

Package ‘Rfuzzycoco’

May 7, 2026

Type Package

Title Provides an R Interface to the 'FuzzyCoCo' C++ Library and
Extends It

Version 0.1.0

Description Provides and extends the 'Fuzzy Coco' algorithm by wrapping the 'FuzzyCoCo' 'C++' Library, cf <https://github.com/Lonza-RND-Data-Science/fuzzycoco>. 'Fuzzy Coco' constructs systems that predict the outcome of a human decision-making process while providing an understandable explanation of a possible reasoning leading to it. The constructed fuzzy systems are composed of rules and linguistic variables. This package provides a 'S3' classic interface (`fit_xy()/fit()/predict()/evaluate()`) and a 'tidymodels'/parsnip' interface, a custom engine with custom iteration stop criterion and progress bar support as well as a systematic implementation that do not rely on genetic programming but rather explore all possible combinations.

License GPL (>= 3)

URL <https://github.com/Lonza-RND-Data-Science/Rfuzzycoco>

BugReports <https://github.com/Lonza-RND-Data-Science/Rfuzzycoco/issues>

Depends R (>= 4.1.0)

Imports generics, methods, Rcpp, stats, utils

Suggests knitr, parsnip, progressr, rlang, rmarkdown, testthat

LinkingTo Rcpp

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

SystemRequirements C++17

NeedsCompilation yes

Author Karl Forner [aut, cre]

Maintainer Karl Forner <karl.forner@gmail.com>

Repository CRAN

Date/Publication 2025-10-21 09:10:51 UTC

Contents

| | |
|--|-----------|
| Rfuzzycoco-package | 2 |
| compute_optimal_quantile_fuzzy_set_positions | 3 |
| evaluate.fuzzycoco_fit | 4 |
| evaluate_fuzzy_system | 5 |
| example_iris36 | 5 |
| example_iris_binary_categorical | 6 |
| example_mtcars | 7 |
| fit.fuzzycoco_model | 7 |
| fit_to_df | 8 |
| fit_xy.fuzzycoco_model | 9 |
| fs_rules_to_df | 10 |
| fs_used_vars_to_df | 11 |
| fuzzycoco | 12 |
| fuzzycoco_fit_df_hybrid | 12 |
| fuzzy_coco_parsnip | 13 |
| fuzzy_coco_systematic_fit | 14 |
| params | 15 |
| predict.fuzzycoco_fit | 17 |
| predict_fuzzy_system | 18 |
| stop_engine_if_stalling | 18 |
| stop_engine_on_first_of | 19 |
| Index | 20 |

| | |
|--------------------|--|
| Rfuzzycoco-package | <i>Rfuzzycoco: Provides an R Interface to the 'FuzzyCoCo' C++ Library and Extends It</i> |
|--------------------|--|

Description

Provides and extends the 'Fuzzy Coco' algorithm by wrapping the 'FuzzyCoCo' 'C++' Library, cf <https://github.com/Lonza-RND-Data-Science/fuzzycoco>. 'Fuzzy Coco' constructs systems that predict the outcome of a human decision-making process while providing an understandable explanation of a possible reasoning leading to it. The constructed fuzzy systems are composed of rules and linguistic variables. This package provides a 'S3' classic interface (fit_xy()/fit()/predict()/evaluate()) and a 'tidymodels'/'parsnip' interface, a custom engine with custom iteration stop criterion and progress bar support as well as a systematic implementation that do not rely on genetic programming but rather explore all possible combinations.

