MCCE: Monte Carlo sampling of valid and realistic counterfactual explanations for tabular data

Annabelle Redelmeier, Martin Jullum, Kjersti Aas and Anders Løland

Norwegian Computing Center, P.O. Box 114, Oslo, NO-0314, Norway.

*Corresponding author(s). E-mail(s): martin.jullum@nr.no;
Contributing authors: aredelmeier@gmail.com; kjersti.aas@nr.no; anders.loland@nr.no;

Abstract

We introduce MCCE: **M**onte **C**arlo sampling of valid and realistic **C**ounterfactual **E**xplanations for tabular data, a novel counterfactual explanation method that generates on-manifold, actionable and valid counterfactuals by modeling the joint distribution of the mutable features given the immutable features and the decision. Unlike other on-manifold methods that tend to rely on variational autoencoders and have strict prediction model and data requirements, MCCE handles any type of prediction model and categorical features with more than two levels. MCCE first models the joint distribution of the features and the decision with an autoregressive generative model where the conditionals are estimated using decision trees. Then, it samples a large set of observations from this model, and finally, it removes the samples that do not obey certain criteria. We compare MCCE with a range of state-of-the-art on-manifold counterfactual methods using four well-known data sets and show that MCCE outperforms these methods on all common performance metrics and speed. In particular, including the decision in the modeling process improves the efficiency of the method substantially.

Keywords: explainable AI, counterfactual explanations, Gower distance, conditional distribution

1 Introduction

It is exceedingly apparent that complex, black-box AI, machine learning, and statistical models deployed in industry need sound and efficient explanations. In this paper, we discuss counterfactual explanations, a type of local prediction explanation method. Counterfactual explanations (CEs)[1] explain predictions, $f(\boldsymbol{x})$, from a fitted prediction model $f(\cdot)$ or some other deterministic ML/AI system. It does so by providing examples that yield a different *decision* than the feature vector, $\boldsymbol{x}$, one is trying to explain[2]. These examples, $\tilde{\boldsymbol{x}}$, are called *counterfactuals*, because they give an idea of what features could be changed to obtain a different decision. Generating a feature vector with a different decision is sometimes referred to as generating *recourse* (Karimi et al, 2022; Guidotti, 2022).

As a simple example, imagine that a bank utilizes a black-box machine learning model to decide whether or not an individual should receive a loan. The model takes the four features `age`, `sex`, `salary`, and `defaulted_last_year` as inputs, and outputs the probability of defaulting on the loan $\in [0, 1]$. If the probability is in the desired decision interval $c = [0, 0.1)$, the bank customer is granted the loan. Now suppose that a 36-year-old female customer with a salary of $68,000 who defaulted on a loan last year, is denied a loan. To *explain* to the customer why her loan was denied, a set of counterfactuals could be provided. One counterfactual may show that if she had a `salary` of $80,000 rather than $68,000 she would have received the loan. Another may show that only increasing her salary to $72,000, but without having defaulted on her loan last year would also make her eligible for the loan.

For counterfactuals to be useful, they should be of low *cost*, i.e., limit the *number* and *magnitude* of feature changes. Low-cost counterfactuals are easier for individuals to implement and/or case handlers to understand. Counterfactuals should also be *actionable*, i.e., not change features that are fixed by the individual. Otherwise, they will be impossible to implement in practice. Further, the counterfactuals should lie on the data manifold, since this ensures a realistic and plausible combination of features. Finally, counterfactuals should be *valid*, i.e., obtain the desired decision. Without this, the counterfactual would not be a *counter*factual at all.

1.1 Related work

Recently there has been an explosion of papers proposing counterfactual explainers. Surveys are given by e.g., Verma et al (2021), Stepin et al (2021) and Guidotti (2022). In the most recent survey (Guidotti, 2022), the different approaches are categorized according to (i) the strategy used to generate the counterfactuals, (ii) whether the approach is model-agnostic, (iii)

[1] We use 'counterfactual explanation' or 'CE' to refer to the literature or explanation type and 'counterfactual' or 'example' to refer to the instance produced.

[2] A *decision* is derived from a *prediction*, $f(\boldsymbol{x})$, using a pre-defined cutoff value or interval c, characterizing the desired decision. For example, if $f(\boldsymbol{x}) = 0.39$ and $c = (0.5, 1]$, then since $f(\boldsymbol{x}) \notin c$, we give instance $\boldsymbol{x}$ a decision of 0 and say $\boldsymbol{x}$ has received an undesirable decision.

whether the approach can be used for tabular data, text or images, (iv) whether the approach is able to handle categorical data, (v) whether the approach guarantees validity, and finally, (vi) whether the approach is able to ensure actionability. The survey shows that most counterfactual explanation approaches are tailor-made for either tabular data, text, or images. The strategy used to generate the counterfactuals is categorized into four groups: (i) optimization-based, (ii) heuristic search-based, (iii) decision-tree-based, and (iv) instance-based. The majority of the approaches belong to the first two categories. In what follows we describe and briefly review the four groups of approaches.

The optimization-based approaches usually first define a loss function of the form

$$\text{dist}_1(f(\tilde{\boldsymbol{x}}), y') + \lambda \cdot \text{dist}_2(\boldsymbol{x}, \tilde{\boldsymbol{x}}), \tag{1}$$

where $f(\cdot)$ is a prediction model, $\boldsymbol{x}$ is the original feature vector, $\tilde{\boldsymbol{x}}$ is the counterfactual, λ is some tuning parameter, $y' = \mathbb{1}(f(\tilde{\boldsymbol{x}}) \in c)$ is an indicator for the desired decision (see footnote 2 on page 2), and $\text{dist}_1(\cdot, \cdot)$ and $\text{dist}_2(\cdot, \cdot)$ are two appropriate distance functions (e.g., weighted L_0, L_1, L_2 norms or the median absolute deviation). The counterfactual $\tilde{\boldsymbol{x}}$ is then found by e.g., stochastic gradient descent (Wachter et al, 2017), integer programming (Ustun et al, 2019), random walks (Laugel et al, 2018), genetic algorithms (Dandl et al, 2020; Rasouli and Chieh Yu, 2022), or through a sequence of satisfiability (SAT) problems (Karimi et al, 2020). The optimization-based approaches often restrict $f(\cdot)$ to be differentiable, meaning that they do not work for tree-based classifiers like XGBoost (Chen and Guestrin, 2016) or random forest (Breiman et al, 1984). Further, these methods usually cannot guarantee that the counterfactual will be actionable, and most of them do not handle categorical features with more than two levels. Finally, most of the optimization-based methods produce counterfactuals that do not lie on the data manifold (Pawelczyk et al, 2021). There are, however, some exceptions. The methods `CEM-VAE` (Dhurandhar et al, 2018), `REViSE` (Joshi et al, 2019), `CLUE` (Antorán et al, 2021), `C-CHVAE` (Pawelczyk et al, 2020), and `CRUDS` (Downs et al, 2020) all use variational autoencoders (VAE) to find counterfactuals that are proximal and connected to input data.

The approaches that use heuristic search strategies, of which two examples are `VICE` (Gomez et al, 2020) and `MOC` (Dandl et al, 2020), are quite similar to the optimization-based ones, but instead of using optimization to minimize a loss function, heuristic search strategies are used. Such strategies are usually much faster than optimization algorithms. The downside is that the solutions are not necessarily optimal. Moreover, the heuristic search-based methods have many of the same weaknesses as the optimization-based approaches listed above.

The decision-tree based methods approximate the behavior of the prediction model with a decision tree and then exploit the logic revealed by the tree for building counterfactual explanations. One example is `FT` (Tolomei et al,

2017). A weakness of this approach is that the logic learned by the surrogate decision tree need not be the same as that of the original prediction model.

The last category, instance-based approaches, retrieves counterfactuals by selecting the most similar examples from a (simulated) data set. The approach proposed in this paper belongs to this category. According to Guidotti (2022) the instance-based methods are the best performers with respect to all the properties benchmarked in their survey. Four instance-based methods are discussed in Guidotti (2022). Three of these methods, `CBCE` (Keane and Smyth, 2020), `NICE` (Brughmans et al, 2023), and `FACE` (Poyiadzi et al, 2020), do not treat immutable features in an appropriate way. The fourth approach, `NNCE` (Wexler et al, 2020), should not be used for privacy reasons, because it selects counterfactuals as actual rows in the training data set.

As shown above, every existing method presents one or more limitations. Given that the instance-based approaches seem to be the most productive, we believe there is a demand for a method of this kind that successfully addresses and resolves the identified shortcomings.

1.2 Our contribution

In this paper, we introduce MCCE (**M**onte **C**arlo sampling of valid and realistic **C**ounterfactual **E**xplanations for tabular data), a novel instance-based method that handles any type of prediction model and feature cardinality. MCCE consists of three main steps. In step 1, MCCE models the data distribution and the decision using an autoregressive generative model, e.g., MADE (Germain et al, 2015), with conditionals estimated using decision trees. In step 2, MCCE generates a large set of samples that respect the learned data manifold using the model from step 1. Finally, in step 3, MCCE obtains counterfactual explanations by extracting the low-cost and valid samples.

MCCE is different from previously proposed CE methods in that it:

1. Uses a simultaneous model for the features *and* the decision to ensure on-manifold and valid counterfactuals. It models the decision through an additional fixed binary feature $y' = \mathbb{1}(f(\boldsymbol{x}) \in c)$, for a desired decision interval c (see footnote 2 on page 2). When generating samples in step 2, MCCE conditions on $y' = 1$, which guides the samples to the correct decision. This increases the efficiency of the method substantially. To the best of our knowledge, this trick has not been exploited by CE methods before.
2. Generates examples that are guaranteed to obey the immutable features (e.g., age, sex), i.e., the samples are actionable. This guarantee stems from the way the model in step 1 is specified.
3. Uses decision trees to estimate the data distribution. This means that MCCE does not require the underlying prediction model to be a gradient-based classifier like CE methods that use VAEs.
4. Handles categorical features with an arbitrary number of levels.
5. Breaks up the task into independent steps that can easily be altered to target specific needs.

The rest of the paper is organized as follows. In Section 2, we describe our method, MCCE, in detail. In Section 3, we compare the counterfactual explanations from MCCE with those of various competing methods and show how MCCE surpasses all other methods when it comes to speed and performance metrics. We also investigate how sensitive MCCE is to its main tuning parameter, study the characteristics of the data generated in the second step of the method, and examine the scalability of the different steps of the MCCE procedure. Section 4 examines the privacy implications of MCCE before we conclude in Section 5.

2 Generating counterfactual explanations with MCCE

We now present MCCE, our algorithm for generating counterfactuals. Suppose that we have a tabular data set $\boldsymbol{X} \in \mathcal{X}$ with dimension $n_{\text{train}} \times p$ and binary response $\boldsymbol{Y} \in \mathcal{Y}$. Suppose further that we have a prediction model $f(\cdot)$, a set of predictions, $f(\boldsymbol{x}_i)$, and a set of observations H that have obtained an undesirable decision, i.e., $H := \{i : f(\boldsymbol{x}_i) \notin c\}$ for decision interval c, that we wish to explain. Note that given $\boldsymbol{x} \in \mathcal{X}$, $f(\boldsymbol{x})$ is deterministic, i.e., always produces the same prediction. We divide the feature space, $\mathcal{X}$, into mutable features $\mathcal{X}^m$ and immutable features $\mathcal{X}^f$ such that $\mathcal{X} = \mathcal{X}^m \times \mathcal{X}^f$. Then, for each observation $h \in H$ with factual $\boldsymbol{x}_h$, our aim is to create a set of J counterfactuals $\boldsymbol{E}_h = \{\boldsymbol{e}_{h,1}, \ldots, \boldsymbol{e}_{h,J}\}$, such that:

- **Criterion 1**: $\boldsymbol{e}_{h,j}$ is *on-manifold*, i.e., $p(\boldsymbol{X}^m = \boldsymbol{e}^m_{h,j} \mid \boldsymbol{X}^f = \boldsymbol{e}^f_{h,j}) > \epsilon$, for some $\epsilon > 0$;
- **Criterion 2**: $\boldsymbol{e}_{h,j}$ is *actionable*, i.e., does not violate any of the immutable features;
- **Criterion 3**: $\boldsymbol{e}_{h,j}$ is *valid*, i.e., $f(\boldsymbol{e}_{h,j}) \in c$, for the desired decision interval c;
- **Criterion 4**: $\boldsymbol{e}_{h,j}$ is *low cost*, i.e., close to the factual, $\boldsymbol{x}_h$.

