NICE: An Algorithm for Nearest Instance Counterfactual Explanations

Dieter Brughmans[1], Pieter Leyman[1,2,3], and David Martens[1]

[1]Department of Engineering Management, University of Antwerp, Prinsstraat 13, 2000 Antwerp, Belgium, dieter.brughmans@uantwerpen.be, pieter.leyman@uantwerpen.be, david.martens@uantwerpen.be

[2]Department of Industrial Systems Engineering and Production Design, Ghent University, Technologiepark Zwijnaarde 46, 9052 Zwijnaarde, Belgium, pieter.leyman@ugent.be

[3]Department of Engineering Management, University of Antwerp, Prinsstraat 13, 2000 Antwerp, Belgium

Abstract

In this paper we propose a new algorithm, named NICE, to generate counterfactual explanations for tabular data that specifically takes into account algorithmic requirements that often emerge in real-life deployments: (1) the ability to provide an explanation for *all* predictions, (2) being able to handle any classification model (also non-differentiable ones), (3) being efficient in run time, and (4) providing multiple counterfactual explanations with different characteristics. More specifically, our approach exploits information from a nearest unlike neighbor to speed up the search process, by iteratively introducing feature values from this neighbor in the instance to be explained. We propose four versions of NICE, one without optimization and, three which optimize the explanations for one of the following properties: sparsity, proximity or plausibility. An extensive empirical comparison on 40 datasets shows that our algorithm outperforms the current state-of-the-art in terms of these criteria. Our analyses show a trade-off between on the one hand plausibility and on the other hand proximity or sparsity, with our different optimization methods offering users the choice to select the types of counterfactuals that they prefer. An open-source implementation of NICE can be found at https://github.com/ADMAntwerp/NICE.

Keywords: XAI, Counterfactual Explanations, Machine Learning

1 Introduction

1.1 The need for explainability

In the past decade, machine learning (ML) models have been successfully deployed in many high-stakes decision making domains such as credit scoring (Lessmann et al., 2015), fraud detection (Ngai et al., 2011; Whitrow et al., 2009; Callahan and Shah, 2017), and clinical healthcare (Callahan and Shah, 2017; Huang et al., 2015). However, due to the complexity of the models or high-dimensionality of the underlying data, for many models high performance has come at a cost of explainability (Martens and Provost, 2014; Wachter et al., 2018; Ramon et al., 2020). The

inability to explain automated decisions that impact individuals undermines the trust between data subject and data controllers (Wachter et al., 2018). Post-hoc explanation methods such as counterfactual explanations aim to reinstall this trust while keeping the performance of the decision mechanism (Wachter et al., 2018). Several types of legislation have also pushed on providing explanations for algorithmic decision-making. One example is the Fair Credit Reporting Act in the United States (United States Congress, 1970). It requires data controllers to provide specific reasons that negatively influence a data subject's credit score. Counterfactual explanations are a great fit here as they provide a set of minimum features required to change the predicted outcome. Another example comes from the General Data Protection Regulation (GDPR) in the European Union. Article 14 states that data subjects have the right to obtain meaningful information about the logic involved in automated decision making (European Parliament, 2016).

Current classification models have a high complexity and many parameters. Consequently, explaining the inner working of such a model will not be meaningful to a data subject. Counterfactual explanations, on the contrary, highlight a set of input features that, when changed, alter the predicted decision (Martens and Provost, 2014). These input features are a lot more understandable to humans as the form of counterfactual explanations has deep foundations in philosophy (Kment, 2006; Lewis, 2013; Ruben, 2015) and social sciences (Miller, 2019); for it is similar to how a person thinks about a decision by asking the question: what could I have changed to achieve a different outcome? Additionally, counterfactual explanations allow data controllers to explain instances without disclosing any trade secrets or private data (Barocas et al., 2020).