Features

Rfuzzycoco provides the FuzzyCoCo algorithm, cf *Fuzzy CoCo: a cooperative-coevolutionary approach to fuzzy modeling* from [Carlos Andrés Peña-Reyes](#)

Author(s)

Maintainer: Karl Forner <karl.forner@gmail.com>

See Also

Useful links:

- <https://github.com/Lonza-RND-Data-Science/Rfuzzycoco>
- Report bugs at <https://github.com/Lonza-RND-Data-Science/Rfuzzycoco/issues>

compute_optimal_quantile_fuzzy_set_positions

computes the optimal fuzzy set positions based on the distribution of the data

Description

computes the optimal fuzzy set positions based on the distribution of the data

Usage

```
compute_optimal_quantile_fuzzy_set_positions(df, nb_sets)
```

Arguments

| | |
|---------|--------------------------|
| df | the data as a data frame |
| nb_sets | the number of fuzzy sets |

Value

a list, named after the df column names, holding the vector of positions per variable

Examples

```
pos <- compute_optimal_quantile_fuzzy_set_positions(mtcars, 3)
print("position for 2nd fuzzy set for qsec var", pos$qsec[[2]])
```

`evaluate.fuzzycoco_fit`*evaluate the fuzzy system from a fit on some given data*

Description

N.B: just a S3 method method wrapping the `evaluate_fuzzy_system()` function

Usage

```
## S3 method for class 'fuzzycoco_fit'  
evaluate(x, data, verbose = FALSE, ...)
```

Arguments

| | |
|----------------------|--|
| <code>x</code> | the fuzzycoco_fit object containing the fuzzy system to evaluate |
| <code>data</code> | the data to evaluate the fuzzy system on |
| <code>verbose</code> | whether to be verbose |
| <code>...</code> | not used. Only for S3 generic consistency |

Value

the evaluation as a named list:

- `fitness`: the fitness value
- `metrics`: the evaluation metrics as a named list

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params, seed = 123)  
df <- mtcars[c("mpg", "hp", "wt", "qsec")]  
fit <- fit(model, qsec ~ ., df, engine = "rcpp", seed = 456, max_generations = 20)  
  
res <- evaluate(fit, df)  
print(res$fitness)
```

evaluate_fuzzy_system *evaluate the fuzzy system from a fit on some given data*

Description

evaluate the fuzzy system from a fit on some given data

Usage

```
evaluate_fuzzy_system(fs, data, params, verbose = FALSE)
```

Arguments

| | |
|---------|--|
| fs | the fuzzy system to evaluate (as a named list) |
| data | the data to evaluate the fuzzy system on |
| params | the fuzzycoco parameters. probably not needed... |
| verbose | whether to be verbose |

Value

the evaluation as a named list:

- fitness: the fitness value
- metrics: the evaluation metrics as a named list

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params, seed = 123)
x <- mtcars[c("mpg", "hp", "wt")]
y <- mtcars["qsec"]
fit <- fit_xy(model, x, y, progress = FALSE)

res <- evaluate_fuzzy_system(fit$fuzzy_system, cbind(x, y), fit$params)
print(res$metrics$rmse)
```

example_iris36 *model parameters and data for the IRIS36 classification example*

Description

a small (36 rows) dataset extracted from iris with a binary 0/1 outcome OUT response variable

Usage

```
example_iris36()
```

Value

the example as a named list with:

- params: the model parameters
- data: the data to fit as a data frame

Examples

```
model <- fuzzycoco("classification", example_iris36())$params, seed = 123)
fit <- fit(model, OUT ~ ., example_iris36())$data,
max_generations = 20, progress = FALSE)
```

example_iris_binary_categorical

model parameters and data for the IRIS36 classification example

Description

an example dataset based on iris with a binary categorical (non-factor) response

Usage

```
example_iris_binary_categorical()
```

Value

the example as a named list with:

- params: the model parameters
- data: the data to fit as a data frame

Examples

```
model <- fuzzycoco("classification", example_iris_binary_categorical())$params)
fit <- fit(model, Species ~ ., example_iris_binary_categorical())$data,
max_generations = 20, progress = FALSE)
```

example_mtcars *model parameters and data for the mtcars regression example*

Description

model parameters and data for the mtcars regression example

Usage

```
example_mtcars()
```

Value

the example as a named list with:

