# Package 'counterfactuals'

October 17, 2024

**Type** Package

**Title** Counterfactual Explanations

**Version** 0.1.6

**Maintainer** Susanne Dandl <dandls.datascience@gmail.com>

**Description** Modular and unified R6-based interface for counterfactual explanation methods. The following methods are currently implemented: Burghmans et al. (2022) <doi:10.48550/arXiv.2104.07411>, Dandl et al. (2020) <doi:10.1007/978-3-030-58112-1_31> and Wexler et al. (2019) <doi:10.1109/TVCG.2019.2934619>. Optional extensions allow these methods to be applied to a variety of models and use cases. Once generated, the counterfactuals can be analyzed and visualized by provided functionalities.

**URL** https://github.com/dandls/counterfactuals

**BugReports** https://github.com/dandls/counterfactuals/issues

**Depends** R (>= 3.5.0)

**Imports** R6, checkmate, StatMatch, iml, data.table, paradox, miesmuschel, bbotk

**Suggests** gower, randomForest, GGally, trtf, testthat, MASS, R.rsp, cowplot, covr, ggplot2, keras, rchallenge, gamlss.data, partykit, mlt, variables, basefun, rmarkdown, rpart, mlr3, mlr3learners, mlr3pipelines, tidymodels, caret, mlr

**License** LGPL-3

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**VignetteBuilder** R.rsp

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Susanne Dandl [aut, cre] (<https://orcid.org/0000-0003-4324-4163>), Andreas Hofheinz [aut], Martin Binder [ctb], Giuseppe Casalicchio [ctb]

**Repository** CRAN

**Date/Publication** 2024-10-17 12:00:06 UTC

# Contents



---

`CounterfactualMethod` *Base class for Counterfactual Explanation Methods*

---

## Description

Abstract base class for counterfactual explanation methods.

## Inheritance

Child classes: CounterfactualMethodClassif, CounterfactualMethodRegr

## Methods

### Public methods:

- `CounterfactualMethod$new()`
- `CounterfactualMethod$print()`
- `CounterfactualMethod$clone()`

**Method** `new()`**:** Creates a new `CounterfactualMethod` object.

*Usage:*

```
CounterfactualMethod$new(
  predictor,
  lower = NULL,
  upper = NULL,
  distance_function = NULL
)
```

*Arguments:*

`predictor` (Predictor)
: The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`lower` (`numeric()` | `NULL`)
: Vector of minimum values for numeric features. If `NULL` (default), the element for each numeric feature in `lower` is taken as its minimum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`upper` (`numeric()` | `NULL`)
: Vector of maximum values for numeric features. If `NULL` (default), the element for each numeric feature in `upper` is taken as its maximum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`distance_function` (`character(1)` | `function()`)
: Either the name of an already implemented distance function (currently 'gower' or 'gower_c') or a function having three arguments: `x`, `y`, and `data`. The function should return a `double` matrix with `nrow(x)` rows and maximum `nrow(y)` columns.

**Method** `print()`**:** Prints a `CounterfactualMethod` object. The method calls a (private) `$print_parameters()` method which should be implemented by the leaf classes.

*Usage:*

```
CounterfactualMethod$print()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

`CounterfactualMethodClassif` *Base class for Counterfactual Explanation Methods for Classification Tasks*

---

## Description

Abstract base class for counterfactual explanation methods for classifcation tasks.

`CounterfactualMethodClassif` can only be initialized for classification tasks. Child classes inherit the (public) `$find_counterfactuals()` method, which calls a (private) `$run()` method. This `$run()` method should be implemented by the child classes and return the counterfactuals as a `data.table` (preferably) or a `data.frame`.

## Inheritance

Child classes: MOCClassif, WhatIfClassif, NICEClassif

## Super class

counterfactuals::CounterfactualMethod -> CounterfactualMethodClassif

## Methods

### Public methods:

- CounterfactualMethodClassif$new()
- CounterfactualMethodClassif$find_counterfactuals()
- CounterfactualMethodClassif$clone()

**Method** new(): Creates a new CounterfactualMethodClassif object.

*Usage:*

```
CounterfactualMethodClassif$new(
  predictor,
  lower = NULL,
  upper = NULL,
  distance_function = NULL
)
```

*Arguments:*

predictor (Predictor)
: The object (created with iml::Predictor$new()) holding the machine learning model and the data.

lower (numeric() | NULL)
: Vector of minimum values for numeric features. If NULL (default), the element for each numeric feature in lower is taken as its minimum value in predictor$data$X. If not NULL, it should be named with the corresponding feature names.

upper (numeric() | NULL)
: Vector of maximum values for numeric features. If NULL (default), the element for each numeric feature in upper is taken as its maximum value in predictor$data$X. If not NULL, it should be named with the corresponding feature names.

distance_function (function() | NULL)
: A distance function that may be used by the leaf classes. If specified, the function must have three arguments: x, y, and data and return a double matrix with nrow(x) rows and nrow(y) columns.

**Method** find_counterfactuals(): Runs the counterfactual method and returns the counterfactuals. It searches for counterfactuals that have a predicted probability in the interval desired_prob for the desired_class.

*Usage:*

```
CounterfactualMethodClassif$find_counterfactuals(
  x_interest,
  desired_class = NULL,
  desired_prob = c(0.5, 1)
)
```

*Arguments:*

`x_interest` (`data.table(1)` | `data.frame(1)`)
A single row with the observation of interest.

`desired_class` (`character(1)` | `NULL`)
The desired class. If `NULL` (default) then `predictor$class` is taken.

`desired_prob` (`numeric(1)` | `numeric(2)`)
The desired predicted probability of the `desired_class`. It can be a numeric scalar or a vector with two numeric values that specify a probability interval. For hard classification tasks this can be set to `0` or `1`, respectively. A scalar is internally converted to an interval.

*Returns:* A Counterfactuals object containing the results.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

`CounterfactualMethodRegr` *Base class for Counterfactual Explanation Methods for Regression Tasks*

---

## Description

Abstract base class for counterfactual explanation methods for regression tasks.

`CounterfactualMethodRegr` can only be initialized for regression tasks. Child classes inherit the (public) `$find_counterfactuals()` method, which calls a (private) `$run()` method. This `$run()` method should be implemented by the child classes and return the counterfactuals as a `data.table` (preferably) or a `data.frame`.

## Inheritance

Child classes: MOCRegr, WhatIfRegr, NICERegr

## Super class

`counterfactuals::CounterfactualMethod` -> `CounterfactualMethodRegr`

## Methods

### Public methods:

- `CounterfactualMethodRegr$new()`
- `CounterfactualMethodRegr$find_counterfactuals()`
- `CounterfactualMethodRegr$clone()`

**Method** `new()`**:** Creates a new CounterfactualMethodRegr object.

*Usage:*

```
CounterfactualMethodRegr$new(
  predictor,
  lower = NULL,
  upper = NULL,
  distance_function = NULL
)
```

*Arguments:*

`predictor` (Predictor)
: The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`lower` (`numeric()` | `NULL`)
: Vector of minimum values for numeric features. If `NULL` (default), the element for each numeric feature in `lower` is taken as its minimum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`upper` (`numeric()` | `NULL`)
: Vector of maximum values for numeric features. If `NULL` (default), the element for each numeric feature in `upper` is taken as its maximum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`distance_function` (`function()` | `NULL`)
: A distance function that may be used by the leaf classes. If specified, the function must have three arguments: `x`, `y`, and `data` and return a `double` matrix with `nrow(x)` rows and `nrow(y)` columns.

**Method** `find_counterfactuals()`**:** Runs the counterfactual method and returns the counterfactuals. It searches for counterfactuals that have a predicted outcome in the interval `desired_outcome`.

*Usage:*