It is clear that, in theory, counterfactual explanations fit the legislative requirements and have the ability to make black-box ML models transparent and accountable. Spurred by these benefits, in recent years many counterfactual algorithms have been developed for tabular data (see e.g. the overviews by Verma et al. (2020); Karimi et al. (2020b)). However, most of them focus on generating counterfactuals without taking into account the algorithmic requirements in deployment. Consider for example custom fraud detection. In a country such as Belgium, custom administration processes around 9.5 declarations every second (Vanhoeyveld et al., 2020). Each of these cases have the potential for different forms of fraud such as illegal drug traffic, importation of counterfeit goods, valuation fraud, smuggling, product misclassification and the manipulation of the origin of goods. Predictive algorithms are used in this context to identify high-risk targets which are further investigated by custom officers (Vanhoeyveld et al., 2020; Digiampietri et al., 2008). Counterfactual explanations can be of great value here to improve collaboration. The set of features in this explanation can clarify which form of fraud might be committed, or what the main reasons for predicted fraud are (e.g. country risk, article code, weight and so on), thereby speeding up further investigations. Custom officers check many high-risk cases each minute, so explanation algorithms will have to match this speed to make them useful. This computational efficiency requirement also guarantees that these algorithms can be easily scaled without the need of excessive infrastructure. An additional requirement is that *all* observations can be explained in this domain. The absence of an explanation might give the (potentially wrong) impression that the predictive model is not certain about its prediction, undermining the trust in the predictive model and making it difficult for custom officers to act upon the output.

If we return to the example of credit scoring in the US, we notice the same requirements. In this

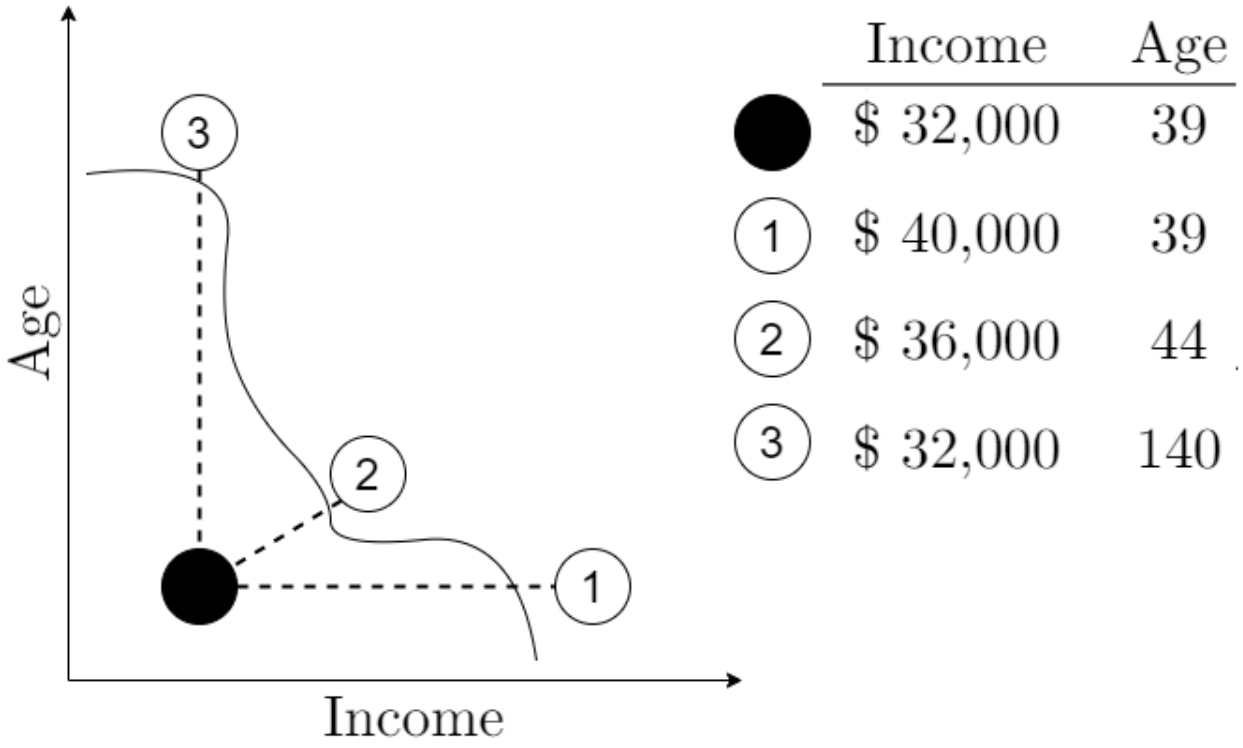


Figure 1: Example of counterfactual explanations for loan approval.