- params: the model parameters
- data: the data to fit as a data frame

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params
fit <- fit(model, qsec ~ ., example_mtcars())$data, max_generations = 20, progress = FALSE)
```

fit.fuzzycoco_model *fit the FuzzyCoco model using the formula interface*

Description

N.B: fix_xy() is the workhorse, fit() is a simple formula-based layer

Usage

```
## S3 method for class 'fuzzycoco_model'
fit(
  object,
  formula,
  data,
  engine = FUZZY_COCO_HYBRID_ENGINE,
  max_generations = object$params$global_params$max_generations,
  max_fitness = object$params$global_params$max_fitness,
  seed = object$seed,
  verbose = object$verbose,
  ...
)
```

Arguments

| | |
|-----------------|--|
| object | the <i>fuzzycoco_model</i> object to fit |
| formula | the fuzzy coco model as a formula |
| data | the data to fit as a data frame. The output variables must be grouped AFTER the input variables |
| engine | the fuzzy coco fit engine to use, one of rcpp and hybrid |
| max_generations | The maximum number of iterations of the algorithm. Each iteration produces a new generation of the rules and membership functions populations. |
| max_fitness | a stop condition: the iterations stop as soon as a generated fuzzy system fitness reaches that threshold. |
| seed | the RNG seed to use (to fit the model) |
| verbose | whether to be verbose |
| ... | Arguments passed on to <code>fit_xy.fuzzycoco_model</code> |
| | x the input variables data (usually to fit) as a dataframe |
| | y the output variables data (usually to fit) as a dataframe |

Value

the fit as a named list

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params, seed = 123)
df <- mtcars[c("mpg", "hp", "wt", "qsec")]
fit <- fit(model, qsec ~ ., df, seed = 456, max_generations = 10, progress = FALSE)
print(names(fit))
```

| | |
|-----------|---|
| fit_to_df | <i>a one-row overview of a fuzzy system with the usage of variables, the fitness, number of generations and optionally a metric</i> |
|-----------|---|

Description

a one-row overview of a fuzzy system with the usage of variables, the fitness, number of generations and optionally a metric

Usage

```
fit_to_df(fit, metric = NULL)
```

Arguments

| | |
|--------|---|
| fit | a fit object, as returned by [fit. |
| metric | an optional metric name to report (e.g. rmse) |

Value

a one-row data frame

See Also

Other fit_utils: [fs_rules_to_df\(\)](#), [fs_used_vars_to_df\(\)](#)

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params, seed = 123)
df <- mtcars[c("mpg", "hp", "wt", "qsec")]
fit <- fit(model, qsec ~ ., df, seed = 456, max_generations = 10, progress = FALSE)

print(fit_to_df(fit))
```

fit_xy.fuzzycoco_model

fit the FuzzyCoco model using the dataframe interface

Description

N.B: the underlying C++ implementation is able to automatically set some missing parameters (NA). The final parameters are those returned by the function, is the params slot.

Usage

```
## S3 method for class 'fuzzycoco_model'
fit_xy(
  object,
  x,
  y,
  engine = FUZZY_COCO_HYBRID_ENGINE,
  max_generations = object$params$global_params$max_generations,
  max_fitness = object$params$global_params$max_fitness,
  seed = object$seed,
  verbose = object$verbose,
  ...
)
```

Arguments

| | |
|--------|---|
| object | the <i>fuzzycoco_model</i> object to fit |
| x | the input variables data (usually to fit) as a dataframe |
| y | the output variables data (usually to fit) as a dataframe |
| engine | the fuzzy coco fit engine to use, one of rpp and hybrid |

max_generations The maximum number of iterations of the algorithm. Each iteration produces a new generation of the rules and membership functions populations.
max_fitness a stop condition: the iterations stop as soon as a generated fuzzy system fitness reaches that threshold.
seed the RNG seed to use (to fit the model)
verbose whether to be verbose
... Arguments passed on to [fuzzycoco_fit_df_hybrid](#)
model a Fuzzy Coco model as a `fuzzy_coco` object
progress whether to display the computation progress (using `progressr`, if available)
until function that takes an engine and returns TRUE if and only if the evolution must stop. It is a way for the user to customize the stop conditions of the algorithm.

Value

the fit as a named list

Examples

