Package 'comparer'

March 25, 2020

Type Package

Title Compare Output and Run Time

Version 0.2.1

Description Quickly run experiments to compare the run time and output of code blocks. The function mbc() can make fast comparisons of code, and will calculate statistics comparing the resulting outputs. It can be used to compare model fits to the same data or see which function runs faster. The function ffexp() runs a function using all possible combinations of selected inputs. This is useful for comparing the effect of different parameter values. It can also run in parallel and automatically save intermediate results, which is very useful for long computations.

License GPL-3

Encoding UTF-8

LazyData true

Imports R6

Suggests plyr, progress, testthat, covr, knitr, ggplot2, parallel, snow, rmarkdown, reshape, tibble, microbenchmark

RoxygenNote 7.1.0

URL https://github.com/CollinErickson/comparer

BugReports https://github.com/CollinErickson/comparer/issues

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Collin Erickson [aut, cre]

Maintainer Collin Erickson <collinberickson@gmail.com>

Repository CRAN

Date/Publication 2020-03-25 18:20:03 UTC

R topics documented:

`ffexp` *Full factorial experiment*

Description

A class for easily creating and evaluating full factorial experiments.

Usage

```
e1 <- ffexp$new(eval_func=, )

e1$run_all()

e1$plot_run_times()

e1$save_self()
```

Arguments

`eval_func` The function called to evaluate each design point.

`...` Factors and their levels to be evaluated at.

`save_output` Should the output be saved?

`parallel` If `TRUE`, function evaluations are done in parallel.

`parallel_cores` Number of cores to be used in parallel. If `"detect"`, `parallel::detectCores()` is used to determine number. `"detect-1"` may be used so that the computer isn't running at full capacity, which can slow down other tasks.

Methods

`$new()` Initialize an experiment. The preprocessing is done, but no function evaluations are run.

`$run_all()` Run all factor combinations.

`$run_one()` Run a single factor combination.

`$add_result_of_one()` Used to add result of evaluation to data set, don't manually call.

`$plot_run_times()` Plot the run times. Especially useful when they have been run in parallel.

`$save_self()` Save ffexp R6 object.

`$recover_parallel_temp_save()` If you ran the experiment using parallel with `parallel_temp_save=TRUE` and it crashes partway through, call this to recover the runs that were completed. Runs that were stopped mid-execution are not recoverable.

Public fields

`outrawdf` Raw data frame of output.

`outcleandf` Clean output in data frame.

`rungrid` matrix specifying which inputs will be run for each experiment.

`nvars` Number of variables

`allvars` All variables

`varlist` Character vector of objects to pass to a parallel cluster.

`arglist` List of values for each argument

`number_runs` Total number of runs

`completed_runs` Logical vector of whether each run has been completed.

`eval_func` The function that is called for each experiment trial.

`outlist` A list of the output from each run.

`save_output` Logical of whether the output should be saved.

`parallel` Logical whether experiment runs should be run in parallel. Allows for massive speedup.

`parallel_cores` How many cores to use when running in parallel. Can be an integer, or 'detect' will detect how many cores are available, or 'detect-1' will do one less than that.

`parallel_cluster` The parallel cluster being used.

`folder_path` The path to the folder where output will be saved.

`verbose` How much should be printed when running. 0 is none, 2 is average.

Methods

Public methods:

- `ffexp$new()`
- `ffexp$run_all()`
- `ffexp$run_one()`
- `ffexp$add_result_of_one()`
- `ffexp$plot_run_times()`
- `ffexp$calculate_effects()`
- `ffexp$save_self()`
- `ffexp$create_save_folder_if_nonexistent()`
- `ffexp$delete_save_folder_if_empty()`
- `ffexp$recover_parallel_temp_save()`
- `ffexp$rungrid2()`
- `ffexp$add_variable()`
- `ffexp$add_level()`
- `ffexp$print()`

- ffexp$stop_cluster()
- ffexp$finalize()
- ffexp$clone()

Method `new()`**:** Create an 'ffexp' object.

Usage:

```
ffexp$new(
  ...,
  eval_func,
  save_output = FALSE,
  parallel = FALSE,
  parallel_cores = "detect",
  folder_path,
  varlist = NULL,
  verbose = 2
)
```

Arguments:

`...` Input arguments for the experiment

`eval_func` The function to be run. It must take named arguments matching the names of ...

`save_output` Should output be saved to file?

`parallel` Should a parallel cluster be used?

`parallel_cores` When running in parallel, how many cores should be used.

`folder_path` Where the data and files should be stored. If not given, a folder in the existing directory will be created.

`varlist` Character vector of names of objects that need to be passed to the parallel environment.

`verbose` How much should be printed when running. 0 is none, 2 is average.