```
CounterfactualMethodRegr$find_counterfactuals(x_interest, desired_outcome)
```

*Arguments:*

`x_interest` (`data.table(1)` | `data.frame(1)`)
: A single row with the observation of interest.

`desired_outcome` (`numeric(1)` | `numeric(2)`)
: The desired predicted outcome. It can be a numeric scalar or a vector with two numeric values that specify an outcome interval. A scalar is internally converted to an interval.

*Returns:* A Counterfactuals object containing the results.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

`Counterfactuals` *Counterfactuals Class*

## Description

A `Counterfactuals` object should be created by the `$find_counterfactuals` method of CounterfactualMethodRegr or CounterfactualMethodClassif. It contains the counterfactuals and has several methods for their evaluation and visualization.

## Active bindings

`desired (list(1) | list(2))`
: A `list` with the desired properties of the counterfactuals. For regression tasks it has one element `desired_outcome` (CounterfactualMethodRegr) and for classification tasks two elements `desired_class` and `desired_prob` (CounterfactualMethodClassif).

`data (data.table)`
: The counterfactuals for `x_interest`.

`x_interest (data.table(1))`
: A single row with the observation of interest.

`distance_function (function())`
: The distance function used in the second and fourth evaluation measure. The function must have three arguments: `x`, `y`, and `data` and return a `numeric` matrix. If set to `NULL` (default), then Gower distance (Gower 1971) is used.

`method (character)`
: A single row with the observation of interest.

## Methods

### Public methods:

- `Counterfactuals$new()`
- `Counterfactuals$evaluate()`
- `Counterfactuals$evaluate_set()`
- `Counterfactuals$predict()`
- `Counterfactuals$subset_to_valid()`
- `Counterfactuals$revert_subset_to_valid()`
- `Counterfactuals$plot_parallel()`
- `Counterfactuals$plot_freq_of_feature_changes()`
- `Counterfactuals$get_freq_of_feature_changes()`
- `Counterfactuals$plot_surface()`
- `Counterfactuals$print()`
- `Counterfactuals$clone()`

**Method** `new()`**:** Creates a new `Counterfactuals` object. This method should only be called by the `$find_counterfactuals` methods of CounterfactualMethodRegr and CounterfactualMethodClassif.

*Usage:*

```
Counterfactuals$new(
  cfactuals,
  predictor,
  x_interest,
  param_set,
  desired,
  method = NULL
)
```

*Arguments:*

`cfactuals (data.table)`
: The counterfactuals. Must have the same column names and types as `predictor$data$X`.

`predictor (Predictor)`
: The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`x_interest (data.table(1) | data.frame(1))`
: A single row with the observation of interest.

`param_set (ParamSet)`
: A ParamSet based on the features of `predictor$data$X`.

`desired (list(1) | list(2))`
: A `list` with the desired properties of the counterfactuals. It should have one element `desired_outcome` for regression tasks (CounterfactualMethodRegr) and two elements `desired_class` and `desired_prob` for classification tasks (CounterfactualMethodClassif).

`method (character)`
: Name of the method with which counterfactuals were generated. Default is NULL which means that no name is provided.

**Method** `evaluate()`**:** Evaluates the counterfactuals. It returns the counterfactuals together with the evaluation `measures`.

*Usage:*

```
Counterfactuals$evaluate(
  measures = c("dist_x_interest", "dist_target", "no_changed", "dist_train",
    "minimality"),
  show_diff = FALSE,
  k = 1L,
  weights = NULL
)
```

*Arguments:*

`measures (character)`
: The name of one or more evaluation measures. The following measures are available:

- `dist_x_interest`: The distance of a counterfactual to `x_interest` measured by Gower's dissimilarity measure (Gower 1971).
- `dist_target`: The absolute distance of the prediction for a counterfactual to the interval `desired_outcome` (regression tasks) or `desired_prob` (classification tasks).
- `no_changed`: The number of feature changes w.r.t. `x_interest`.

- `dist_train`: The (weighted) distance to the `k` nearest training data points measured by Gower's dissimilarity measure (Gower 1971).
- `minimality`: The number of changed features that each could be set to the value of `x_interest` while keeping the desired prediction value.

`show_diff (logical(1))`
Should the counterfactuals be displayed as their differences to `x_interest`? Default is `FALSE`. If set to `TRUE`, positive values for numeric features indicate an increase compared to the feature value in `x_interest`, negative values indicate a decrease. For factors, the feature value is displayed if it differs from `x_interest`; `NA` means "no difference" in both cases.

`k (integerish(1))`
How many nearest training points should be considered for computing the `dist_train` measure? Default is `1L`.

`weights (numeric(k) | NULL)`
How should the `k` nearest training points be weighted when computing the `dist_train` measure? If `NULL` (default) then all `k` points are weighted equally. If a numeric vector of length `k` is given, the i-th element specifies the weight of the i-th closest data point.

**Method** `evaluate_set()`**:** Evaluates a set of counterfactuals. It returns the evaluation `measures`.

*Usage:*

```
Counterfactuals$evaluate_set(
  measures = c("diversity", "no_nondom", "frac_nondom", "hypervolume"),
  nadir = NULL
)
```

*Arguments:*

`measures (character)`
The name of one or more evaluation measures. The following measures are available:

- `diversity`: Diversity of returned counterfactuals in the feature space
- `no_nondom`: Number of counterfactuals that are not dominated by other counterfactuals.
- `frac_nondom`: Fraction of counterfactuals that are not dominated by other counterfactuals
- `hypervolume`: Hypervolume of the induced Pareto front

`nadir (numeric)`
Max objective values to calculate dominated hypervolume. Only considered, if `hypervolume` is one of the `measures`. May be a scalar, in which case it is used for all four objectives, or a vector of length 4. Default is NULL, meaning the nadir point by Dandl et al. (2020) is used: (min distance between prediction of `x_interest` to `desired_prob/_outcome`, 1, number of features, 1).

**Method** `predict()`**:** Returns the predictions for the counterfactuals.

*Usage:*

```
Counterfactuals$predict()
```

**Method** `subset_to_valid()`**:** Subset data to those meeting the desired prediction, Process could be reverted using `revert_subset_to_valid()`.

*Usage:*

```
Counterfactuals$subset_to_valid()
```

**Method** `revert_subset_to_valid()`**:** Subset data to those meeting the desired prediction, Process could be reverted using `revert_subset_to_valid()`.

*Usage:*

```
Counterfactuals$revert_subset_to_valid()
```

**Method** `plot_parallel()`**:** Plots a parallel plot that connects the (scaled) feature values of each counterfactual and highlights `x_interest` in blue.

*Usage:*

```
Counterfactuals$plot_parallel(
  feature_names = NULL,
  row_ids = NULL,
  digits_min_max = 2L
)
```

*Arguments:*

`feature_names (character | NULL)`
: The names of the (numeric) features to display. If `NULL` (default) all features are displayed.

`row_ids (integerish | NULL)`
: The row ids of the counterfactuals to display. If `NULL` (default) all counterfactuals are displayed.

`digits_min_max` Maximum number of digits for the minimum and maximum features values. Default is `2L`.

**Method** `plot_freq_of_feature_changes()`**:** Plots a bar chart with the frequency of feature changes across all counterfactuals.

*Usage:*

```
Counterfactuals$plot_freq_of_feature_changes(subset_zero = FALSE)
```

*Arguments:*

`subset_zero (logical(1))`
: Should unchanged features be excluded from the plot? Default is `FALSE`.

**Method** `get_freq_of_feature_changes()`**:** Returns the frequency of feature changes across all counterfactuals.

*Usage:*

```
Counterfactuals$get_freq_of_feature_changes(subset_zero = FALSE)
```

*Arguments:*

`subset_zero (logical(1))`
: Should unchanged features be excluded? Default is `FALSE`.

*Returns:* A (named) `numeric` vector with the frequency of feature changes.

**Method** `plot_surface()`**:** Creates a surface plot for two features. `x_interest` is represented as a white dot and all counterfactuals that differ from `x_interest` **only** in the two selected features are represented as black dots. The tick marks next to the axes show the marginal distribution of the observed data (`predictor$data$X`).
The exact plot type depends on the selected feature types and number of features:

- 2 numeric features: surface plot
- 2 non-numeric features: heatmap
- 1 numeric or non-numeric feature: line graph

*Usage:*

```
Counterfactuals$plot_surface(feature_names, grid_size = 250L)
```

*Arguments:*

`feature_names (character(2))`
The names of the features to plot.

`grid_size (integerish(1))`
The grid size of the plot. It is ignored in case of two `non-numeric` features. Default is `250L`.

**Method** `print()`**:** Prints the `Counterfactuals` object.

*Usage:*

```
Counterfactuals$print()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

Gower, J. C. (1971), "A general coefficient of similarity and some of its properties". Biometrics, 27, 623–637.

---

`dist_to_interval` *Computes the (absolute, pairwise) distance between the vector elements and an interval*

---

## Description

Computes the (absolute, pairwise) distance between the vector elements and an interval

## Usage

```
dist_to_interval(x, interval)
```

## Arguments

| | |
|---|---|
| `x` | (`numeric()`)<br>A numeric vector. |
| `interval` | (`numeric(2)`)<br>An interval. |

## eval_distance *Evaluates a distance function and checks for correct output format*

### Description

This function serves as an evaluation wrapper for some distance function. It checks that the output of `distance_function` is a `numeric` matrix with `nrow(x)` rows and `nrow(y)` columns as expected.