Given a training data set $\boldsymbol{X}$ and these criteria, we propose to generate a set of counterfactuals with three steps:

1. Model the distribution of the mutable features given the immutable features and the decision, y'. In our implementation, we use an autoregressive generative model based on decision trees.
2. For each observation $h \in H$, generate a large data set of dimension $K \times (p + 1)$ by sampling from this fitted distribution. Denote this data set $\boldsymbol{D}_h$ for each $h \in H$.
3. Create $\boldsymbol{E}_h$ for each $h \in H$ by removing the rows in $\boldsymbol{D}_h$ that do not obey criteria 3 and 4. Return $\boldsymbol{E}_h \subseteq \boldsymbol{D}_h$ as the set of counterfactual explanations for this instance.

Figure 1 shows an outline of the three steps, and Sections 2.1-2.3 describe them in more detail.

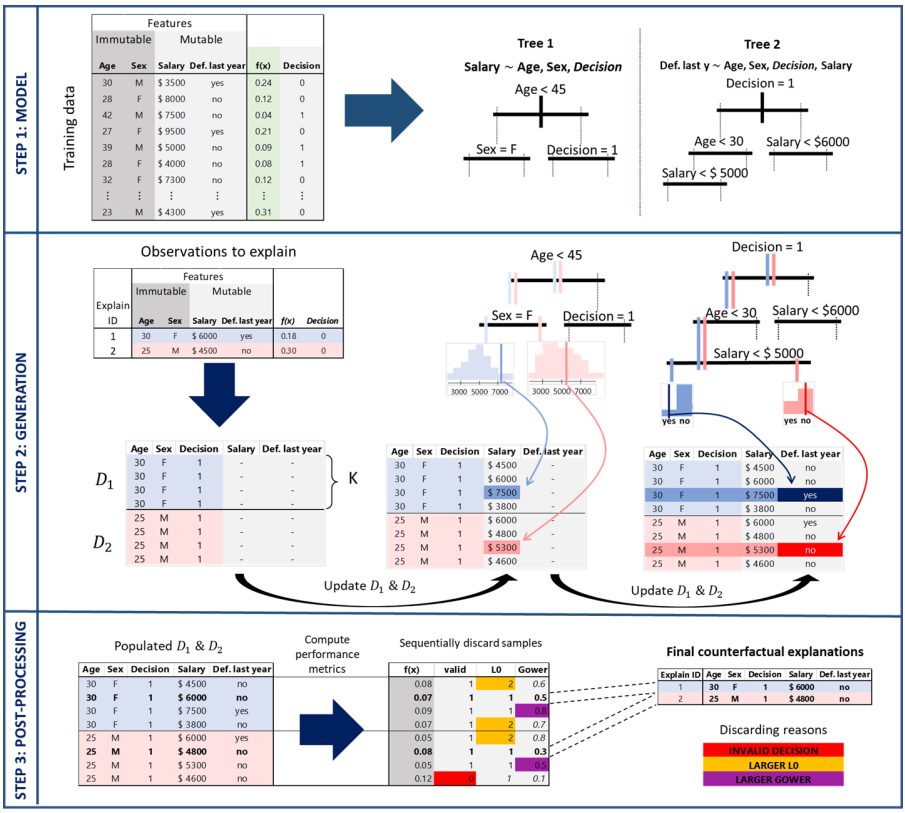

Fig. 1 MCCE's three steps illustrated on the introductory example with two immutable and two mutable features. Step 1: Fit a decision tree for each mutable feature iteratively on the previously fit features and the decision. Step 2: For each observation/prediction to explain, trace the trees down based on the immutable feature values, decision = 1, and previous sampled values, and then randomly sample a training observation in the leaf nodes and update $\boldsymbol{D}_h$. Step 3: Among the sampled rows, find the instance closest to the original vector over the decision boundary.

2.1 Model

Step 1 (*model*) To model the distribution of the q mutable features, $\boldsymbol{X}^m = \{X_1^m, \ldots, X_q^m\}$, given the u immutable features, $\boldsymbol{X}^f = \{X_1^f, \ldots, X_u^f\}$, and decision, Y', we use an autoregressive generative model to decompose $\boldsymbol{X}^m \mid \boldsymbol{X}^f, Y'$ into products of conditional probability distributions as follows

$$p(\boldsymbol{X}^m \mid \boldsymbol{X}^f, Y') = p(X_1^m \mid \boldsymbol{X}^f, Y') \prod_{i=2}^{q} p(X_i^m \mid \boldsymbol{X}^f, Y', X_1^m, \ldots, X_{i-1}^m). \quad (2)$$

Then, we estimate each of the univariate conditional distributions in (2) separately. The main reason for rewriting $p(\boldsymbol{X}^m \mid \boldsymbol{X}^f, Y')$ in this way is that

it is typically much easier to correctly model distributions with only a single dependent variable, rather than many.

The question is then how to model each of these conditional distributions. Options include the use of basic parametric probability distributions, copulas (Sklar, 1959), non-parametric methods, and deep learning approaches such as MADE (Germain et al, 2015). We choose to model the $q-1$ conditional distributions in (2) using $q-1$ decision trees. For example, to model $X_2^m \mid \boldsymbol{X}^f, Y', X_1^m$, we fit a decision tree to $X_2^m \sim (\boldsymbol{X}^f, Y', X_1^m)$ based on the data. We use decision trees because they are fast and robust to extrapolation, scale well, handle mixed (continuous and categorical) data automatically, handle arbitrary orders of interactions between the features, and do not require extra preprocessing or scaling of the data.

Inspired by Drechsler and Reiter (2011) and Reiter (2005), we choose to use CART (categorical and regression trees) (Breiman et al, 1984) to model these conditional distributions. CART is a tree algorithm that builds trees by recursively making binary splits on the feature space until a given stopping criterion is fulfilled. Depending on the class of the response, either the Gini impurity index or the squared-error loss can be used to measure the quality of the split. The tree can be controlled by setting other parameters like the maximum depth or number of features to check when splitting a node.

We choose CART over other non-parametric models since CART is consistent (Scornet et al, 2015; Chi et al, 2022) and allows for high dimensionality. In addition, CART has previously been used for synthetic data generation with good results. For example, Reiter (2005) uses CART to impute sensitive data and then compares the generated observations with the true observations. Reiter show that CART generates data that are close to the corresponding population statistics and covers the corresponding data intervals. Further, Drechsler and Reiter (2011) compare different non-parametric approaches to generating data. They show that CART is one of two methods that gives the best results when it comes to recreating population means and variances.

2.2 Generation

Step 2 (*generation*) Suppose that in step 1 we fit $q-1$ decision trees: $T_1, \ldots, T_{q-1}$ where T_i is fit to $X_i^m \sim (\boldsymbol{X}^f, Y', X_1^m, \ldots, X_{i-1}^m)$. Step 2 consists of generating $K \times p$ dimensional data sets, $\boldsymbol{D}_h$, $h \in H$, with samples from the fitted trees. The full generation procedure is outlined in Algorithm 1, but in brief, for each observation, h, we loop over each mutable feature, j, and sample a set of feature values that we append to $\boldsymbol{D}_h$. To obtain the samples for feature j, we find the end node of the fitted tree T_j based on the current row values of $\boldsymbol{D}_h$, and sample from this node.

While the feature values in $\boldsymbol{D}_h$ only take the values of the features in the training data, our method *combines* the feature values in new ways such that the rows of $\boldsymbol{D}_h$ are rarely observed in the training data. See Section 4 for a discussion on privacy concerns. Note that in practice, Algorithm 1 is made more efficient by only following the trace down the branches of T_j for each

Algorithm 1 Outline of step 2 in MCCE

Input:

$u > 0$ number of immutable features
$q > 0$ number of mutable features
($p = u + q$ is the total number of features)
$H > 0$ number of observations to explain
$K > 0$ number of samples from each tree
$T_1, \ldots, T_{q-1}$ fitted decision trees

for $h \in H$ **do**
  Let $\boldsymbol{D}_h$ be a $K \times u$ matrix where each row is a copy of the vector of u immutable features $\boldsymbol{x}_h^f$
  Append a $K \times 1$ column vector to $\boldsymbol{D}_h$ with $y' = 1$ repeated K times
  for $1 \leq j \leq q - 1$ **do**
    Let $\boldsymbol{d}$ be an empty vector of length K
    for $1 \leq i \leq K$ **do**
      Find the end node of tree T_j based on vector $\boldsymbol{D}_h[i,]$.
      Let $\boldsymbol{d}[i]$ be a single (Monte Carlo) sample from the training observations belonging to this end node.
    end for
    $\boldsymbol{D}_h \leftarrow [\boldsymbol{D}_h \mid \boldsymbol{d}]$; ▷ Append vector $\boldsymbol{d}$ as a new column of $\boldsymbol{D}_h$
  end for
  Remove the $y' = 1$ column vector from $\boldsymbol{D}_h$
  return $\boldsymbol{D}_h$
end for

Output:

data sets $\boldsymbol{D}_h$ of dimension $K \times p$

unique row of $\boldsymbol{D}_h$ and then returning multiple samples from that end node (one for each duplicate row). This saves time in the first stages of the algorithm. Observations $h \in H$ with the same immutable feature values can also share the same data set, $\boldsymbol{D}_h$.

2.3 Post-processing

Step 3 (*post-processing*) The last step of MCCE removes the rows of $\boldsymbol{D}_h$ that do not satisfy the criteria listed at the start of Section 2. The rows that remain become the set of counterfactuals for observation $h \in H$. Criteria 1 and 2 are already fulfilled since the samples come from an approximation to the data distribution conditioned on the immutable features. In addition, most samples also fulfill criterion 3 since we condition on the decision when building the trees. Nevertheless, we remove the few rows of $\boldsymbol{D}_h$ where $f(\boldsymbol{D}_h[i,]) \notin c$.

To fulfill criterion 4, we calculate **sparsity** and the **Gower distance** between each $\boldsymbol{D}_h[i,]$ and factual $\boldsymbol{x}_h$. For factual $\boldsymbol{x} = \{x_1, \ldots, x_p\}$ and row

$\boldsymbol{d} = \{d_1, \ldots, d_p\}$, sparsity is equal to the number of features changed between $\boldsymbol{x}$ and $\boldsymbol{d}$ and

$$\text{Gower distance} = \frac{1}{p}\sum_{j=1}^{p} \delta_G(d_j, x_j) \in [0, 1], \tag{3}$$

where

$$\delta_G(d_j, x_j) = \begin{cases} \frac{1}{R_j} \mid d_j - x_j \mid & \text{if } x_j \text{ is numerical,} \\ \mathbb{1}_{d_j \neq x_j} & \text{if } x_j \text{ is categorical,} \end{cases} \tag{4}$$

and R_j normalizes the jth feature so that it lies between 0 and 1. Although we could have used a distance function other than the Gower distance, it is our impression that the Gower distance is the most commonly used distance function in the counterfactual explanation literature, see e.g., Wachter et al (2017), Mothilal et al (2020) and Guidotti (2022).

