Journal of Statistical Software

December 2025, Volume 115, Issue 9. *doi: 10.18637/jss.v115.i09*

counterfactuals: An R Package for Counterfactual Explanation Methods

Susanne Dandl
LMU Munich
MCML

Andreas Hofheinz
LMU Munich

Martin Binder
LMU Munich
MCML

Bernd Bischl
LMU Munich
MCML

Giuseppe Casalicchio
LMU Munich
MCML

Abstract

Counterfactual explanation methods provide information on how feature values of individual observations must be changed to obtain a desired prediction. Despite the increasing amount of proposed methods in research, only a few implementations exist, whose interfaces and requirements vary widely. In this work, we introduce the **counterfactuals** R package, which provides a modular and unified **R6**-based interface for counterfactual explanation methods. We implemented three existing counterfactual explanation methods and propose some optional methodological extensions to generalize these methods to different scenarios and to make them more comparable. We explain the structure and workflow of the package using real use cases and show how to integrate additional counterfactual explanation methods into the package. In addition, we compared the implemented methods for a variety of models and datasets with regard to the quality of their counterfactual explanations and their runtime behavior.

Keywords: counterfactual explanations, interpretable machine learning, R.

1. Introduction and related work

In recent years, counterfactual explanation methods have emerged as valuable techniques for explaining single predictions of black-box models. Denied loan applications serve as a common example; here, a counterfactual explanation (or *counterfactual* for short) could be: "You were denied a loan because your annual income was £30,000. If your income had been £45,000, you would have been offered a loan" (Wachter, Mittelstadt, and Russell 2018). More generally,

counterfactuals address questions of the form: "For input $\mathbf{x}^\star$, the model predicted y. What needs to be changed in $\mathbf{x}^\star$ so that the model predicts a desired outcome y' instead?" One advantage of counterfactuals is their human-friendly interpretability: As they simply suggest feature changes to obtain a desired outcome, they are comprehensible even to non-experts (Molnar 2022).

Most methods for generating counterfactual explanations do not consider causal relationships (Verma, Boonsanong, Hoang, Hines, Dickerson, and Shah 2022). Counterfactual explanations, therefore, differ from the counterfactuals in the causality literature (Pearl 2009; Lewis 1973). As Baron (2023) showed, they can nevertheless be useful, e.g., to understand which features locally affect a prediction (e.g., income) and why a model made a certain decision (e.g., the income was too low). They can also help to identify biased or inaccurate decisions when the explanations are compared with the user's domain/causal knowledge (Wachter *et al.* 2018). For example, changes in gender or a lower income should not lead to a loan offer, but if these changes are suggested by the counterfactual, this may indicate a biased model. Counterfactual explanations that do not consider causal relations are, in general, not suitable for recourse, i.e., recommendations to achieve a desired output in the future. For recourse, we are interested in the effects of real-world actions, and thus, the causal relationships must be taken into account (Karimi, Schölkopf, and Valera 2021; König, Freiesleben, and Grosse-Wentrup 2023). For example, a counterfactual suggesting changes in income only implies a favorable prediction if all other features are left unchanged; in the real world, changing the income does not happen in isolation but, for example, as an effect of changing the position and due to increased experience which might be reflected in a higher age of the applicant. Absent causal knowledge, counterfactuals may, nevertheless, hint at the directions one should develop, although they might not fully explain how to get there (Ustun, Spangher, and Liu 2019).

There are several ways to change features to obtain a desired outcome, but not all of them are feasible. Therefore, counterfactual methods that provide multiple (reasonable) counterfactuals and allow the user to assess their usefulness using domain knowledge are preferable (Dandl, Molnar, Binder, and Bischl 2020b). Counterfactual explanations are related to adversarial examples (Szegedy *et al.* 2014), but the latter aim to deceive a model instead of explaining it (Freiesleben 2021).

Over the past few years, a variety of counterfactual explanation methods have been proposed. Overviews are given in Verma *et al.* (2022), Karimi *et al.* (2021), and Stepin, Alonso, Catala, and Pereira-Fariña (2021). Most of the methods focus on classification models and use either optimization techniques or heuristic rules to search for counterfactuals. Existing methods are either model-specific in the sense that they are only applicable to certain model classes (e.g., linear or tree-based models) or model-agnostic, i.e., they are applicable to arbitrary models. Furthermore, the methods differ in whether and to what extent access to the underlying data is necessary, the number of counterfactuals they return, and the properties of counterfactuals targeted by a method (e.g., sparsity or actionability). We will present the most frequently targeted properties in Definition 1.

Despite the increasing amount of proposed counterfactual methods in research, the current software landscape is rather sparse. To the best of our knowledge, the only counterfactual methods available in R (R Core Team 2025) as dedicated packages are the multi-objective counterfactual explanation method (MOC, Dandl *et al.* 2020b; Dandl, Molnar, and Binder 2020a) and feature tweaking (Tolomei, Silvestri, Haines, and Lalmas 2017; Kato 2018). Fea-

ture tweaking is a model-specific method tailored to random forests and its R implementation only allows forests specifically trained with the **randomForest** package (Liaw and Wiener 2002). In contrast, MOC is a model-agnostic method and its implementation allows all regression or classification models fitted with popular toolboxes such as **caret** (Kuhn 2008) and **mlr3** (Lang *et al.* 2019). Models of other packages can also be processed using a wrapper function. In Python (Van Rossum and Drake Jr 1995), the **CARLA** library (Pawelczyk, Bielawski, Van den Heuvel, Richter, and Kasneci 2021) provides a variety of (model-agnostic and model-specific) counterfactual explanation methods for classification models. **CARLA** currently calls the original Python implementations of the methods, which often only allow models of specific ML libraries as an input. Furthermore, a library for the model-agnostic nearest instance counterfactual explanations method (**NICE**, Brughmans, Leyman, and Martens 2023; Brughmans 2021) exists which could process all models fitted with **scikit-learn** (Pedregosa *et al.* 2011). Implementations of the methods MACE (Karimi, Barthe, Balle, and Valera 2020), MINT (Karimi *et al.* 2021), and LORE (Guidotti, Monreale, Ruggieri, Pedreschi, Turini, and Giannotti 2018) are available (Karimi and Mohammadi 2021; Guidotti 2018), but these are only meant to reproduce the experiments of the original paper, and are therefore limited to certain datasets and models. Apart from MOC, the mentioned methods are not able to return multiple counterfactuals (in a run).

In summary, existing implementations are predominantly available in Python in different repositories or libraries and at different stages of development. R users can only access a limited number of methods, and the usability and comparability of these methods are severely limited because there is no common user interface. Most Python libraries only allow methods for classification models and focus primarily on methods returning a single counterfactual.

Contributions. With the **counterfactuals** package, we offer the first R package that provides a user-friendly and unified interface for model-specific as well as model-agnostic counterfactual explanation methods. Therefore, it complements other R-based toolkits for interpreting machine learning models such as **iml** (Molnar 2022) and **DALEX** (Biecek 2018). The package provides common functionalities to evaluate and visualize counterfactuals of diverse methods. It is flexible enough to be easily extended by other counterfactual methods for classification or regression models. Currently, the package provides three counterfactual explanations methods. We discuss some (optional) extensions we have made to these methods: first, to generalize them to diverse scenarios (for example, to regression models or multiclass classifiers), and second, to improve their comparability, for example, by letting the two methods that return only one counterfactual, return several ones just like the third method. Our work is therefore one of the few that explicitly advocate methods that simultaneously generate multiple, qualitatively comparable counterfactuals rather than a single one. We are also among the first to provide an evaluation approach for sets of counterfactuals of *different sizes* by comparing the three implemented methods in a benchmark study. In contrast, previous work primarily focused on one counterfactual per method (de Oliveira and Martens 2021; Pawelczyk *et al.* 2021; Moreira, Chou, Hsieh, Ouyang, Jorge, and Pereira 2022). Because the package and benchmark study code are openly available, we encourage readers to add counterfactual approaches to our R package and compare them to the ones that have already been implemented.

In the upcoming section, we present the three currently implemented methods. In Section 3, we explain the overall structure and handling of the package as well as its most important

functionalities. We present use cases for a regression and classification task to show the main functionalities of the package in Section 4, followed by an example in Section 5 illustrating how additional counterfactual explanation methods can be easily integrated into our package. In Section 6, we show the general setup and results of the benchmark study. We summarize our findings as well as open questions in Section 7.

2. Methodological background and extensions

Our definition of counterfactual explanations is based on the work of Dandl *et al.* (2020b) and Verma *et al.* (2022).

Definition 1 (Counterfactual explanation)**.** Let $\hat{f} : \mathcal{X} \to \mathbb{R}$ be a prediction function with $\mathcal{X}$ as the p-dimensional feature space. While our definition naturally covers regression models, for classification tasks, we assume that $\hat{f}$ returns the score or probability for a predefined class of interest, usually the so-called positive class. Let further $\mathbf{X} := (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)})$ with $\mathbf{x}^{(i)} \in \mathcal{X}, i \in \{1, \ldots, n\}$ be the observed data and $Y' = [Y'_l, \ Y'_u]$ be an interval of desired predictions. We define a point $\mathbf{x}$ as a counterfactual explanation for an observation $\mathbf{x}^\star$ if $\mathbf{x}$ fulfills (at least some of) the following desired properties:

i Validity: $\mathbf{x}$ leads to a desired prediction, i.e., $\hat{f}(\mathbf{x}) \in Y'$. This could be assessed, e.g., by (Dandl *et al.* 2020b)

$$o_{\text{valid}}\left(\hat{f}(\mathbf{x}), Y'\right) = \begin{cases} 0, & \text{if } \hat{f}(\mathbf{x}) \in Y' \\ \min_{y' \in Y'} \left|\hat{f}(\mathbf{x}) - y'\right|, & \text{otherwise} \end{cases}. \tag{1}$$

ii Proximity: $\mathbf{x}$ is close to $\mathbf{x}^\star$, which could be measured, e.g., by the Gower distance d_G (Gower 1971) for mixed feature spaces

$$o_{\text{prox}}\left(\mathbf{x}, \mathbf{x}^\star\right) = d_G(\mathbf{x}, \mathbf{x}^\star) := \frac{1}{p} \sum_{j=1}^{p} \delta_G\left(x_j, x_j^\star\right) \in [0, 1] \tag{2}$$

with

$$\delta_G(x_j, x_j^\star) = \begin{cases} \frac{1}{\hat{R}_j} \left|x_j - x_j^\star\right| & \text{if } x_j \text{ is numerical} \\ \mathbb{I}_{x_j \neq x_j^\star} & \text{if } x_j \text{ is categorical} \end{cases}$$

where $\hat{R}_j = \max(\mathbf{X_j}) - \min(\mathbf{X_j})$ is the value range of feature j in $\mathbf{X}$.

iii Sparsity: $\mathbf{x}$ differs from $\mathbf{x}^\star$ in only a few features. This can be measured by the L_0 norm

$$o_{\text{sparse}}\left(\mathbf{x}, \mathbf{x}^\star\right) = \|\mathbf{x} - \mathbf{x}^\star\|_0 = \sum_{j=1}^{p} \mathbb{I}_{x_j \neq x_j^\star}. \tag{3}$$

iv Plausibility: $\mathbf{x}$ is realistic, i.e., close to the data manifold. Metrics are the (weighted) Gower distance to the k closest training samples $\mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[k]} \in \mathbf{X}$ (Dandl *et al.* 2020b)

$$o_{\text{plaus}}\left(\mathbf{x}, \mathbf{X}\right) = \sum_{i=1}^{k} w^{[i]} d_G\left(\mathbf{x}^{[i]}, \mathbf{x}\right) \in [0, 1] \text{ where } \sum_{i=1}^{k} w^{[i]} = 1 \tag{4}$$

or the reconstruction error of a variational autoencoder (VAE) trained on the training samples (Brughmans *et al.* 2023).

v Actionability: $\mathbf{x}$ does not alter immutable features (e.g., country of birth) and only proposes changes within an actionable range (e.g., non-negative age).

vi Causality: $\mathbf{x}$ reflects the underlying causal structure and takes causal relations of features into account. This property could be only examined if the causal graph (Pearl 2009) is (at least partially) known (Karimi *et al.* 2020, 2021; Mahajan, Tan, and Sharma 2020). Since this is rarely the case, most counterfactual methods (including the ones implemented in the **counterfactuals** package) disregard this property (Verma *et al.* 2022).

While some desired properties have a common tendency, others are rather opposed: if an explanation is sparse (iii), it also tends to be proximal (ii), since a counterfactual tends to be close to the original data point when only a few features are changed. However, a counterfactual that is close to the original data point tends to have a similar prediction, which may be far from a desired prediction, thus making the counterfactual less valid (i). The exact interdependence between the properties depends on the prevailing circumstances. Existing counterfactual methods vary in the desired properties they consider and how they measure and optimize them. An overview of methods is given in Verma *et al.* (2022). The methods also vary in whether a single counterfactual or a set of diverse ones is generated for an $\mathbf{x}^\star$. We argue that a set of counterfactuals is more valuable than a single one. This is because there could exist different equally good counterfactuals with the desired prediction (Rashomon effect, see Breiman 2001) and it is more likely that a set contains a counterfactual that satisfies a user's (hidden) preferences (Dandl *et al.* 2020b).

Below, we introduce the three counterfactual methods currently available in the **counterfactuals** package: MOC (Dandl *et al.* 2020b), the What-If tool (WhatIf, Wexler, Pushkarna, Bolukbasi, Wattenberg, Viégas, and Wilson 2019), and NICE (Brughmans *et al.* 2023). By addressing their limitations, we motivate optional extensions of the methods that we implemented in our package. In particular, these extensions enable all methods to return multiple counterfactuals for binary and multiclass classification models, as well as regression models.

2.1. Multi-objective counterfactual explanations

Original method

MOC by Dandl *et al.* (2020b) searches for counterfactuals by solving a multi-objective minimization problem

$$\min_{\mathbf{x}} \mathbf{o}(\mathbf{x}) := \min_{\mathbf{x}} \left(o_{\text{valid}}(\hat{f}(\mathbf{x}), Y'), o_{\text{prox}}\left(\mathbf{x}, \mathbf{x}^\star\right), o_{\text{sparse}}\left(\mathbf{x}, \mathbf{x}^\star\right), o_{\text{plaus}}(\mathbf{x}, \mathbf{X}) \right). \tag{5}$$

The single objectives correspond to the desired properties validity, proximity, sparsity, and plausibility formalized in Equations 1 to 4 as part of Definition 1. MOC also considers actionability by allowing the specification of "fixed features" that remain unchanged and of alteration ranges for continuous features.

To tackle the optimization problem in Equation 5, MOC uses a customized version of the non-dominated sorting genetic algorithm (NSGA-II) of Deb, Pratap, Agarwal, and Meyarivan (2002): unlike the original algorithm, MOC employs mixed-integer evolution strategies (Li *et al.* 2013) to handle mixed feature spaces and computes the crowding distance not only in

the objective space but also in the feature space. A description of the steps of the algorithm as implemented in the **counterfactuals** package is given in Algorithm 1 in Appendix A.

The algorithm first initializes a population. The authors proposed several strategies:

- Random: Feature values of new individuals are sampled uniformly from the range of observed values. Subsequently, some features are randomly reset to their initial value in $\mathbf{x}^\star$ to induce sparsity.
- ICE curve: As for the "random" strategy, feature values are sampled from the range of observed values. Then, however, features are reset with probabilities relative to their feature importance: the higher the importance of a feature $\mathbf{x}_j$, the higher the probability that its values differ from $\mathbf{x}_j^\star$. The importance of one feature is measured using the standard deviation of its corresponding individual conditional expectation (ICE) curve (Goldstein, Kapelner, Bleich, and Pitkin 2015).
- Standard deviation: This method is similar to "random", except that the sample ranges of numerical features are limited to one standard deviation from their value in $\mathbf{x}^\star$.
- Training data: Contrary to the other strategies, individuals are drawn from non-dominated previous observations in the dataset. If insufficient observations are available, the remaining individuals are initialized by random sampling. Subsequently, some features are randomly reset to their initial value in $\mathbf{x}^\star$.

Dandl *et al.* (2020b) discussed only the first two strategies in their paper, although the third and fourth strategies were also available in their implementation (Dandl *et al.* 2020a). In subsequent generations, the algorithm recombines and mutates individuals of the population and their features with predefined probabilities so that the initial population evolves. For mutation, the authors state two approaches: the first is to apply a scaled Gaussian mutator to numerical features and a uniform discrete mutator to categorical features (Li *et al.* 2013); the second approach aims to take feature distributions into account by sampling conditionally on the other feature values using a transformation tree (Hothorn and Zeileis 2021).