case explanations for negative decisions are required by law, rendering counterfactual algorithms that cannot explain all of these credit rejection predictions, useless. Also computational inefficiency is costly here. In credit scoring, the data subjects are potential customers. Imagine a consumer applying for a loan at a bank. A number of variables are asked, such as income, profession, etc. to assess the credit risk. The classification model, which is efficient by design as well, as not to have the consumer wait for minutes or hours to get a decision, will provide a decision swiftly. In case the application is rejected, it is just as important to have an efficient explanation algorithm to come up with a motivation for the rejection. Having the consumer just sit there and wait is arguably unacceptable or at least bad business practice.

In Figure 1, we show a simplified example of a counterfactual explanation in the domain of credit scoring. In this example the loans of individuals will be approved or denied based on their income and age. The graph shows a two-dimensional feature space. The black line represents a classification model that splits the feature space into two areas: the loans of individuals who land on the right will be approved, while the individuals who land on the left will be denied. As an example, consider a person, 39 years old and with an income of $32,000, who applies for a loan. This person is represented by the black dot and the classification model denies the loan. If this person receives a raise, increasing her income by $8,000, she would land on white dot number 1 in the feature space where her loan would be approved. This $8,000 increase in income is an example of a counterfactual explanation. It is the change that has to be made to an observation in order to change its predicted

class. This raise would result in the person being 39 years old and earning $40,000, which is called the counterfactual instance.

1.2 Research objectives

ML in general is a fast evolving field where models vary over applications and time. Current state-of-the-art classification models, might be outperformed in a few years. To ensure that counterfactual algorithms remain useful when classification models change, and to give data controllers full freedom over the choice of these models, there is a preference for model-agnostic explanation algorithms. This implies that the classification model is used simply as an output generating machine, based on provided input. We have identified four important algorithmic requirements:

- Perfect coverage, which means we want the counterfactual algorithm to be able to provide an explanation for each prediction.
- Required model access, which preferably limits itself to the inputs and outputs of the models, making the algorithm model-agnostic.
- Computational efficiency.
- Versatility, which allows to optimize the explanation based on user-provided preferences, such as for short, near or realistic explanations.

In this paper, we propose Nearest Instance Counterfactual Explanations (NICE), a new algorithm to find counterfactual explanations for tabular data with both numerical and categorical variables. This type of data is widely used in ML applications where individuals are impacted, such as credit scoring, clinical healthcare, recruitment or fraud detection. Explanations are extremely important in this context as a wrong decision can have dire consequences (Doshi-Velez and Kim, 2017).

Our contribution to the XAI literature, and more specifically to the counterfactual literature, amounts to the introduction of a new algorithm, which allows for **simultaneously** achieving the following:

1. **100% coverage:** NICE always finds a counterfactual and hence can always find an explanation for a given instance and a given classification model.
2. **Model-agnostic:** NICE works for any kind of classification model and does not require any kind of information regarding the internal workings of the classification model.
3. **Fast counterfactual generation:** NICE is fast, especially in light of the required 100% coverage (see Section 4 for the computational experiments).
4. **Versatility:** NICE has the ability to optimize for multiple counterfactual properties. Research has shown that the preference for counterfactual properties is context-dependant (Ramon et al., 2021; Fürnkranz et al., 2020). An algorithm that can provide multiple counterfactual explanations with different characteristics allows for personalized explanations per user.

Whereas other algorithms permit one or two of the above (see Section 2.2), NICE guarantees

all of these simultaneously (see Section 3). NICE furthermore employs existing counterfactual properties (i.e., sparsity, proximity and plausibility), and is built in such a way that properties can be integrated as objectives in a modular way, depending on the users' requirements.