To generate one counterfactual for each $h \in H$, we find the minimum sparsity across all instances of $\boldsymbol{D}_h$ and remove the instances with a sparsity larger than this value. Then we find the instance in the remaining set with the smallest Gower distance. This becomes the counterfactual for observation $h \in H$.

In this paper, we concentrate on generating a single counterfactual per instance to be explained. However, if we want to use our method to return several (say l) counterfactuals for each $h \in H$, we can set upper bounds for sparsity and the Gower distance and present the best l instances in $\boldsymbol{D}_h$ with a sparsity and Gower distance less than these bounds. See Section 5 for further discussion on this subject.

3 Experiments

In this section, we perform four experiments. In the first experiment, we compare MCCE to a set of previously proposed methods for generating counterfactual explanations using four well-known real data sets. The second investigates how MCCE's performance varies with the tuning parameter K, while the third studies the characteristics of the data generated in the second step of the algorithm. Finally, the fourth experiment considers the scalability of the different steps of the MCCE procedure. See also Appendices A.1 and A.2 for additional results for non-binarized data and a non-gradient based prediction model, and for a study of the importance of conditioning on the decision.

All computations in this section are run on a desktop computer with a 16-core AMD Ryzen Threadripper 1950X, 3.4 GHz CPU, and an NVIDIA GeForce GTX 1080 Ti GPU, which runs Ubuntu 20.04 with Python 3.7. MCCE runs on CPU only, while most of the competing methods can benefit from being run on the GPU. Therefore, below, the competing methods are run using a GPU (GeForce GTX 1080 Ti) whereas MCCE is run on a CPU.

3.1 Experiment 1: How does MCCE compare with competing methods?

In this section, we compare MCCE to a set of previously proposed methods for counterfactual explanations using four well-known data sets.

Data sets *The Adult data set* is from the 1994 Census database consisting of four continuous and eight categorical features and 49,000 observations. The goal is to classify individuals with an income of more than $50,000 USD per year or not. We explain the predictions of individuals who are predicted to not reach this income threshold. The features `age` and `sex` are set as immutable.

The Give Me Some Credit (GMC) data set is from a 2011 Kaggle Competition in credit scoring consisting of 150,000 observations and 10 continuous features. The goal is to classify individuals as experiencing financial distress or not. Individuals with financial distress are given a response of 0 and those without are given a response of 1. We explain the predictions of individuals who are predicted to experience financial distress. We drop the rows with missing data and set `age` as immutable.

The FICO data set is from the 2018 FICO Explainable Machine Learning Challenge. The data set is of Home Equity Line of Credit (HELOC) applications made by homeowners. The response is a binary feature called RiskPerformance that takes the value 1 ('Bad') if the customer is more than 90 days late on his or her payment and 0 ('Good') otherwise. There are 23 features (21 continuous and two categorical) and 10,459 observations. We explain those predicted to be 'Bad' and set the feature `ExternalRiskEstimate` as immutable.

The German Credit Data set is a publically available data set downloaded from the UCI Machine Learning Repository. It consists of 20 features (seven are continuous and 13 are categorical) and has 1000 observations. Each entry represents a person who takes credit at a bank. Each person is classified as having good or bad credit risk according to the set of attributes. We explain the predictions of individuals who are predicted to have bad credit risk. The features `Purpose`, `Age`, and `Sex` are set as immutable.

Competing methods We compare MCCE with the instance-based method FACE (Poyiadzi et al, 2020), and the optimization-based methods `C-CHVAE` (Pawelczyk et al, 2020), `CEM-VAE` (Dhurandhar et al, 2018), `CLUE` (Antorán et al, 2021), `CRUDS` (Downs et al, 2020), and `REViSE` (Joshi et al, 2019). The specific optimization-based methods were chosen because they produce counterfactuals that lie on the data manifold. In our opinion, this is an important requirement for obtaining realistic counterfactuals. The Python package `CARLA` (Pawelczyk et al, 2021) is used to generate counterfactuals from the selected methods. For all four data sets, we use the default parameter values in `CARLA` for all the competing methods, except for `CLUE` where we use other values for the FICO and German Credit Data sets such that the method is able to generate counterfactuals for a larger proportion of the individuals. The parameter values are shown in Appendix A.3. The counterfactuals were generated with code from commit 192 in `CARLA`.

Prediction model For all four data sets, we use a multi-layer perceptron (ANN) with three hidden layers as the prediction model. The number of hidden nodes, epochs, and batch sizes for each data set are shown in Table 1 together with the out-of-sample AUC values. The ANN is chosen instead of e.g., XGBoost or Random Forest, since with the exception of `C-CHVAE`, all the competing methods require gradient-based classifiers. Before fitting the models, the categorical features are binarized by partitioning them into the most frequent class and its counterpart, since most of the competing methods do not handle categorical features with more than two levels. MCCE was also tested with non-binarized features and a random forest model in addition to the experiments described in this section. See Appendices A.1 and A.2 for the results. All continuous variables are scaled to have mean equal to 0 and standard deviation equal to 1 before fitting the prediction models. Finally, for the sake of simplicity, the desired decision interval c is chosen to be $[0.5, 1]$ for all data sets.

Table 1 Predictive model settings.

	Adult	GMC	FICO	German Credit
Hidden nodes	(18, 9, 3)	(18, 9, 3)	(81, 27, 3)	(81, 16, 3)
Batch-size	1024	2048	8	16
Epochs	20	20	20	20
Learning rate	0.002	0.002	0.0005	0.002
AUC for test set	0.90	0.82	0.76	0.80

Our Method For MCCE, we fit the conditional distributions in (2) with CART. We use the Gini impurity index to measure the quality of the split if the 'response' feature is categorical and the mean squared error if it is continuous. To generate deep trees, we set the minimum number of samples required to split to 2 and the minimum number of samples in each leaf node to 5. We do not set a max depth of the tree. These selections closely resemble the default values for CART models in the widely-used synthetic data generating `R`-package `synthpop` (Nowok et al, 2016). Unless otherwise noted, we use $K = 1000$ in step 2 of the algorithm.

Performance metrics There is no standard agreement on how to perform an objective evaluation of methods generating counterfactual explanations (Guidotti, 2022). However, like Pawelczyk et al (2021) and Guidotti (2022) we think it is most natural to evaluate the performance of counterfactual explainers with respect to the four criteria listed in Section 2. The counterfactuals should (i) lie on the data manifold, (ii) be valid, (iii) be actionable, and (iv) be of low cost.

Data manifold closeness is the property that guarantees that the counterfactual is as similar to the training data as possible (Wachter et al, 2017). We follow Guidotti (2022) and say a counterfactual lies on the data manifold if it is 'similar' to the training data set and obeys the observed correlations among

the features. A counterfactual is *valid* if the counterfactual produces the correct decision (i.e., *success* = 1). A counterfactual is *actionabile* if it has as few violations as possible. A violation is when an immutable feature is changed. Finally, a counterfactual is of *low-cost* if it minimizes sparsity (denoted L_0) and the Gower distance (denoted L_1), see Section 2.3 for definitions.

Results For each data set, except the German Credit Data set, we generate one counterfactual for $n_{\text{test}} = 1000$ observations with an undesirable decision using MCCE and the competing methods. The German Credit Data set only has 1000 observations in total; therefore, we generate counterfactuals for 200 observations. We denote the number of counterfactuals generated by each method as N_{CE}. If $N_{\text{CE}} < n_{\text{test}}$ for some method, that method did not obtain counterfactuals for all observations. We calculate the *validity*, *actionability*, and *low-cost* metrics defined above and present the average and standard deviations across all test observations that obtained a counterfactual. The evaluation of the *data manifold closeness* is treated in Section 3.3.

The average and standard deviations of each metric, method, and data set are presented in Table 2. 'time(s)' indicates the time (in seconds) it takes to generate counterfactuals for n_{test} observations. Although almost all methods produce only valid counterfactuals (average success = 1). MCCE, `C-CHVAE` and `REViSE` are the only methods that always produce actionable counterfactuals (average violation = 0), and MCCE and `C-CHVAE` are the only ones that obtain counterfactuals for all test observations in all data sets. MCCE produces counterfactuals with 2-6 fewer feature changes than the best performing competing methods (L_0), and for three of the four data sets, MCCE produces counterfactuals closest to the corresponding test observations on average (L_1). For the FICO data set, the counterfactuals produced by `REViSE` are of slightly lower cost. Finally, MCCE is faster than all the other methods for all four data sets.

To get some insight into the counterfactuals generated by the different methods, we manually analyzed the Adult counterfactuals to see whether we could detect patterns across methods. One example of the counterfactuals generated for the same random observation is shown in Table 3. We show that `CLUE` and `CRUDS` produce counterfactuals that are far outside acceptable bounds and `CEM-VAE` produces counterfactuals that are not valid. The remaining methods tend to change the same three features (namely `fnlwgt`, `education-number`, and `capital-gain`). However, they don't always change them equally. For example, `education-number` is increased by MCCE for all test observations, while it is increased for only 50% of test observations by `C-CHVAE` and `REViSE`. In addition, the methods differ in how much they change the features. `C-CHVAE` changes nearly all features by a small amount while `FACE` changes fewer features but by larger amounts. This analysis shows that, unsurprisingly, there are systematic differences between the methods.

Table 2 Experiment 1: Average and standard deviation (in parentheses) of performance metrics for counterfactuals generated with MCCE and the competing methods. The **bold** values indicate the best values per metric and data set. 'time(s) all' indicates the time (seconds) it takes to generate counterfactuals for n_{test} observations.

Data set: Adult, $n_{\text{test}} = 1000$, $K = 1000$						
Method	$L_0 \downarrow$	$L_1 \downarrow$	violation$\downarrow$	success$\uparrow$	$N_{\text{CE}} \uparrow$	time(s)$\downarrow$
C-CHVAE	7.76 (1.02)	3.13 (1.10)	**0.00 (0.00)**	**1.00**	**1000**	86.29
CEM-VAE	6.92 (2.06)	3.18 (1.65)	1.38 (0.59)	0.50	**1000**	646.29
CLUE	13.00 (0.00)	7.83 (0.31)	1.36 (0.48)	**1.00**	**1000**	2754.35
CRUDS	7.87 (1.08)	3.90 (1.11)	**0.00 (0.00)**	**1.00**	**1000**	10017.76
FACE	7.03 (1.58)	3.27 (1.54)	1.40 (0.52)	**1.00**	**1000**	4151.14
REViSE	5.54 (0.96)	1.13 (0.83)	**0.00 (0.00)**	**1.00**	**1000**	8400.42
MCCE	**2.70 (0.73)**	**0.56 (0.45)**	**0.00 (0.00)**	**1.00**	**1000**	**15.00**
Data set: Give Me Some Credit, $n_{\text{test}} = 1000$, $K = 1000$						
Method	$L_0 \downarrow$	$L_1 \downarrow$	violation$\downarrow$	success$\uparrow$	$N_{\text{CE}} \uparrow$	time(s)$\downarrow$
C-CHVAE	8.98 (0.15)	0.95 (0.29)	**0.00 (0.00)**	**1.00**	**1000**	94.38
CEM-VAE	8.62 (1.08)	1.61 (0.57)	0.96 (0.19)	0.93	**1000**	685.88
CLUE	10.00 (0.04)	1.41 (0.32)	1.00 (0.03)	**1.00**	**1000**	2533.57
CRUDS	9.00 (0.00)	1.68 (0.36)	**0.00 (0.00)**	**1.00**	**1000**	8137.49
FACE	8.53 (1.08)	1.65 (0.53)	0.98 (0.16)	**1.00**	**1000**	13986.24
REViSE	8.36 (1.06)	0.70 (0.27)	**0.00 (0.00)**	**1.00**	**1000**	5722.87
MCCE	**4.52 (0.97)**	**0.61 (0.32)**	**0.00 (0.00)**	**1.00**	**1000**	**19.58**
Data set: German Credit, $n_{\text{test}} = 200$, $K = 1000$						
Method	$L_0 \downarrow$	$L_1 \downarrow$	violation$\downarrow$	success$\uparrow$	$N_{\text{CE}} \uparrow$	time(s)$\downarrow$
C-CHVAE	10.62 (1.23)	4.79 (1.31)	**0.00 (0.00)**	**1.00**	**200**	12.88
CEM-VAE	10.16 (1.89)	6.97 (1.93)	1.44 (0.53)	0.67	63	147.86
CLUE	20.09 (0.00)	13.4 (0.53)	1.55 (0.50)	**1.00**	**200**	719.94
CRUDS	13.01 (1.29)	9.22 (1.28)	**0.00 (0.00)**	**1.00**	**200**	2453.17
FACE	9.75 (2.10)	6.68 (1.97)	1.50 (0.55)	**1.00**	**200**	73.68
REViSE	7.56 (1.56)	2.28 (1.29)	**0.00 (0.00)**	**1.00**	172	2021.15
MCCE	**3.62 (0.93)**	**1.90 (0.76)**	**0.00 (0.00)**	**1.00**	**200**	**3.71**
Data set: FICO, $n_{\text{test}} = 1000$, $K = 1000$						
Method	$L_0 \downarrow$	$L_1 \downarrow$	violation$\downarrow$	success$\uparrow$	$N_{\text{CE}} \uparrow$	time(s)$\downarrow$
C-CHVAE	21.99 (0.09)	2.08 (0.84)	**0.00 (0.00)**	**1.00**	**1000**	16.39
CEM-VAE	19.52 (2.46)	3.11 (0.87)	0.95 (0.21)	0.31	478	734.83
CLUE	23.00 (0.00)	2.40 (0.56)	1.00 (0.00)	**1.00**	99	3523.77
CRUDS	22.00 (0.00)	9.32 (2.20)	**0.00 (0.00)**	**1.00**	592	7639.12
FACE	19.11 (2.32)	3.01 (0.90)	0.98 (0.16)	**1.00**	**1000**	428.44
REViSE	21.71 (0.78)	**1.64 (0.54)**	**0.00 (0.00)**	**1.00**	926	5589.01
MCCE	**12.67 (1.91)**	1.95 (0.78)	**0.00 (0.00)**	**1.00**	**1000**	**15.47**