After recombination and mutation, some features are randomly reset to their initial value in $\mathbf{x}^\star$ with prespecified probabilities to induce sparsity. The recombination and mutation steps in the algorithm can be customized via multiple control parameters. An overview is given in Appendix B.2. To emphasize validity (i), individuals whose prediction exceeds a specified target distance $\epsilon \in \mathbb{R}_{\geq 0}$ can be penalized using the approach of Deb *et al.* (2002). MOC terminates either after a prespecified number of generations or when the hypervolume (HV) indicator (Zitzler and Thiele 1998) of the objectives in (5) does not improve for a prespecified number of consecutive generations. As counterfactuals, MOC returns all (unique) non-dominated individuals across all generations.

Contrary to most other methods, MOC is inherently applicable to both classification and regression tasks. Moreover, MOC does not require the user to weigh the objectives *a priori* and thus avoids the risk of arbitrarily affecting the solution set. Instead, it returns a Pareto set of counterfactuals so that the objectives can be weighted *a posteriori*.

Modifications

We did not rely on the previous implementation of MOC on Github (Dandl *et al.* 2020a). Instead, we reimplemented an updated version of MOC: we replaced the NSGA-II implemen-

tation in **mosmafs** (Binder, Dandl, and Moosbauer 2020) with its extended and more versatile successor **miesmuschel** (Binder, Schneider, Dandl, and Hofheinz 2023), and parameter spaces are now defined by the **paradox** package (Lang, Bischl, Richter, Sun, and Binder 2023) instead of **ParamHelpers** (Bischl, Lang, Richter, Bossek, Horn, and Kerschke 2020).

2.2. WhatIf

Original method

WhatIf is the counterfactual method for classification models proposed by Wexler *et al.* (2019) as part of the **What-If Tool** (`https://pair-code.github.io/what-if-tool/`). Wexler *et al.* (2019) assume that the underlying model $\hat{h} : \mathcal{X} \to \mathcal{Y}$ predicts a class label. They define the set of desired predictions Y' as the set of all labels other than the current one. As a counterfactual $\mathbf{x}'$ for an observation $\mathbf{x}^\star$, WhatIf returns the data point most similar to $\mathbf{x}^\star$ from previous observations $\tilde{\mathbf{X}} = \{\mathbf{x} \in \mathbf{X} : \hat{h}(\mathbf{x}) \neq \hat{h}(\mathbf{x}^\star)\}$ whose predicted class is different from that of $\mathbf{x}^\star$. This leads to the minimization problem:

$$\mathbf{x}' \in \underset{\mathbf{x} \in \tilde{\mathbf{X}}}{\operatorname{argmin}}\, d(\mathbf{x}, \mathbf{x}^\star). \tag{6}$$

The function d is a slightly adapted version of the Gower distance (Equation 2): for numerical features, the authors scale the distances with the standard deviations $\hat{\sigma}_j$; for categorical features, the feature distances are set equal "to the probability that any two examples across the entire dataset would share the same value for that feature" if their values differ, and 0 otherwise (Wexler *et al.* 2019). By definition, WhatIf aims for valid (i), proximal (ii), and plausible (iv) counterfactuals. WhatIf often serves as a baseline method in benchmark studies (Dandl *et al.* 2020b; Schleich, Geng, Zhang, and Suciu 2021; Carreira-Perpiñán and Hada 2021) because it is easily implementable and adaptable.

Modifications

For better comparability with MOC, we use the original Gower distance as the default for d in the **counterfactuals** package. We allow users to replace this with other dissimilarity measures (see Section 4.2). We also extended the method to work with $\hat{f}$ that returns the probability of a prespecified class of interest for classification tasks instead of a hard label classifier $\hat{h}$. This allows us to define the set of desired predictions Y' as a probability interval $[Y'_l,\ Y'_u] \subseteq [0, 1]$. Additionally, our approach makes WhatIf applicable to regression tasks without further modifications. In this case, Y' can simply be any real interval. $\tilde{\mathbf{X}}$ is then redefined as $\tilde{\mathbf{X}} = \{\mathbf{x} \in \mathbf{X} : \hat{f}(\mathbf{x}) \in Y'\}$.

As argued in Section 1, methods that can find multiple counterfactuals for a single observation are preferable. Therefore, we implemented an extended WhatIf version that returns the $l \in \mathbb{N}$ closest data points of $\tilde{\mathbf{X}}$ to $\mathbf{x}^\star$ with the desired prediction. This is equivalent to minimizing the following objective instead of (6)

$$\{\mathbf{x}'_1, \ldots, \mathbf{x}'_l\} \in \underset{\mathbf{Z} \subseteq \tilde{\mathbf{X}},\, |\mathbf{Z}| = l}{\operatorname{argmin}} \sum_{\mathbf{z} \in \mathbf{Z}} d_G(\mathbf{z}, \mathbf{x}^\star). \tag{7}$$

2.3. Nearest instance counterfactual explanations

Original method

NICE introduced by Brughmans *et al.* (2023) is a counterfactual explanation method for binary score classifiers $\hat{f} : \mathcal{X} \to [-1, 1]$. Accordingly, they define the set of desired predictions Y' as the set of all scores that lead to a different class than the current one. NICE starts the counterfactual search for an observation $\mathbf{x}^\star$ by finding its most similar instance $\mathbf{x}_{nn}$ for which a class other than $\mathbf{x}^\star$ was *correctly* predicted. Brughmans *et al.* (2023) assess similarity by the heterogeneous Euclidean overlap method (Wilson and Martinez 1997) with L_1-norm aggregation, which corresponds to the Gower distance without averaging (i.e., Equation 2 without $\frac{1}{p}$).

Once $\mathbf{x}_{nn}$ is found, NICE generates new instances in the first iteration ($m = 1$) by replacing single feature values of $\mathbf{x}^\star$ with the corresponding value of $\mathbf{x}_{nn}$. NICE evaluates the created instances with a reward function that optimizes either sparsity, proximity, or plausibility (see Equations (2)-(4) in Brughmans *et al.* 2023, for details).

If the prediction of the instance with the highest reward value is in Y', the algorithm terminates and returns this instance as a counterfactual. Otherwise, NICE creates new instances in the next iteration by replacing single feature values of the best performing instance of the previous iteration with the corresponding value of $\mathbf{x}_{nn}$. The search continues as long as the prediction for the highest reward value instance is not in Y'.

Modifications

We generalized NICE for regression models and multiclass classifiers: first, we extend $\hat{f}$ to predict real-values (regression) or the probability of a predefined class k, respectively (see Definition 1). Second, we conceptualize the search for $\mathbf{x}_{nn}$ as the following minimization problem:

$$\mathbf{x}_{nn} = \underset{\mathbf{x} \in \mathring{\mathbf{X}}'}{\operatorname{argmin}}\, o_{\text{prox}}(\mathbf{x}, \mathbf{x}^\star) \tag{8}$$

with o_{prox} as defined in Equation 2. For classification, $\mathring{\mathbf{X}}' = \{\mathbf{x} \in \mathbf{X} : \hat{f}(\mathbf{x}) \in Y' \wedge h(\hat{f}(\mathbf{x})) = y\}$ is the set of all correctly classified observations whose prediction is in the set of desired predictions Y'. y is the true class label of $\mathbf{x}$ and $h(\cdot)$ is a transformation function that maps class scores onto class labels. For regression, $\mathring{\mathbf{X}}' = \{\mathbf{x} \in \mathbf{X} : \hat{f}(\mathbf{x}) \in Y' \wedge |\hat{f}(\mathbf{x}) - y| \leq \epsilon\}$ is the set of all observations with a prediction in the desired real interval Y' and a prediction error of less than a user-specified $\epsilon \in \mathbb{R}_{\geq 0}$. Similar to WhatIf, o_{prox} in Equation 8 could be replaced with user-defined distance measures in our implementation (demonstrated in Section 4.2).

The whole process after finding $\mathbf{x}_{nn}$ is already applicable to both multiclass classification and regression tasks. We only updated the proposed reward functions for an iteration m to

$$R_O(\mathbf{x}) = \frac{o_{\text{valid}}(\hat{f}(\mathbf{x}_{m-1,R_{\max}}), Y') - o_{\text{valid}}(\hat{f}(\mathbf{x}), Y')}{O(\mathbf{x}, \mathbf{x}_{m-1,R_{\max}} \mid \mathbf{x}^\star)}, \tag{9}$$

where $\mathbf{x}_{i-1,R_{\max}}$ is the highest reward instance of the previous iteration ($m - 1$), and o_{valid} is defined in Equation 1. The denominator $O(\cdot, \cdot)$ corresponds to the originally proposed functions aiming at sparsity, proximity, or plausibility (see Equations 2–4 in Brughmans *et al.* 2023).

Although multiple instances could have the desired prediction (and similar reward values), the original NICE algorithm only returns a single counterfactual. In the **counterfactuals** package, we implemented two (optional) extensions that enable NICE to return multiple counterfactuals. Our first extension returns all created instances (from all iterations) with a desired prediction as counterfactuals after termination. Our second extension does not terminate when the prediction of the highest reward instance is in the desired interval. Instead, it continues until $\mathbf{x}_{nn}$ is recreated. This leads to a total number of $(d^2+d)/2$ created instances, where d is the number of feature values that differ between $\mathbf{x}^\star$ and $\mathbf{x}_{nn}$. Like our first extension, it then returns all created instances with a desired prediction as counterfactuals. Compared to counterfactuals in earlier iterations, a counterfactual created in a later iteration is inferior w.r.t. proximity (ii) and sparsity (iii) (as more feature values are changed), but may be superior w.r.t. plausibility (iv). The pseudocode of our modified NICE version is shown in Algorithm 2 in Appendix A.

In contrast to MOC, NICE does not consider all the desired counterfactual properties (listed in Definition 1) simultaneously: while NICE guarantees validity by design (provided that a correctly classified observation with a desired prediction exists), the user must prioritize the other desired properties under the given circumstances and choose the reward function accordingly. If there is no clear preference for the properties *a priori*, we recommend running our second NICE extension for each of the reward functions, combining the counterfactuals, removing duplicates, and evaluating the remaining counterfactuals *a posteriori*. We chose this strategy for our benchmark study in Section 6.

A not yet implemented extension is to set lower and upper bounds on $\mathbf{x}_{nn}$ to constrain the feature values of the counterfactuals, enhancing their actionability (v). Another extension would be to run the algorithm multiple times, defining $\mathbf{x}_{nn}$ in the l-th run as the l-th most similar (correctly classified) data point of $\mathbf{x}^\star$, which increases the diversity of the counterfactuals.

3. counterfactuals R package

In this section, we introduce the **counterfactuals** R package and explain its structure and workflow. The package is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=counterfactuals` (Dandl, Hofheinz, Binder, and Casalicchio 2026).

Inspired by the **iml** package (Molnar, Bischl, and Casalicchio 2018), each counterfactual method described in the previous section is implemented in `R6` classes (Chang 2021). Datasets and counterfactuals are represented as **data.table** objects (Dowle and Srinivasan 2022) to allow efficient data manipulations and computations. Depending on whether a counterfactual method supports classification or regression tasks, its class inherits from the (abstract) `CounterfactualMethodClassif` or `CounterfactualMethodRegr` `R6` classes, respectively. Counterfactual methods that support both tasks are split into two separate classes. Figure 1 illustrates the inheritance structure. For instance, as MOC is applicable to classification and regression tasks, we implemented two classes: `MOCClassif` and `MOCRegr`. Both classes rely on the same (private) code base (`moc_algo()`) to generate counterfactuals to avoid code repetitions. `MOCClassif` inherits features from its superclass `CounterfactualMethodClassif`, while `MOCRegr` inherits from `CounterfactualMethodRegr`. Both of these superclasses in turn have the `CounterfactualMethod` as their superclass.

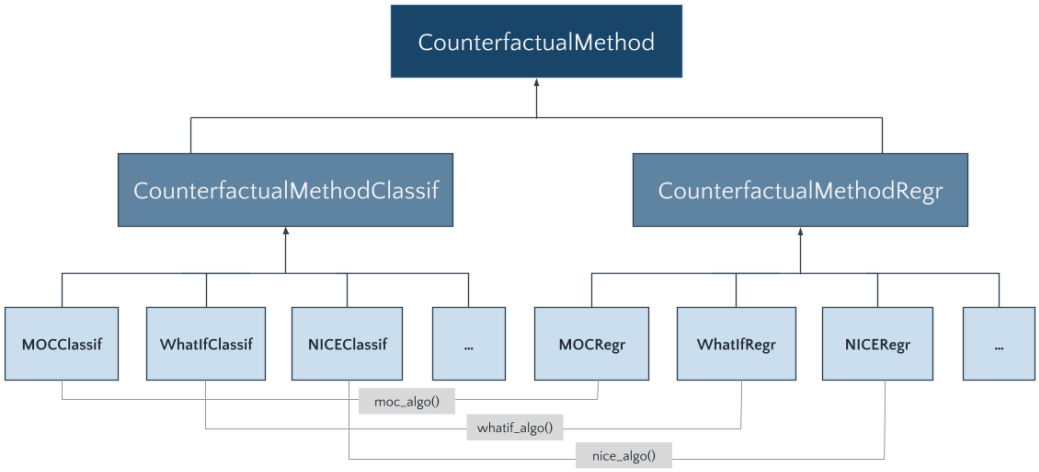


Figure 1: Inheritance diagram of the **counterfactuals** package; a more detailed version is included in Appendix B.1.

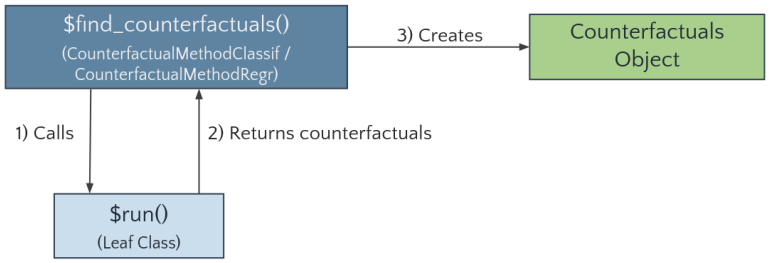


Figure 2: Call graph of the **counterfactuals** package. The `find_counterfactuals()` method (1) calls a private `run()` method – implemented by the leaf classes – which performs the search and (2) returns the counterfactuals as a `data.table`; `find_counterfactuals()` then (3) creates a `Counterfactuals` object, which contains the counterfactuals and provides several methods for their evaluation and visualization.

To generate counterfactuals for an arbitrary model with a specific counterfactual explanation method, the following steps are necessary: First, an `iml::Predictor` object which encapsulates a fitted model and the underlying data must be initialized. The `Predictor` object is a wrapper for any machine learning model and ensures a unified interface and output for model predictions. It offers the necessary flexibility to generate counterfactuals for models fitted with a variety of popular machine learning interfaces, e.g., fitted with the **caret** (Kuhn 2008), **mlr** (Bischl *et al.* 2016), or **mlr3** packages (Lang *et al.* 2019). We showcase this in the upcoming sections and Appendix B.3. The instantiated `Predictor` object serves as an input for the `predictor` field of the initialization method of the `WhatIfClassif/-Regr`, `MOCClassif/-Regr` or `NICEClassif/-Regr` classes. Additionally, the user can change the parameters of the used methods when initializing the object – such as the mutation probability for MOC or the used reward function for NICE. Overviews of the parameters are given in Tables 2–4 in Appendix B.2.

Counterfactuals are generated by calling the `$find_counterfactuals()` method of the initialized object inherited from the classes `CounterfactualMethodClassif/-Regr`. Figure 2 illustrates the internal call graph. As input, `find_counterfactuals()` requires the observation of interest $\mathbf{x}^\star$ for which we seek counterfactuals as well as the desired prediction. The method then calls the `$run()` method, which is implemented in the leaf classes, and creates a `Counterfactuals` object that contains the generated counterfactuals.

How the computational burden scales with the number of observations and number of features for the different methods is assessed in Section 6. Several tools are available to visualize and evaluate the counterfactuals. They are showcased and explained in more detail in the upcoming section. These tools are primarily based on the codebase underlying Dandl *et al.* (2020b). More tools will be added in the future.

4. Use cases

In this section, we illustrate the `counterfactuals` workflow by applying MOC (Section 2.1) to a classification task and our NICE extension (Section 2.3) to a regression task.