The remainder of this paper is structured as follows. We discuss related work in Section 2 and explicitly consider potential benchmark algorithms for NICE. We formally introduce NICE itself in Section 3 and show detailed computational results in Section 4. Finally, we present our conclusions along with future research avenues in Section 5.

2 Related work

Explainable AI (XAI), of which the goal is to explain model predictions resulting from black-box models, has seen tremendous growth over the past decade. Established and recent explanation algorithms include LIME (Ribeiro et al., 2016), SHAP (Lundberg and Lee, 2017), CAVs (Kim et al., 2018), Anchors (Ribeiro et al., 2018), ProtoPNet (Chen et al., 2019) and SpRAy (Lapuschkin et al., 2019). Since in this research we focus on counterfactuals as a technique for explainability, we mainly restrict ourselves to counterfactual research in the following subsections. For in-depth overviews on XAI and its existing algorithms, we refer to Molnar (2019) and to Barredo Arrieta et al. (2020).

In Section 2.1 we give an overview of the recent and relevant counterfactual literature in general, whereas in Section 2.2 we go into detail about possible benchmark algorithms. In Section 2.3 we discuss properties of counterfactuals, and finally we touch upon user interpretability in Section 2.4.

2.1 Counterfactual explanation

Martens and Provost (2014) have been argued to be the first to formally introduce a counterfactual evidence method (see overview by Karimi et al. (2020b)), which they call SEDC[1]. Martens and Provost (2014) applied SEDC to textual data, while in subsequent work it has been employed for behavioral data (Ramon et al., 2020), image data (Vermeire et al., 2022) and finally tabular data (Fernández-Loría et al., 2020) as well. Other interesting forerunners of counterfactuals are the case-based reasoning approaches by Nugent and Cunningham (2005) and by Nugent et al. (2009), which provide insight into the impact of feature value changes on model predictions, although they focus on actual past cases while counterfactuals discuss required changes in the current situation. Given the similarities between case-based reasoning and counterfactuals as explanation methods, we discuss the case-based reasoning approach by Keane and Smyth (2020), specifically applied to counterfactuals, in more detail below, and also compare our results with theirs (see Section 4).

Wachter et al. (2018) stated the problem as a loss function to be optimized. With this approach it is easy to impose desired properties on the explanations by adding extra terms to this loss function. Mothilal et al. (2020) added a diversity parameter to generate multiple counterfactual explanations for each observation. Others implemented an auto-encoder (AE) (Van Looveren and Klaise, 2021; Dhurandhar et al., 2018) or prototype (Van Looveren and Klaise, 2021) loss, resulting in explanations closer to the data manifold. However, for tabular data, these loss functions face some challenges. First of all, they have difficulties handling nominal variables. Some solve this

[1]Pronounced as "Set See".

problem by one-hot encoding the variables and adding an extra loss term to enforce a correct encoding (Mothilal et al., 2020; Joshi et al., 2019). Others map the features into an ordinal vector space (Dhurandhar et al., 2019; Van Looveren and Klaise, 2021). A more substantial problem is that these loss functions can only be solved efficiently when gradients are available, which is only the case for differentiable models. To date, the best performing models for tabular data often include tree-based ensembles (Olson et al., 2017; Lessmann et al., 2015). For these non-differentiable models, the gradients have to be calculated numerically which causes a computational bottleneck. Van Looveren and Klaise (2021) claim to have reduced this bottleneck by adding the distance to the training data to the loss function, causing the optimization to converge faster.

Delaney et al. (2020) discuss counterfactuals in case of time series data and develop Native Guide as solution approach, which adapts existing counterfactual instances. The authors provide details of characteristics of good counterfactuals, and use these in their algorithm. The results show the superiority of Native Guide compared to existing techniques for time series data.

Multi-objective counterfactual explanations are for the first time discussed by Dandl et al. (2020), whose approach allows for a better trade-off between different objectives and is based on the multi-objective optimization literature. An adjusted version of the non-dominated sorting genetic algorithm (NSGA-II) is used to solve the multi-objective problem.