3.2 Experiment 2: How does MCCE's performance vary with K?

Compared to all competing methods other than `FACE`, MCCE has only one tuning parameter, the number of samples generated, K (see Appendix A.3).

Table 3 Experiment 1: The counterfactuals generated for the same random factual of Adult. The **bold** values indicate the values that are the same as the original factual values.

Method	Age	FNLWGT	Edc.	Gain	Loss	Hr.	MS	Co.	Occ.	Race	Rel.	Sex	Work
Original	**20**	**266,015**	**10**	**0**	**0**	**44**	**NM**	**US**	**O**	**NW**	**NH**	**M**	**P**
C-CHVAE	**20**	247,240	**10**	652	1679	42	M	**US**	**O**	**NW**	H	**M**	**P**
CEM-VAE	23	190,709	12	10,296	**0**	52	**NM**	**US**	**O**	W	**NH**	**M**	NP
CLUE	11	398,962	**10**	10,725	-62	49	**NM**	**US**	**O**	W	**NH**	**M**	**P**
CRUDS	**20**	138,021	21	4833	210	79	M	**US**	MS	W	H	M	**P**
FACE	46	220,979	**10**	13,550	**0**	40	**NM**	**US**	**O**	**NW**	**NH**	**M**	**P**
REViSE	**20**	76,456	14	10	379	68	M	**US**	**O**	W	H	**M**	**P**
MCCE	**20**	348,148	**10**	34,095	**0**	48	**NM**	**US**	**O**	**NW**	**NH**	**M**	**P**

In this section, we study how MCCE's performance varies with K. We repeat the analysis for the Adult data set in Experiment 1 varying K in $[10, 50, 100, 1000, 10{,}000, 25{,}000]$ and examine both MCCE's metric averages and run times. We show the results in Table 4. 'model(s)', 'gener(s)', and 'post-proc(s)' indicate the time (in seconds) it takes for MCCE's modeling, generating, and post-processing steps to run, respectively. 'total(s)' is the total time.

We see that a K as low as 10 generates valid counterfactuals for 99.9% of test observations and gives results of L_0 and L_1 that are better than all competing methods. Although the metrics become better with larger K, conditioning on the decision eliminates the risk of not sampling any valid counterfactuals. Conditioning on the immutable features also ensures the samples are actionable and relatively close to the factual.

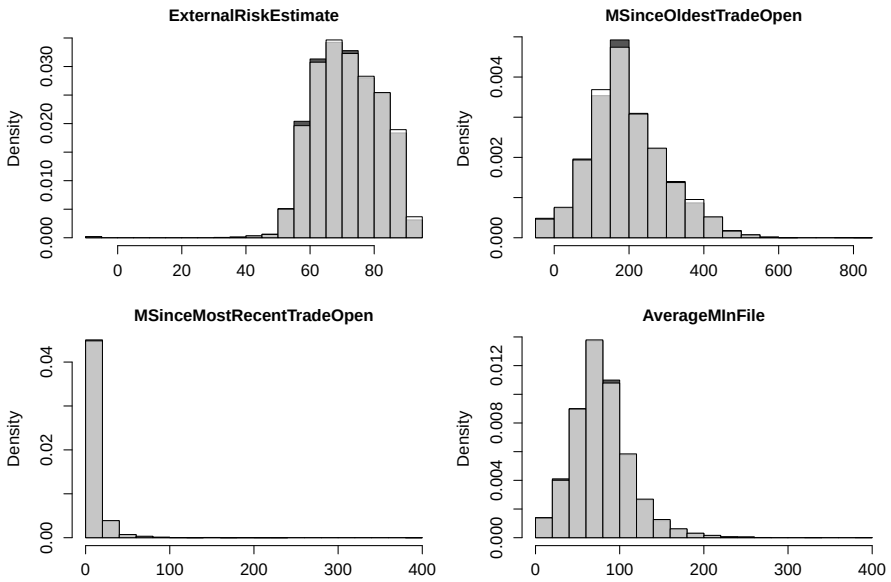

Fig. 2 Histograms for three of the variables in the generated data set (white) with the histograms for the real data superimposed (dark grey). Where the histograms overlap, the blend of of white and dark grey gives a light grey color.

3.3 Experiment 3: Are MCCE counterfactuals on-manifold?

According to Guidotti (2022), a plausible counterfactual is 'realistic' if it is 'similar' to the known data set and adheres to observed correlations among the features. In this section, we study the characteristics of the data generated in step 2 of the MCCE method for the FICO data set, using $K = 10{,}000$.

Table 4 Experiment 2: Average and standard deviation (in parentheses) of performance metrics for counterfactuals generated with MCCE as we vary K.

Data set: Adult, $n_{\text{test}} = 1000$, K varies								
K	$L_0 \downarrow$	$L_1 \downarrow$	violation↓	$N_{\text{CE}} \uparrow$	model(s)↓	gener(s)↓	post-proc(s)↓	total(s)↓
10	3.98 (1.1)	1.59 (0.94)	0 (0.0)	999	3.8	0.38	5.56	9.74
50	3.39 (0.94)	1.1 (0.77)	0 (0.0)	1000	3.8	0.58	5.61	9.99
100	3.18 (0.9)	0.93 (0.69)	0 (0.0)	1000	3.8	0.76	5.72	10.28
1000	2.7 (0.73)	0.56 (0.45)	0 (0.0)	1000	3.8	4.39	7.23	15.43
5000	2.51 (0.66)	0.45 (0.39)	0 (0.0)	1000	3.8	21.29	12.34	37.43
10,000	2.43 (0.64)	0.43 (0.37)	0 (0.0)	1000	3.8	42.61	17.71	64.11
25,000	2.39 (0.64)	0.41 (0.36)	0 (0.0)	1000	3.8	106.93	29.50	140.23

Figure 2 shows histograms for four of the variables in this data set (white) with histograms of the real data superimposed (dark grey). Where the histograms overlap, the blend of white and dark grey gives a light grey color. As can be seen from the figure, the generated marginal distributions for both the continuous and categorical variables are very close to the data distributions. Appendix A.5 shows that the figures for the rest of the variables are very similar. In Figures 3-5 we show the correlation matrices for the real and generated data, as well as the difference between the correlation matrices. It is evident that even the correlations are very well captured by the generative model.

Another way of checking whether the generated data can easily be told apart from the original data is to compute the discriminator measure described by Borisov et al (2023). To do so, we fit a Random Forest model on a mix of generated training samples and samples from the real training data set, where the response indicates whether the observation comes from the generated or real data (when performing this experiment, the real response is discarded). Then, we use this model to predict the probability of belonging to the real data on a set of new samples and compute the model's out-of-sample accuracy using a threshold of 0.5. If the discriminator cannot differentiate between the generated and real samples, the model's accuracy will be around 50%. In our case, the mean accuracy over five trials with different random seeds for the Adult, GMC, FICO, and German Gredit data sets are 62.8%, 56.5%, 59.1%, and 55.7%, respectively. Although these numbers are slightly higher than 50%, the numbers for Adult and FICO are lower than the lowest accuracy reported for the same datasets in Borisov et al (2023)[3].

These experiments show that it is difficult to differentiate between counterfactuals generated by MCCE and the underlying data set.

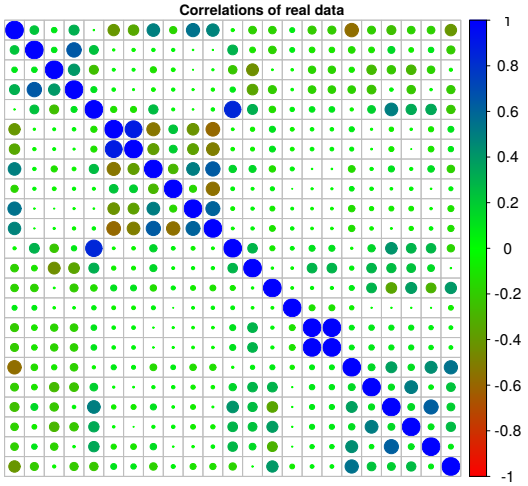

Fig. 3 Correlation matrix for the FICO data set. The areas of the circles are proportional to the absolute value of the corresponding correlation coefficients.

[3]In Borisov et al (2023) the FICO dataset is referred to as the HELOC dataset.

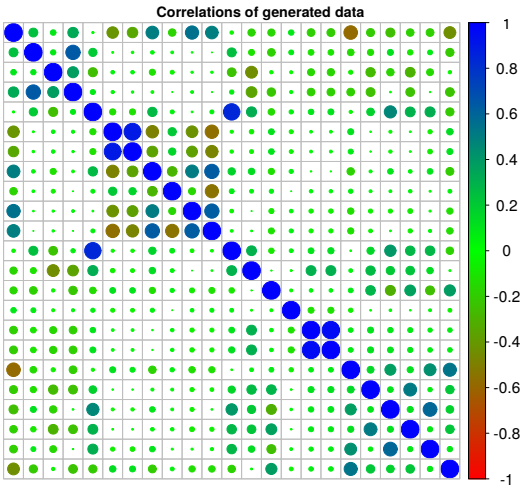

Fig. 4 Correlation matrix for the data generated from the autoregressive generative model ($K = 10{,}000$). The areas of the circles are proportional to the absolute value of the corresponding correlation coefficients.