### Usage

```
eval_distance(distance_function, x, y, data = NULL)
```

### Arguments

| | |
|---|---|
| `distance_function` | (`function()`)<br>A distance function to evaluate. |
| `x` | (`data.frame()` \| `numeric()`)<br>A matrix or a data frame containing variables that should be used in the computation of the distance. |
| `y` | (`data.frame()` \| `numeric()`)<br>A matrix or a data.frame containing variables that should be used in the computation of the distance. |
| `data` | (`data.frame()` \| `NULL`)<br>A data.frame or data.table containing the entire data set. This can be used to compute statistics used in the computation of the distance, e.g., standard deviation or range. |

## make_param_set *Creates a ParamSet for the features of a data.table.*

### Description

Creates a ParamSet for the columns of `dt`. Depending on the class of a column, a different Domain is created:

- `double`: `p_dbl()`
- `integer`: `p_int()`
- `character`: `p_fct()` (with unique values as levels)
- `factor`: `p_fct()` (with factor levels as levels)

### Usage

```
make_param_set(dt, lower = NULL, upper = NULL)
```

### Arguments

| | |
|---|---|
| dt | (data.table())<br>The data for the ParamSet. |
| lower | (numeric() \| NULL)<br>Vector of minimum values for numeric features. If not NULL, it should be named with the corresponding feature names. If NULL (default) lower is taken for each numeric feature as its minimum value in dt. |
| upper | (numeric() \| NULL)<br>Vector of maximum values for numeric features. If not NULL, it should be named with the corresponding feature names. If NULL (default) upper is taken for each numeric feature as its maximum value in dt. |

### Value

A ParamSet for the features of dt.

---

## MOCClassif *MOC (Multi-Objective Counterfactual Explanations) for Classification Tasks*

---

### Description

MOC (Dandl et. al 2020) solves a multi-objective optimization problem to find counterfactuals. The four objectives to minimize are:

1. dist_target: Distance to desired_prob (classification tasks) or desired_prob (regression tasks).
2. dist_x_interest: Dissimilarity to x_interest measured by Gower's dissimilarity measure (Gower 1971).
3. no_changed: Number of feature changes.
4. dist_train: (Weighted) sum of dissimilarities to the k nearest data points in predictor$data$X.

For optimization, it uses the NSGA II algorithm (Deb et. al 2002) with mixed integer evolutionary strategies (Li et al. 2013) and some tailored adjustments for the counterfactual search (Dandl et al. 2020). Default values for the hyperparameters are based on Dandl et al. 2020.

### Details

Several population initialization strategies are available:

1. random: Feature values of new individuals are sampled from the feature value ranges in predictor$data$X. Some features values are randomly reset to their initial value in x_interest.
2. sd: Like random, except that the sample ranges of numerical features are limited to one standard deviation from their initial value in x_interest.

3. `icecurve`: As in `random`, feature values are sampled from the feature value ranges in `predictor$data$X`. Then, however, features are reset with probabilities relative to their importance: the higher the importance of a feature, the higher the probability that its values differ from its value in `x_interest`. The feature importance is measured using ICE curves (Goldstein et al. 2015).
4. `traindata`: Contrary to the other strategies, feature values are drawn from (non-dominated) data points in `predictor$data$X`; if not enough non-dominated data points are available, remaining individuals are initialized by random sampling. Subsequently, some features values are randomly reset to their initial value in `x_interest` (as for `random`).

If `use_conditional_mutator` is set to TRUE, a conditional mutator samples feature values from the conditional distribution given the other feature values with the help of transformation trees (Hothorn and Zeileis 2017). For details see Dandl et al. 2020.

## Super classes

`counterfactuals::CounterfactualMethod` -> `counterfactuals::CounterfactualMethodClassif` -> `MOCClassif`

## Active bindings

`optimizer` (OptimInstanceBatchMultiCrit)
The object used for optimization.

## Methods

### Public methods:

- `MOCClassif$new()`
- `MOCClassif$plot_statistics()`
- `MOCClassif$get_dominated_hv()`
- `MOCClassif$plot_search()`
- `MOCClassif$clone()`

**Method** `new()`**:** Create a new `MOCClassif` object.

*Usage:*

```
MOCClassif$new(
  predictor,
  epsilon = NULL,
  fixed_features = NULL,
  max_changed = NULL,
  mu = 20L,
  termination_crit = "gens",
  n_generations = 175L,
  p_rec = 0.71,
  p_rec_gen = 0.62,
  p_mut = 0.73,
  p_mut_gen = 0.5,
  p_mut_use_orig = 0.4,
  k = 1L,
```

```
  weights = NULL,
  lower = NULL,
  upper = NULL,
  init_strategy = "icecurve",
  use_conditional_mutator = FALSE,
  quiet = FALSE,
  distance_function = "gower"
)
```

*Arguments:*

`predictor` (Predictor)
: The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`epsilon` (`numeric(1)` | `NULL`)
: If not `NULL`, candidates whose prediction for the `desired_class` is farther away from the interval `desired_prob` than `epsilon` are penalized. `NULL` (default) means no penalization.

`fixed_features` (`character()` | `NULL`)
: Names of features that are not allowed to be changed. `NULL` (default) allows all features to be changed.

`max_changed` (`integerish(1)` | `NULL`)
: Maximum number of feature changes. `NULL` (default) allows any number of changes.

`mu` (`integerish(1)`)
: The population size. Default is `20L`.

`termination_crit` (`character(1)`|`NULL`)
: Termination criterion, currently, two criterions are implemented: "gens" (default), which stops after `n_generations` generations, and "genstag", which stops after the hypervolume did not improve for `n_generations` generations (the total number of generations is limited to 500).

`n_generations` (`integerish(1)`)
: The number of generations. Default is `175L`.

`p_rec` (`numeric(1)`)
: Probability with which an individual is selected for recombination. Default is `0.71`.

`p_rec_gen` (`numeric(1)`)
: Probability with which a feature/gene is selected for recombination. Default is `0.62`.

`p_mut` (`numeric(1)`)
: Probability with which an individual is selected for mutation. Default is `0.73`.

`p_mut_gen` (`numeric(1)`)
: Probability with which a feature/gene is selected for mutation. Default is `0.5`.

`p_mut_use_orig` (`numeric(1)`)
: Probability with which a feature/gene is reset to its original value in `x_interest` after mutation. Default is `0.4`.

`k` (`integerish(1)`)
: The number of data points to use for the forth objective. Default is `1L`.

`weights` (`numeric(1)` | `numeric(k)` | `NULL`)
: The weights used to compute the weighted sum of dissimilarities for the forth objective. It is either a single value or a vector of length `k`. If it has length `k`, the i-th element specifies the weight of the i-th closest data point. The values should sum up to `1`. `NULL` (default) means all data points are weighted equally.

`lower (numeric() | NULL)`
: Vector of minimum values for numeric features. If `NULL` (default), the element for each numeric feature in `lower` is taken as its minimum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`upper (numeric() | NULL)`
: Vector of maximum values for numeric features. If `NULL` (default), the element for each numeric feature in `upper` is taken as its maximum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`init_strategy (character(1))`
: The population initialization strategy. Can be `icecurve` (default), `random`, `sd` or `traindata`. For more information, see the `Details` section.

`use_conditional_mutator (logical(1))`
: Should a conditional mutator be used? The conditional mutator generates plausible feature values based on the values of the other feature. Default is `FALSE`.

`quiet (logical(1))`
: Should information about the optimization status be hidden? Default is `FALSE`.

`distance_function (function() | 'gower' | 'gower_c')`
: The distance function to be used in the second and fourth objective. Either the name of an already implemented distance function ('gower' or 'gower_c') or a function. If set to 'gower' (default), then Gower's distance (Gower 1971) is used; if set to 'gower_c', a C-based more efficient version of Gower's distance is used. A function must have three arguments `x`, `y`, and `data` and should return a `double` matrix with `nrow(x)` rows and maximum `nrow(y)` columns.

**Method** `plot_statistics()`**:** Plots the evolution of the mean and minimum objective values together with the dominated hypervolume over the generations. All values for a generation are computed based on all non-dominated individuals that emerged until that generation.