```

model <- fuzzycoco("regression", example_mtcars())$params, seed = 123)
x <- mtcars[c("mpg", "hp", "wt")]
y <- mtcars["qsec"]
fit <- fit_xy(model, x, y, progress = FALSE)
print(names(fit))

```

fs_rules_to_df *format the fuzzy rules as a data frame*

Description

format the fuzzy rules as a data frame

Usage

```
fs_rules_to_df(fuzzy_system_desc)
```

Arguments

fuzzy_system_desc
 a fuzzy system description as a named list

Value

a data frame, one row per rule, including the default rule, in columns the input and output variables. The values are the corresponding fuzzy set number.

See Also

Other fit_utils: [fit_to_df\(\)](#), [fs_used_vars_to_df\(\)](#)

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params, seed = 123)
df <- mtcars[c("mpg", "hp", "wt", "qsec")]
fit <- fit(model, qsec ~ ., df, seed = 456, max_generations = 10, progress = FALSE)

print(fs_rules_to_df(fit$fuzzy_system))
```

fs_used_vars_to_df *extract the usage of the variables by a fuzzy system*

Description

extract the usage of the variables by a fuzzy system

Usage

```
fs_used_vars_to_df(fuzzy_system_desc)
```

Arguments

fuzzy_system_desc
a fuzzy system description as a named list

Value

a one-row data frame, in columns the input and output variables, with TRUE iff the variable is used.

See Also

Other fit_utils: [fit_to_df\(\)](#), [fs_rules_to_df\(\)](#)

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params, seed = 123)
df <- mtcars[c("mpg", "hp", "wt", "qsec")]
fit <- fit(model, qsec ~ ., df, seed = 456, max_generations = 10, progress = FALSE)

print(fs_rules_to_df(fit$fuzzy_system))
```

fuzzycoco *creates a model for the Fuzzy Coco algorithm*

Description

creates a model for the Fuzzy Coco algorithm

Usage

```
fuzzycoco(
  mode = c("classification", "regression"),
  params,
  seed = sample.int(10^5, 1),
  verbose = FALSE
)
```

Arguments

| | |
|---------|---|
| mode | the type of model, either classification or regression |
| params | fuzzy coco parameters, as a recursive named list, cf params() |
| seed | the RNG seed to use (to fit the model) |
| verbose | whether to be verbose |

Value

a *fuzzycoco_model* object (named list)

Examples

```
model <- fuzzycoco("regression", params(nb_rules = 1, nb_max_var_per_rule = 3), seed = 123)
```

fuzzycoco_fit_df_hybrid
*lowest-level implementation of the fitting of a fuzzy coco model using
the **hybrid engine***

Description

lowest-level implementation of the fitting of a fuzzy coco model using the **hybrid engine**

Usage

```
fuzzycoco_fit_df_hybrid(
  model,
  x,
  y,
  until = stop_engine_on_first_of(max_generations =
    model$params$global_params$max_generations, max_fitness =
    model$params$global_params$max_fitness),
  verbose = model$verbose,
  progress = TRUE
)
```

Arguments

| | |
|----------|--|
| model | a Fuzzy Coco model as a fuzzy_coco object |
| x | the input variables data (usually to fit) as a dataframe |
| y | the output variables data (usually to fit) as a dataframe |
| until | function that takes an engine and returns TRUE if and only if the evolution must stop. It is a way for the user to customize the stop conditions of the algorithm. |
| verbose | whether to be verbose |
| progress | whether to display the computation progress (using progressr, if available) |

Value

a named list as a fuzzy_coco fit object

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params
fit <- fuzzycoco_fit_df_hybrid(model, mtcars[c("mpg", "hp", "wt")], mtcars["qsec"])
```

fuzzy_coco_parsnip *parsnip model function*

Description

parsnip model function

Usage

```
fuzzy_coco_parsnip(
  mode = "unknown",
  params,
  engine = FUZZY_COCO_HYBRID_ENGINE,
  seed = sample.int(10^5, 1),
  verbose = FALSE
)
```