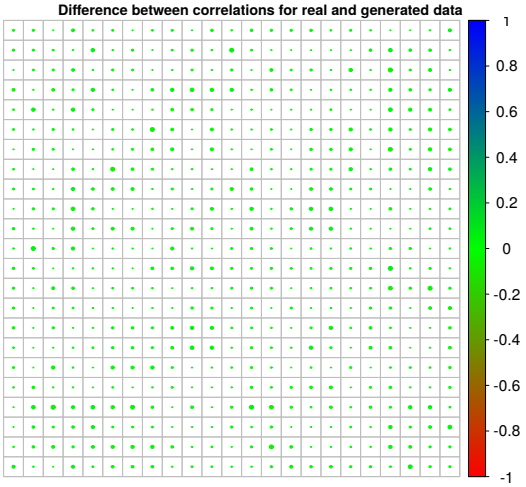

Fig. 5 Difference between the correlation matrices in Figures 3 and 4. The areas of the circles are proportional to the absolute value of the corresponding correlation coefficients.

3.4 Experiment 4: Is MCCE scalable?

For explanability methods to be applicable in a wide range of practical situations, it is crucial that they scale well computationally. We now study the scalability of the different steps of the MCCE procedure. The first subsection provides actual measurements of the computation time for the three MCCE

steps as we vary the different quantities, using a controlled simulation setup. The second subsection provides the corresponding theoretical computational complexity in big-O notation.

3.4.1 Computational complexity in practice

Let the p-dimensional feature vector $\boldsymbol{X}$ be simulated from a zero-mean Gaussian distribution with covariance $\text{Cov}(X_j, X_k) = 0.5$ and $\text{Var}(X_j) = 1$ for all $j \neq k$. In this simulated setup, we explain simple linear models of the form $f(\boldsymbol{x}) = \sum_{j=1}^{p} \beta_j x_j$, where $\beta_j = 1$ for all j.

We generate counterfactual explanations for predictions of this model as we vary the following quantities: p, n_{train}, n_{test}, and K. For each simulated model, we set the decision interval for an acceptable decision to $c = (0, \infty)$, which on average covers half of the observations regardless of the dimension p. We always only explain test observations that are originally outside of this interval. Moreover, we do not fix any of the features in the explanations, i.e., all features are mutable ($q = p$).

As an initial broad overview, Table 5 shows the computation times for the three steps of MCCE for all combinations of $p = (5, 30)$, $n_{\text{train}} = (1000, 10{,}000)$, $n_{\text{test}} = (1, 50)$, and $K = (10{,}000, 100{,}000)$. The displayed computation times represent the mean of 10 repeated computations.

Table 5 Overview of the computational times (in seconds) of the three steps of MCCE when varying n_{test}, n_{train}, p, and K. The displayed computation times represent the mean of 10 repeated computations.

n_{test}	n_{train}	p	K	model(s)↓	gener(s)↓	post-proc(s)↓	total(s)↓
1	1000	5	10,000	0.04	0.11	0.09	0.24
1	1000	5	100,000	0.04	0.75	0.12	0.91
1	1000	30	10,000	1.25	0.83	0.09	2.17
1	1000	30	100,000	1.25	5.10	0.15	6.50
1	10,000	5	10,000	0.48	0.07	0.09	0.64
1	10,000	5	100,000	0.48	0.46	0.12	1.06
1	10,000	30	10,000	12.05	0.95	0.10	13.10
1	10,000	30	100,000	12.05	5.39	0.15	17.59
50	1000	5	10,000	0.05	3.70	0.99	4.74
50	1000	5	100,000	0.04	38.26	3.36	41.66
50	1000	30	10,000	1.24	24.89	1.03	27.16
50	1000	30	100,000	1.24	264.63	6.21	272.08
50	10,000	5	10,000	0.48	2.07	0.70	3.25
50	10,000	5	100,000	0.49	25.16	3.01	28.66
50	10,000	30	10,000	12.06	24.87	1.04	37.97
50	10,000	30	100,000	12.00	268.76	6.03	286.79

Altering all quantities affects the total computation times. Even for as many as $p = 30$ features, performing the modeling with $n_{\text{train}} = 10{,}000$ only takes 12 seconds. The dimension p does, however, play a crucial role for both

the modeling time and the generation time. Further, the number of samples generated per test observation, K, and the number of test observations n_{test}, naturally affects the generation time significantly. The post-processing time is quite limited even for $n_{\text{test}} = 50$, $p = 30$ and $K = 100{,}000$. That is, for the model and quantity sizes used here, the generation time is the driving factor.

In Figures 6-9 we take a more detailed look at how each of the four quantities affects the runtime of the different steps when altering one at a time and keeping the others fixed. When the quantities are not altered, we fix them at the following values: $n_{\text{test}} = 1$, $n_{\text{train}} = 1000$, $p = 10$, $K = 10{,}000$.

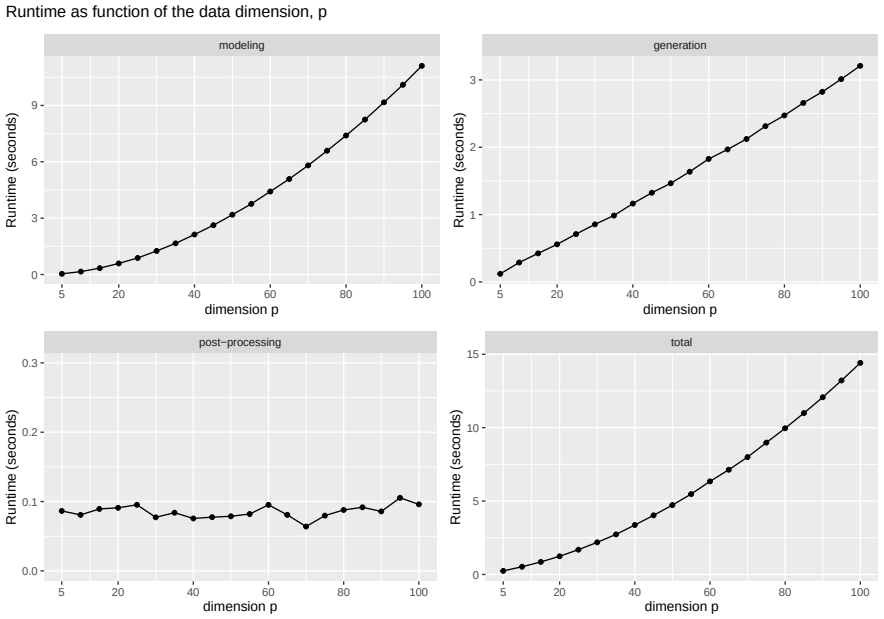

Fig. 6 Runtime as function of dimension p. The other quantities are fixed to $n_{\text{test}} = 1$, $n_{\text{train}} = 1000$, $K = 10{,}000$.

From Figure 6, we see that increasing the dimension p, increases the modeling time faster-than-linearly, the generation time linearly, and the post-processing time not at all. The higher rate of the modeling time indicates that for very high dimensions (> 100), the modeling time may be the driving factor.

Figure 7 shows that the generation time scales linearly at a significant rate in terms of K. The picture is not as clear for the post-processing time, but this is less important, as the actual run times are rather small for this component. The modeling time is obviously unaffected by K and is therefore not displayed.

Figure 8 shows that the generation time scales approximately linearly at quite a high rate. The effect of n_{train} on the generation time is unclear from these simulations. For the smallest values of n_{train}, this component is probably

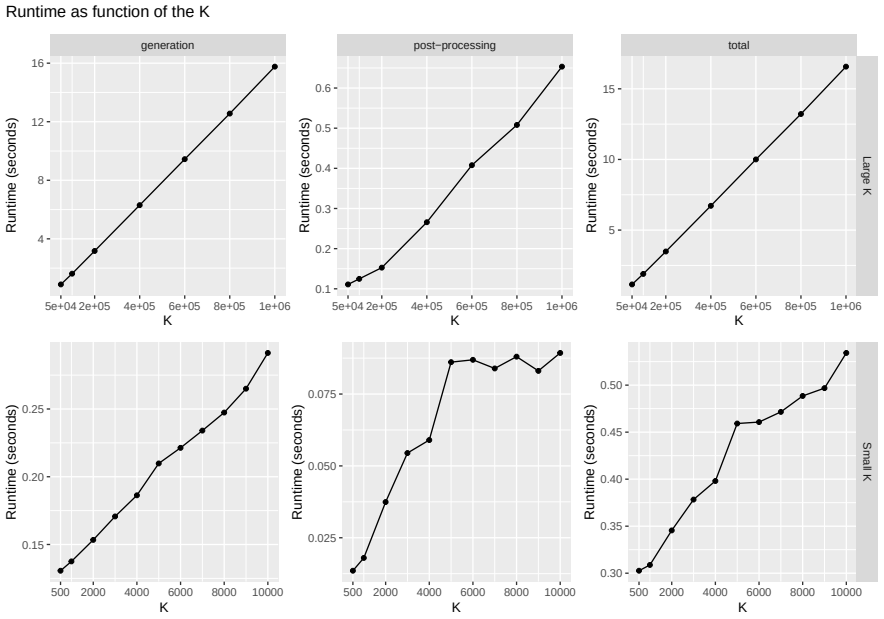

Fig. 7 Runtime as function of the number of generated samples K. The other quantities are fixed to $n_{\text{test}} = 1$, $p = 10$, $n_{\text{train}} = 1000$. The upper panels show $K > 10^4$, and lower panels show $K \leq 10^4$. The modeling time is obviously unaffected by K and is therefore not displayed.

influenced by inaccuracies in the computational timing. The post-processing time is not affected by n_{train} and is therefore not displayed.

Figure 9 shows that both the generation and post-processing times naturally scale approximately linearly in terms of n_{test}. The modeling time is clearly not affected by n_{test} and is therefore not displayed. For this specific setup, the total computation time seems to increase by about 0.2 seconds per extra test observation.

To showcase that these scalability results are relevant and valid in a broader context, we also run simulations with quantities mimicking the four real data sets from Section 3.1. Table A4 in Appendix A.6 displays the computation times for these simulations and shows a relatively good match between the computation times in the simulations and the real data experiments. This suggests that the scalability observed in the above simulation study is likely to extend roughly to real-world data scenarios.

3.4.2 Theoretical computational complexity

The simulation study above shows that MCCE scales well in all four quantities. In this section, we discuss the theoretical computational complexity in terms of big-O notation. Recall that u and q are the number of immutable and mutable features, respectively.

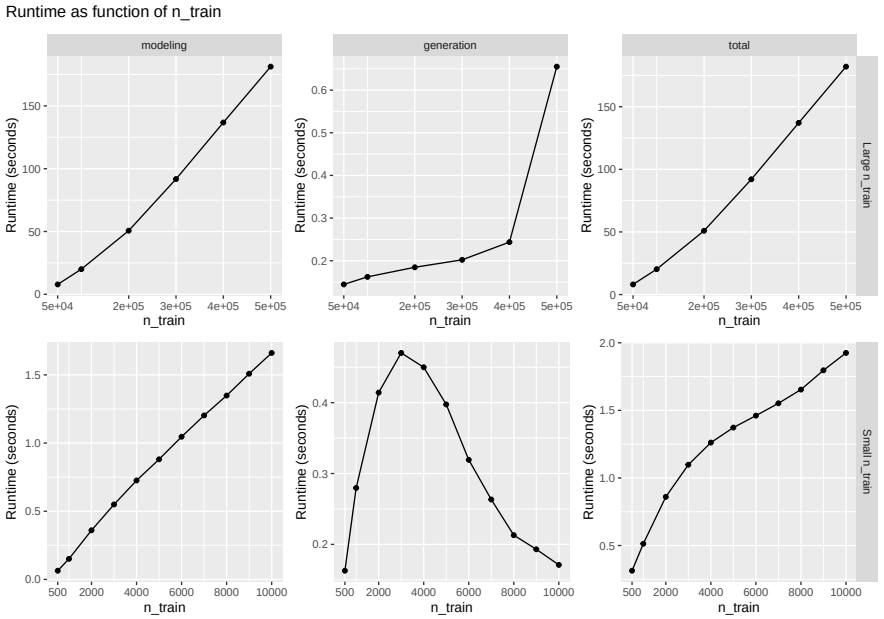

Fig. 8 Runtime as function of number of training samples n_{train}. The other quantities are fixed to $n_{\text{test}} = 1$, $p = 10$, $K = 10{,}000$. The upper panels shows $n_{\text{train}} > 10^4$ and the lower panels shows $n_{\text{train}} \leq 10^4$. The post-processing time is not affected by n_{train} and is therefore not displayed.