*Usage:*

```
MOCClassif$plot_statistics(centered_obj = TRUE)
```

*Arguments:*

`centered_obj (logical(1))`
: Should the objective values be centered? If set to `FALSE`, each objective value is visualized in a separate plot, since they (usually) have different scales. If set to `TRUE` (default), they are visualized in a single plot.

**Method** `get_dominated_hv()`**:** Calculates the dominated hypervolume of each generation.

*Usage:*

```
MOCClassif$get_dominated_hv()
```

*Returns:* A `data.table` with the dominated hypervolume of each generation.

**Method** `plot_search()`**:** Visualizes two selected objective values of all emerged individuals in a scatter plot.

*Usage:*

```
MOCClassif$plot_search(objectives = c("dist_target", "dist_x_interest"))
```

*Arguments:*

`objectives (character(2))`
: The two objectives to be shown in the plot. Possible values are "dist_target", "dist_x_interest, "no_changed", and "dist_train".

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References


Dandl, S., Molnar, C., Binder, M., and Bischl, B. (2020). "Multi-Objective Counterfactual Explanations". In: Parallel Problem Solving from Nature – PPSN XVI, edited by Thomas Bäck, Mike Preuss, André Deutz, Hao Wang, Carola Doerr, Michael Emmerich, and Heike Trautmann, 448–469, Cham, Springer International Publishing, doi:10.1007/9783030581121_31.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". IEEE transactions on evolutionary computation, 6(2), 182-197.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015). "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation". Journal of Computational and Graphical Statistics 24 (1): 44–65. doi:10.1080/10618600.2014.907095.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. Biometrics, 27, 623–637.

Hothorn, T., Zeileis, A. (2017), "Transformation Forests". Technical Report, arXiv 1701.02110.

Li, Rui, L., Emmerich, M. T. M., Eggermont, J. Bäck, T., Schütz, M., Dijkstra, J., Reiber, J. H. C. (2013). "Mixed Integer Evolution Strategies for Parameter Optimization." Evolutionary Computation 21 (1): 29–64. doi:10.1162/EVCO_a_00059.


## Examples

```
if (require("randomForest")) {

  # Train a model
  rf = randomForest(Species ~ ., data = iris)
  # Create a predictor object
  predictor = iml::Predictor$new(rf, type = "prob")
  # Find counterfactuals for x_interest
  moc_classif = MOCClassif$new(predictor, n_generations = 15L, quiet = TRUE)

  cfactuals = moc_classif$find_counterfactuals(
    x_interest = iris[150L, ], desired_class = "versicolor", desired_prob = c(0.5, 1)
  )
  # Print the counterfactuals
  cfactuals$data
  # Plot evolution of hypervolume and mean and minimum objective values
  moc_classif$plot_statistics()

}
```

MOCRegr *MOC (Multi-Objective Counterfactual Explanations) for Regression Tasks*

## Description

MOC (Dandl et. al 2020) solves a multi-objective optimization problem to find counterfactuals. The four objectives to minimize are:

1. `dist_target`: Distance to `desired_prob` (classification tasks) or `desired_prob` (regression tasks).
2. `dist_x_interest`: Dissimilarity to `x_interest` measured by Gower's dissimilarity measure (Gower 1971).
3. `no_changed`: Number of feature changes.
4. `dist_train`: (Weighted) sum of dissimilarities to the `k` nearest data points in `predictor$data$X`.

For optimization, it uses the NSGA II algorithm (Deb et. al 2002) with mixed integer evolutionary strategies (Li et al. 2013) and some tailored adjustments for the counterfactual search (Dandl et al. 2020). Default values for the hyperparameters are based on Dandl et al. 2020.

## Details

Several population initialization strategies are available:

1. `random`: Feature values of new individuals are sampled from the feature value ranges in `predictor$data$X`. Some features values are randomly reset to their initial value in `x_interest`.
2. `sd`: Like `random`, except that the sample ranges of numerical features are limited to one standard deviation from their initial value in `x_interest`.
3. `icecurve`: As in `random`, feature values are sampled from the feature value ranges in `predictor$data$X`. Then, however, features are reset with probabilities relative to their importance: the higher the importance of a feature, the higher the probability that its values differ from its value in `x_interest`. The feature importance is measured using ICE curves (Goldstein et al. 2015).
4. `traindata`: Contrary to the other strategies, feature values are drawn from (non-dominated) data points in `predictor$data$X`; if not enough non-dominated data points are available, remaining individuals are initialized by random sampling. Subsequently, some features values are randomly reset to their initial value in `x_interest` (as for `random`).

If `use_conditional_mutator` is set to TRUE, a conditional mutator samples feature values from the conditional distribution given the other feature values with the help of transformation trees (Hothorn and Zeileis 2017). For details see Dandl et al. 2020.

## Super classes

`counterfactuals::CounterfactualMethod` -> `counterfactuals::CounterfactualMethodRegr` -> `MOCRegr`

## Active bindings

`optimizer` (OptimInstanceBatchMultiCrit)
The object used for optimization.

## Methods

### Public methods:

- `MOCRegr$new()`
- `MOCRegr$plot_statistics()`
- `MOCRegr$get_dominated_hv()`
- `MOCRegr$plot_search()`
- `MOCRegr$clone()`

**Method** `new()`**:** Create a new `MOCRegr` object.

*Usage:*

```
MOCRegr$new(
  predictor,
  epsilon = NULL,
  fixed_features = NULL,
  max_changed = NULL,
  mu = 20L,
  termination_crit = "gens",
  n_generations = 175L,
  p_rec = 0.71,
  p_rec_gen = 0.62,
  p_mut = 0.73,
  p_mut_gen = 0.5,
  p_mut_use_orig = 0.4,
  k = 1L,
  weights = NULL,
  lower = NULL,
  upper = NULL,
  init_strategy = "icecurve",
  use_conditional_mutator = FALSE,
  quiet = FALSE,
  distance_function = "gower"
)
```

*Arguments:*

`predictor` (Predictor)
The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`epsilon` (`numeric(1)` | `NULL`)
If not `NULL`, candidates whose prediction is farther away from the interval `desired_outcome` than `epsilon` are penalized. `NULL` (default) means no penalization.

`fixed_features` (`character()` | `NULL`)
Names of features that are not allowed to be changed. `NULL` (default) allows all features to be changed.

`max_changed` (`integerish(1)` | `NULL`)
: Maximum number of feature changes. `NULL` (default) allows any number of changes.

`mu` (`integerish(1)`)
: The population size. Default is `20L`.

`termination_crit` (`character(1)`|`NULL`)
: Termination criterion, currently, two criterions are implemented: "gens" (default), which stops after `n_generations` generations, and "genstag", which stops after the hypervolume did not improve for `n_generations` generations (the total number of generations is limited to 500).

`n_generations` (`integerish(1)`)
: The number of generations. Default is `175L`.

`p_rec` (`numeric(1)`)
: Probability with which an individual is selected for recombination. Default is `0.71`.

`p_rec_gen` (`numeric(1)`)
: Probability with which a feature/gene is selected for recombination. Default is `0.62`.

`p_mut` (`numeric(1)`)
: Probability with which an individual is selected for mutation. Default is `0.73`.

`p_mut_gen` (`numeric(1)`)
: Probability with which a feature/gene is selected for mutation. Default is `0.5`.

`p_mut_use_orig` (`numeric(1)`)
: Probability with which a feature/gene is reset to its original value in `x_interest` after mutation. Default is `0.4`.

`k` (`integerish(1)`)
: The number of data points to use for the forth objective. Default is `1L`.

`weights` (`numeric(1)` | `numeric(k)` | `NULL`)
: The weights used to compute the weighted sum of dissimilarities for the forth objective. It is either a single value or a vector of length `k`. If it has length `k`, the i-th element specifies the weight of the i-th closest data point. The values should sum up to `1`. `NULL` (default) means all data points are weighted equally.

`lower` (`numeric()` | `NULL`)
: Vector of minimum values for numeric features. If `NULL` (default), the element for each numeric feature in `lower` is taken as its minimum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`upper` (`numeric()` | `NULL`)
: Vector of maximum values for numeric features. If `NULL` (default), the element for each numeric feature in `upper` is taken as its maximum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`init_strategy` (`character(1)`)
: The population initialization strategy. Can be `icecurve` (default), `random`, `sd` or `traindata`. For more information, see the `Details` section.

`use_conditional_mutator` (`logical(1)`)
: Should a conditional mutator be used? The conditional mutator generates plausible feature values based on the values of the other feature. Default is `FALSE`.