Arguments

| | |
|---------|---|
| mode | the type of model, either classification or regression |
| params | fuzzy coco parameters, as a recursive named list, cf params() |
| engine | the fuzzy coco fit engine to use, one of rcpp and hybrid |
| seed | the RNG seed to use (to fit the model) |
| verbose | whether to be verbose |

Value

a parsnip model

Examples

```
spec <- fuzzy_coco_parsnip("regression", params = example_mtcars()$params, seed = 123)
fit <- spec |> parsnip::set_engine("hybrid") |> parsnip::fit(qsec ~ ., data = example_mtcars()$data)
```

fuzzy_coco_systematic_fit
systematic search

Description

This is a R implementation of a systematic search, where the fuzzy set positions are determined by the distribution of the data (cf [compute_optimal_quantile_fuzzy_set_positions\(\)](#)) and the rules are systematically explored

Usage

```
fuzzy_coco_systematic_fit(x, y, params, fitter)
```

Arguments

| | |
|--------|--|
| x | the input variables data (usually to fit) as a dataframe |
| y | the output variables data (usually to fit) as a dataframe |
| params | fuzzy coco parameters, as a recursive named list, cf params() |
| fitter | a function metrics → fitness value providing the objective/fitness function to optimize TODO: describe the metrics |

Details

N.B: this is experimental, only possible for a small number of variables. Not all parameters are used, obviously, and currently `fitness_params$output_vars_defuzz_thresholds` has to be set explicitly.

Value

a list of the best results (all ties). Each result is also a named list(metric=,fs=) holding the corresponding metric value and the fuzzy system.

Examples

```
fitter <- function(metrics) 2^-metrics$rms
params <- example_mtcars()$params
params$fitness_params$output_vars_defuzz_thresholds <- 0
params$global_params$nb_rules <- 1
params$global_params$nb_max_var_per_rule <- 2
params$output_vars_params$nb_sets <- 2

x <- mtcars[c("mpg", "hp", "wt")]
y <- mtcars["qsec"]
fit <- fuzzy_coco_systematic_fit(x, y, params, fitter)
```

 params

utility to build the Fuzzy Coco parameters data structure

Description

utility to build the Fuzzy Coco parameters data structure

Usage

```
params(
  nb_rules,
  nb_max_var_per_rule,
  max_generations = 100,
  max_fitness = 1,
  nb_cooperators = 2,
  influence_rules_initial_population = FALSE,
  influence_evolution_ratio = 0.8,
  ivars.nb_sets = 3,
  ivars.nb_bits_vars = NA_integer_,
  ivars.nb_bits_sets = NA_integer_,
  ivars.nb_bits_pos = NA_integer_,
  ovars.nb_sets = 3,
  ovars.nb_bits_vars = NA_integer_,
  ovars.nb_bits_sets = NA_integer_,
  ovars.nb_bits_pos = NA_integer_,
  rules.pop_size = 100,
  rules.elite_size = 5,
  rules.cx_prob = 0.5,
  rules.mut_flip_genome = 0.5,
  rules.mut_flip_bit = 0.025,
```

```

mfs.pop_size = 100,
mfs.elite_size = 5,
mfs.cx_prob = 0.5,
mfs.mut_flip_genome = 0.5,
mfs.mut_flip_bit = 0.025,
output_vars_defuzz_thresholds = NA,
metricsw.sensitivity = 1,
metricsw.specificity = 0.8,
metricsw.accuracy = 0,
metricsw.ppv = 0,
metricsw.rmse = 0,
metricsw.rrse = 0,
metricsw.rae = 0,
metricsw.mse = 0,
metricsw.distanceThreshold = 0,
metricsw.distanceMinThreshold = 0,
metricsw.nb_vars = 0,
metricsw.overLearn = 0,
metricsw.true_positives = 0,
metricsw.false_positives = 0,
metricsw.true_negatives = 0,
metricsw.false_negatives = 0,
features_weights = list()
)

```

Arguments

`nb_rules` (mandatory) the number of rules in the fuzzy system

`nb_max_var_per_rule`

(mandatory) The maximum number of antecedents (input variables) to use in each rule.