4.1. MOC applied to a classification task

As training data, we use the German Credit data set from the **rchallenge** package (Todeschini 2021).[1] The dataset originally contains 20 features on credit and personal information of 1000 bank customers. For illustrative purposes, we only consider the seven features: `duration`, `amount`, `purpose`, `age`, `employment_duration`, `housing`, and `number_credits`. The target variable `credit_risk` indicates whether a credit is a good/low or bad/high risk for the bank.

```
R> library("counterfactuals")
R> library("iml")
R> library("randomForest")
R> data("german", package = "rchallenge")
R> credit <- german[, c("duration", "amount", "purpose", "age",
+    "employment_duration", "housing", "number_credits", "credit_risk")]
```

We train a random forest with the **randomForest** package (Liaw and Wiener 2002) to predict the `credit_risk`. We omit observation 998 from the training data, which is $\mathbf{x}^\star$, to imitate the situation of finding counterfactuals for a new observation.[2]

```
R> set.seed(20210816)
R> rf <- randomForest(credit_risk ~ ., data = credit[-998L,])
```

An `iml::Predictor` object serves as a wrapper for different model types. It contains the model and the data for its analysis. We set `type = "prob"` such that class probabilities instead of hard labels are predicted. For our observation of interest $\mathbf{x}^\star$ – denoted in the code as `x_interest` – the model predicts a probability of being a good credit risk of 38.2%:

[1] The dataset was originally donated to UCI (Dua and Graff 2017) by Prof. Dr. Hofmann from Universität Hamburg and was later corrected by Grömping (2019).

[2] This does not rule out the possibility to generate counterfactuals for training data points.

```
R> predictor <- iml::Predictor$new(rf, type = "prob")
R> x_interest <- credit[998L,]
R> predictor$predict(x_interest)

    bad  good
1 0.618 0.382
```

Generation of counterfactuals

Now, we examine which risk factors must be changed to increase the predicted probability of being a good credit risk to at least 60%. Since we want to apply MOC to a classification model, we initialize a `MOCClassif` object. As explained in Section 2.1, individuals whose prediction is farther away from the desired interval than a prespecified value `epsilon` can be penalized. Here, we set `epsilon = 0` to penalize all individuals whose prediction is outside the desired interval. With the `fixed_features` argument, we fix the non-actionable features `age` and `employment_duration` to the respective value of $\mathbf{x}^\star$. By setting the termination criterion to `genstag`, we stop once the HV indicator does not increase for `n_generations = 10L` consecutive generations. By setting `quiet = TRUE`, no information on the optimization is printed.

```
R> moc_classif <- MOCClassif$new(predictor, epsilon = 0,
+    fixed_features = c("age", "employment_duration"),
+    termination_crit = "genstag", n_generations = 10L, quiet = TRUE)
```

We use the `$find_counterfactuals()` method to search for counterfactuals for `x_interest`. As we aim to find counterfactuals with a predicted probability of being a good credit risk of at least 60%, we set the `desired_class` to `"good"` and the `desired_prob` to `c(0.6, 1)`; this is equivalent to setting the `desired_class` to `"bad"` and `desired_prob` to `c(0, 0.4)`.

```
R> cfactuals <- moc_classif$find_counterfactuals(x_interest,
+    desired_class = "good", desired_prob = c(0.6, 1))
```

The `Counterfactuals` *object*

The resulting `Counterfactuals` object holds the counterfactuals in the `data` field and possesses several methods for their evaluation and visualization. Printing a `Counterfactuals` object gives an overview of the results. Overall, we generated 53 counterfactuals.

```
R> print(cfactuals)

53 Counterfactual(s)

Desired class: good
Desired predicted probability range: [0.6, 1]

Head:
```

```
   duration amount purpose age employment_duration housing number_credits
1:       21   7460  others  30            >= 7 yrs     own              1
2:       21   7054  others  30            >= 7 yrs     own              1
3:       21   6435  others  30            >= 7 yrs     own              1
```

The `$predict()` method returns the predictions for the counterfactuals.

```
R> head(cfactuals$predict(), 3L)
```

```
    bad  good
1 0.322 0.678
2 0.318 0.682
3 0.296 0.704
```

The `$evaluate()` method returns the counterfactuals along with some predefined quality measures `dist_x_interest`, `no_changed`, `dist_train`, and `dist_target` for the desired properties proximity, sparsity, plausibility, and validity (listed in Definition 1). The quality measures are equal to the objectives of MOC. Setting the `show_diff` argument to `TRUE` displays the counterfactuals as their difference from `x_interest`: for a numeric feature, positive values indicate an increase compared to the feature value in `x_interest` and negative values indicate a decrease; for factors, the feature value is displayed if it differs from `x_interest`; `NA` means "no difference".

```
R> head(cfactuals$evaluate(show_diff = TRUE, measures = c("dist_x_interest",
+    "dist_target", "no_changed", "dist_train")), 3L)
```

```
   duration amount purpose age employment_duration housing number_credits
1:       NA  -5220    <NA>  NA                <NA>    <NA>           <NA>
2:       NA  -5626    <NA>  NA                <NA>    <NA>           <NA>
3:       NA  -6245    <NA>  NA                <NA>    <NA>           <NA>
   dist_x_interest no_changed dist_train dist_target
1:      0.04103193          1 0.04215022           0
2:      0.04422330          1 0.03895885           0
3:      0.04908897          1 0.03409318           0
```

By design, there is no guarantee that all counterfactuals generated with MOC have a prediction $\in Y'$. Therefore, we use the `$subset_to_valid()` method to omit all non-valid counterfactuals. The method `$revert_subset_to_valid()` can reverse this step.

```
R> cfactuals$subset_to_valid()
R> nrow(cfactuals$data)
[1] 22
```

Of the 53 counterfactuals, 22 have the desired predictions. To detect which features are the most important levers to obtain a certain prediction, the relative frequency of feature changes across all counterfactuals can be plotted via the `$plot_freq_of_feature_changes()` method. Setting `subset_zero = TRUE` excludes all unchanged features from the plot. Figure 3 shows that all counterfactuals require changes in the credit amount.

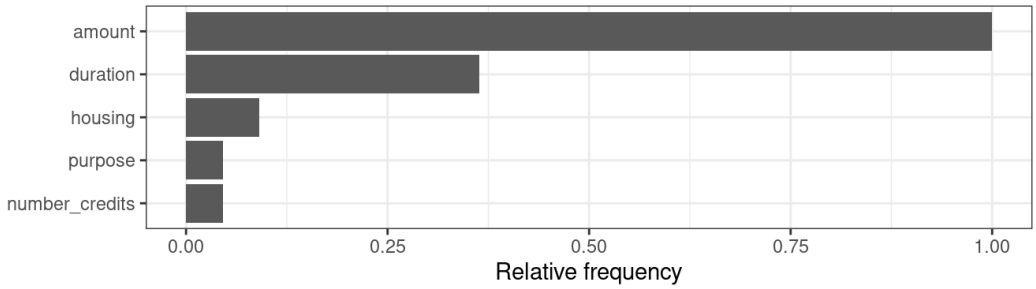


Figure 3: Relative frequency of feature changes across all counterfactuals. Features without proposed changes are omitted.

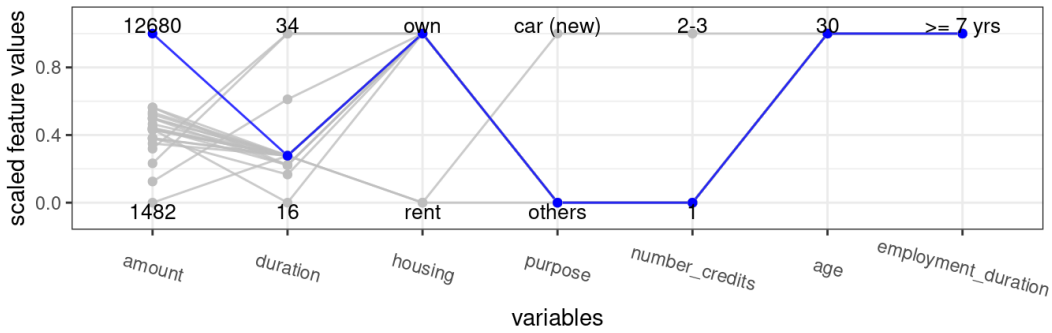


Figure 4: Parallel plot along (standardized) features. The blue line represents $\mathbf{x}^\star$ (`x_interest`), whereas gray lines represent generated counterfactuals.

```
R> cfactuals$plot_freq_of_feature_changes(subset_zero = TRUE)
```

The parallel plot (Figure 4) – created with the `$plot_parallel()` method – compares the feature values of the counterfactuals among each other (one gray line per counterfactual) and with `x_interest` (blue line). Equal to Dandl *et al.* (2020b), all features are scaled between 0 and 1. The argument `feature_names` filters the features and orders them, `NULL` means "all". Using `$get_freq_of_feature_changes()`, we order the features according to their frequency of changes. The `digits_min_max` argument specifies the maximum number of digits for plotted values. The default value is `2L`. All counterfactuals propose a decrease in the credit `amount` while the `duration` either needs no modifications, an increase or a decrease. For one counterfactual, additionally, the `purpose` was set to a new car, the `housing` type was set to rented and the `number_credits` was increased.

```
R> library(GGally)
R> cfactuals$plot_parallel(feature_names = names(
+    cfactuals$get_freq_of_feature_changes()), digits_min_max = 2L)
```

The `$plot_surface()` method generates prediction surface plots/2-dimensional ICE plots (Dandl *et al.* 2020b). The method requires the names of two features (`feature_names`) as an input. The white dot in Figure 5 represents `x_interest`. All counterfactuals that differ from `x_interest` *only* in the two selected features (here, `duration` and `amount`) are displayed as black dots. We observe that either a change in `amount` alone, or in `amount` *and* the `duration`

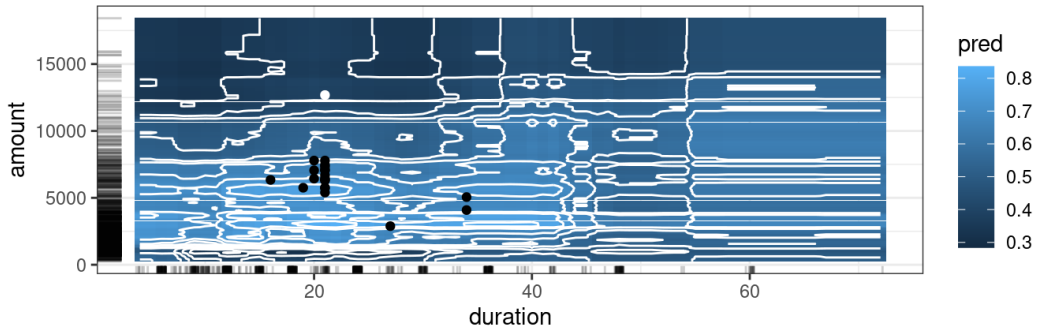


Figure 5: Prediction surface plotted along features `duration` and `amount`. Other feature values are held constant at $\mathbf{x}^\star$. The white point displays $\mathbf{x}^\star$. Black points are counterfactuals with variations only in the two displayed features. Rugs represent marginal distributions of the observed data.

is advocated. The rug lines next to the axes indicate the marginal distribution of the training data. It should be noted that the multi-objective approach does not consider counterfactuals farther away from `x_interest` as suboptimal because these counterfactuals outperform others in their proximity to the observed data points (plausibility property (iv)).

```
R> cfactuals$plot_surface(feature_names = c("duration", "amount"))
```

MOC diagnostics

The aforementioned plotting and evaluation methods are part of the class `Counterfactuals` and all counterfactuals created by MOC, WhatIf, or NICE can be evaluated with them. For MOC, additional diagnostic tools are available. Since they are only applicable to MOC, they cannot be called by the `Counterfactuals` class but rather by instances from the `MOCClassif` and `MOCRegr` class after counterfactuals were generated. To evaluate the estimated Pareto front, Dandl *et al.* (2020b) use a HV indicator (Zitzler and Thiele 1998) with reference point $s = (\inf_{y' \in Y'} |\hat{f}(\mathbf{x}^\star) - y'|, 1, p, 1)$ representing the maximal values of the objectives (o_{valid}, o_{prox}, o_{sparse}, o_{plaus} of Equations 1 to 4). The evolution of the HV indicator can be plotted together with the evolution of mean and minimum objective values using the `$plot_statistics()` method. The `centered_obj` argument allows the user to control whether the objective values should 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, as shown in Figure 6.

```
R> moc_classif$plot_statistics(centered_obj = TRUE)
```

Ideally, the mean value of each objective decreases, while the HV increases over the generations. However, there is often a trade-off between the objectives in the sense that when the mean value of one objective slightly decreases, it might slightly increase for another objective. This trade-off is also visible in the scatter plot created with the `$plot_search()` method that visualizes the values of two specified `objectives` of all emerged individuals. Ideally, one would like to have a point shift to the lower-left corner over the generations, which implies lower and thus better objective values.

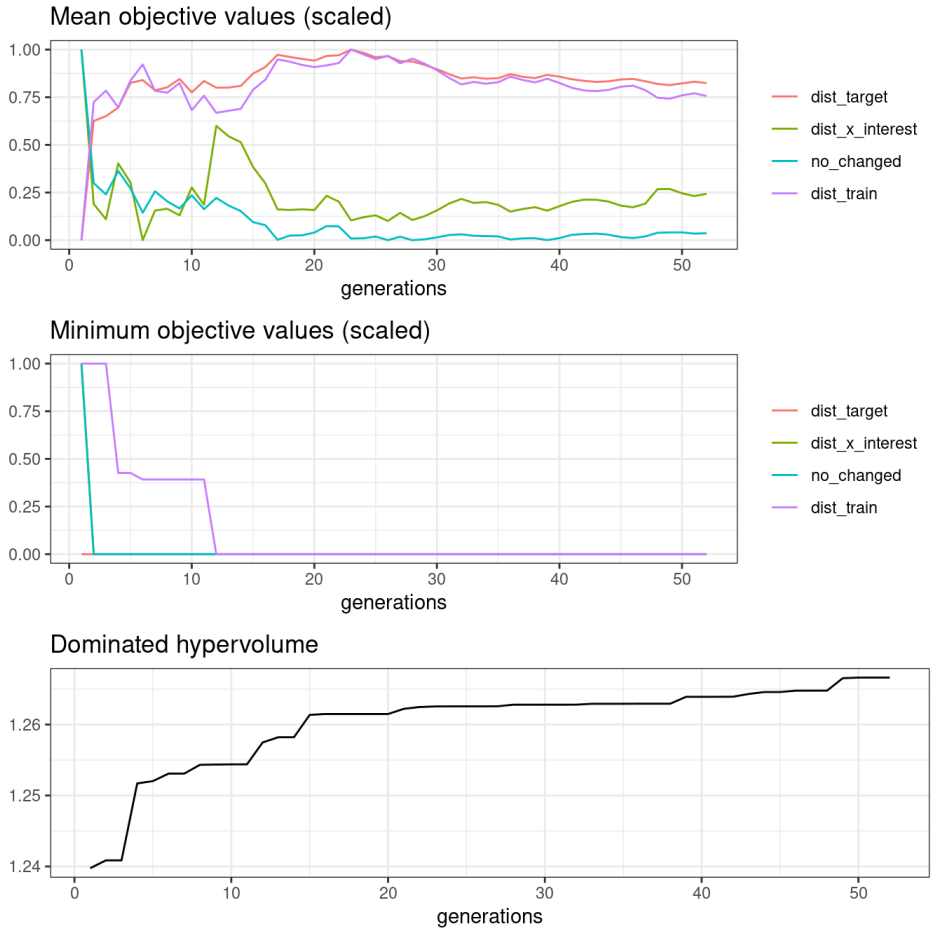


Figure 6: Evolution of the mean and minimum objective values together with the dominated HV over the generations. The mean and minimum objective values were scaled between 0 and 1.