`quiet` (`logical(1)`)
: Should information about the optimization status be hidden? Default is `FALSE`.

`distance_function (function() | 'gower' | 'gower_c')`
: The distance function to be used in the second and fourth objective. Either the name of an already implemented distance function ('gower' or 'gower_c') or a function. If set to 'gower' (default), then Gower's distance (Gower 1971) is used; if set to 'gower_c', a C-based more efficient version of Gower's distance is used. A function must have three arguments `x`, `y`, and `data` and should return a `double` matrix with `nrow(x)` rows and maximum `nrow(y)` columns.

**Method** `plot_statistics()`**:** Plots the evolution of the mean and minimum objective values together with the dominated hypervolume over the generations. All values for a generation are computed based on all non-dominated individuals that emerged until that generation.

*Usage:*

```
MOCRegr$plot_statistics(centered_obj = TRUE)
```

*Arguments:*

`centered_obj (logical(1))`
: Should the objective values be centered? If set to `FALSE`, each objective value is visualized in a separate plot, since they (usually) have different scales. If set to `TRUE` (default), they are visualized in a single plot.

**Method** `get_dominated_hv()`**:** Calculates the dominated hypervolume of each generation.

*Usage:*

```
MOCRegr$get_dominated_hv()
```

*Returns:* A `data.table` with the dominated hypervolume of each generation.

**Method** `plot_search()`**:** Visualizes two selected objective values of all emerged individuals in a scatter plot.

*Usage:*

```
MOCRegr$plot_search(objectives = c("dist_target", "dist_x_interest"))
```

*Arguments:*

`objectives (character(2))`
: The two objectives to be shown in the plot. Possible values are "dist_target", "dist_x_interest, "no_changed", and "dist_train".

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

Dandl, S., Molnar, C., Binder, M., and Bischl, B. (2020). "Multi-Objective Counterfactual Explanations". In: Parallel Problem Solving from Nature – PPSN XVI, edited by Thomas Bäck, Mike Preuss, André Deutz, Hao Wang, Carola Doerr, Michael Emmerich, and Heike Trautmann, 448–469, Cham, Springer International Publishing, doi:10.1007/9783030581121_31.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". IEEE transactions on evolutionary computation, 6(2), 182-197.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015). "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation". Journal of Computational and Graphical Statistics 24 (1): 44–65. doi:10.1080/10618600.2014.907095.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. Biometrics, 27, 623–637.

Hothorn, T., Zeileis, A. (2017), "Transformation Forests". Technical Report, arXiv 1701.02110.

Li, Rui, L., Emmerich, M. T. M., Eggermont, J. Bäck, T., Schütz, M., Dijkstra, J., Reiber, J. H. C. (2013). "Mixed Integer Evolution Strategies for Parameter Optimization." Evolutionary Computation 21 (1): 29–64. doi:10.1162/EVCO_a_00059.

## Examples

```
if (require("randomForest")) {
  # Train a model
  rf = randomForest(mpg ~ ., data = mtcars)
  # Create a predictor object
  predictor = iml::Predictor$new(rf)
  # Find counterfactuals for x_interest

  moc_regr = MOCRegr$new(predictor, n_generations = 15L, quiet = TRUE)
 cfactuals = moc_regr$find_counterfactuals(x_interest = mtcars[1L, ], desired_outcome = c(22, 26))
  # Print the counterfactuals
  cfactuals$data
  # Plot evolution of hypervolume and mean and minimum objective values
  moc_regr$plot_statistics()

}
```

---

`NICEClassif` *NICE (Nearest Instance Counterfactual Explanations) for Classification Tasks*

---

## Description

NICE (Brughmans and Martens 2021) searches for counterfactuals by iteratively replacing feature values of `x_interest` with the corresponding value of its most similar (optionally correctly classified) instance `x_nn`.

## Details

NICE starts the counterfactual search for `x_interest` by finding its most similar (optionally) correctly classified neighbor `x_nn`.
In the first iteration, NICE creates new instances by replacing a different feature value of `x_interest` with the corresponding value of `x_nn` in each new instance. Thus, if `x_nn` differs from `x_interest`

in d features, d new instances are created.
Then, the reward values for the created instances are computed with the chosen reward function. Available reward functions are `sparsity`, `proximity`, and `plausibility`.
In the second iteration, NICE creates `d-1` new instances by replacing a different feature value of the highest reward instance of the previous iteration with the corresponding value of `x_interest`, and so on.
If `finish_early = TRUE`, the algorithm terminates when the predicted `desired_class` probability for the highest reward instance is in the interval `desired_prob`; if `finish_early = FALSE`, the algorithm continues until `x_nn` is recreated.
Once the algorithm terminated, it depends on `return_multiple` which instances are returned as counterfactuals: if `return_multiple = FALSE`, then only the highest reward instance in the last iteration is returned as counterfactual; if `return_multiple = TRUE`, then all instances (of all iterations) whose predicted `desired_class` probability is in the interval `desired_prob` are returned as counterfactuals.

If `finish_early = FALSE` and `return_multiple = FALSE`, then `x_nn` is returned as single counterfactual.

This NICE implementation corresponds to the original version of Brughmans and Martens (2021) when `return_multiple = FALSE`, `finish_early = TRUE`, and `x_nn_correct = TRUE`.

## Super classes

`counterfactuals::CounterfactualMethod` -> `counterfactuals::CounterfactualMethodClassif` -> `NICEClassif`

## Active bindings

`x_nn (logical(1))`
: The most similar (optionally) correctly classified instance of `x_interest`.

`archive (list())`
: A list that stores the history of the algorithm run. For each algorithm iteration, it has one element containing a `data.table`, which stores all created instances of this iteration together with their reward values and their predictions.

## Methods

### Public methods:

- `NICEClassif$new()`
- `NICEClassif$clone()`

**Method** `new()`**:** Create a new NICEClassif object.

*Usage:*

```
NICEClassif$new(
  predictor,
  optimization = "sparsity",
  x_nn_correct = TRUE,
  return_multiple = FALSE,
  finish_early = TRUE,
```

```
  distance_function = "gower"
)
```

*Arguments:*

`predictor` (Predictor)
: The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`optimization (character(1))`
: The reward function to optimize. Can be `sparsity` (default), `proximity` or `plausibility`.

`x_nn_correct (logical(1))`
: Should only *correctly* classified data points in `predictor$data$X` be considered for the most similar instance search? Default is `TRUE`.

`return_multiple (logical(1))`
: Should multiple counterfactuals be returned? If TRUE, the algorithm returns all created instances whose `desired_class` prediction is in the interval `desired_prob`. For more information, see the `Details` section.

`finish_early (logical(1))`
: Should the algorithm terminate after an iteration in which the `desired_class` prediction for the highest reward instance is in the interval `desired_prob`. If `FALSE`, the algorithm continues until `x_nn` is recreated.

`distance_function (function() | 'gower' | 'gower_c')`
: The distance function used to compute the distances between `x_interest` and the training data points for finding `x_nn`. If `optimization` is set to `proximity`, the distance function is also used for calculating the distance between candidates and `x_interest`. Either the name of an already implemented distance function ('gower' or 'gower_c') or a function is allowed as input. If set to 'gower' (default), then Gower's distance (Gower 1971) is used; if set to 'gower_c', a C-based more efficient version of Gower's distance is used. A function must have three arguments `x`, `y`, and `data` and should return a `double` matrix with `nrow(x)` rows and maximum `nrow(y)` columns.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

Brughmans, D., & Martens, D. (2021). NICE: An Algorithm for Nearest Instance Counterfactual Explanations. arXiv 2104.07411 v2.

Gower, J. C. (1971), "A general coefficient of similarity and some of its properties". Biometrics, 27, 623–637.

## Examples

```
if (require("randomForest")) {
  # Train a model
  rf = randomForest(Species ~ ., data = iris)
```

```
  # Create a predictor object
  predictor = iml::Predictor$new(rf, type = "prob")
  # Find counterfactuals
  nice_classif = NICEClassif$new(predictor)
  cfactuals = nice_classif$find_counterfactuals(
    x_interest = iris[150L, ], desired_class = "versicolor", desired_prob = c(0.5, 1)
  )
  # Print the results
  cfactuals$data
  # Print archive
  nice_classif$archive
}
```

---

NICERegr *NICE (Nearest Instance Counterfactual Explanations) for Regression Tasks*

---

## Description

NICE (Brughmans and Martens 2021) searches for counterfactuals by iteratively replacing feature values of `x_interest` with the corresponding value of its most similar (optionally correctly predicted) instance `x_nn`. While the original method is only applicable to classification tasks (see NICEClassif), this implementation extend it to regression tasks.