According to Hastie et al (2009, Ch 9.7), the training time for the i-th tree (with $u + i$ features) is $O((u + i)n_{\text{train}} \log(n_{\text{train}}))$. Since we have $q - 1$ such trees and $\sum_{i=1}^{q-1}(u + i) = (q - 1)(u + (q - 2)/2)$, the computational complexity of step 1 is $O(q(u + q)n_{\text{train}} \log(n_{\text{train}}))$.

Generating a sample from one of the conditional distributions in step 1 requires traversing the decision tree from root to leaf, which requires going through roughly $O(\log(n_{\text{train}}))$ nodes (Géron, 2019). Hence, the computational complexity of step 2 is $O(n_{\text{test}} Kq \log(n_{\text{train}}))$.

In step 3, we compute validity, sparsity, and the Gower distance for every sample to find the best counterfactual. The computational complexity of the validity computation is non-trivial since it depends on the choice of prediction model. Assuming that the computation time of the prediction model increases no faster than linearly in $u + q$, the computational complexity of step 3 is $O\left(n_{\text{test}}\, K\, (u + q)\right)$.

Removing smaller order terms, the total computational complexity of MCCE becomes $O\left(q(u + q)n_{\text{train}} \log(n_{\text{train}}) + n_{\text{test}} K(q \log(n_{\text{train}}) + u)\right)$, showing that MCCE scales very well in all quantities (i.e., quadratic in q, at rate $n_{\text{train}} \log(n_{\text{train}})$ in n_{train}, and linear in u, K, and n_{test}).

Overall, these theoretical scalability terms are in accordance with our findings from the simulations: For the modeling time, the quadratic increase in q and log-linear increase in n_{train} match the findings from our simulations. The

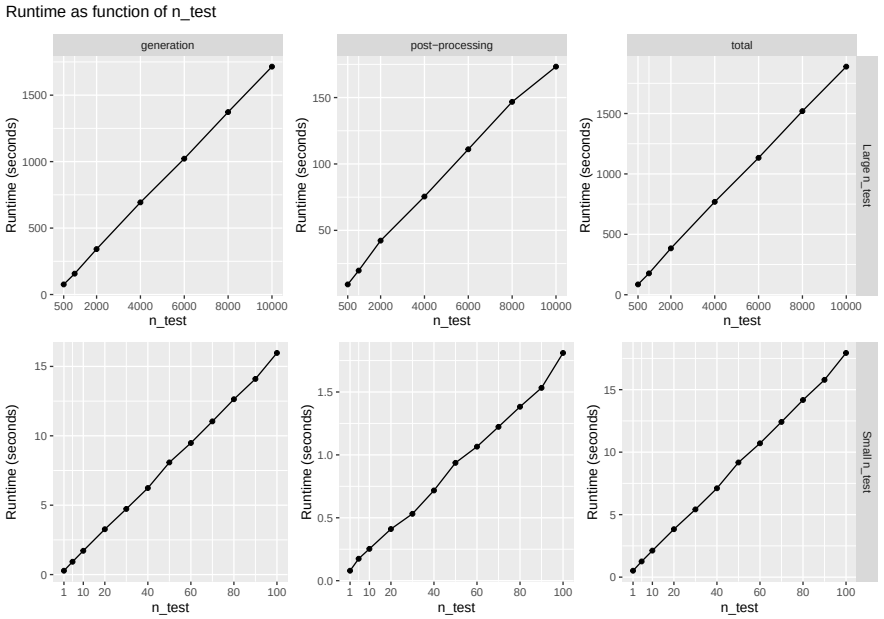

Fig. 9 Runtime as function of number of test observations n_{test}. The other quantities are fixed to $n_{\text{train}} = 1000$, $p = 10$, $K = 10{,}000$. The upper panels shows $n_{\text{test}} > 10^4$, and upper panels shows $n_{\text{test}} \leq 10^4$. The modeling time is clearly not affected by n_{test} and is therefore not displayed.

linear increase in n_{test}, K, and q for the generation time is also in accordance with our simulation results. The scalability in terms of n_{train} was, as mentioned, fairly uncertain, partly due to the low computation time (< 1 sec). Thus, it is not unreasonable that the theoretically found logarithmic rate is valid also for our implementation (with quite a small factor). For the post-processing time, the computation times in our simulations are so small that the theoretically found linearity in q, K and n_{test} seems reasonable.

4 Privacy

Naive counterfactual explanations can leak sensitive or personal data in a variety of ways. First, if a counterfactual explanation is an exact copy of an individual in the training data, this individual may be recovered, leading to a breach of privacy. Table 6 presents both the number and percentage of MCCE counterfactuals with exact copies in each of the datasets discussed in this paper. Since there are at most 1.1% of counterfactuals with copies, this type of privacy breach is rare for MCCE.

However, in certain circumstances, it may be necessary for a strict assurance that the counterfactuals generated by MCCE do not match training observations. To achieve this, one can easily add a step in MCCE's post-processing step to remove all rows of $\mathbf{D}_h$ that exactly match an observation

Table 6 The number and proportion of counterfactuals being copies of original training observations for each dataset.

	Adult	GMC	FICO	German Credit
# counterfactuals	1000	1000	1000	200
# copies	11	4	0	2
Proportion of copies	1.1%	0.4%	0%	1.0%

in the training data. Although this approach may slightly diminish the quality of the counterfactuals based on other performance measures, it provides a safeguard against privacy breaches of this type.

Another privacy breach may occur due to so-called explanation linkage attacks (Goethals et al, 2022) or membership inference attacks. A linkage attack is an attempt to re-identify individuals in a dataset by combining the generated data with background information. A membership inference attack attempts to re-identify individuals by exploiting the fact that their counterfactual explanation may reveal whether they were used in training the model (Pawelczyk et al, 2023). Even though MCCE's explanations are rarely observed in the underlying data, it is an open question as to whether MCCE counterfactuals are vulnerable to explanation linkage or membership inference attacks. One way to alleviate these potential privacy problems is to incorporate differential privacy (Dwork, 2006). In the case of MCCE, incorporating differential privacy could entail introducing noise in its data generating step (see e.g., Mahiou et al (2022) or Nowok et al (2016)). Further work is required to investigate the efficiency and practical considerations of such an approach.

5 Conclusion

To the best of our knowledge, MCCE is the first counterfactual method that models both the underlying data distribution and the decision. As shown in our experiments, this is a powerful setup because even without the post-processing step, MCCE generates samples that are almost always valid and actionable. In addition, modeling the data with an autoregressive generating model produces counterfactuals with lower costs than all the competing methods. Finally, as opposed to other on-manifold methods that typically use variational autoencoders and strict prediction model and data requirements, MCCE handles any type of prediction model and categorical features with more than two levels.

5.1 Future steps

MCCE's three steps are flexible and can be improved independently of each other. We will discuss possible changes to each step below. In MCCE's first step, the decision trees can easily be replaced with another generation method like conditional generative adversarial nets (CGAN) (Mirza and Osindero, 2014) or conditional tabular GANs (CTGAN) (Xu et al, 2019). Note, however, that both CGANs and GANs are restricted to gradient-based classifiers.

If formal privacy guarantees are required, differential privacy can be built into MCCE's data-generating process with little effort.

In MCCE's second step (the autoregressive generative model), the visit sequence has to be chosen. In the experiments conducted in this paper, the visit sequence corresponded to the order of variables in the original data sets. One might get even better correspondence between the generated and original data if the visit sequence for example is chosen based on the correlation between the variables, modeling the most correlated variables first. This would be an interesting and relevant topic for further work.

Finally, MCCE's third step can easily be adjusted to handle other performance metrics. Since the Gower distance penalizes changes to continuous features less than categorical features, one might choose to replace it with another type of distance function like the Heterogeneous or Interpolated Value Difference Metrics (Wilson and Martinez, 1997). If, for some reason, the number of features being changed is irrelevant for a specific application, the initial filtering with the L_0 norm may also be dropped.

We have developed MCCE for tabular data. One avenue for further work is to adapt MCCE to non-tabular data. As long as the prediction model is used for a binary (or multiclass) decision, a prerequisite for counterfactual explanations, it should be possible to adapt MCCE's three steps to non-tabular data, such as time series, text, or images. One challenge might be that the number of samples needed to generate explanations with low enough cost becomes higher with more complex data.

In this paper, the aim was to generate one counterfactual for each individual. However, as mentioned, MCCE can easily be extended to return the $k \in K$ best counterfactuals if this is desired. To enhance the practical relevance of a set of multiple counterfactuals, it might be desirable for the generated counterfactuals to also exhibit *diversity* (Mothilal et al, 2020). Finding the satisfactory balance between performance scores for single counterfactuals and their diversity remains a subject for future research.

Acknowledgments. This work was supported by the Norwegian Research Council grant 237718 (BigInsight). This paper is supported by the European Union's HORIZON Research and Innovation Programme under grant agreement No 101120657, project ENFIELD (European Lighthouse to Manifest Trustworthy and Green AI).

Authors' contributions. AR: Methodology, Software, Validation, Data Curation, Original Draft, Writing, Review & Editing, Visualization. MJ: Conceptualization, Methodology, Software, Validation, Original Draft, Writing, Review & Editing, Visualization. KA: Conceptualization, Methodology, Validation, Original Draft, Writing, Review & Editing, Visualization. AL: Conceptualization, Methodology, Writing, Original Draft, Review & Editing, Visualization.

Availability of data and materials. The Adult, FICO and German Credit Data Sets can be downloaded from https://github.com/riccotti/Scamander/

blob/main/dataset and the Give Me Some Credit dataset from https://www.kaggle.com/c/GiveMeSomeCredit/data.

Declarations

Conflict of interest. The authors declare that they have no conflicts of interest.

Code availability. The Python code used in this paper is open source, and can be downloaded at https://github.com/NorskRegnesentral/mccepy. A similar R package is available at https://github.com/NorskRegnesentral/mcceR.

Ethics approval. Not applicable.

Consent to participate. Not applicable.

Consent for publication. The authors declare that they provide consent for publication.

Appendix A Further experiments

A.1 How is MCCE's performance when the categorical features are not binarized?

Since none of the on-manifold methods investigated in the main paper handle categorical features with more than two levels, we had to restrict the models in the method comparisons to models with only two levels. The MCCE method can, however, handle models with categorical features with an arbitrary number of levels. To exemplify this, we here provide performance results for the MCCE applied to the Adult data set *without* binarizing the seven categorical features. The results are shown in Table A1.

Since the explanation is carried out on different data/model, the performance scores are not directly comparable. Note, however, that the costs are slightly higher for these counterfactuals, and that the computation time also is higher. Both of these effects are expected since the data are much richer – one of the categorical features (country) has, for instance, 41 different levels. Changes in the categorical features are then more likely. Training the decision trees is also more time-consuming as many more splits need to be considered.

Table A1 Experiment 5: Average and standard deviation (in parentheses) of performance metrics for counterfactuals generated by MCCE when the categorical features are **not binarized**.

Data set: Adult, $n_{\text{test}} = 1000$, $K = 1000$					
$L_0 \downarrow$	$L_1 \downarrow$	violation$\downarrow$	success$\uparrow$	$N_{\text{CE}} \uparrow$	time(s)$\downarrow$
3.14 (0.98)	0.9 (0.76)	0 (0.0)	1	1000	29.95

A.2 How is MCCE's performance when the prediction model is non-gradient based?