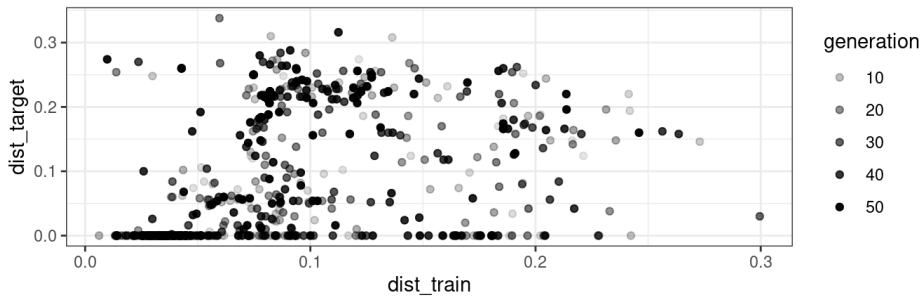


Figure 7: Evolution of the objectives `dist_train` and `dist_target` over the generations.

```
R> moc_classif$plot_search(objectives = c("dist_train", "dist_target"))
```

According to Figure 7, many counterfactuals have predictions in the desired prediction range (`dist_target` = 0). However, many points for the objectives `dist_train` and `dist_target` are also located in the middle region. This underlines the difficulty of minimizing both

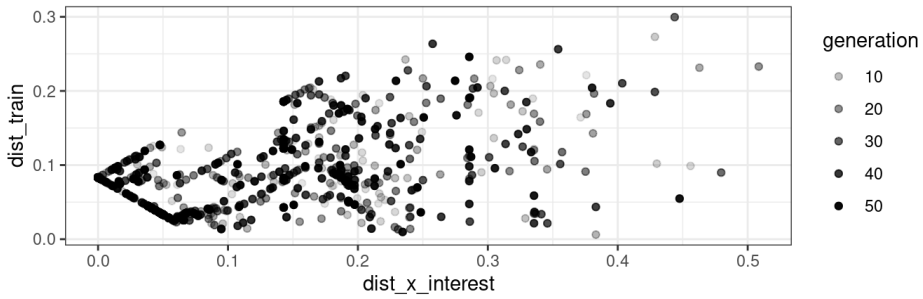


Figure 8: Evolution of the objectives `dist_x_interest` and `dist_train` over the generations.

objectives simultaneously. For the objectives `dist_train` and `dist_x_interest` (Figure 8), on the other hand, there is a clearer shift to the lower-left corner over the generations. The distinct boundary on the lower left indicates that the optimization potential for these two objectives might be fully exploited.

```
R> moc_classif$plot_search(objectives = c("dist_x_interest", "dist_train"))
```

4.2. NICE applied to a regression task

Searching for counterfactuals for regression models works analogously to classification models. In this example, we use our NICE extension for regression models to search for a counterfactual for a predictor of plasma retinol concentration. This is interesting because low concentrations are associated with an increased risk for some types of cancer (see Xie *et al.* (2019) for an overview).

As training data, we use the plasma dataset (Nierenberg, Stukel, Baron, Dain, Greenberg, and The Skin Cancer Prevention Study Group 1989) from the **gamlss.data** package (Stasinopoulos, Rigby, and De Bastiani 2021). The dataset contains 315 observations with 13 features describing personal and dietary factors (e.g., age, number of alcoholic drinks per week or the measured plasma beta-carotene level) and the (continuous) target variable `retplasma` – the plasma retinol concentration in ng/ml. We train a regression tree with the **mlr3** package to predict `retplasma` (Lang *et al.* 2019). We reserve the 100th row of the data for $\mathbf{x}^\star$ – denoted as `x_interest`.

```
R> library("mlr3")
R> data("plasma", package = "gamlss.data")
R> x_interest <- plasma[100L,]
R> tsk <- mlr3::TaskRegr$new(id = "plasma", backend = plasma[-100L,],
+    target = "retplasma")
R> tree <- lrn("regr.rpart")
R> model <- tree$train(tsk)
```

Then, we initialize an `iml::Predictor` object. For `x_interest`, the model predicts a plasma concentration of 342.92 ng/ml.

```
R> predictor <- Predictor$new(model, data = plasma, y = "retplasma")
R> predictor$predict(x_interest)
```

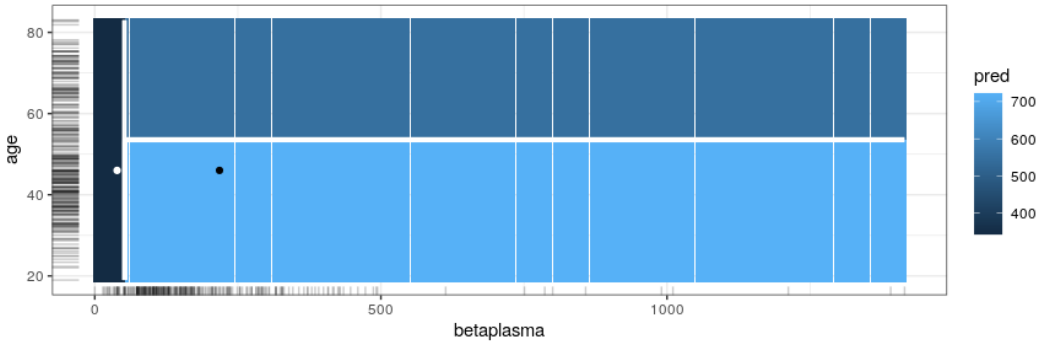


Figure 9: Prediction surface plotted along features `betaplasma` and `age`. Other feature values are held constant at $\mathbf{x}^\star$. The white point displays $\mathbf{x}^\star$. Black points are counterfactuals with variations only in the two displayed features. Rugs represent marginal distributions of the observed data. White horizontal lines are plotting artifacts.

```
  .prediction
1    342.9231
```

Since we want to apply NICE to a regression model, we initialize a `NICERegr` object. The initial version of NICE restricted to classification models starts the search by finding the most similar correctly classified data point. For regression models, we define a correctly predicted data point when its prediction is less than a user-specified value (`margin_correct`) away from the true outcome. In this example, we allow for a deviation of 0.5. The argument `optimization` specifies the reward function we want to optimize. We aim for the most proximal counterfactual by setting this argument to `proximity` and by setting `return_multiple` to `FALSE`.

We call the `$find_counterfactuals()` method to search for counterfactuals for `x_interest` with a predicted concentration of more than 500 ng/ml, i.e., a concentration in the interval $[500, \text{Inf}[$.

```
R> nice_reg <- NICERegr$new(predictor, optimization = "proximity",
+    margin_correct = 0.5, return_multiple = FALSE)
R> cfactuals <- nice_reg$find_counterfactuals(x_interest,
+    desired_outcome = c(500, Inf))
```

The result is a `Counterfactuals` object, which we can analyze with the same methods as in Section 4.1. The surface plot of plasma beta-carotene (`betaplasma`) and age (Figure 9), for example, reveals that increasing the beta-carotene concentration (e.g., by eating more kale, carrots, etc.) is sufficient for predicting a plasma concentration $\geq$ 500 ng/ml for $\mathbf{x}^\star$, while changing the age alone has no effect on the prediction.

```
R> cfactuals$plot_surface(feature_names = c("betaplasma", "age"),
+    grid_size = 200)
```

User-defined distance function

As stated in Equation 8, NICE determines the most similar (correctly classified) data point by minimizing the Gower distance. However, the input parameter `distance_measure` of the initialization method of `NICERegr` (and `NICEClassif`) allows a different distance measure. The parameter requires a function with arguments `x`, `y`, and `data`, that returns a numeric matrix with number of rows and columns corresponding to the number of observations in `x` and `y`, respectively. As an example, we replace the Gower function with the L_0 norm. First, we set up the function and illustrate its functionality in a short example.

```
R> l0_norm <- function(x, y, data) {
+    res <- matrix(NA, nrow = nrow(x), ncol = nrow(y))
+    for (i in seq_len(nrow(x))) {
+      for (j in seq_len(nrow(y))) {
+        res[i, j] <- sum(x[i,] != y[j,])
+      }
+    }
+    res
+  }
R> xt <- data.frame(a = 0.5, b = "a")
R> yt <- data.frame(a = c(0.5, 3.2, 0.1), b = c("a", "b", "a"))
R> l0_norm(xt, yt, data = NULL)

     [,1] [,2] [,3]
[1,]    0    2    1
```

Next, we forward this function to the `distance_function` argument of `NICERegr`.

```
R> nice_reg <- NICERegr$new(predictor, optimization = "proximity",
+    margin_correct = 0.5, return_multiple = FALSE,
+    distance_function = l0_norm)
R> nice_reg$find_counterfactuals(x_interest, desired_outcome = c(500, Inf))

1 Counterfactual(s)

Desired outcome range: [500, Inf]

Head:
   age sex smokstat    bmi vituse calories   fat fiber alcohol cholesterol
1:  46   1        3  35.26      3   2667.5 131.6  10.1       0       550.5
   betadiet retdiet betaplasma
1:     1210    1291        218
```

The initialization methods of MOC and WhatIf also have a `distance_function` argument: for MOC, its input replaces the Gower distances used for o_{prox} and o_{plaus} (Equations 2 & 4); for WhatIf, its input replaces the Gower distance in Equation 7.

5. Extending the package

We have designed the **counterfactuals** package to be quickly extensible by new methods. Here, we illustrate how to add new methods to the package by integrating the **featureTweakR** package (Kato 2018), which implements feature tweaking (Tolomei *et al.* 2017), a counterfactual method that can be applied to (classification) tree ensembles fitted with the **randomForest** package. Feature tweaking starts the search for counterfactuals for an observation $\mathbf{x}^\star$ by finding all trees in the ensemble that do not predict the desired class. For each of these trees, it attempts to change (or "tweak") $\mathbf{x}^\star$ as little as possible to switch the prediction of that tree to the desired class. From all tweaked instances that also switch the ensemble prediction to the desired class, it returns the tweaked instance that changes $\mathbf{x}^\star$ the least as a counterfactual.

The **featureTweakR** package has various limitations, e.g., factors in the training data cause problems or that it is only applicable to random forests trained on standardized features with the **randomForest** package (Liaw and Wiener 2002). Due to these limitations, **featureTweakR** is not part of the **counterfactuals** package but does serve as a suitable example here. First, we install **featureTweakR** and its dependency **pforeach** (Makiyama 2015) and load the required libraries.

```
R> devtools::install_github("katokohaku/featureTweakR")
R> devtools::install_github("hoxo-m/pforeach")
R> library("featureTweakR")
R> library("pforeach")
R> library("R6")
```

5.1. Class structure

At least two methods must be implemented for a new class: `$initialize()` and `$run()`. The `$print_parameters()` method is not mandatory but still strongly recommended, as it gives objects of that class an informative `print()` output. As elaborated above, a new class inherits from either `CounterfactualMethodClassif` or `CounterfactualMethodRegr`, depending on which task it supports. Since feature tweaking supports classification tasks, the new `FeatureTweakerClassif` class inherits from the former.

```
R> FeatureTweakerClassif <- R6::R6Class("FeatureTweakerClassif",
+    inherit = CounterfactualMethodClassif,
+    public = list(
+      initialize = function() {
+      }
+   ),
+   private = list(
+     run = function() {
+     },
+     print_parameters = function() {
+     }
+   )
+ )
```

Implementation of the `$initialize()` *method*

In the next step, we implement the `$initialize()` method, which must have a `predictor` argument that takes an `iml::Predictor` object. In addition, it may have further arguments specific to the counterfactual method. Feature tweaking has the following hyperparameters: `ktree` representing the number of trees to be considered, `epsiron`[3] as the upper threshold of feature changes, and `resample` indicating whether trees are randomly selected or not.

```
R> initialize <- function(predictor, ktree = NULL, epsiron = 0.1,
+    resample = FALSE) {
+      super$initialize(predictor)
+      private$ktree = ktree
+      private$epsiron = epsiron
+      private$resample = resample
+  }
```

We also fill the `$print_parameters()` method with the parameters of feature tweaking.

```
R> print_parameters <- function() {
+    cat(" - epsiron: ", private$epsiron, "\n")
+    cat(" - ktree: ", private$ktree, "\n")
+    cat(" - resample: ", private$resample)
+  }
```

Implementation of the `$run()` *method*

The `$run()` method performs the search for counterfactuals. Its structure is completely free, which makes it flexible to add new counterfactual methods to the **counterfactuals** package. The only requirement is that a `data.table` with the generated counterfactuals is returned at the end. The columns display the features and rows the counterfactuals.

The `$run()` method is called by the method `$find_counterfactuals()` implemented in the `CounterfactualMethodClassif` class. As shown in Section 4.1, `$find_counterfactuals()` requires as input `x_interest`, `desired_class`, and `desired_prob`, which are saved in private fields. Thus, `$run()` could directly access the information and preprocesses them before it passes them on to the implemented methods of **featureTweakR**.

The workflow of finding counterfactuals for `x_interest` with the **featureTweakR** package for a fitted random forest model `rf` consists of three steps: First, decision trees are transformed to data frames of paths by `getRules()`. Then, `set.eSatisfactory()` generates new instances by slightly altering feature values. Finally, `tweak()` generates counterfactuals for a specific instance $\mathbf{x}^\star$. Further information could be found in the documentation of the package (Kato 2018). The `$run()` method encapsulates these steps and returns a `data.frame` of generated counterfactuals.

```
R> run <- function() {
+    predictor <- private$predictor
+    y_hat_interest <- predictor$predict(private$x_interest)
```

[3]Please note that this is not a typo on our part, but the naming in the original implementation (Kato 2018).

```
+    class_x_interest <- names(y_hat_interest)[which.max(y_hat_interest)]
+    rf <- predictor$model
+
+    rules <- getRules(rf, ktree = private$ktree,
       resample = private$resample)
+    es <- set.eSatisfactory(rules, epsiron = private$epsiron)
+    tweaks <- tweak(
+      es, rf, private$x_interest, label.from = class_x_interest,
+      label.to = private$desired_class, .dopar = FALSE
+    )
+    return(tweaks$suggest)
+  }
```

The composite code of our new class can be seen in Appendix B.4.

5.2. Feature tweaking applied to a classification task

For demonstration purposes, we apply the implemented feature tweaking to the `iris` dataset (Fisher 1936; Anderson 1936). We train a random forest on the dataset and set up the `iml::Predictor` object, again omitting `x_interest` (here, row 150) from the training data.

```
R> set.seed(78546)
R> X <- subset(iris, select = -Species)[-150L,]
R> y <- iris$Species[-150L]
R> rf <- randomForest(X, y, ntree = 20L)
R> predictor <- iml::Predictor$new(rf, data = iris[-150L,],
+    y = "Species", type = "prob")
```

For `x_interest`, the model predicts a probability of 10% for `versicolor`.

```
R> x_interest <- iris[150L,]
R> predictor$predict(x_interest)

  setosa versicolor virginica
1      0        0.1       0.9
```

Now, we use feature tweaking to address the question: "What changes in `x_interest` are necessary for the model to predict a probability of at least 60% for `versicolor`?".

```
R> ft_classif <- FeatureTweakerClassif$new(predictor, ktree = 10L,
+    resample = TRUE)
R> cfactuals <- ft_classif$find_counterfactuals(x_interest = x_interest,
+    desired_class = "versicolor", desired_prob = c(0.6, 1))
```

As for MOC and NICE, the result is a `Counterfactuals` object which could be visualized and evaluated as shown in Section 4.1.

6. Benchmarking

In this section, we use a benchmark study to answer the following research questions:

OpenML ID	Name	Obs	Cont	Cat
31	credit_g	1,000	7	13
37	diabetes	768	8	0
50	tic_tac_toe	958	0	9
725	bank8FM	8,192	8	0
1479	hill_valley	1,212	100	0
40922	run_or_walk_info	88,588	6	0

Table 1: Description of the OpenML datasets used for benchmarking. Obs displays the number of observations, Cont the number of continuous features and Cat the number of categorical features.

1. How do the different methods implemented in the **counterfactuals** R package perform according to the properties validity (i), proximity (ii), sparsity (iii) and plausibility (iv) of Definition 1, and according to the HV indicator and number of non-dominated counterfactuals?

2. How do the methods differ in their runtime for an increasing number of observations (n) and number of features (p)?

The overall design of our benchmark study is strongly inspired by the work of Dandl *et al.* (2020b) who also compared different methods according to the four properties of Definition 1. Additionally, we evaluate the methods with regard to their runtime behavior and HV. Furthermore, we added NICE as another comparison method. Since our source code is openly available, (in the supplementary materials and at `https://github.com/slds-lmu/benchmark_2022_counterfactuals`) we encourage readers to add other counterfactual methods to our R package and to compare them to the already implemented ones using our study code.

6.1. Setup

We used six datasets from the OpenML platform (Vanschoren, Van Rijn, Bischl, and Torgo 2014) with binary classes, no missing values, and varying numbers of observations and features. Table 1 provides an overview of the datasets. To study the runtime behavior, we also ran all available methods on row-wise subsets (with differing number of observations $n \in \{886\ (1\%),\ 8859\ (10\%),\ 88588\ (100\%)\}$) of the `run_or_walk_info` dataset and column-wise subsets (with differing number of features $p \in \{10,\ 30,\ 100\}$) of the `hill_valley` dataset. The subsets were randomly generated and identical for all models and methods.

On each dataset, we tuned and trained five models using the **mlr3** R package (Lang *et al.* 2019): a random forest (ranger), an xgboost model, an RBF support vector machine (svm), a logistic regression (logreg), and a neural network with one hidden layer (neuralnet).[4] Beforehand, we standardized numerical features and one-hot-encoded categorical ones. For tuning, we employed random search with 30 evaluations and 5-fold cross-validation (CV) using the misclassification error as a performance measure. Further details on the tuning search space and the classification accuracies are given in Appendix C.1. Before training, we randomly selected ten observations from each dataset as $\mathbf{x}^\star$ and omitted them from the training data.