`max_generations,` `max_fitness,` `nb_cooperators,`
`influence_rules_initial_population,` `influence_evolution_ratio,`
`ivars.nb_sets,` `ivars.nb_bits_vars,` `ivars.nb_bits_sets,`
`ivars.nb_bits_pos,` `ovars.nb_sets,` `ovars.nb_bits_vars,`
`ovars.nb_bits_sets,` `ovars.nb_bits_pos,` `rules.pop_size,`
`rules.elite_size,` `rules.cx_prob,` `rules.mut_flip_genome,`
`rules.mut_flip_bit,` `mfs.pop_size,` `mfs.elite_size,`
`mfs.cx_prob,` `mfs.mut_flip_genome,` `mfs.mut_flip_bit,`
`output_vars_defuzz_thresholds,` `metricsw.sensitivity,`
`metricsw.specificity,` `metricsw.accuracy,` `metricsw.ppv,`
`metricsw.rmse,` `metricsw.rrse,` `metricsw.rae,` `metricsw.mse,`
`metricsw.distanceThreshold,` `metricsw.distanceMinThreshold,`
`metricsw.nb_vars,` `metricsw.overLearn,` `metricsw.true_positives,`
`metricsw.false_positives,` `metricsw.true_negatives,`
`metricsw.false_negatives,` `features_weights`

cf [fuzzycoco doc](#)

Value

a nested named list

Examples

```
pms <- params(
  nb_rules = 2, nb_max_var_per_rule = 3, rules.pop_size = 20, mfs.pop_size = 20,
  ivars.nb_sets = 3, ivars.nb_bits_vars = 3, ivars.nb_bits_sets = 2, ivars.nb_bits_pos = 8,
  ovars.nb_sets = 3, ovars.nb_bits_vars = 1, ovars.nb_bits_sets = 2, ovars.nb_bits_pos = 8,
  metricsw.sensitivity = 0, metricsw.specificity = 0, metricsw.rmse = 1,
  output_vars_defuzz_thresholds = list(3, 17)
)
```

predict.fuzzycoco_fit *predict the outcome on some input data using a fitted model*

Description

N.B: just a S3 method method wrapping the [predict_fuzzy_system\(\)](#) function

Usage

```
## S3 method for class 'fuzzycoco_fit'
predict(object, x, verbose = FALSE, bin = TRUE, ...)
```

Arguments

| | |
|---------|---|
| object | the fuzzycoco_fit object containing the fuzzy system to predict on |
| x | the input data to use with the fuzzy system to predict the output |
| verbose | whether to be verbose |
| bin | whether to transform the output data into a binary response. Only applies to classification models. |
| ... | not used. Only for S3 generic consistency |

Value

the predicted output data as a data frame

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params, seed = 123)
x <- mtcars[c("mpg", "hp", "wt")]
y <- mtcars["qsec"]
fit <- fit_xy(model, x, y, progress = FALSE)

y2 <- predict(fit, x)
```

predict_fuzzy_system *predict the outcome of a fuzzy system on some input data*

Description

predict the outcome of a fuzzy system on some input data

Usage

```
predict_fuzzy_system(fs, x, verbose = FALSE)
```

Arguments

| | |
|---------|---|
| fs | the fuzzy system to predict on (as a named list) |
| x | the input data to use with the fuzzy system to predict the output |
| verbose | whether to be verbose |

Value

the predicted output data as a data frame

Examples