## Details

NICE starts the counterfactual search for `x_interest` by finding its most similar (optionally) correctly predicted neighbor `x_nn` with(in) the desired prediction (range). Correctly predicted means that the prediction of `x_nn` is less than a user-specified `margin_correct` away from the true outcome of `x_nn`. This is designed to mimic the search for `x_nn` for regression tasks. If no `x_nn` satisfies this constraint, a warning is returned that no counterfactual could be found.
In the first iteration, NICE creates new instances by replacing a different feature value of `x_interest` with the corresponding value of `x_nn` in each new instance. Thus, if `x_nn` differs from `x_interest` in `d` features, `d` new instances are created.
Then, the reward values for the created instances are computed with the chosen reward function. Available reward functions are `sparsity`, `proximity`, and `plausibility`.
In the second iteration, NICE creates `d-1` new instances by replacing a different feature value of the highest reward instance of the previous iteration with the corresponding value of `x_interest`, and so on.
If `finish_early = TRUE`, the algorithm terminates when the predicted outcome for the highest reward instance is in the interval `desired_outcome`; if `finish_early = FALSE`, the algorithm continues until `x_nn` is recreated.
Once the algorithm terminated, it depends on `return_multiple` which instances are returned as counterfactuals: if `return_multiple = FALSE`, then only the highest reward instance in the last iteration is returned as counterfactual; if `return_multiple = TRUE`, then all instances (of all iterations) whose predicted outcome is in the interval `desired_outcome` are returned as counterfactuals.

If finish_early = FALSE and return_multiple = FALSE, then x_nn is returned as single counterfactual.

The function computes the dissimilarities using Gower's dissimilarity measure (Gower 1971).

## Super classes

counterfactuals::CounterfactualMethod -> counterfactuals::CounterfactualMethodRegr -> NICERegr

## Active bindings

x_nn (logical(1))
: The most similar (optionally) correctly classified instance of x_interest.

archive (list())
: A list that stores the history of the algorithm run. For each algorithm iteration, it has one element containing a data.table, which stores all created instances of this iteration together with their reward values and their predictions.

## Methods

### Public methods:

- NICERegr$new()
- NICERegr$clone()

**Method** new(): Create a new NICERegr object.

*Usage:*

```
NICERegr$new(
  predictor,
  optimization = "sparsity",
  x_nn_correct = TRUE,
  margin_correct = NULL,
  return_multiple = FALSE,
  finish_early = TRUE,
  distance_function = "gower"
)
```

*Arguments:*

predictor (Predictor)
: The object (created with iml::Predictor$new()) holding the machine learning model and the data.

optimization (character(1))
: The reward function to optimize. Can be sparsity (default), proximity or plausibility.

x_nn_correct (logical(1))
: Should only *correctly* classified data points in predictor$data$X be considered for the most similar instance search? Default is TRUE.

margin_correct (numeric(1) | NULL)
: The accepted margin for considering a prediction as "correct". Ignored if x_nn_correct = FALSE. If NULL, the accepted margin is set to half the median absolute distance between the true and predicted outcomes in the data (predictor$data).

`return_multiple (logical(1))`
: Should multiple counterfactuals be returned? If TRUE, the algorithm returns all created instances whose prediction is in the interval `desired_outcome`. For more information, see the `Details` section.

`finish_early (logical(1))`
: Should the algorithm terminate after an iteration in which the prediction for the highest reward instance is in the interval `desired_outcome`. If `FALSE`, the algorithm continues until `x_nn` is recreated.

`distance_function (function() | 'gower' | 'gower_c')`
: The distance function used to compute the distances between `x_interest` and the training data points for finding `x_nn`. If `optimization` is set to `proximity`, the distance function is also used for calculating the distance between candidates and `x_interest`. Either the name of an already implemented distance function ('gower' or 'gower_c') or a function is allowed as input. If set to 'gower' (default), then Gower's distance (Gower 1971) is used; if set to 'gower_c', a C-based more efficient version of Gower's distance is used. A function must have three arguments `x`, `y`, and `data` and should return a `double` matrix with `nrow(x)` rows and maximum `nrow(y)` columns.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

Brughmans, D., & Martens, D. (2021). NICE: An Algorithm for Nearest Instance Counterfactual Explanations. arXiv 2104.07411 v2.

Gower, J. C. (1971), "A general coefficient of similarity and some of its properties". Biometrics, 27, 623–637.

## Examples

```
if (require("randomForest")) {
  set.seed(123456)
  # Train a model
  rf = randomForest(mpg ~ ., data = mtcars)
  # Create a predictor object
  predictor = iml::Predictor$new(rf)
  # Find counterfactuals
  nice_regr = NICERegr$new(predictor)
  cfactuals = nice_regr$find_counterfactuals(
     x_interest = mtcars[1L, ], desired_outcome = c(22, 26)
  )
  # Print the results
  cfactuals$data
  # Print archive
  nice_regr$archive
}
```

RandomSearchClassif *Random Search for Classification Tasks*

## Description

RandomSearch randomly samples a population of candidates and returns non-dominated candidates w.r.t to the objectives of MOC (Dandl et. al 2020) as counterfactuals. RandomSearch is equivalent to MOC with zero generations and the `random` initialization strategy.

The four objectives of MOC (Dandl et. al 2020) to are:

1. Distance to `desired_prob` (classification tasks) or `desired_prob` (regression tasks).
2. Dissimilarity to `x_interest` measured by Gower's dissimilarity measure (Gower 1971).
3. Number of feature changes.
4. (Weighted) sum of dissimilarities to the `k` nearest data points in `predictor$data$X`.

## Details

RandomSearch is typically used as a baseline in benchmark comparisons with MOC. The total number of samples drawn is `mu` * `n_generations`. Using separate parameters `mu` and `n_generations` is only required to make certain statistics comparable with MOC (e.g. the evolution of the dominated hypervolume).

## Super classes

`counterfactuals::CounterfactualMethod` -> `counterfactuals::CounterfactualMethodClassif` -> `RandomSearchClassif`

## Active bindings

`optimizer` (OptimInstanceBatchMultiCrit)
The object used for optimization.

## Methods

### Public methods:

- `RandomSearchClassif$new()`
- `RandomSearchClassif$plot_statistics()`
- `RandomSearchClassif$get_dominated_hv()`
- `RandomSearchClassif$plot_search()`
- `RandomSearchClassif$clone()`

**Method** `new()`**:** Create a new `RandomSearchClassif` object.

*Usage:*

```
RandomSearchClassif$new(
  predictor,
  fixed_features = NULL,
  max_changed = NULL,
  mu = 20L,
  n_generations = 175L,
  p_use_orig = 0.5,
  k = 1L,
  weights = NULL,
  lower = NULL,
  upper = NULL,
  distance_function = "gower"
)
```

*Arguments:*

`predictor` (Predictor)
: The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`fixed_features` (`character()` | `NULL`)
: Names of features that are not allowed to be changed. `NULL` (default) allows all features to be changed.

`max_changed` (`integerish(1)` | `NULL`)
: Maximum number of feature changes. `NULL` (default) allows any number of changes.

`mu` (`integerish(1)`)
: The population size. Default is `20L`. The total number of random samples is set to `mu * n_generations`. See the `Details` for further details.

`n_generations` (`integerish(1)`)
: The number of generations. Default is `175L`. The total number of random samples is set to `mu * n_generations`. See the `Details` section for further details.

`p_use_orig` (`numeric(1)`)
: Probability with which a feature/gene is reset to its original value in `x_interest` after random sampling. Default is `0.5`.

`k` (`integerish(1)`)
: The number of data points to use for the forth objective. Default is `1L`.

`weights` (`numeric(1)` | `numeric(k)` | `NULL`)
: The weights used to compute the weighted sum of dissimilarities for the forth objective. It is either a single value or a vector of length `k`. If it has length `k`, the i-th element specifies the weight of the i-th closest data point. The values should sum up to `1`. `NULL` (default) means all data points are weighted equally.