[4]For the `hill_valley` dataset with 100 features, two dense layers were used.

For each $\mathbf{x}^\star$, we set the desired class probability interval Y' to the opposite of the predicted class (based on a threshold of 0.5):

$$Y' = \begin{cases}]0.5, 1] & \text{if } \hat{f}(\mathbf{x}^\star) \leq 0.5 \\ [0, 0.5] & \text{else} \end{cases} . \tag{10}$$

For each dataset, model, and $\mathbf{x}^\star$, we computed counterfactuals with WhatIf, NICE and MOC. Apart from the stopping criterion, all MOC control parameters were set to their default values selected through iterated F-racing (López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari, and Stützle 2016) (see Appendix B.2). Notably, we used different datasets for tuning than for the benchmark study. The stopping criterion was convergence of the HV over 10 generations, with a total maximum of 500 generations. For all three counterfactual methods, we set the `distance_function` to `'gower_c'` – a C-based, more efficient version of Gower's distance based on the **gower** R package (Van der Loo 2022).

As stated in Section 2, we prefer a set of counterfactuals over a single one. MOC is designed to return multiple counterfactuals and we also let NICE and WhatIf return multiple ones. Therefore, the NICE control parameter `finish_early` was set to `FALSE`, corresponding to our second NICE extension (Section 2.3). In addition, we computed counterfactuals for each of the three different reward functions by varying the `optimization` hyperparameter and combined them for a final set of counterfactuals, as recommended in Section 2.3. For WhatIf, the number of counterfactuals was set to 10 via the `n_counterfactuals` parameter, in accordance with Dandl *et al.* (2020b). All other NICE and WhatIf control parameters (except the `distance_function`, see above) were set to their default values (Appendix B.2).

For the evaluation, we only considered the counterfactuals that (1) achieve the desired prediction such that $o_{\text{valid}} = 0$ and (2) are not dominated by other counterfactuals produced by the same method according to the remaining three objectives (o_{prox}, o_{sparse} and o_{plaus}). By design of the three methods, criterion (1) always holds for counterfactuals of WhatIf and NICE and (2) always for MOC.

For Research Question 1, we evaluated the generated counterfactuals by means of the desired properties stated in Definition 1: validity (i, o_{valid}), proximity (ii, o_{prox}), sparsity (iii, o_{sparse}) and plausibility (iv, o_{plaus}). We ranked all counterfactuals per dataset, model, and $\mathbf{x}^\star$ by their values in the desired properties, normalized the ranks between 0 and 1, and compared the normalized ranks between the methods. The ranking ensures that counterfactuals are comparable over all datasets and models. To take into account all three properties at once, we also computed the HV indicator, which measures the HV in the objective space between the non-dominated counterfactuals and a (worst-case) reference point (1 for o_{prox}, no. features for o_{sparse} and 1 for o_{plaus}). For Research Question 2, we tracked the runtime behavior for all methods in generating counterfactuals for (row-wise or column-wise subsets of) the `run_or_walk_info` and `hill_valley` datasets.

6.2. Results

In the following, we present the results for the two stated research questions.

Research Question 1

Figure 10 compares the ranking of counterfactuals according to the desired properties for MOC, NICE and WhatIf for each dataset separately. Figure 14 in the Appendix does the

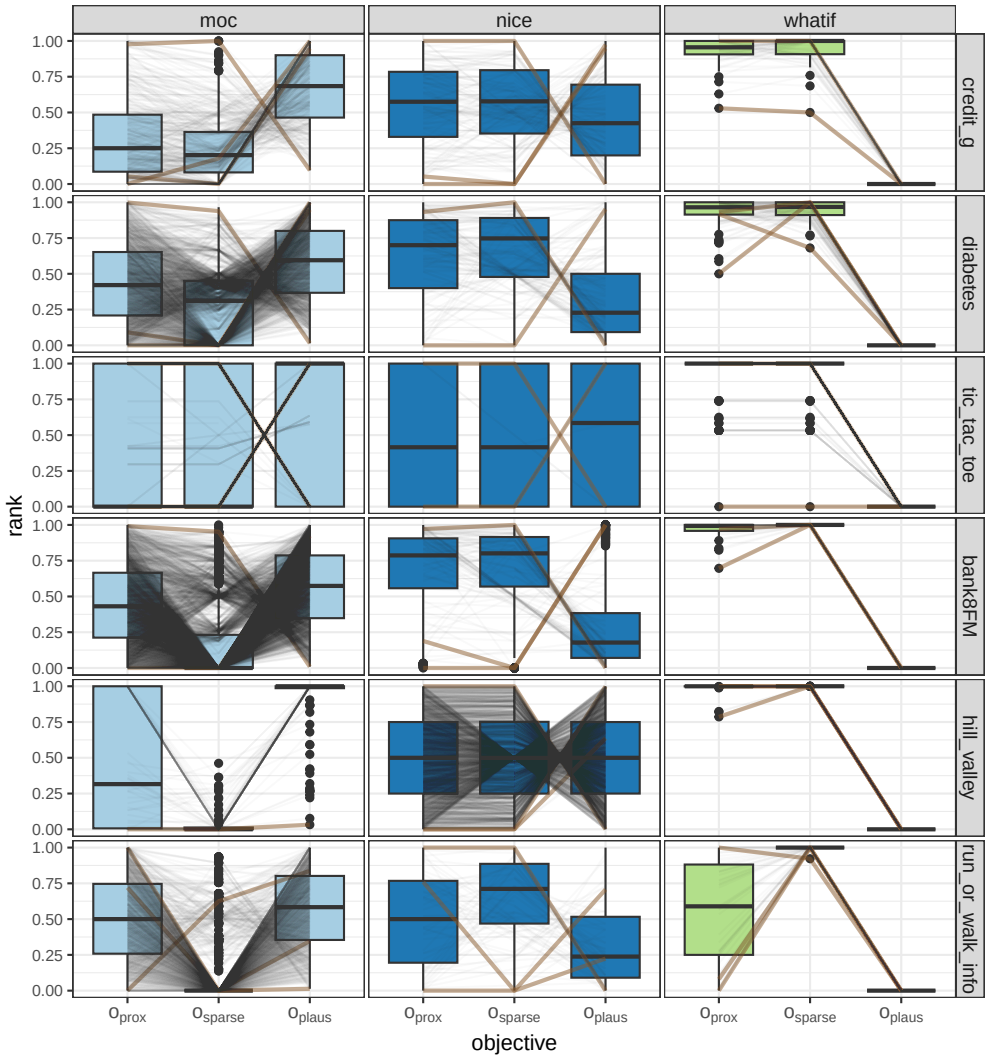


Figure 10: Comparison of NICE, WhatIf, and MOC w.r.t. their rank in the properties proximity (ii, o_{prox}), sparsity (iii, o_{sparse}) and plausibility (iv, o_{plaus}). Each gray line reflects a counterfactual (for clarity purposes, only a maximum of 2000 counterfactuals are displayed). The counterfactuals with the lowest and therefore best rank in an objective display the brown lines. Lower values are better.

same for each model separately. Since our setup ensured that all compared counterfactuals achieved the desired prediction, we omitted the results for the first property validity (i, o_{valid}). Each gray line reflects a counterfactual. The counterfactuals with the lowest and therefore best rank in one of the three remaining objectives display the brown lines. Appendix C.2 shows the results on the property instead of the raking scale for each model and dataset separately. They agree with the results shown here.

WhatIf's counterfactuals changed on average more features (o_{sparse}) and had the highest distances to $\mathbf{x}^\star$ (o_{prox}), making WhatIf inferior to the other methods w.r.t. the desired counterfactual properties sparsity (iii) and proximity (ii). However, its counterfactuals have low training data distances (o_{plaus}) by design, guaranteeing plausibility (iv).

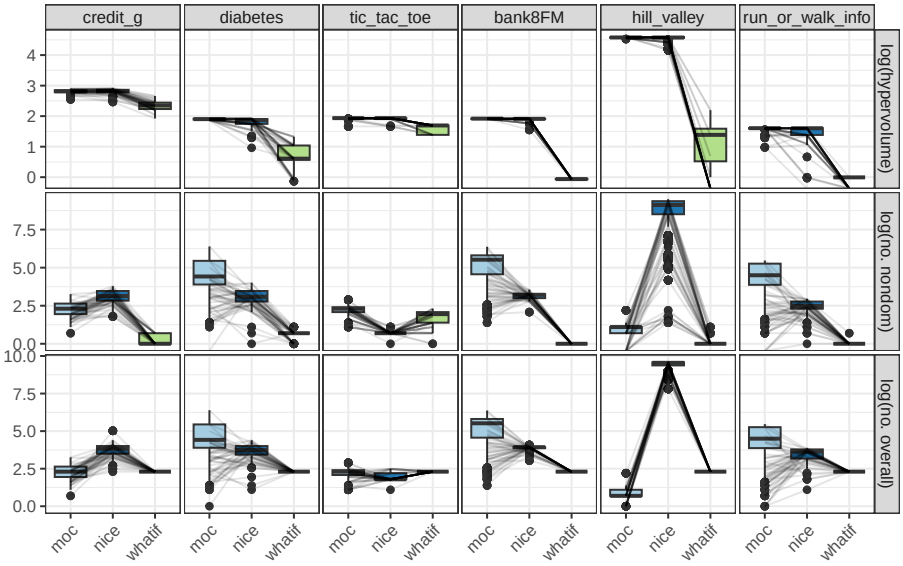
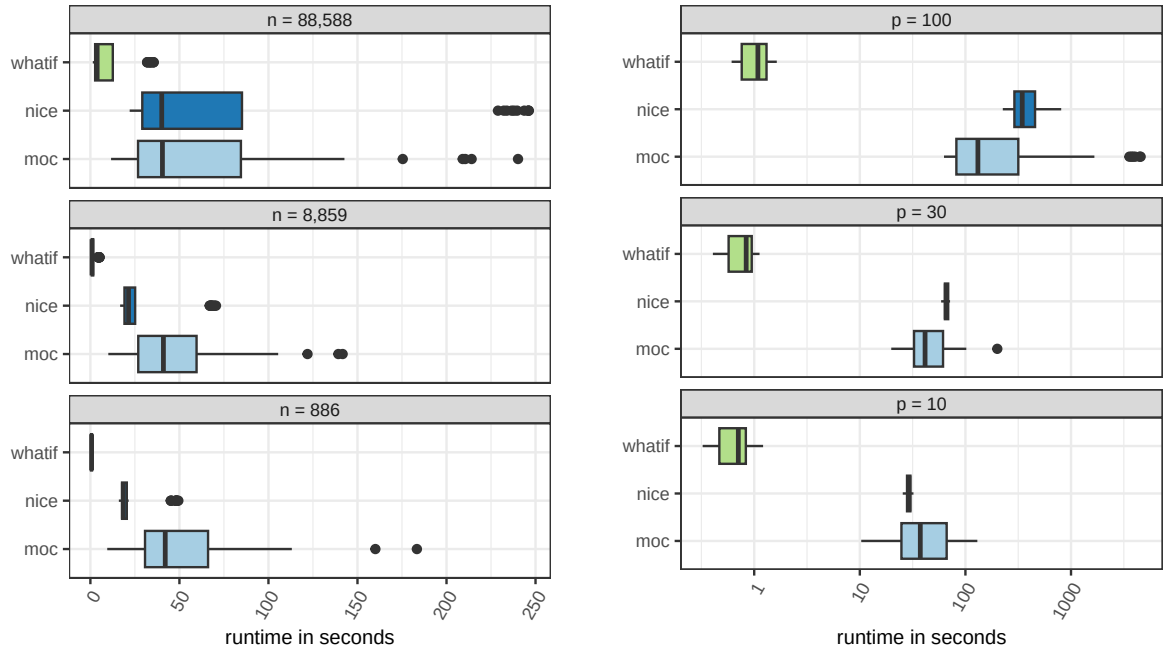


Figure 11: Comparison of NICE, WhatIf, and MOC w.r.t. their HV, the number of non-dominated and valid counterfactuals (no. nondom) and the number of all returned counterfactuals (no. overall). The values were logarithmized. Higher values are better.



(a) increasing n (`run_or_walk_info`)

(b) increasing p (`hill_valley`)

Figure 12: Speed comparison of NICE, WhatIf, and MOC based on row-wise subsets of the `run_or_walk_info` dataset and column-wise subsets of the `hill_valley` dataset. The runtimes of NICE were aggregated for its three reward function configurations.

Compared with MOC, the counterfactuals of NICE on average changed more features and had often a higher distance to $\mathbf{x}^\star$, indicating that NICE was overall inferior to MOC w.r.t. sparsity and proximity. However, on average, the counterfactuals of NICE had lower training data distances (measuring plausibility) than MOC's counterfactuals.

Figure 11, displays the HV, the number of non-dominated, valid counterfactuals, and the overall number of returned counterfactuals (including dominated and/or non-valid ones) on the log scale for each dataset and method. Overall, MOC's counterfactuals achieved the highest HV closely followed by NICE, indicating that MOC is slightly superior when considering all objectives simultaneously. The HV of WhatIf's counterfactuals is comparably low except for the `tic_tac_toe` dataset with a low number of categorical features. While all counterfactuals of MOC are (by design) non-dominated by other counterfactuals returned by the method, many of the counterfactuals of NICE or WhatIf are dominated by others generated by the same method. Apart from the `tic_tac_toe` dataset, WhatIf produced the least non-dominated counterfactuals. MOC generated the most non-dominated counterfactuals except for the `credit_g` and `hill_valley` datasets.

Research Question 2

Figure 12 compares the runtimes of our extended WhatIf and NICE versions with MOC. WhatIf was the fastest and best scaling method. NICE ran on average 2 times longer than MOC for high p. This is because for the `hill_valley` dataset with $p = 100$ features, the method at worst needs to evaluate $(p^2+p)/2 = 5050$ observations for each of the three reward functions. For low n, NICE was on average faster than MOC.

6.3. Discussion

In the following, we briefly discuss the suitability of each method for different scenarios based on the results of our benchmark study. MOC returned on average the most non-dominated counterfactuals of highest-quality when considering all desired properties simultaneously. Our extended NICE version had comparatively high runtimes for a medium to high number of features. WhatIf was the fastest method, but (by design) its counterfactuals suggested changes to many features, impeding the interpretation. The method is suitable in time-critical scenarios for datasets with a few categorical features.

7. Conclusion

In this work, we introduced the **counterfactuals** R package, which to the best of our knowledge is the first R package that provides several counterfactual methods via a unified interface. The package includes the method MOC as well as extended versions of WhatIf and NICE, which are all capable of returning multiple counterfactuals for regression and (binary and multiclass) classification models. In addition, we illustrated that the **counterfactuals** package is quickly extensible with new methods. This is crucial, as the variety of counterfactual methods proposed in research is growing rapidly, but the number of implemented methods in R is very limited. Furthermore, the package offers a variety of functionalities for evaluating and visualizing the counterfactuals. Thus, our package facilitates the application of counterfactual methods in practice for auditing machine learning models.

The results of our benchmark study and other research (e.g., Verma *et al.* 2022) suggest that no existing counterfactual method is superior in all situations. This underlines the benefit of the **counterfactuals** package, which makes a variety of methods readily available to the user. Furthermore, the object-oriented concept of our package and the openly available benchmark code allow new methods to easily compete with those currently available.

Computational details

The results in this work were obtained using R 4.3.2 (R Core Team 2025). R itself and most of the packages used are available from CRAN – including the **counterfactuals** R package (Dandl *et al.* 2026). We included all data examples of Sections 4 and 5 in dedicated vignettes. To facilitate full reproducibility of the benchmark study of Section 6, we created a dedicated Github repository: `https://github.com/slds-lmu/benchmark_2022_counterfactuals`. The experiments were run in parallel with the help of the **batchtools** package (Lang, Bischl, and Surmann 2017) on a computer with a 2.60 GHz Intel Xeon processor, and 32 CPUs. Training (incl. tuning) the models took 53 hours spread over 15 CPUs, and generating the counterfactuals took 37 hours spread over 14 CPUs.

Acknowledgments

This work has been partially supported by the Federal Statistical Office of Germany. We thank Gunnar König for insightful discussions on causality in counterfactual explanations.

References

Anderson E (1936). "The Species Problem in Iris." *Annals of the Missouri Botanical Garden*, **23**(3), 457–509. doi:10.2307/2394164.

Baron S (2023). "Explainable AI and Causal Understanding: Counterfactual Approaches Considered." *Minds and Machines*, **33**(2), 347–377. doi:10.1007/s11023-023-09637-x.

Biecek P (2018). "**DALEX**: Explainers for Complex Predictive Models in R." *Journal of Machine Learning Research*, **19**(84), 1–5.