```
model <- fuzzycoco("regression", example_mtcars())$params, seed = 123)
x <- mtcars[c("mpg", "hp", "wt")]
y <- mtcars["qsec"]
fit <- fit_xy(model, x, y, progress = FALSE)

y2 <- predict_fuzzy_system(fit$fuzzy_system,x)
```

stop_engine_if_stalling

an utility function to easily generate a stop function that stops when the convergence is stalling

Description

an utility function to easily generate a stop function that stops when the convergence is stalling

Usage

```
stop_engine_if_stalling(nb_iterations)
```

Arguments

| | |
|---------------|--|
| nb_iterations | number of iterations of the stalling: stops if the fitness has not increased during that number of iterations. |
|---------------|--|

Value

a function: (engine) -> logical that stops (i.e/ returns TRUE) if the convergence is stalling

Examples

```
until <- stop_engine_on_first_of(max_generations = 1000, other_func = stop_engine_if_stalling(5))
```

```
stop_engine_on_first_of
```

an utility function to easily generate the commonly used until parameter, as used by [fuzzycoco_fit_df_hybrid\(\)](#)

Description

an utility function to easily generate the commonly used until parameter, as used by [fuzzycoco_fit_df_hybrid\(\)](#)

Usage

```
stop_engine_on_first_of(
  max_generations = NULL,
  max_fitness = NULL,
  other_func = NULL
)
```

Arguments

`max_generations`

The maximum number of iterations of the algorithm. Each iteration produces a new generation of the rules and membership functions populations.

`max_fitness`

a stop condition: the iterations stop as soon as a generated fuzzy system fitness reaches that threshold.

`other_func`

if not NULL, a function: (engine) ->logical that should return TRUE to stop the evolution (cf [stop_engine_if_stalling\(\)](#))

Value

a function: (engine) -> logical that stops (i.e/ returns TRUE) when the number of generations or the fitness are reached, or when the other_func if provided returns TRUE

Examples

```
until <- stop_engine_on_first_of(max_generations = 100)
until <- stop_engine_on_first_of(max_generations = 100, max_fitness = 0.8)
until <- stop_engine_on_first_of(max_fitness = 0.9, other_func = stop_engine_if_stalling(5))
```

Index

- * **fit_utils**
 - fit_to_df, [8](#)
 - fs_rules_to_df, [10](#)
 - fs_used_vars_to_df, [11](#)

- compute_optimal_quantile_fuzzy_set_positions,
[3](#)
- compute_optimal_quantile_fuzzy_set_positions(),
[14](#)

- evaluate.fuzzycoco_fit, [4](#)
- evaluate_fuzzy_system, [5](#)
- evaluate_fuzzy_system(), [4](#)
- example_iris36, [5](#)
- example_iris_binary_categorical, [6](#)
- example_mtcars, [7](#)

- fit.fuzzycoco_model, [7](#)
- fit_to_df, [8](#), [11](#)
- fit_xy.fuzzycoco_model, [8](#), [9](#)
- fs_rules_to_df, [9](#), [10](#), [11](#)
- fs_used_vars_to_df, [9](#), [11](#), [11](#)
- fuzzy_coco_parsnip, [13](#)
- fuzzy_coco_systematic_fit, [14](#)
- fuzzycoco, [12](#)
- fuzzycoco_fit_df_hybrid, [10](#), [12](#)
- fuzzycoco_fit_df_hybrid(), [19](#)

- params, [15](#)
- params(), [12](#), [14](#)
- predict.fuzzycoco_fit, [17](#)
- predict_fuzzy_system, [18](#)
- predict_fuzzy_system(), [17](#)

- Rfuzzycoco (Rfuzzycoco-package), [2](#)
- Rfuzzycoco-package, [2](#)

- stop_engine_if_stalling, [18](#)
- stop_engine_if_stalling(), [19](#)
- stop_engine_on_first_of, [19](#)