`lower` (`numeric()` | `NULL`)
: Vector of minimum values for numeric features. If `NULL` (default), the element for each numeric feature in `lower` is taken as its minimum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`upper` (`numeric()` | `NULL`)
: Vector of maximum values for numeric features. If `NULL` (default), the element for each numeric feature in `upper` is taken as its maximum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

distance_function (function() | 'gower' | 'gower_c')
: The distance function to be used in the second and fourth objective. Either the name of an already implemented distance function ('gower' or 'gower_c') or a function. If set to 'gower' (default), then Gower's distance (Gower 1971) is used; if set to 'gower_c', a C-based more efficient version of Gower's distance is used. A function must have three arguments x, y, and data and should return a double matrix with nrow(x) rows and maximum nrow(y) columns.

**Method** plot_statistics(): Plots the evolution of the mean and minimum objective values together with the dominated hypervolume over the generations. All values for a generation are computed based on all non-dominated individuals that emerged until that generation. The randomly drawn samples are therefore split into n_generations folds of size mu. This function mimics MOCs plot_statistics() method. See the Details section for further information.

*Usage:*

```
RandomSearchClassif$plot_statistics(centered_obj = TRUE)
```

*Arguments:*

centered_obj (logical(1))
: Should the objective values be centered? If set to FALSE, each objective value is visualized in a separate plot, since they (usually) have different scales. If set to TRUE (default), they are visualized in a single plot.

**Method** get_dominated_hv(): Calculates the dominated hypervolume of each generation. The randomly drawn samples are therefore split into n_generations folds of size mu. This function mimics MOCs get_dominated_hv() method. See the Details section for further information.

*Usage:*

```
RandomSearchClassif$get_dominated_hv()
```

*Returns:* A data.table with the dominated hypervolume of each generation.

**Method** plot_search(): Visualizes two selected objective values of all emerged individuals in a scatter plot. The randomly drawn samples are therefore split into n_generations folds of size mu. This function mimics MOCs plot_search() method. See the Details section for further information.

*Usage:*

```
RandomSearchClassif$plot_search(
  objectives = c("dist_target", "dist_x_interest")
)
```

*Arguments:*

objectives (character(2))
: The two objectives to be shown in the plot. Possible values are "dist_target", "dist_x_interest, "no_changed", and "dist_train".

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

## References

Dandl, S., Molnar, C., Binder, M., and Bischl, B. (2020). "Multi-Objective Counterfactual Explanations". In: Parallel Problem Solving from Nature – PPSN XVI, edited by Thomas Bäck, Mike Preuss, André Deutz, Hao Wang, Carola Doerr, Michael Emmerich, and Heike Trautmann, 448–469, Cham, Springer International Publishing, doi:10.1007/9783030581121_31.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". IEEE transactions on evolutionary computation, 6(2), 182-197.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015). "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation". Journal of Computational and Graphical Statistics 24 (1): 44–65. doi:10.1080/10618600.2014.907095.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. Biometrics, 27, 623–637.

Li, Rui, L., Emmerich, M. T. M., Eggermont, J. Bäck, T., Schütz, M., Dijkstra, J., Reiber, J. H. C. (2013). "Mixed Integer Evolution Strategies for Parameter Optimization." Evolutionary Computation 21 (1): 29–64. doi:10.1162/EVCO_a_00059.

## Examples

```
if (require("randomForest")) {
  # Train a model
  rf = randomForest(Species ~ ., data = iris)
  # Create a predictor object
  predictor = iml::Predictor$new(rf, type = "prob")
  # Find counterfactuals for x_interest
  rs_classif = RandomSearchClassif$new(predictor, n_generations = 30L)
  cfactuals = rs_classif$find_counterfactuals(
    x_interest = iris[150L, ], desired_class = "versicolor", desired_prob = c(0.5, 1)
  )
  # Print the counterfactuals
  cfactuals$data
  # Plot evolution of hypervolume and mean and minimum objective values
  rs_classif$plot_statistics()
}
```

---

`RandomSearchRegr` *Random Search for Regression Tasks*

---

## Description

RandomSearch randomly samples a population of candidates and returns non-dominated candidates w.r.t to the objectives of MOC (Dandl et. al 2020) as counterfactuals. RandomSearch is equivalent to MOC with zero generations and the `random` initialization strategy.

The four objectives of MOC (Dandl et. al 2020) to are:

1. Distance to `desired_prob` (classification tasks) or `desired_prob` (regression tasks).

2. Dissimilarity to `x_interest` measured by Gower's dissimilarity measure (Gower 1971).
3. Number of feature changes.
4. (Weighted) sum of dissimilarities to the `k` nearest data points in `predictor$data$X`.

## Details

RandomSearch is typically used as a baseline in benchmark comparisons with MOC. The total number of samples drawn is `mu` * `n_generations`. Using separate parameters `mu` and `n_generations` is only required to make certain statistics comparable with MOC (e.g. the evolution of the dominated hypervolume).

## Super classes

`counterfactuals::CounterfactualMethod` -> `counterfactuals::CounterfactualMethodRegr` -> `RandomSearchRegr`

## Active bindings

`optimizer` (OptimInstanceBatchMultiCrit)
The object used for optimization.

## Methods

### Public methods:

- `RandomSearchRegr$new()`
- `RandomSearchRegr$plot_statistics()`
- `RandomSearchRegr$get_dominated_hv()`
- `RandomSearchRegr$plot_search()`
- `RandomSearchRegr$clone()`

**Method** `new()`**:** Create a new `RandomSearchRegr` object.

*Usage:*

```
RandomSearchRegr$new(
  predictor,
  fixed_features = NULL,
  max_changed = NULL,
  mu = 20L,
  n_generations = 175L,
  p_use_orig = 0.5,
  k = 1L,
  weights = NULL,
  lower = NULL,
  upper = NULL,
  distance_function = "gower"
)
```

*Arguments:*

`predictor` (Predictor)
: The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`fixed_features` (`character()` | `NULL`)
: Names of features that are not allowed to be changed. `NULL` (default) allows all features to be changed.

`max_changed` (`integerish(1)` | `NULL`)
: Maximum number of feature changes. `NULL` (default) allows any number of changes.

`mu` (`integerish(1)`)
: The population size. Default is `20L`. The total number of random samples is set to `mu * n_generations`. See the `Details` section for further details.

`n_generations` (`integerish(1)`)
: The number of generations. Default is `175L`. The total number of random samples is set to `mu * n_generations`. See the `Details` section for further details.

`p_use_orig` (`numeric(1)`)
: Probability with which a feature/gene is reset to its original value in `x_interest` after random sampling. Default is `0.5`.

`k` (`integerish(1)`)
: The number of data points to use for the forth objective. Default is `1L`.

`weights` (`numeric(1)` | `numeric(k)` | `NULL`)
: The weights used to compute the weighted sum of dissimilarities for the forth objective. It is either a single value or a vector of length `k`. If it has length `k`, the i-th element specifies the weight of the i-th closest data point. The values should sum up to `1`. `NULL` (default) means all data points are weighted equally.

`lower` (`numeric()` | `NULL`)
: Vector of minimum values for numeric features. If `NULL` (default), the element for each numeric feature in `lower` is taken as its minimum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`upper` (`numeric()` | `NULL`)
: Vector of maximum values for numeric features. If `NULL` (default), the element for each numeric feature in `upper` is taken as its maximum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`distance_function` (`function()` | `'gower'` | `'gower_c'`)
: The distance function to be used in the second and fourth objective. Either the name of an already implemented distance function ('gower' or 'gower_c') or a function. If set to 'gower' (default), then Gower's distance (Gower 1971) is used; if set to 'gower_c', a C-based more efficient version of Gower's distance is used. A function must have three arguments `x`, `y`, and `data` and should return a `double` matrix with `nrow(x)` rows and maximum `nrow(y)` columns.

**Method** `plot_statistics()`**:** Plots the evolution of the mean and minimum objective values together with the dominated hypervolume over the generations. All values for a generation are computed based on all non-dominated individuals that emerged until that generation. The randomly drawn samples are therefore split into `n_generations` folds of size `mu`. This function mimics MOCs `plot_statistics()` method. See the `Details` section for further information.

*Usage:*

```
RandomSearchRegr$plot_statistics(centered_obj = TRUE)
```

*Arguments:*

`centered_obj (logical(1))`
: Should the objective values be centered? If set to `FALSE`, each objective value is visualized in a separate plot, since they (usually) have different scales. If set to `TRUE` (default), they are visualized in a single plot.

**Method** `get_dominated_hv()`**:** Calculates the dominated hypervolume of each generation. The randomly drawn samples are therefore split into `n_generations` folds of size `mu`. This function mimics MOCs `get_dominated_hv()` method. See the `Details` section for further information.