Binder M, Dandl S, Moosbauer J (2020). ***mosmafs**: Multi-Objective Simultaneous Model and Feature Selection*. doi:10.32614/CRAN.package.mosmafs. R package version 0.1.2.

Binder M, Pfisterer F, Lang M, Schneider L, Kotthoff L, Bischl B (2021). "**mlr3pipelines** - Flexible Machine Learning Pipelines in R." *Journal of Machine Learning Research*, **22**(184), 1–7. doi:10.1201/9781003402848.

Binder M, Schneider L, Dandl S, Hofheinz A (2023). ***miesmuschel**: Mixed Integer Evolution Strategies*. doi:10.32614/CRAN.package.miesmuschel. R package version 0.0.4-2.

Bischl B, Lang M, Kotthoff L, Schiffner J, Richter J, Studerus E, Casalicchio G, Jones ZM (2016). "**mlr**: Machine Learning in R." *Journal of Machine Learning Research*, **17**(170), 1–5.

Bischl B, Lang M, Richter J, Bossek J, Horn D, Kerschke P (2020). ***ParamHelpers**: Helpers for Parameters in Black-Box Optimization, Tuning and Machine Learning*. doi:10.32614/CRAN.package.paramhelpers. R package version 1.14.1.

Breiman L (2001). "Statistical Modeling: The Two Cultures." *Statistical Science*, **16**(3), 199–231. doi:10.1214/ss/1009213726.

Brughmans D (2021). "**NICE**: Nearest Instance Counterfactual Explanations." Github repository. URL https://github.com/DBrughmans/NICE. Commit 6389a39692e9b98ecb1734f6603167029987f870.

Brughmans D, Leyman P, Martens D (2023). "NICE: An Algorithm for Nearest Instance Counterfactual Explanations." *Data Mining and Knowledge Discovery*, **38**(5). doi:10.1007/s10618-023-00930-y.

Carreira-Perpiñán MÁ, Hada SS (2021). "Counterfactual Explanations for Oblique Decision Trees: Exact, Efficient Algorithms." *Proceedings of the AAAI Conference on Artificial Intelligence*, **35**(8), 6903–6911. doi:10.1609/aaai.v35i8.16851.

Chang W (2021). ***R6**: Encapsulated Classes with Reference Semantics*. doi:10.32614/CRAN.package.r6. R package version 2.5.1.

Dandl S, Hofheinz A, Binder M, Casalicchio G (2026). ***counterfactuals**: An R Package for Counterfactual Explanation Methods*. doi:10.32614/CRAN.package.counterfactuals. R package version 1.0.0.

Dandl S, Molnar C, Binder M (2020a). "**counterfactuals**: Counterfactual Explanations." Github repository. URL https://github.com/susanne-207/moc. Commit d2fa9e0918d157c5d46a822b4ef110e641b45b76.

Dandl S, Molnar C, Binder M, Bischl B (2020b). "Multi-Objective Counterfactual Explanations." In T Bäck, M Preuss, A Deutz, H Wang, C Doerr, M Emmerich, H Trautmann (eds.), *Parallel Problem Solving from Nature – PPSN XVI*, pp. 448–469. Springer-Verlag, Cham. doi:10.1007/978-3-030-58112-1_31.

de Oliveira RMB, Martens D (2021). "A Framework and Benchmarking Study for Counterfactual Generating Methods on Tabular Data." *Applied Sciences*, **11**(16). doi:10.3390/app11167274.

Deb K, Pratap A, Agarwal S, Meyarivan T (2002). "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." *IEEE Transactions on Evolutionary Computation*, **6**(2), 182–197. doi:10.1109/4235.996017.

Dowle M, Srinivasan A (2022). ***data.table**: Extension of 'data.frame'*. doi:10.32614/CRAN.package.data.table. R package version 1.16.0.

Dua D, Graff C (2017). "UCI Machine Learning Repository." URL http://archive.ics.uci.edu/ml.

Erickson N, Mueller J, Shirkov A, Zhang H, Larroy P, Li M, Smola A (2020). "AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data." *arXiv 2003.06505*, arXiv.org E-Print Archive. doi:10.48550/arxiv.2003.06505.

Fisher RA (1936). "The Use of Multiple Measurements in Taxonomic Problems." *Annals of Eugenics*, **7**(2), 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x.

Freiesleben T (2021). "The Intriguing Relation between Counterfactual Explanations and Adversarial Examples." *Minds and Machines*, **32**(1), 77–109. doi:10.1007/s11023-021-09580-9.

Goldstein A, Kapelner A, Bleich J, 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 JC (1971). "A General Coefficient of Similarity and Some of Its Properties." *Biometrics*, **27**(4), 857–871. doi:10.2307/2528823.

Grömping U (2019). "South German Credit Data: Correcting a Widely Used Data Set." *Report 04/2019*, Department II, Beuth University of Applied Sciences Berlin. URL http://www1.beuth-hochschule.de/FB_II/reports/Report-2019-004.pdf.

Guidotti R (2018). "**LORE**: LOcal Rule-Based Explanations." Github repository. URL https://github.com/riccotti/LORE. Commit 710ffb42bf764bae90e9295e14349f0250fc2628.

Guidotti R, Monreale A, Ruggieri S, Pedreschi D, Turini F, Giannotti F (2018). "Local Rule-Based Explanations of Black Box Decision Systems." *arXiv 1805.10820*, arXiv.org E-Print Archive. doi:10.48550/arxiv.1805.10820.

Hothorn T, Zeileis A (2021). "Predictive Distribution Modelling Using Transformation Forests." *Journal of Computational and Graphical Statistics*, **30**(4), 1181–1196. doi:10.1080/10618600.2021.1872581.

Karimi AH, Barthe G, Balle B, Valera I (2020). "Model-Agnostic Counterfactual Explanations for Consequential Decisions." In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 108 of *Proceedings of Machine Learning Research*, pp. 895–905. PMLR. URL https://proceedings.mlr.press/v108/karimi20a.html.

Karimi AH, Mohammadi K (2021). "**mace**: Model Agnostic Counterfactual Explanations." Github repository. URL https://github.com/amirhk/mace. Commit 01e6a405ff74e24dc3438a005cd60892154d189d.

Karimi AH, Schölkopf B, Valera I (2021). "Algorithmic Recourse: From Counterfactual Explanations to Interventions." In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pp. 353–362. doi:10.1145/3442188.3445899.

Kato S (2018). "**featureTweakR**: R Package for Actionable Feature Tweaking." Github repository. URL https://github.com/katokohaku/featureTweakR. Commit 6f3e614531fe2c9e475703afdc4deb8aaf62f78f.

Kingma DP, Ba J (2017). "Adam: A Method for Stochastic Optimization." *arXiv 1412.6980 v9*, arXiv.org E-Print Archive. doi:10.48550/arxiv.1412.6980.

König G, Freiesleben T, Grosse-Wentrup M (2023). "Improvement-Focused Causal Recourse (ICR)." *Proceedings of the AAAI Conference on Artificial Intelligence*, **37**(10), 11847–11855. doi:10.1609/aaai.v37i10.26398.

Kuhn M (2008). "Building Predictive Models in R Using the **caret** Package." *Journal of Statistical Software*, **28**(5), 1–26. doi:10.18637/jss.v028.i05.

Kuhn M, Wickham H (2020). ***tidymodels**: A Collection of Packages for Modeling and Machine Learning Using **tidyverse** Principles*. URL https://www.tidymodels.org/.

Lang M, Binder M, Richter J, Schratz P, Pfisterer F, Coors S, Au Q, Casalicchio G, Kotthoff L, Bischl B (2019). "**mlr3**: A Modern Object-Oriented Machine Learning Framework in R." *Journal of Open Source Software*, **4**(44). doi:10.21105/joss.01903.

Lang M, Bischl B, Richter J, Sun X, Binder M (2023). ***paradox**: Define and Work with Parameter Spaces for Complex Algorithms*. doi:10.32614/CRAN.package.paradox. R package version 1.0.1.

Lang M, Bischl B, Surmann D (2017). "**batchtools**: Tools for R to Work on Batch Systems." *Journal of Open Source Software*, **2**(10). doi:10.21105/joss.00135.

Lewis DK (1973). *Counterfactuals*. Blackwell, Malden.

Li R, Emmerich MTM, Eggermont J, Bäck T, Schütz M, Dijkstra J, Reiber JHC (2013). "Mixed Integer Evolution Strategies for Parameter Optimization." *Evolutionary Computation*, **21**(1), 29–64. doi:10.1162/evco_a_00059.

Liaw A, Wiener M (2002). "Classification and Regression by **randomForest**." *R News*, **2**(3), 18–22. URL https://journal.R-project.org/articles/RN-2002-022/.

López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016). "The **irace** Package: Iterated Racing for Automatic Algorithm Configuration." *Operations Research Perspectives*, **3**, 43–58. doi:10.1016/j.orp.2016.09.002.

Mahajan D, Tan C, Sharma A (2020). "Preserving Causal Constraints in Counterfactual Explanations for Machine Learning Classifiers." *arXiv 1912.03277 v3*, arXiv.org E-Print Archive. doi:10.48550/arxiv.1912.03277.

Makiyama K (2015). "**pforeach**: An Easy Way to Parallel Processing in R." Github repository. URL https://github.com/hoxo-m/pforeach. Commit c44f3bf651a4b2d5d5657bf8be3a94f93769871.

Molnar C (2022). *Interpretable Machine Learning*. 2nd edition. Lulu. URL https://christophm.github.io/interpretable-ml-book.

Molnar C, Bischl B, Casalicchio G (2018). "**iml**: An R Package for Interpretable Machine Learning." *Journal of Open Source Software*, **3**(26), 786. doi:10.21105/joss.00786.

Moreira C, Chou YL, Hsieh C, Ouyang C, Jorge J, Pereira JM (2022). "Benchmarking Counterfactual Algorithms for XAI: From White Box to Black Box." *arXiv 2203.02399v2*, arXiv.org E-Print Archive. doi:10.48550/arxiv.2203.02399.

Nierenberg DW, Stukel TA, Baron JA, Dain BJ, Greenberg ER, The Skin Cancer Prevention Study Group (1989). "Determinants of Plasma Levels of Beta-Carotene and Retinol." *American Journal of Epidemiology*, **130**(3), 511–521. doi:10.1093/oxfordjournals.aje.a115365.

Pawelczyk M, Bielawski S, Van den Heuvel J, Richter T, Kasneci G (2021). "**CARLA**: A Python Library to Benchmark Algorithmic Recourse and Counterfactual Explanation Algorithms." In J Vanschoren, S Yeung (eds.), *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks*. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/b53b3a3d6ab90ce0268229151c9bde11-Paper-round1.pdf.

Pearl J (2009). *Causality: Models, Reasoning and Inference*. 2nd edition. Cambridge University Press. doi:10.1017/cbo9780511803161.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011). "**scikit-learn**: Machine Learning in Python." *Journal of Machine Learning Research*, **12**, 2825–2830.

Pfisterer F, Poon J, Lang M (2022). "**mlr3keras**: **mlr3 Keras** Extension." Github repository. URL https://github.com/mlr-org/mlr3keras. Commit bad8434b7898b51b2143fc680594057c00dc7080.

R Core Team (2025). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. doi:10.32614/R.manuals. URL https://www.R-project.org/.

Schleich M, Geng Z, Zhang Y, Suciu D (2021). "GeCo: Quality Counterfactual Explanations in Real Time." *Proceedings of the VLDB Endowment*, **14**(9), 1681–1693. doi:10.14778/3461535.3461555.

Stasinopoulos M, Rigby B, De Bastiani F (2021). ***gamlss.data**: Data for Generalised Additive Models for Location Scale and Shape*. doi:10.32614/CRAN.package.gamlss.data. R package version 6.0.2.

Stepin I, Alonso JM, Catala A, Pereira-Fariña M (2021). "A Survey of Contrastive and Counterfactual Explanation Generation Methods for Explainable Artificial Intelligence." *IEEE Access*, **9**, 11974–12001. doi:10.1109/access.2021.3051315.

Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R (2014). "Intriguing Properties of Neural Networks." *arXiv 1312.6199 v4*, arXiv.org E-Print Archive. doi:10.48550/arxiv.1312.6199.

Therneau T, Atkinson B (2019). ***rpart**: Recursive Partitioning and Regression Trees*. doi:10.32614/CRAN.package.rpart. R package version 4.1.23.

Todeschini A (2021). ***rchallenge**: A Simple Data Science Challenge System*. doi:10.32614/CRAN.package.rchallenge. R package version 1.3.4.

Tolomei G, Silvestri F, Haines A, Lalmas M (2017). "Interpretable Predictions of Tree-Based Ensembles via Actionable Feature Tweaking." In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 465–474. doi:10.1145/3097983.3098039.

Ustun B, Spangher A, Liu Y (2019). "Actionable Recourse in Linear Classification." In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pp. 10–19. doi:10.1145/3287560.3287566.

Van der Loo M (2022). ***gower**: Gower's Distance*. doi:10.32614/CRAN.package.gower. R package version 1.0.1.

Van Rossum G, Drake Jr FL (1995). "Python Tutorial." *Technical Report CS-R9526*, Centrum voor Wiskunde en Informatica.

Vanschoren J, Van Rijn JN, Bischl B, Torgo L (2014). "OpenML: Networked Science in Machine Learning." *SIGKDD Explorations Newsletter*, **15**(2), 49–60. doi:10.1145/2641190.2641198.

Verma S, Boonsanong V, Hoang M, Hines KE, Dickerson JP, Shah C (2022). "Counterfactual Explanations and Algorithmic Recourses for Machine Learning: A Review." *arXiv 2010.10596 v3*, arXiv.org E-Print Archive. doi:10.48550/arxiv.2010.10596.

Wachter S, Mittelstadt B, Russell C (2018). "Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR." *Harvard Journal of Law & Technology*, **31**(2), 841–887. doi:10.2139/ssrn.3063289.

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. doi:10.1109/tvcg.2019.2934619.

Wilson DR, Martinez TR (1997). "Improved Heterogeneous Distance Functions." *Journal of Artificial Intelligence Research*, **6**, 1–34. doi:10.1613/jair.346.

Xie L, Song Y, Lin T, Guo H, Wang B, Tang G, Liu C, Huang W, Yang Y, Ling W, Zhang Y, Li J, Huo Y, Wang X, Zhang H, Qin X, Xu X (2019). "Association of Plasma Retinol Levels with Incident Cancer Risk in Chinese Hypertensive Adults: A Nested Case-Control Study." *British Journal of Nutrition*, **122**(3), 293–300. doi:10.1017/s000711451900120x.

Zitzler E, Thiele L (1998). "Multiobjective Optimization Using Evolutionary Algorithms – A Comparative Case Study." In AE Eiben, T Bäck, M Schoenauer, HP Schwefel (eds.), *Parallel Problem Solving from Nature – PPSN V*, pp. 292–301. Springer-Verlag, Heidelberg. doi:10.1007/bfb0056872.