In addition to the restriction to categorical features with two levels, our method comparison required a gradient-based predictive model to be applicable to all the alternative methods. Again, MCCE is not restricted to gradient-based predictive models. Thus, below, we use MCCE to explain a random forest model on the Adult data set, to showcase that it is directly applicable to non gradient-based predictive models. We use a random forest model with 200 trees.

The counterfactual method `C-CHVAE` is the only other on-manifold method in our list of benchmark methods that supposedly handles non-gradient-based models. Thus, it would have been interesting to compare the performance with this method. Unfortunately, a non-gradient-based version of this method is not currently implemented in `CARLA`.

The average performance metrics for MCCE are reported in Table A2. As in Section A.1, the results are not directly comparable. Note, however, that

the cost is quite similar to those for the ANN model. The computation time, on the other hand, is quite a bit higher for the random forest model. This is a result of increased prediction time (used when computing validity) for the random forest implementation compared to the ANN model.

Table A2 Average and standard deviation (in parentheses) of performance metrics for counterfactuals generated by MCCE when the predictive model is a **random forest**.

Data set: Adult, $n_{\text{test}} = 1000$, $K = 1000$					
$L_0 \downarrow$	$L_1 \downarrow$	violation$\downarrow$	success$\uparrow$	$N_{\text{CE}} \uparrow$	time(s)$\downarrow$
2.48 (0.83)	0.42 (0.43)	0 (0.0)	1	1000	36.53

A.3 Parameter values for competing methods

For all four data sets we used the default parameter values in `CARLA` for all the competing methods, except for `CLUE` where we had to use other values for the FICO and German Credit Data sets. The parameter values used for the different methods are shown in Figure A1. The values in parenthesis are the ones used for the FICO and German Credit Data sets.

CEM-VAE:
Hyperparams:

batch_size:	1
kappa:	0.1
init_learning_rate:	0.01
binary_search_steps:	9
max_iterations:	100
initial_const:	10
beta:	0.9
gamma:	1.0
mode:	"PN"
num_classes:	2

AE-params:

hidden_layer:	[20, 10, 7]
train_ae:	True
epochs:	5

FACE:
fraction: 0.15

CRUDS:
Hyperparams:

lambda_param:	0.001
optimizer:	"RMSprop"
lr:	0.008
max_iter:	2000

VAE-params:

layers:	[16, 8]
train:	True
epochs:	5
lr:	0.001
batch_size:	32

REVISE:
Hyperparams:

lambda:	0.5
optimizer:	"adam"
lr:	0.1
max_iter:	1500
target_class:	[0, 1]
binary_cat_features:	True

VAE-params:

layers:	[512, 256, 8]
train:	True
lambda_reg:	0.000001
epochs:	5
lr:	0.001
batch_size:	32

C-CHVAE:
Hyperparams:

n_search_samples:	100
p_norm:	1
step:	0.1
max_iter:	1000
binary_cat_features:	True

VAE-params:

layers:	[512, 256, 8]
train:	True
lambda_reg:	0.000001
epochs:	5
lr:	0.001
batch_size:	32

CLUE:

train_vae:	True
width:	10
depth:	3
latent_dim:	12
batch_size:	64 (16)
epochs:	1
lr:	0.001 (0.0005)
early_stop:	10

Fig. A1 Parameters used for the competing methods.

Table A3 Average and standard deviation (in parentheses) of performance metrics for counterfactuals generated with MCCE when **we do not condition on the decision**.

Data set: Adult, $n_{\text{test}} = 1000$, K varies								
K	$L_0 \downarrow$	$L_1 \downarrow$	violation$\downarrow$	$N_{\text{CE}} \uparrow$	model(s)$\downarrow$	gener(s)$\downarrow$	post-proc(s)$\downarrow$	total(s)$\downarrow$
10	4.66 (1.4)	2.19 (1.22)	0 (0.0)	633	3.87	1.09	2.44	7.41
50	4.04 (1.27)	1.71 (1.09)	0 (0.0)	837	3.87	1.59	3.91	9.37
100	3.79 (1.24)	1.50 (1.05)	0 (0.0)	875	3.87	1.77	4.12	9.76
1000	3.19 (1.13)	1.00 (0.85)	0 (0.0)	974	3.87	5.56	4.92	14.35
5000	2.78 (0.9)	0.69 (0.59)	0 (0.0)	1000	3.87	22.69	6.20	32.76
10,000	2.64 (0.82)	0.62 (0.54)	0 (0.0)	1000	3.87	43.95	7.45	55.28
25,000	2.45 (0.81)	0.55 (0.50)	0 (0.0)	1000	3.87	108.74	10.95	123.56

A.4 How do the metrics and computation times compare when we *do not* condition on the desired decision?

One of the novel contributions of this paper is the modeling of the decision alongside the mutable features, to then condition on the desired decision (and the immutable features) when generating the data set of potential counterfactuals. The idea behind this notion is to bias the method toward generating samples that are more likely to yield the desired decision. In this section, we investigate whether including the decision on the modeling and then conditioning on the desired decision really ensures a higher proportion of valid samples. For the test observations in the Adult data set we generate counterfactuals *without* conditioning on the decision for various K and show the usual metrics and run times in Table A3, to be compared to Table 4. As suspected, this variation of MCCE is much less effective in generating *valid* counterfactuals (N_{CE} in Table A3 is much smaller than in Table 4).

Furthermore, the fitting and generation times exhibit a slight decrease compared to the regular MCCE approach. Although one might have expected the training time to decrease due to fewer candidate features per split, the omission of the decision variable can result in larger trees as the available features convey less information. This was precisely the case for the given dataset, also leading to a slight increase in generation time.

On the other hand, the post-processing time is significantly reduced with this alternative approach, as it only generates around 25% unique and valid samples (compared to 86% with the normal MCCE), requiring fewer L_0 and L_1 calculations. In total, the computation time is smaller for the MCCE approach which does *not* condition on the desired decision.

However, it is essential to highlight that this alternative approach requires a significantly higher value of K in order to generate valid counterfactuals for all 1000 test observations. When not conditioning on the desired decision, a K value of $5,000$ is needed compared to only 50 when conditioning on the desired response. Consequently, this negates the initial speed-up of the former method, making the original MCCE a clearly superior alternative.

A.5 Quality of generated data

In Section 3.3 we showed the histograms for the generated data for four of the variables in the FICO data set. Here, we show the histograms for the rest of the variables. The histograms for the generated data are in white while the histograms for the real data are in dark grey. As we can see, the marginal distributions of the generated data match the original ones very well also for these features.

A.6 Simulations mimicking real data experiments

In Section 3.4 we performed simulation experiments with a linear model when varying the dimension p (with no fixed features, i.e., $u = 0$), n_{train}, n_{test} and K. To showcase that these scalability results are relevant and valid in a

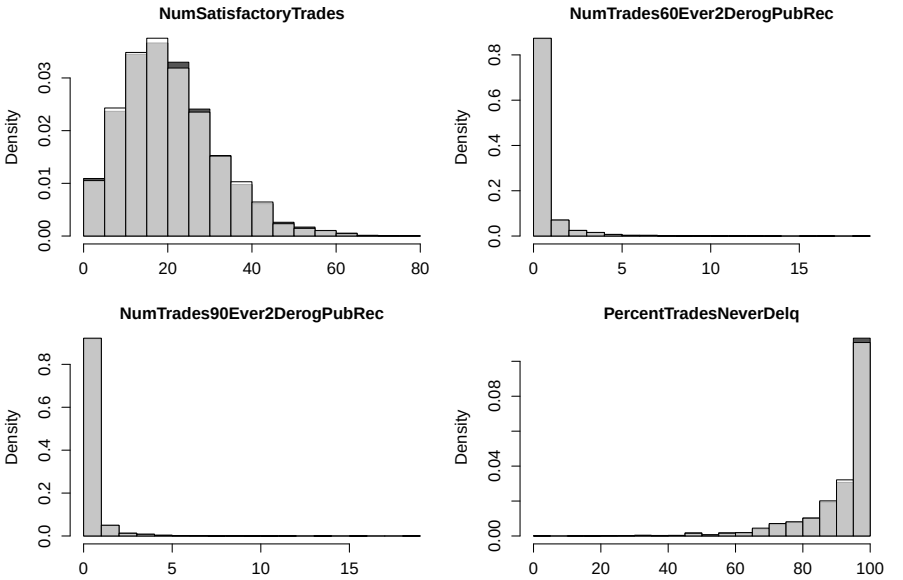

Fig. A2 Histograms for four of the variables in the generated data set (white) with the histograms for the real data superimposed (dark grey). Where the histograms overlap, the blend of white and dark grey gives a light grey color.

broader context, we also ran simulations with quantities mimicking the four real data sets in Section 3.1. As seen from the table, the total computation times in the simulation (27, 50, 7, and 49 seconds) are similar in magnitude to those recorded in the real data experiments (25, 32, 6, and 34 seconds), despite the model, the feature dependence and most likely the tree depth, differing significantly. This indicates that the scalability in our basic simulation study in Section 3.4 generalizes roughly to real data settings as well.

Table A4 Computation times (in seconds) of the three steps of MCCE when mimicking n_{test}, n_{train}, and $p/u/q$, with $K = 1000$ for the four real data experiments in Section 3.1. The displayed computation times represent the mean of 10 repeated computations.

mimicked dataset	model(s)↓	gener(s)↓	post-proc(s)↓	total(s)↓
Adult	10.28	9.60	7.30	27.18
GMC	34.28	8.05	7.19	49.52
German credit	0.56	5.38	1.46	7.40
FICO	7.60	33.42	7.53	48.55

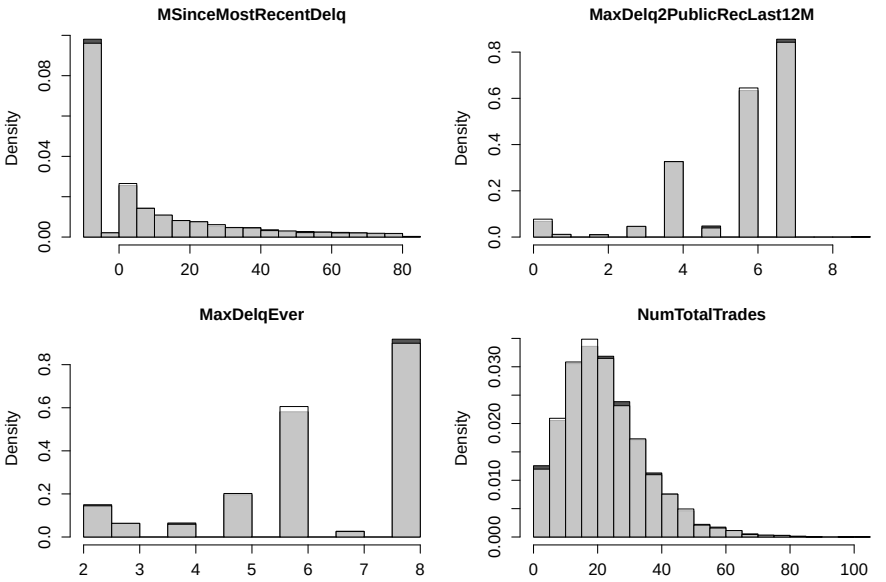

Fig. A3 Histograms for four of the variables in the generated data set (white) with the histograms for the real data superimposed (dark grey). Where the histograms overlap, the blend of white and dark grey gives a light grey color.