*Usage:*

```
RandomSearchRegr$get_dominated_hv()
```

*Returns:* A `data.table` with the dominated hypervolume of each generation.

**Method** `plot_search()`**:** Visualizes two selected objective values of all emerged individuals in a scatter plot. The randomly drawn samples are therefore split into `n_generations` folds of size `mu`. This function mimics MOCs `plot_search()` method. See the `Details` section for further information.

*Usage:*

```
RandomSearchRegr$plot_search(objectives = c("dist_target", "dist_x_interest"))
```

*Arguments:*

`objectives (character(2))`
: The two objectives to be shown in the plot. Possible values are "dist_target", "dist_x_interest, "no_changed", and "dist_train".

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

Dandl, S., Molnar, C., Binder, M., and Bischl, B. (2020). "Multi-Objective Counterfactual Explanations". In: Parallel Problem Solving from Nature – PPSN XVI, edited by Thomas Bäck, Mike Preuss, André Deutz, Hao Wang, Carola Doerr, Michael Emmerich, and Heike Trautmann, 448–469, Cham, Springer International Publishing, doi:10.1007/9783030581121_31.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". IEEE transactions on evolutionary computation, 6(2), 182-197.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015). "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation". Journal of Computational and Graphical Statistics 24 (1): 44–65. doi:10.1080/10618600.2014.907095.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. Biometrics, 27, 623–637.

Li, Rui, L., Emmerich, M. T. M., Eggermont, J. Bäck, T., Schütz, M., Dijkstra, J., Reiber, J. H. C. (2013). "Mixed Integer Evolution Strategies for Parameter Optimization." Evolutionary Computation 21 (1): 29–64. doi:10.1162/EVCO_a_00059.

## Examples

```
if (require("randomForest")) {
  # Train a model
  rf = randomForest(mpg ~ ., data = mtcars)
  # Create a predictor object
  predictor = iml::Predictor$new(rf)
  # Find counterfactuals for x_interest
  rs_regr = RandomSearchRegr$new(predictor, n_generations = 30L)
 cfactuals = rs_regr$find_counterfactuals(x_interest = mtcars[1L, ], desired_outcome = c(22, 26))
  # Print the counterfactuals
  cfactuals$data
  # Plot evolution of hypervolume and mean and minimum objective values
  rs_regr$plot_statistics()
}
```

---

# smallest_n_indices *Returns the indices of the n smallest elements in a vector*

---

## Description

Returns the indices of the n smallest elements in a vector

## Usage

```
smallest_n_indices(x, n = 1L)
```

## Arguments

| | |
|---|---|
| x | (numeric())<br>A numeric vector. |
| n | (numeric(1))<br>A integer indicating how many elements should be returned |

---

# WhatIfClassif *WhatIf for Classification Tasks*

---

## Description

WhatIf returns the n_counterfactual most similar observations to x_interest from observations in predictor$data$X whose prediction for the desired_class is in the desired_prob interval.

## Details

By default, the dissimilarities are computed using Gower's dissimilarity measure (Gower 1971). Only observations whose features values lie between the corresponding values in lower and upper are considered counterfactual candidates.

## Super classes

counterfactuals::CounterfactualMethod -> counterfactuals::CounterfactualMethodClassif -> WhatIfClassif

## Methods

### Public methods:

- WhatIfClassif$new()
- WhatIfClassif$clone()

**Method** `new()`**:** Create a new WhatIfClassif object.

*Usage:*

```
WhatIfClassif$new(
  predictor,
  n_counterfactuals = 1L,
  lower = NULL,
  upper = NULL,
  distance_function = "gower"
)
```

*Arguments:*

`predictor` (Predictor)
: The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`n_counterfactuals` (`integerish(1)`)
: The number of counterfactuals to return. Default is `1L`.

`lower` (`numeric()` | `NULL`)
: Vector of minimum values for numeric features. If `NULL` (default), the element for each numeric feature in `lower` is taken as its minimum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`upper` (`numeric()` | `NULL`)
: Vector of maximum values for numeric features. If `NULL` (default), the element for each numeric feature in `upper` is taken as its maximum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`distance_function` (`function()` | `'gower'` | `'gower_c'`)
: The distance function used to compute the distances between `x_interest` and the training data points for finding `x_nn`. Either the name of an already implemented distance function ('gower' or 'gower_c') or a function. If set to 'gower' (default), then Gower's distance (Gower 1971) is used; if set to 'gower_c', a C-based more efficient version of Gower's distance is used. A function must have three arguments `x`, `y`, and `data` and should return a `double` matrix with `nrow(x)` rows and maximum `nrow(y)` columns.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

Gower, J. C. (1971), "A general coefficient of similarity and some of its properties". Biometrics, 27, 623–637.

Wexler, J., Pushkarna, M., Bolukbasi, T., Wattenberg, M., Viégas, F., & Wilson, J. (2019). The what-if tool: Interactive probing of machine learning models. IEEE transactions on visualization and computer graphics, 26(1), 56–65.

## Examples

```
if (require("randomForest")) {
  # Train a model
  rf = randomForest(Species ~ ., data = iris)
  # Create a predictor object
  predictor = iml::Predictor$new(rf, type = "prob")
  # Find counterfactuals for x_interest
  wi_classif = WhatIfClassif$new(predictor, n_counterfactuals = 5L)
  cfactuals = wi_classif$find_counterfactuals(
    x_interest = iris[150L, ], desired_class = "versicolor", desired_prob = c(0.5, 1)
  )
  # Print the results
  cfactuals$data
}
```

---

`WhatIfRegr` *WhatIf for Regression Tasks*

---

## Description

WhatIf returns the `n_counterfactual` most similar observations to `x_interest` from observations in `predictor$data$X` whose prediction is in the `desired_outcome` interval.

## Details

Only observations whose features values lie between the corresponding values in `lower` and `upper` are considered counterfactual candidates.

## Super classes

`counterfactuals::CounterfactualMethod` -> `counterfactuals::CounterfactualMethodRegr` -> `WhatIfRegr`

## Methods

### Public methods:

- `WhatIfRegr$new()`
- `WhatIfRegr$clone()`

**Method** `new()`**:** Create a new WhatIfRegr object.

*Usage:*

```
WhatIfRegr$new(
  predictor,
  n_counterfactuals = 1L,
  lower = NULL,
  upper = NULL,
  distance_function = "gower"
)
```

*Arguments:*

`predictor` (Predictor)
: The object (created with `iml::Predictor$new()`) holding the machine learning model and the data.

`n_counterfactuals` (`integerish(1)`)
: The number of counterfactuals to return Default is `1L`.

`lower` (`numeric()` | `NULL`)
: Vector of minimum values for numeric features. If `NULL` (default), the element for each numeric feature in `lower` is taken as its minimum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`upper` (`numeric()` | `NULL`)
: Vector of maximum values for numeric features. If `NULL` (default), the element for each numeric feature in `upper` is taken as its maximum value in `predictor$data$X`. If not `NULL`, it should be named with the corresponding feature names.

`distance_function` (`function()` | `'gower'` | `'gower_c'`)
: The distance function used to compute the distances between `x_interest` and the training data points for finding `x_nn`. Either the name of an already implemented distance function ('gower' or 'gower_c') or a function. If set to 'gower' (default), then Gower's distance (Gower 1971) is used; if set to 'gower_c', a C-based more efficient version of Gower's distance is used. A function must have three arguments `x`, `y`, and `data` and should return a `double` matrix with `nrow(x)` rows and maximum `nrow(y)` columns.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

Gower, J. C. (1971), "A general coefficient of similarity and some of its properties". Biometrics, 27, 623–637.

Wexler, J., Pushkarna, M., Bolukbasi, T., Wattenberg, M., Viégas, F., & Wilson, J. (2019). The what-if tool: Interactive probing of machine learning models. IEEE transactions on visualization and computer graphics, 26(1), 56–65.

## Examples

```
if (require("randomForest")) {
  set.seed(123456)
  # Train a model
  rf = randomForest(mpg ~ ., data = mtcars)
  # Create a predictor object
  predictor = iml::Predictor$new(rf)
  # Find counterfactuals for x_interest
  wi_regr = WhatIfRegr$new(predictor, n_counterfactuals = 5L)
  cfactuals = wi_regr$find_counterfactuals(
    x_interest = mtcars[1L, ], desired_outcome = c(22, 26)
  )
  # Print the results
  cfactuals
}
```

# Index