Finally, Mothilal et al. (2021) seek to combine feature importance analysis with counterfactual generation, since both types of approaches may highlight different features. The authors study the necessity and sufficiency of features and find that not only the top features reported by feature importance methods can prove to be meaningful for explanations.

2.2 Potential benchmark algorithms

We provide an overview of counterfactual algorithms worth considering to benchmark NICE (experiments discussed in detail in Section 4) in Table 1. For a recent overview on and classification of the counterfactual literature in general, we refer to Karimi et al. (2020b), whereas de Oliveira and Martens (2021) explicitly perform a benchmarking study for ten counterfactual algorithms on tabular data.

As we focus on a new optimization approach, the algorithms in Table 1 have been grouped into "families" according to the type of algorithm used, that is, gradient descent, neighbor-based, genetic algorithm and SAT solver. Combined with the more general survey on counterfactual algorithms by Karimi et al. (2020b), we can conclude that gradient descent and neighbor-based heuristics are the most prevalent types of algorithms currently used. The overview in Table 1 furthermore takes three properties from the literature used by NICE into account as objectives (sparsity, proximity and plausibility, see Sections 2.3 and 3.2), along with custom specifications 100% coverage guarantee and compatibility with tabular data (numerical and categorical features). For algorithm specifics, aside from the type of algorithm, we include the model-agnosticism of the algorithm and whether it requires access to the classification model. Finally, we show whether Python (compatible) code is publicly available or not. In the following paragraphs, we briefly discuss each of the articles in Table 1. Note that some of these articles have already been touched upon in earlier paragraphs, in order to better situate our own work.

Name	Spars.	Prox.	Plaus.	Cover.	Mixed data	Algorithm Type	Algorithm Model-agn.	Code
DiCE		x			x	GD	Y*	Y
CFproto	x	x	x		x	GD	Y*	Y
GS	x	x				NB	Y	Y
WIT		x	x		x	NB	Y	N*
CBR	x		x		x	NB	Y	N*
SEDC	x				x	NB	Y	Y
NICE	x	x	x	x	x	NB	Y	Y
GeCo	x	x	x		x	GA	Y	Y
MACE	x	x	x	x	x	SAT	Y	Y†

Table 1: Overview of the counterfactual algorithms for tabular data **potentially relevant for comparison**. The following algorithms are considered: DiCE (Mothilal et al., 2020), GS (Laugel et al., 2017), WIT (Wexler et al., 2019), CBR (Keane and Smyth, 2020), CBR (Keane and Smyth, 2020), GeCo (Schleich et al., 2021), and MACE (Karimi et al., 2020a). Objectives present in a paper are marked with an "x" in the corresponding column for sparsity (Spars.), proximity (Prox.) and plausibility (Plaus.), as are the coverage guarantee (Cover.) and the use of tabular data consisting of both discrete and continuous features (mixed data). Related to the algorithms themselves, the type (GD for gradient descent, NB for neighbor-based, GA for genetic algorithm and SAT for SAT solver) is included, along with whether it is model-agnostic (yes or no, an asterisk means model access is an optional feature) and whether the code is publicly available and usable in Python without further assumptions (yes or no, an asterisk * means we implemented the algorithm ourselves, a dagger † means specific code assumptions are made).

In terms of **gradient descent** algorithms, DiCE (Diverse Counterfactual Explanations) by Mothilal et al. (2020) allows for the generation of diverse counterfactuals for the same instance by combining multiple objectives into a single loss function. The approach returns multiple counterfactuals for a given instance, in order to present a user with several diverse counterfactuals. Another gradient descent algorithm is CFproto[2] by Van Looveren and Klaise (2021). Unlike some of its gradient-based peers, it has found a way to work with categorical data in the loss function and claims to be faster in optimizing it. The purpose of this algorithm is very similar to that of NICE (`plaus`). It tries to find a balance between close and plausible explanations and uses the training data to achieve this (Van Looveren and Klaise, 2021).