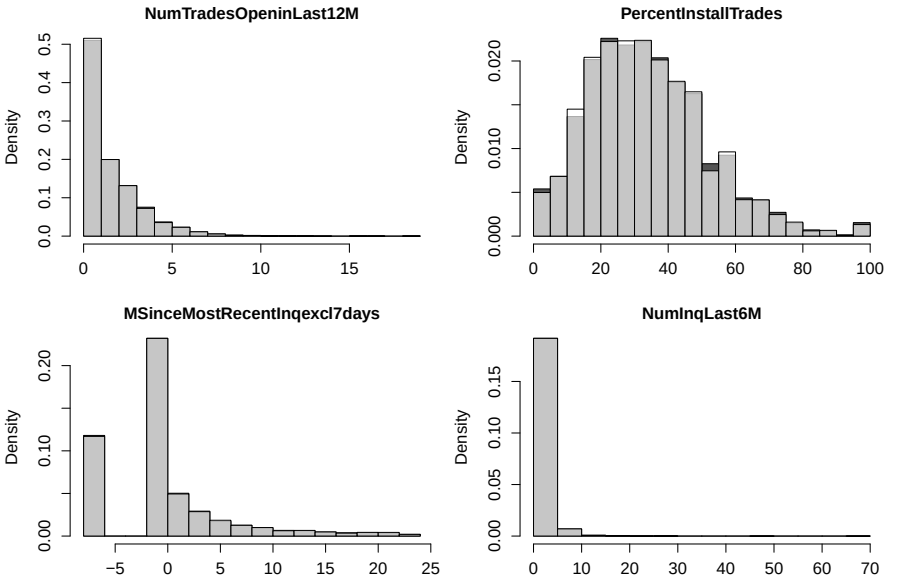

Fig. A4 Histograms for four of the variables in the generated data set (white) with the histograms for the real data superimposed (dark grey). Where the histograms overlap, the blend of white and dark grey gives a light grey color.

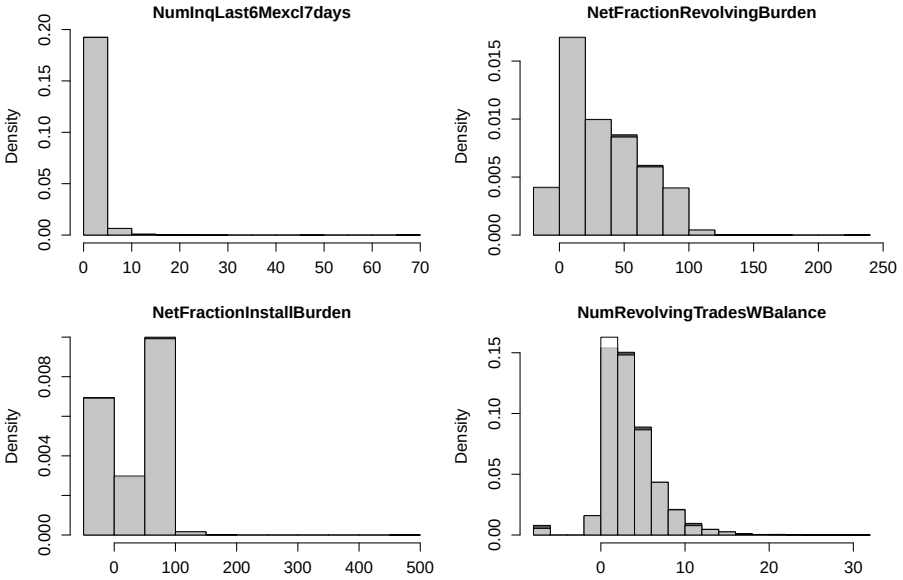

Fig. A5 Histograms for four of the variables in the generated data set (white) with the histograms for the real data superimposed (dark grey). Where the histograms overlap, the blend of white and dark grey gives a light grey color.

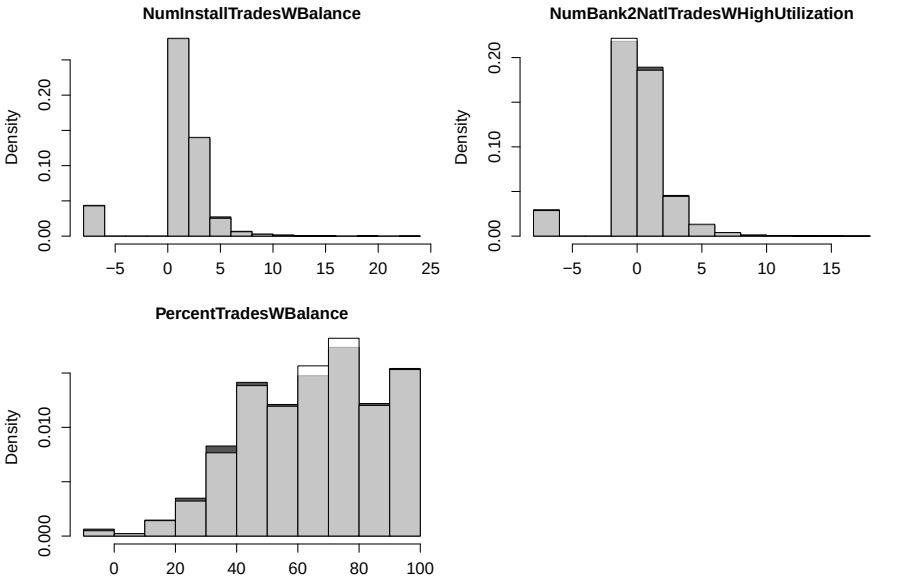

Fig. A6 Histograms for three of the variables in the generated data set (white) with the histograms for the real data superimposed (dark grey). Where the histograms overlap, the blend of of white and dark grey gives a light grey color.

References

Antorán J, Bhatt U, Adel T, et al (2021) Getting a clue: A method for explaining uncertainty estimates. In: International Conference on Learning Representations

Borisov V, Seffler K, Leemann T, et al (2023) Language models are realistic tabular data generators. In: Proceeedings of ICLR 2023

Breiman L, Friedman J, Olshen R, et al (1984) Classification and regression trees. Chapman and Hall

Brughmans D, Leyman P, Martens D (2023) NICE: An algorithm for nearest instance counterfactual explanations. Data Mining and Knowledge Discovery pp 1–39

Chen T, Guestrin C (2016) XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 785–794

Chi CM, Vossler P, Fan Y, et al (2022) Asymptotic properties of high-dimensional random forests. The Annals of Statistics 50(6):3415–3438

Dandl S, Molnar C, Binder M, et al (2020) Multi-objective counterfactual explanations. In: International Conference on Parallel Problem Solving from Nature, Springer, pp 448–469

Dhurandhar A, Chen PY, Luss R, et al (2018) Explanations based on the missing: towards contrastive explanations with pertinent negatives. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp 590–601

Downs M, Chu JL, Yacoby Y, et al (2020) Cruds: Counterfactual recourse using disentangled subspaces. In: ICML Workshop on Human Interpretability in Machine Learning

Drechsler J, Reiter JP (2011) An empirical evaluation of easily implemented, nonparametric methods for generating synthetic datasets. Computational Statistics & Data Analysis 55(12):3232–3243

Dwork C (2006) Differential privacy. In: Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33, Springer, pp 1–12

Germain M, Gregor K, Murray I, et al (2015) Made: Masked autoencoder for distribution estimation. In: International Conference on Machine Learning, PMLR, pp 881–889

Goethals S, Sörensen K, Martens D (2022) The privacy issue of counterfactual explanations: explanation linkage attacks. arXiv preprint arXiv:221012051

Gomez O, Holter S, Yuan J, et al (2020) Vice: Visual counterfactual explanations for machine learning models. In: Proceedings of the 25th International Conference on Intelligent User Interfaces. Association for Computing Machinery, New York, NY, USA, IUI '20, pp 531–535

Guidotti R (2022) Counterfactual explanations and how to find them: literature review and benchmarking. Data Mining and Knowledge Discovery pp 1–55

Géron A (2019) Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd edition. O'Reilly Media, Inc

Hastie T, Tibshirani R, Friedman JH, et al (2009) The Elements of Statistical Learning: Data Mining, Inference, and Prediction, vol 2. Springer

Joshi S, Koyejo O, Vijitbenjaronk W, et al (2019) Towards realistic individual recourse and actionable explanations in black-box decision making systems. Safe Machine Learning workshop at ICLR

Karimi AH, Barthe G, Balle B, et al (2020) Model-agnostic counterfactual explanations for consequential decisions. In: International Conference on Artificial Intelligence and Statistics, PMLR, pp 895–905

Karimi AH, Barthe G, Schölkopf B, et al (2022) A survey of algorithmic recourse: contrastive explanations and consequential recommendations. ACM Computing Surveys 55(5):1–29

Keane MT, Smyth B (2020) Good counterfactuals and where to find them: A case-based technique for generating counterfactuals for explainable AI (XAI). In: Case-Based Reasoning Research and Development: 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8–12, 2020, Proceedings 28, Springer, pp 163–178

Laugel T, Lesot MJ, Marsala C, et al (2018) Comparison-based inverse classification for interpretability in machine learning. In: International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Springer, pp 100–111

Mahiou S, Xu K, Ganev G (2022) dpart: Differentially Private Autoregressive Tabular, a General Framework for Synthetic Data Generation. arXiv preprint arXiv:220705810

Mirza M, Osindero S (2014) Conditional generative adversarial nets. arXiv preprint arXiv:14111784

Mothilal RK, Sharma A, Tan C (2020) Explaining machine learning classifiers through diverse counterfactual explanations. In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, pp 607–617

Nowok B, Raab GM, Dibben C, et al (2016) synthpop: Bespoke creation of synthetic data in R. Journal of Statistical Software 74(11):1–26

Pawelczyk M, Broelemann K, Kasneci G (2020) Learning model-agnostic counterfactual explanations for tabular data. In: Proceedings of The Web Conference 2020, pp 3126–3132

Pawelczyk M, Bielawski S, Van den Heuvel J, et al (2021) Carla: A python library to benchmark algorithmic recourse and counterfactual explanation algorithms. arXiv preprint arXiv:210800783

Pawelczyk M, Lakkaraju H, Neel S (2023) On the privacy risks of algorithmic recourse. In: International Conference on Artificial Intelligence and Statistics, PMLR, pp 9680–9696

Poyiadzi R, Sokol K, Santos-Rodriguez R, et al (2020) Face: Feasible and actionable counterfactual explanations. In: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, pp 344–350

Rasouli P, Chieh Yu I (2022) CARE: Coherent actionable recourse based on sound counterfactual explanations. International Journal of Data Science and Analytics pp 1–26

Reiter JP (2005) Using CART to generate partially synthetic public use microdata. Journal of Official Statistics 21(3):441

Scornet E, Biau G, Vert JP (2015) Consistency of random forests. The Annals of Statistics 43(4):1716–1741

Sklar M (1959) Fonctions de repartition an dimensions et leurs marges. Publ inst statist univ Paris 8:229–231

Stepin I, Alonso JM, Catala A, et al (2021) A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. IEEE Access 9:11,974–12,001

Tolomei G, Silvestri F, Haines A, et al (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. Association for Computing Machinery, New York, NY, USA, KDD '17, pp 465–474

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

Verma S, Dickerson JP, Hines K (2021) Counterfactual explanations for machine learning: Challenges revisited. CoRR abs/2106.07756

Wachter S, Mittelstadt B, Russell C (2017) Counterfactual explanations without opening the black box: Automated decisions and the GDPR. Harv JL & Tech 31:841

Wexler J, Pushkarna M, Bolukbasi T, et al (2020) The what-if tool: Interactive probing of machine learning models. IEEE Transactions on Visualization and Computer Graphics 26(1):56–65. https://doi.org/10.1109/TVCG.2019.2934619

Wilson DR, Martinez TR (1997) Improved heterogeneous distance functions. Journal of Artificial Intelligence Research 6:1–34

Xu L, Skoularidou M, Cuesta-Infante A, et al (2019) Modeling tabular data using conditional GAN. Advances in Neural Information Processing Systems 32