Method `run_all()`**:** Run an experiment. The user can choose to run all rows, or just specified ones, if it should be run in parallel, and what files should be saved.

Usage:

```
ffexp$run_all(
  to_run = NULL,
  redo = FALSE,
  run_order,
  save_output = self$save_output,
  parallel = self$parallel,
  parallel_temp_save = save_output,
  write_start_files = save_output,
  write_error_files = save_output,
  delete_parallel_temp_save_after = FALSE,
  varlist = self$varlist,
  verbose = self$verbose,
  warn_repeat = TRUE
)
```

Arguments:

`to_run` Which rows should be run? If NULL, then all that haven't been run yet.

`redo` Should already completed rows be run again?

`run_order` In what order should the rows by run? Options: random, in_order, and reverse.

`save_output` Should the output be saved?

`parallel` Should it be run in parallel?

`parallel_temp_save` Should temp files be written when running in parallel? Prevents losing results if it crashes partway through.

`write_start_files` Should start files be written?

`write_error_files` Should error files be written for rows that fail?

`delete_parallel_temp_save_after` If using parallel temp save files, should they be deleted afterwards?

`varlist` A character vector of names of variables to be passed the the parallel cluster.

`verbose` How much should be printed when running. 0 is none, 2 is average.

`warn_repeat` Should warnings be given when repeating already completed rows?

Method `run_one()`**:** Run a single row of the experiment. You can specify which one to run. Generally this should not be used by users, use 'run_all' instead.

Usage:

```
ffexp$run_one(
  irow = NULL,
  save_output = self$save_output,
  write_start_files = save_output,
  write_error_files = save_output,
  warn_repeat = TRUE,
  is_parallel = FALSE,
  verbose = self$verbose
)
```

Arguments:

`irow` Which row should be run?

`save_output` Should the output be saved?

`write_start_files` Should a file be written when starting the experiment?

`write_error_files` Should a file be written if there is an error?

`warn_repeat` Should a warning be given if repeating a row?

`is_parallel` Is this being run in parallel?

`verbose` How much should be printed when running. 0 is none, 2 is average.

Method `add_result_of_one()`**:** Add the result of a single experiment to the object. This shouldn't be used by users.

Usage:

```
ffexp$add_result_of_one(
  output,
  systime,
  irow,
  row_grid,
  row_df,
```

```
  start_time,
  end_time,
  save_output
)
```

Arguments:

`output` The output of the experiment.

`systime` The time it took to run

`irow` The row of inputs used.

`row_grid` The corresponding row in the run grid.

`row_df` The corresponding row data frame.

`start_time` The start time of the experiment.

`end_time` The end time of the experiment.

`save_output` Should the output be saved?

Method `plot_run_times()`**:** Plot the run times of each trial.

Usage:

```
ffexp$plot_run_times()
```

Method `calculate_effects()`**:** Calculate the effects of each variable as if this was an experiment using a linear model.

Usage:

```
ffexp$calculate_effects()
```

Method `save_self()`**:** Save this R6 object

Usage:

```
ffexp$save_self()
```

Method `create_save_folder_if_nonexistent()`**:** Create the save folder if it doesn't already exist.

Usage:

```
ffexp$create_save_folder_if_nonexistent()
```

Method `delete_save_folder_if_empty()`**:** Delete the save folder if it is empty. Used to prevent leaving behind empty folders.

Usage:

```
ffexp$delete_save_folder_if_empty()
```

Method `recover_parallel_temp_save()`**:** Running this loads the information saved to files if 'save_parallel_temp_save=TRUE' was used when running. Useful when running long jobs in parallel so that you don't lose all results if it crashes before finishing.

Usage:

```
ffexp$recover_parallel_temp_save(delete_after = TRUE)
```

Arguments:

`delete_after` Should the temp files be deleted after they are recovered? If TRUE, make sure you save the ffexp object after running this function so you don't lose the data.

Method `rungrid2()`**:** Display the input rows of the experiment. rungrid just gives integers, this gives the actual values.

Usage:

```
ffexp$rungrid2(rows = 1:nrow(self$rungrid))
```

Arguments:

`rows` Which rows to display the inputs for? On big experiments, specifying the rows can be much faster.

Method `add_variable()`**:** Add a variable to the experiment. You must specify the value of the variable for all existing rows, and then also the values of the variable which haven't been run yet.

Usage:

```
ffexp$add_variable(name, existing_value, new_values, suppressMessage = FALSE)
```

Arguments:

`name` Name of the variable being added.

`existing_value` Which existing argument is a level being added to?

`new_values` The values of the new variable which have not been run. This should not include 'arg_name', the name of the new variable at the existing values.

`suppressMessage` Should the message be suppressed? The message tells the user a new variable was added and it is being returned in a new object. Default FALSE.

