `.iter` column and replicate the `rset` object multiple times over iterations (at limited additional memory costs).

References

Bohachevsky, Johnson, and Stein (1986) "Generalized Simulated Annealing for Function Optimization", *Technometrics*, 28:3, 209-217

See Also

`tune::tune_grid()`, `control_sim_anneal()`, `yardstick::metric_set()`

Examples

```
library(finetune)
library(rpart)
library(dplyr)
library(tune)
library(rsample)
library(parsnip)
library(workflows)
library(ggplot2)

## -----------------------------------------------------------------------------
if (rlang::is_installed("modeldata")) {
  data(two_class_dat, package = "modeldata")

  set.seed(5046)
  bt <- bootstraps(two_class_dat, times = 5)

  ## -----------------------------------------------------------------------------

  cart_mod <-
    decision_tree(cost_complexity = tune(), min_n = tune()) %>%
    set_engine("rpart") %>%
    set_mode("classification")

  ## -----------------------------------------------------------------------------

  # For reproducibility, set the seed before running.
  set.seed(10)
  sa_search <-
    cart_mod %>%
    tune_sim_anneal(Class ~ ., resamples = bt, iter = 10)
```

```
  autoplot(sa_search, metric = "roc_auc", type = "parameters") +
    theme_bw()

  ## ------------------------------------------------------------------------------
  # More iterations. `initial` can be any other tune_* object or an integer
  # (for new values).

  set.seed(11)
  more_search <-
    cart_mod %>%
    tune_sim_anneal(Class ~ ., resamples = bt, iter = 10, initial = sa_search)

  autoplot(more_search, metric = "roc_auc", type = "performance") +
    theme_bw()
}
```

Index