A. Algorithmic reference

Algorithm 1 MOC based on Dandl *et al.* (2020b) as implemented in the **counterfactuals** R package

Inputs:
Data point to explain prediction for $\mathbf{x}^\star \in \mathcal{X}$
Desired outcome (range) $Y' \subset \mathbb{R}$
Prediction function $\hat{f} : \mathcal{X} \to \mathbb{R}$
Observed data $\mathbf{X}$
Number of generations $n_{\text{generations}}$
Size of population μ
Recombination and mutation methods including probabilities
Selection method and initialization method
Stopping criterion
(Additional user inputs, e.g., range of numerical features, immutable features, distance function)

Initialize population P_0 with $|P_0| = \mu$
Evaluate candidates according to the four objectives of Equation 5
Set $t = 0$
while stopping criterion not met
    $C_t =$ `create_offspring`(P_t), $|C_t| = \mu$ by selecting, recombining and mutating parents with given probabilities
    Combine parents and offspring $R_t = C_t \cup P_t$
    Assign candidates to a front according to their objective values: $(F_1, F_2, \ldots, F_m) =$ `nondominated_sorting`(R_t)
    for $i = 1, \ldots, m$
        Sort candidates within a front with (tailored) crowding distance sorting: $\tilde{F}_i =$ `crowding_distance_sort`(F_i)
    end for
    Set $P_{t+1} = \emptyset$ and $i = 1$
    while $|P_{t+1}| + |\tilde{F}_i| \leq \mu$
        $P_{t+1} = P_{t+1} \cup \tilde{F}_i$
        i = i + 1
    end while
    Choose first $\mu - |P_{t+1}|$ elements of $\tilde{F}_i$: $P_{t+1} = P_{t+1} \cup \tilde{F}_i[1 : (\mu - |P_{t+1}|)]$
    $t = t + 1$
end while
Return unique, non-dominated candidates of $\bigcup_{k=0}^{t} P_k \setminus \mathbf{x}^\star$

Algorithm 2 NICE based on Brughmans *et al.* (2023) as implemented in the **counterfactuals** R package

Inputs:
Data point to explain prediction for $\mathbf{x}^\star \in \mathcal{X}$
Desired outcome (range) $Y' \subset \mathbb{R}$
Prediction function $\hat{f} : \mathcal{X} \to \mathbb{R}$
Observed data $\mathbf{X}$
Reward function R_O, $O \in$\{sparsity, proximity, plausibility\}
Indicator whether multiple counterfactuals should be returned *return_multiple*
Indicator whether to terminate as soon as desired prediction is reached $finish\_early$
(Additional user inputs, e.g., distance function)

Find closest observed data point $\mathbf{x}_{nn} \in \mathbf{X}$ to $\mathbf{x}^\star$ with desired prediction (Equation 8)
Set $\mathbf{x}^{best} = \mathbf{x}^\star$
Initialize archive set $A = \emptyset$
Set $J = \{j \in \{1, \ldots, p\} : x_{nn,j} \neq x_j^{best}\}$
while $(\hat{f}(\mathbf{x}^{best}) \notin Y'$ & $finish\_early ==$ TRUE$) \mid (J \neq \emptyset)$
    $j^{best} = \emptyset$
    for $j \in J$:
        $\mathbf{x} = \mathbf{x}^{best}$
        Create new candidate by replacing one feature: $x_j = x_{nn,j}$
        if $R_O(\mathbf{x}) > R_O(\mathbf{x}^{best})$: $\mathbf{x}^{best} = \mathbf{x}$ and $j^{best} = j$
        Save created candidate in an archive: $A = A \cup \mathbf{x}$
    end for
    Update $J = J \setminus j^{best}$
end while
if *return_multiple*: return $\{\mathbf{a} \in A : \hat{f}(\mathbf{a}) \in Y'\}$
else return $\mathbf{x}^{best}$

B. The counterfactuals R package

B.1. Class diagram

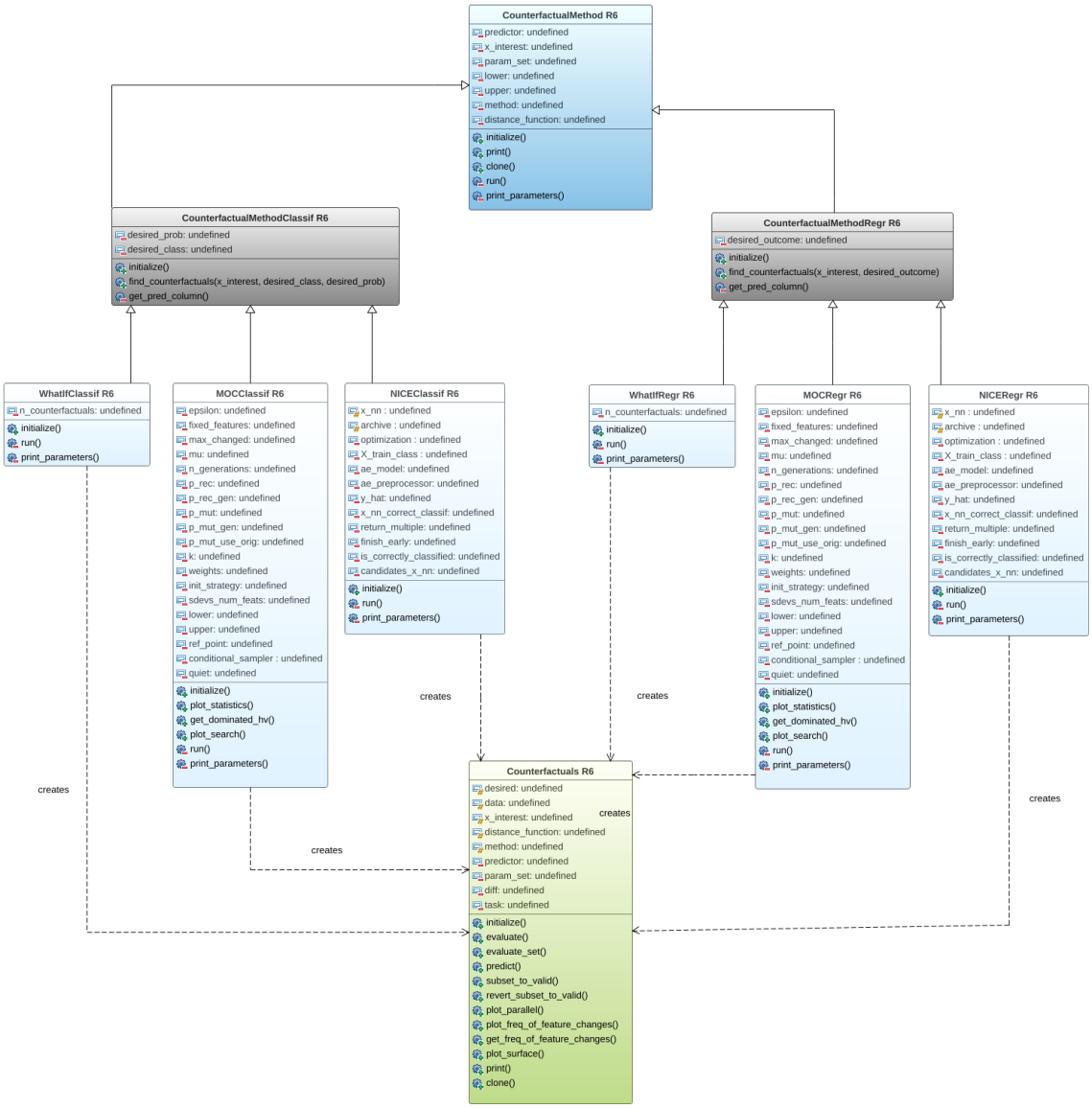


Figure 13: Detailed class diagram of the **counterfactuals** package.

B.2. Default values

The default parameter settings of the implementations of WhatIf and NICE should mimic the originally proposed methods in the corresponding papers (Wexler *et al.* 2019; Brughmans *et al.* 2023). Our MOC implementation has the same parameters as the original MOC implementation proposed in (Dandl *et al.* 2020a) except for `p_rec_use_orig`. Instead of resetting after

recombination *and* after mutation, we simplify things and reset only once after mutation with a probability of `p_mut_use_orig`. Due to the change in the dependency packages (**paradox** and **miesmuschel**, see Section 2.1), we re-tuned the MOC hyperparameters using the iterated F-race described in Dandl *et al.* (2020b) (see Appendix B). The code for tuning can be found here: `https://github.com/dandls/moc/tree/irace_newversion`. Although tuning identified the usage of the conditional mutator as a successor, we set `use_conditional_mutator` to `FALSE`, since it increases the runtime considerably.

Name	Description	Default
`n_counterfactuals`	The number of counterfactuals to be found.	1
`lower`	Vector of minimum values for numeric features named with the corresponding feature names. If `NULL`, the element for a numeric feature in `lower` is taken as its minimum value in observed data.	`NULL`
`upper`	Vector of maximum values for numeric features named with the corresponding feature names. If `NULL`, the element for a numeric feature in `upper` is taken as its maximum value in observed data.	`NULL`
`distance_function`	Distance function to compute the distances between the original and the training data points. 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; `"gower_c"` is a C-based more efficient version of Gower's distance. A function must have three arguments `x`, `y`, and `data`, and must return a numeric matrix.	`"gower"`

Table 2: Parameters of WhatIf and their default values in the **counterfactuals** package.

Name	Description	Default
`epsilon`	If not `NULL`, candidates whose prediction is farther away from the desired interval than epsilon are penalized.	`NULL`
`fixed_features`	Names of features that are not allowed to be changed. `NULL` (default) allows all features to be changed.	`NULL`
`max_changed`	Maximum number of feature changes. `NULL` (default) allows any number of changes.	`NULL`
`mu`	The population size.	20
`n_generations`	The number of generations.	175
`p_rec`	Probability with which an individual is selected for recombination.	0.71

`p_rec_gen`	Probability with which a feature/gene is selected for recombination.	0.62
`p_mut`	Probability with which an individual is selected for mutation.	0.73
`p_mut_gen`	Probability with which a feature/gene is selected for mutation.	0.5
`p_mut_use_orig`	Probability with which a feature/gene is reset to its original value in x_interest after mutation.	0.4
`k`	The number of data points to use for the fourth objective (Equation 4).	1
`weights`	The weights used to compute the weighted sum of dissimilarities for the fourth objective. It is either a single value or a vector of length `k` summing up to 1 (one weight for each of the `k` closest points). `NULL` (default) means all data points are weighted equally.	`NULL`
`lower`	Vector of minimum values for numeric features named with the corresponding feature names. If `NULL`, the element for a numeric feature in lower is taken as its minimum value in observed data.	`NULL`
`upper`	Vector of maximum values for numeric features named with the corresponding feature names. If `NULL`, the element for a numeric feature in upper is taken as its maximum value in observed data.	`NULL`
`init_strategy`	The population initialization strategy. Can be `"random"`, `"sd"`, `"traindata"` or `"icecurve"`.	`"icecurve"`
`use_conditional_mutator`	Should a conditional mutator be used? The conditional mutator generates plausible feature values based on the values of the other features.	`FALSE`
`distance_function`	Distance function for the second and fourth objectives. 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; `"gower_c"` is a C-based more efficient version of Gower's distance. A function must have three arguments `x`, `y`, and `data`, and must return a numeric matrix.	`"gower"`

Table 3: Parameters of MOC and their default values in the **counterfactuals** package.

Name	Description	Default
`optimization`	The reward function to optimize. Can be `"sparsity"` (default), `"proximity"`, or `"plausibility"`.	`"sparsity"`
`x_nn_correct`	Should only correctly predicted observations be considered for the most similar instance search?	TRUE
`margin_correct`	Only for regression models. 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 observed data.	`NULL`
`return_multiple`	Should multiple counterfactuals be returned? If `TRUE`, the algorithm returns all created instances whose prediction is in the desired interval.	`FALSE`
`finish_early`	Should the algorithm terminate after an iteration in which the prediction for the highest reward instance is in the desired interval. If `FALSE`, the algorithm continues until `x_nn` is recreated.	TRUE
`distance_function`	Distance function for computing the distances between the original 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; `"gower_c"` is a C-based more efficient version of Gower's distance. A function must have three arguments `x`, `y`, and `data`, and must return a numeric matrix.	`"gower"`

Table 4: Parameters of NICE and their default values in the **counterfactuals** package.

B.3. Different machine learning interfaces

The **counterfactuals** R package only allows machine learning models as an input that are instances of an `iml::Predictor` object. The `Predictor` class encapsulates a fitted model together with its underlying (training) data. In Section 4, we saw that it works off-the-shelf with models fitted with the **randomForest** and **mlr3** R packages (Liaw and Wiener 2002; Lang *et al.* 2019). In this section, we generate counterfactuals for the plasma retinol example of Section 4.2 for models trained with the **caret**, **tidymodels** and **mlr** packages (Kuhn 2008; Kuhn and Wickham 2020; Bischl *et al.* 2016). While all these machine learning interfaces allow training of a variety of models (linear models, model ensembles, etc.), for illustration, we focus on regression trees. Trees are fitted internally with **rpart** (Therneau and Atkinson 2019), such that – for the sake of completeness – we also show how to generate counterfactuals

for a **rpart** tree. For each tree, we generate a counterfactual for the 100th row of the plasma dataset using the NICE method. The counterfactual should propose changes such that for the observation a plasma concentration larger than 500 ng/ml is predicted.

```
R> data("plasma", package = "gamlss.data")
R> x_interest <- plasma[100L,]
```

caret *package*

First, we fit a regression tree model with the help of **caret**. To avoid tuning of the tree, we manually set the only tuning parameter `cp` to 0.01 – the default of the **rpart** package. Then, we initialize an `iml::Predictor` object with the fitted model as an input.

```
R> library("caret")
R> treecaret <- caret::train(retplasma ~ ., data = plasma[-100L,],
+    method = "rpart", tuneGrid = data.frame(cp = 0.01))
R> predcaret <- Predictor$new(model = treecaret, data = plasma[-100L,],
+    y = "retplasma")
R> predcaret$predict(x_interest)

  .prediction
1    342.9231
```

For the 100th row of the plasma dataset (our `x_interest` or $\mathbf{x}^\star$), we predict a median value of 342.92 – the same as in Section 4.2. Next, we generate counterfactuals by initializing a `NICERegr` object with the instantiated `Predictor`.

```
R> nicecaret <- NICERegr$new(predcaret, optimization = "proximity",
+    margin_correct = 0.5, return_multiple = FALSE)
R> nicecaret$find_counterfactuals(x_interest,
+    desired_outcome = c(500, Inf))

1 Counterfactual(s)

Desired outcome range: [500, Inf]

Head:
   age sex smokstat   bmi vituse calories   fat fiber alcohol cholesterol
1:  46   1        3 35.26      3   2667.5 131.6  10.1       0       550.5
   betadiet retdiet betaplasma
1:     1210    1291        218
```

Since for all the examples shown in this section, we internally fit a `rpart` model to the same data, the prediction and the counterfactual for `x_interest` will be the same. We, therefore, omit the outputs for the prediction and counterfactual for the following machine learning interfaces.

tidymodels *package*

Regression trees of the **tidymodels** package also work off-the-shelf. However, for classification models, the `iml::Predictor` requires a prediction wrapper function (`predict.function`) such that class probabilities are returned instead of class labels. For details, the corresponding help page should be consulted.

```
R> library("tidymodels")
R> treetm <- decision_tree(mode = "regression", engine = "rpart") %>%
     fit(retplasma ~ ., data = plasma[-100L,])
R> predtm <- Predictor$new(model = treetm, data = plasma[-100L,],
+    y = "retplasma")
R> predtm$predict(x_interest)
R> nicetm <- NICERegr$new(predtm, optimization = "proximity",
+    margin_correct = 0.5, return_multiple = FALSE)
R> nicetm$find_counterfactuals(x_interest = x_interest,
+    desired_outcome = c(500, Inf))
```

mlr *package*

For the **mlr** package, the workflow to generate counterfactuals is similar to the one for the **caret** package. We only need `mlr::RegrTask` and `mlr::regr.rpart` objects.

```
R> library("mlr")
R> task <- mlr::makeRegrTask(data = plasma[-100L,], target = "retplasma")
R> mod <- mlr::makeLearner("regr.rpart")
R> treemlr <- mlr::train(mod, task)
R> predmlr <- Predictor$new(model = treemlr, data = plasma[-100L,],
+    y = "retplasma")
R> predmlr$predict(x_interest)
R> nicemlr <- NICERegr$new(predmlr, optimization = "proximity",
+    margin_correct = 0.5, return_multiple = FALSE)
R> nicemlr$find_counterfactuals(x_interest = x_interest,
+    desired_outcome = c(500, Inf))
```

rpart *package*

For sake of completeness, we also show how to generate counterfactuals for a regression model directly fitted with the **rpart** package.

```
R> library("rpart")
R> treerpart <- rpart(retplasma ~ ., data = plasma[-100L,])
R> predrpart <- Predictor$new(model = treerpart, data = plasma[-100L,],
+    y = "retplasma")
R> predrpart$predict(x_interest)
R> nicerpart <- NICERegr$new(predrpart, optimization = "proximity",
+    margin_correct = 0.5, return_multiple = FALSE)
R> nicerpart$find_counterfactuals(x_interest = x_interest,
+    desired_outcome = c(500, Inf))
```

B.4. Class `FeatureTweakerClassif`

```
R> FeatureTweakerClassif <- R6Class("FeatureTweakerClassif",
+    inherit = CounterfactualMethodClassif,
+
+    public = list(
+      initialize = function(predictor, ktree = NULL, epsiron = 0.1,
+        resample = FALSE) {
+        super$initialize(predictor)
+        private$ktree = ktree
+        private$epsiron = epsiron
+        private$resample = resample
+      }
+    ),
+
+    private = list(
+      ktree = NULL,
+      epsiron = NULL,
+      resample = NULL,
+
+      run = function() {
+        predictor = private$predictor
+        y_hat_interest = predictor$predict(private$x_interest)
+        class_x_interest = names(y_hat_interest)[which.max(y_hat_interest)]
+        rf = predictor$model
+
+        rules = getRules(rf, ktree = private$ktree,
+          resample = private$resample)
+        es = set.eSatisfactory(rules, epsiron = private$epsiron)
+        tweaks = featureTweakR::tweak(
+          es, rf, private$x_interest, label.from = class_x_interest,
+          label.to = private$desired_class, .dopar = FALSE
+        )
+        return(tweaks$suggest)
+      },
+
+      print_parameters = function() {
+        cat(" - epsiron: ", private$epsiron, "\n")
+        cat(" - ktree: ", private$ktree, "\n")
+        cat(" - resample: ", private$resample)
+      }
+    )
+  )
```

C. Benchmarking

C.1. Hyperparameter tuning

For hyperparameter tuning, we used random search (with 30 evaluations) and 5-fold CV with the misclassification error as a performance measure. Table 5 shows the tuning search space of each model. Numerical features were standardized and categorical ones were one-hot encoded using the **mlr3pipelines** package (Binder, Pfisterer, Lang, Schneider, Kotthoff, and Bischl 2021). The optimizer for the neural network was ADAM (Kingma and Ba 2017), and early stopping was imposed after 5 patience steps. All other hyperparameters were set to their default values in the packages of the mlr3 ecosystem (Lang *et al.* 2019). For the `hill_valley` dataset we used the default deep and wide architecture (two layers) inspired by Erickson *et al.* (2020) as implemented in the **mlr3keras** package without tuning (Pfisterer, Poon, and Lang 2022). Table 6 shows the accuracies of each model using nested resampling (with 5-fold CV in the inner and outer loop).