Method `add_level()`**:** Add a level to one of the arguments. This returns a new object. The existing object is not changed.

Usage:

```
ffexp$add_level(arg_name, new_values, suppressMessage = FALSE)
```

Arguments:

`arg_name` Which existing argument is a level being added to?

`new_values` The value of the new levels to be added to 'arg_name'.

`suppressMessage` Should the message be suppressed? The message tells the user a new level was added and it is being returned in a new object. Default FALSE.

Method `print()`**:** Printing the object shows some summary information.

Usage:

```
ffexp$print()
```

Method `stop_cluster()`**:** Stop the parallel cluster.

Usage:

```
ffexp$stop_cluster()
```

Method `finalize()`**:** Cleanup after deleting object.

Usage:

```
ffexp$finalize()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ffexp$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Two factors, both with two levels.
#   The evaluation function simply prints out the combination
cc <- ffexp$new(a=1:2,b=c("A","B"),
                eval_func=function(...) {c(...)})
# View the factor settings it will run (each row).
cc$rungrid
# Evaluate all four settings
cc$run_all()


cc <- ffexp$new(a=1:3,b=2, cd=data.frame(c=3:4,d=5:6),
                eval_func=function(...) {list(...)})
```

---

mbc *Model benchmark compare*

Description

Compare the run time and output of various code chunks

Usage

```
mbc(
  ...,
  times = 5,
  input,
  inputi,
  evaluator,
  post,
  target,
  targetin,
  metric = "rmse",
  paired,
  kfold
)
```

Arguments

...	Functions to run
times	Number of times to run
input	Object to be passed as input to each function
inputi	Function to be called with the replicate number then passed to each function.
evaluator	An expression that the ... expressions will be passed as "." for evaluation.
post	Function or expression (using ".") to post-process results.
target	Values the functions are expected to (approximately) return.

targetin	Values that will be given to the result of the run to produce output.
metric	c("rmse", "t", "mis90", "sr27") Metric used to compare output values to target. mis90 is the mean interval score for 90% confidence, see Gneiting and Raftery (2007). sr27 is the scoring rule given in Equation 27 of Gneiting and Raftery (2007).
paired	Should the results be paired for comparison?
kfold	First element should be the number of elements that are being split into groups. If the number of folds is different from 'times', then the second argument is the number of folds. Use 'ki' in 'inputi' and 'targeti' to select elements in the current fold.

Value

Data frame of comparison results

References

Gneiting, T., & Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. Journal of the American Statistical Association, 102(477), 359-378.

Examples

```
# Compare distribution of mean for different sample sizes
mbc(mean(rnorm(1e2)),
    mean(rnorm(1e4)),
    times=20)

# Compare mean and median on same data
mbc(mean(x),
    median(x),
    inputi={x=rexp(1e2)})

# input given, no post
mbc({Sys.sleep(rexp(1, 30));mean(x)},
     {Sys.sleep(rexp(1, 5));median(x)},
    inputi={x=runif(100)})

# input given with post
mbc(mean={Sys.sleep(rexp(1, 30));mean(x)},
    med={Sys.sleep(rexp(1, 5));median(x)},
    inputi={x=runif(100)},
    post=function(x){c(x+1, x^2)})

# input given with post, 30 times
mbc(mean={Sys.sleep(rexp(1, 30));mean(x)+runif(1)},
    med={Sys.sleep(rexp(1, 50));median(x)+runif(1)},
    inputi={x=runif(100)},
    post=function(x){c(x+1, x^2)}, times=10)

# Name one function and post
mbc({mean(x)+runif(1)},
```

```
    a1={median(x)+runif(1)},
    inputi={x=runif(100)},
    post=function(x){c(rr=x+1, gg=x^2)}, times=10)

# No input
m1 <- mbc(mean={x <- runif(100);Sys.sleep(rexp(1, 30));mean(x)},
          med={x <- runif(100);Sys.sleep(rexp(1, 50));median(x)})
```

`plot.mbc` *Plot mbc class*

Description

Plot mbc class

Usage

```
## S3 method for class 'mbc'
plot(x, ...)
```

Arguments

`x`	Object of class mbc
`...`	Additional parameters

Value

None

Examples

```
m1 <- mbc(mn= {Sys.sleep(rexp(1, 30));mean(x)},
          med={Sys.sleep(rexp(1, 5));median(x)},
          input=runif(100))
plot(m1)
```

`print.mbc` *Print mbc class*

Description

Print mbc class

Usage

```
## S3 method for class 'mbc'
print(x, ...)
```

Arguments

x	Object of class mbc
...	Additional parameters

Value

None

Examples

```
m1 <- mbc({Sys.sleep(rexp(1, 30));mean(x)},
          {Sys.sleep(rexp(1, 5));median(x)},
          input=runif(100))
print(m1)
```

Index