The following algorithms all fall under the **neighbor-based** family. Growing Spheres (GS) (Laugel et al., 2017) focuses on finding the minimal required changes to an instance in order to obtain a counterfactual, without relying on existing data. Growing Spheres, however, is only compatible with numerical features and does not take categorical features into account (de Oliveira and Martens, 2021). The What-if Tool (WIT) (Wexler et al., 2019) is an interactive tool which selects the nearest instance from the training set that is classified in a different class. Besides two small differences, this method is exactly the same as NICE (`none`), a basic version of NICE (see Section 3.1). First, WIT selects a counterfactual instance from the complete training set and not only the correctly classified ones. Second, the distance metric is slightly different: for numerical features it standardizes the

[2]This name comes from the GitHub repository of Van Looveren and Klaise (2021), in which CFproto is the name used for the .py file which contains their algorithm.

differences with the standard deviation and not the range of the feature values in the training set. In spite of these two small differences, WIT would make a good benchmark for NICE. Just like NICE and WIT, the Case-Based Reasoning system (CBR) for counterfactual explanations (Keane and Smyth, 2020) use nearest instances to find counterfactuals. The difference is the optimization method, which limits the search to explanations that have a maximum sparsity of two features. Furthermore, this method also does not guarantee that an explanation will be found (coverage), but it could still be worthwhile as a point of comparison. The next algorithm is SEDC for tabular data (Fernández-Loría et al., 2020). The optimization method is very similar to NICE under sparsity. The difference between both algorithms is their search space. Whereas NICE replaces feature values with those of the nearest instance, SEDC replaces them with the respective mean or mode of each feature.

A **genetic algorithm** for counterfactual generation (GeCo), which focuses on finding counterfactuals with minimal changes, is proposed by Schleich et al. (2021). The algorithm prefers explanations with as few as possible changes in feature values, and furthermore allows for the comparatively fast generation of counterfactuals.

Karimi et al. (2020a) build on formal logic and solve a sequence of **satisfiability problems (SAT)**, as part of their counterfactual approach called Model-Agnostic Counterfactual Explanations (MACE). Their results show that MACE has 100% coverage, i.e., the algorithm can always find a counterfactual, though its computation times are negatively affected by the use of an SAT solver. The open-source Python implementation of MACE[3] also sets some specific constraints on the encoding of categorical variables, which are not compatible with other models in our experiments. For these reasons, we decided not to employ MACE as a benchmark algorithm.

Name	**Spars.**	**Prox.**	**Plaus.**	**Cover.**	**Algorithm Type**
DiCE		x			GD
CFproto	x	x	x		GD
WIT		x	x		NB
CBR	x		x		NB
SEDC	x				NB
NICE	x	x	x	x	NB
GeCo	x	x	x		GA

Table 2: Overview of the following counterfactual algorithms for tabular data **used for comparison**: DiCE (Mothilal et al., 2020), CFproto (Van Looveren and Klaise, 2021), WIT (Wexler et al., 2019), CBR (Keane and Smyth, 2020), SEDC (Fernández-Loría et al., 2020), NICE, and GeCo (Schleich et al., 2021). Objectives present in a paper are marked with an "x" in the corresponding column for sparsity (Spars.), proximity (Prox.) and plausibility (Plaus.), as is the coverage guarantee (Cover.). Related to the algorithms themselves, the type (GD for gradient descent, NB for neighbor-based, GA for genetic algorithm) is included.

In order to allow for a proper and fair comparison with NICE, in the sense that benchmarks should have similar characteristics as our approach, we focus on algorithms with the following characteristics: (1) model-agnostic and no model access, (2) compatibility with mixed tabular

[3]https://github.com/amirhk/mace

data, and (3) code available and usable without further assumptions. Though both WIT and CBR, two algorithms very similar to NICE, have no Python code publicly available, we decided to implement both algorithms ourselves (code available online in our GitHub repository). Both CBR and WIT are quite similar to our approach, which warrants a comparison, and are furthermore straightforward enough to implement them ourselves. Combined with the three conditions above, the remaining algorithms are those in Table 2, which means that we have six benchmark algorithms for NICE. From the table, we conclude that in the selected benchmark algorithms there is variation in both the objectives and the algorithm type, while NICE and WIT are the only ones that have 100% coverage.