Model	Hyperparameter	Range
randomForest	ntrees	[0, 1000]
xgboost	nrounds	[0, 1000]
svm	cost	[0.01, 1]
logreg	-	-
neuralnet	lr	[0.00001, 0.1]
	layer_size	[1, 20]

Table 5: Tuning search space of each model. Hyperparameters `ntrees` and `nrounds` were log-transformed.

dataset	logistic_regression	neural_network	ranger	svm	xgboost
`credit_g`	0.72	0.71	0.71	0.73	0.70
`diabetes`	0.75	0.72	0.75	0.73	0.72
`tic_tac_toe`	0.97	0.98	0.95	0.79	0.98
`bank8FM`	0.94	0.94	0.94	0.95	0.94
`hill_valley`	0.60	0.53	0.56	0.48	0.57
`run_or_walk_info`	0.72	0.91	0.99	0.96	0.99

Table 6: Classification accuracies of each model on each dataset. The accuracies were computed using nested resampling with 5-fold CV in the inner and outer loop.

C.2. Additional results

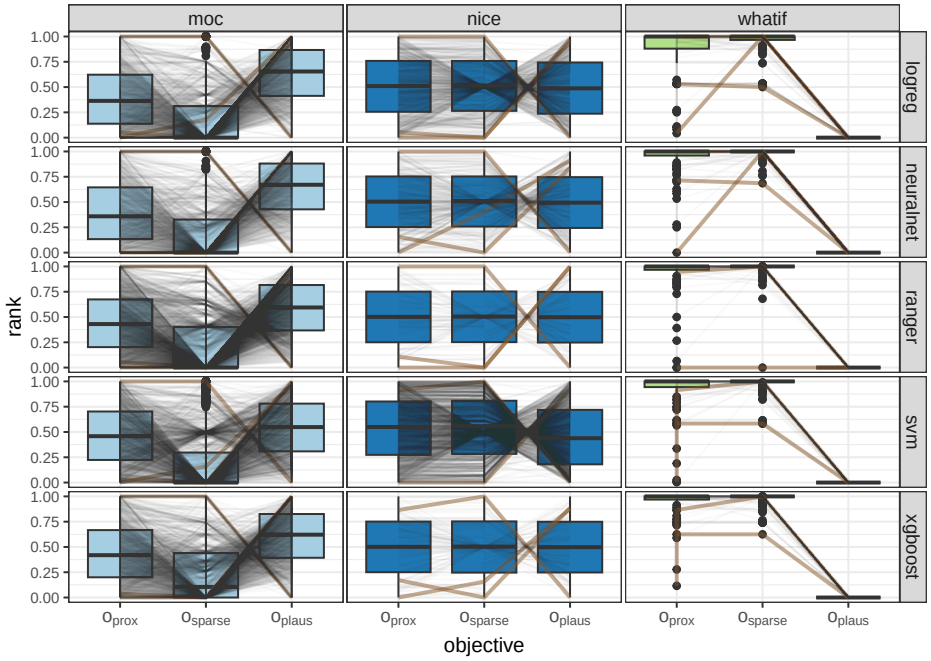
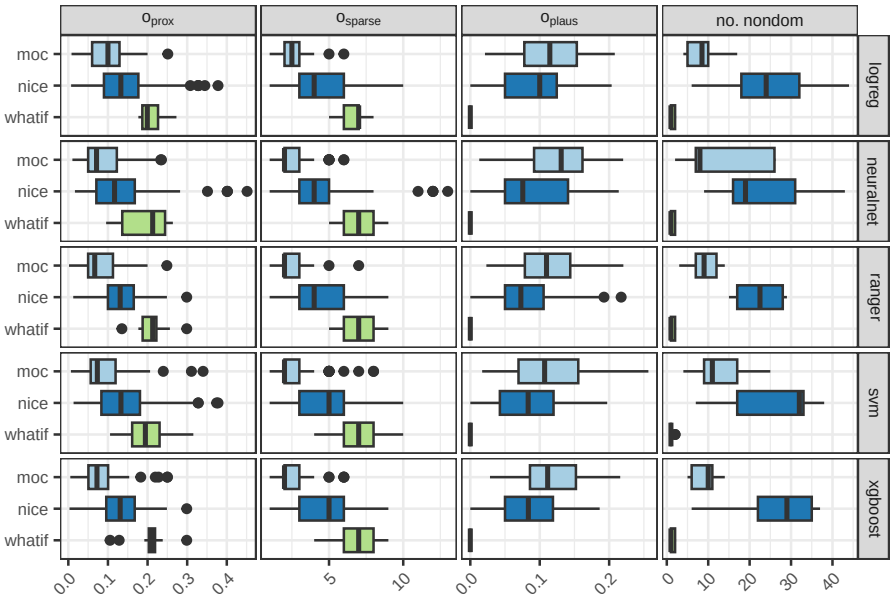
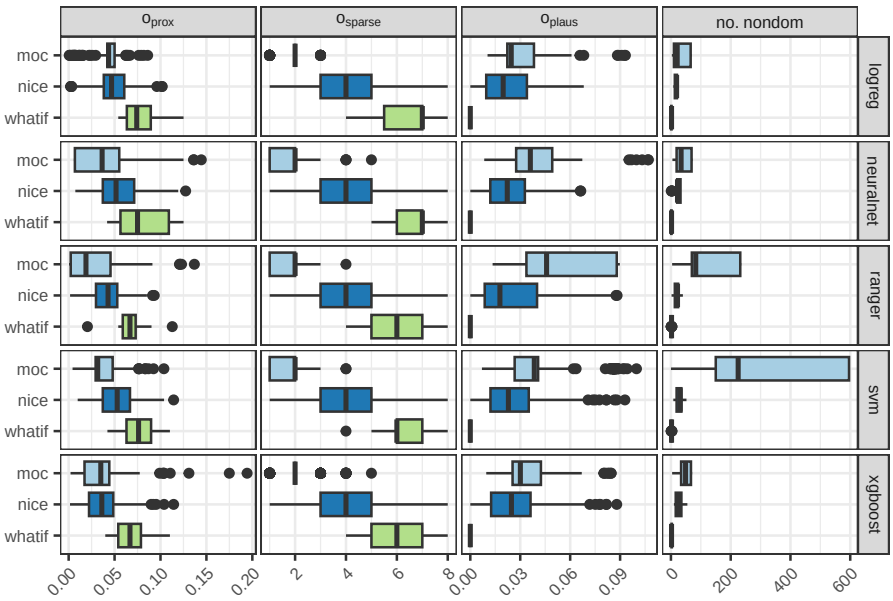


Figure 14: Comparison of NICE, WhatIf, and MOC w.r.t. their rank in the properties proximity (ii, o_{prox}), sparsity (iii, o_{sparse}) and plausibility (iv, o_{plaus}). Each gray line reflects a counterfactual (for clarity purposes, only a maximum of 2000 counterfactuals are displayed). The counterfactuals with the lowest and therefore best rank in an objective display the brown lines. Lower values are better.

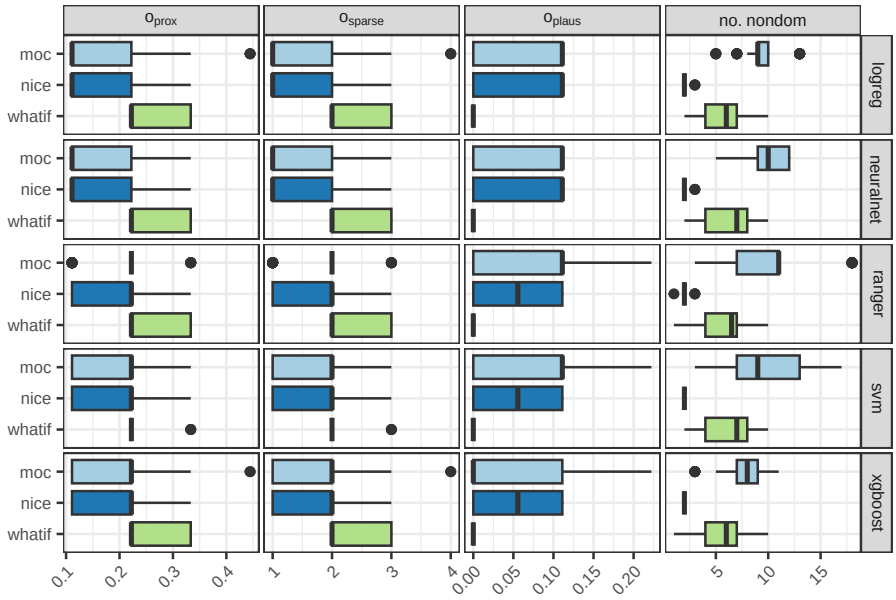


(a) `credit_g`

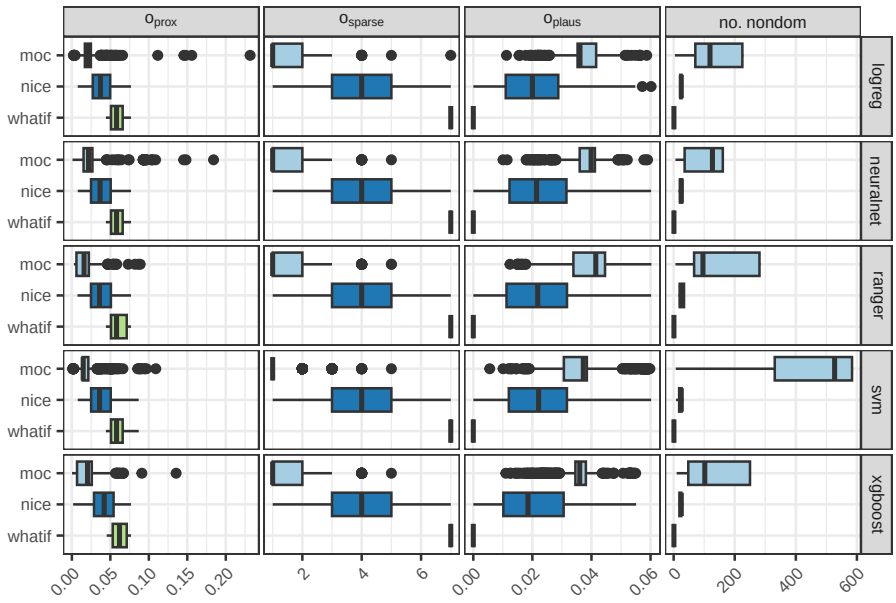


(b) `diabetes`

Figure 15: Comparison of NICE, WhatIf, and MOC w.r.t. the measures `dist_x_interest`, `no_changed`, `dist_train` (explained in Section 4), and no. nondom (number of non-dominated counterfactuals) for several models for the datasets `credit_g` and `diabetes`. o_{valid} was 0 for all counterfactuals. Lower values are better, except for no. nondom. The figure is based on Dandl *et al.* (2020b).

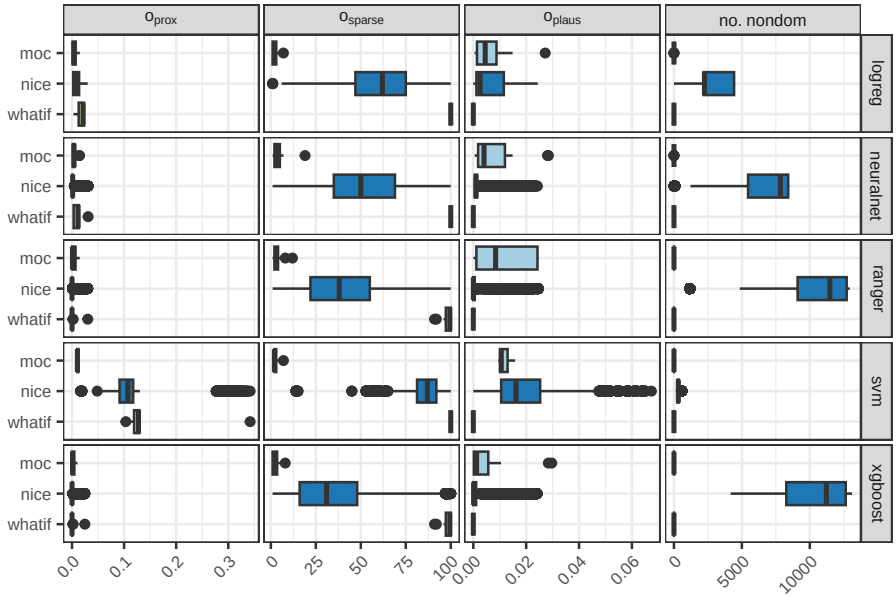


(a) `tic_tac_toe`

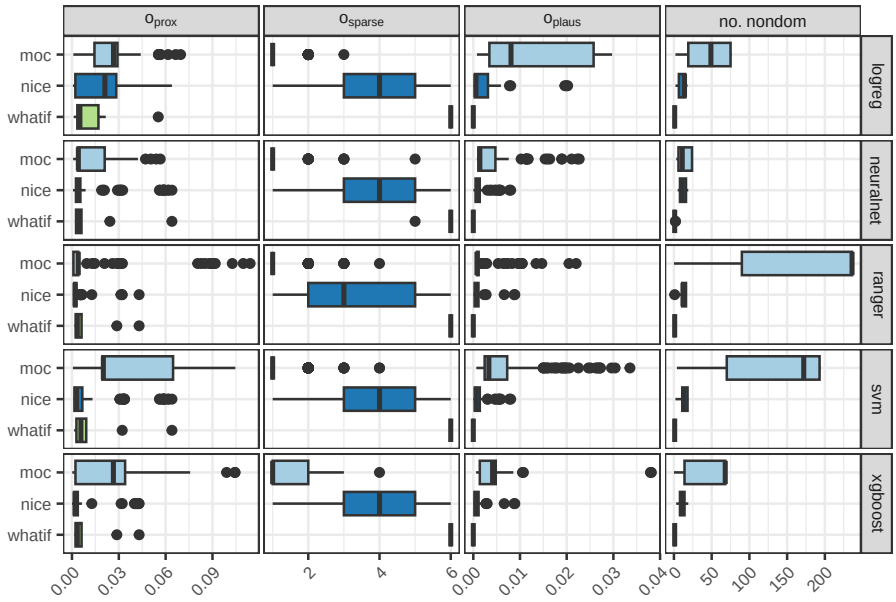


(b) `bank8FM`

Figure 16: Comparison of NICE, WhatIf, and MOC w.r.t. the measures `dist_x_interest`, `no_changed`, `dist_train` (explained in Section 4), and no. nondom (number of non-dominated counterfactuals) for several models for the datasets `tic_tac_toe` and `bank8FM`. o_{valid} was 0 for all counterfactuals. Lower values are better, except for no. nondom. The figure is based on Dandl *et al.* (2020b).



(a) `hill_valley`



(b) `run_or_walk_info`

Figure 17: Comparison of NICE, WhatIf, and MOC w.r.t. the measures `dist_x_interest`, `no_changed`, `dist_train` (explained in Section 4), and no. nondom (number of non-dominated counterfactuals) for several models for the datasets `hill_valley` and `run_or_walk_info`. o_{valid} was 0 for all counterfactuals. Lower values are better, except for no. nondom. The figure is based on Dandl *et al.* (2020b).

Affiliation:

Giuseppe Casalicchio
Department of Statistics
Ludwig-Maximilians-Universität München, Germany
Ludwigstr. 33, 80539 Munich, Germany
Munich Center for Machine Learning (MCML), Germany
E-mail: Giuseppe.Casalicchio@stat.uni-muenchen.de

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
December 2025, Volume 115, Issue 9 *Submitted:* 2023-06-29
doi:10.18637/jss.v115.i09 *Accepted:* 2024-10-17