2.3 Counterfactual properties

Previous research has pointed out several important properties of counterfactual explanations (Karimi et al., 2020b; Verma et al., 2020). In this work, we focus on sparsity, proximity and plausibility.

A commonly used property is **sparsity** (Karimi et al., 2020a; Dandl et al., 2020; Laugel et al., 2018) which refers to the number of features in an explanation. It is often claimed that sparser explanations are better as they are less complex. This statement stems from psychological research which finds that people can only process five to nine pieces of information at once (Miller, 1956; Medin et al., 1987; Edwards et al., 2019; Förster et al., 2020, 2021). Especially when working with high dimensional features spaces, a sparsity constraint is useful to ensure explanations remain comprehensible for humans. In Figure 1, counterfactual 1 is sparser than counterfactual 2. If loan applicants only want to change their income and do not want to wait until they get older, counterfactual 1 is probably a better explanation for them. Furthermore, consider that sparsity is actually equal to the L0 distance between the instance to explain and the counterfactual instance. A critique of sparsity is that it is a very rough measure of distance. For example, when providing explanations in a credit approval context, it is straightforward to get an explanation that suggests a \$200 raise is better than one which suggests a \$10,000 one. However, in terms of sparsity, both explanations are equal. To counter this, previous research has used alternative distance metrics such as L1 distance (Wexler et al., 2019), L2 distance (Mothilal et al., 2020) or ABDM (Van Looveren and Klaise, 2021). In this paper, we use **proximity** as an overlapping term for all these distance metrics. Therefore, sparsity can actually be seen as a specific example of proximity. However, we keep them as separate properties in this article because much of the previous literature specifically focuses on sparsity.

Pawelczyk et al. (2020) pointed out that spars or near counterfactual explanations may be vulnerable to classification model changes over time. The counterfactual instance possibly ends up in an area far from the data manifold, where these models predict with high uncertainty. When a different classification model is trained on the same data, the previous explanation might no longer be valid. In our loan approval example, this would correspond to a case where applicants are told to raise their income by \$8,000. However, when they return to the bank, another model has been put into production and their loan request is again rejected. Such occurrences would diminish confidence in counterfactual explanations. In the rest of this paper, we call this concept cross-model robustness.

One property of counterfactual explanations that can avoid the problem of cross-model robustness is **plausibility** (Pawelczyk et al., 2020). It measures the closeness of the counterfactual instance to the data manifold. If an explanation in the loan approval example, such as counterfactual 3, suggests that applicants wait until they are 140 years old, this explanation is clearly not plausible, as it lies far outside of the data manifold. Compared to the previous two properties, plausibility is more conceptual and cannot be measured directly. Proxies that have been used to measure plausibility are: the distance to the k-nearest neighbors from the training data Dandl et al. (2020), the local outlier factor (Kanamori et al., 2020), IM1 and IM2 (Van Looveren and Klaise, 2021), uncertainty estimation Delaney et al. (2021), and the reconstruction error of an AE trained on the training data (Mahajan et al., 2019; Van Looveren and Klaise, 2021). Some studies have shown that there is an inherent trade-off between sparsity and plausibility (Dandl et al., 2020; Van Looveren and Klaise, 2021).

2.4 User interpretability

We should not ignore the often context-dependent user preferences related to XAI methods. Whereas we mainly discuss user interpretability here, overviews of general and often cross-disciplinary XAI requirements can be found in Miller (2019) and in Langer et al. (2021). As stated by among others Keane et al. (2021), a major deficit of counterfactual research is the small number of user studies, in order to gain insights into the type of counterfactuals (or explanations in general) users may prefer. As discussed before, the claimed preference for sparse explanations stems from psychological research (Miller, 1956; Medin et al., 1987; Edwards et al., 2019; Förster et al., 2020, 2021).

In spite of the above, studies related to XAI have shown that user preferences on interpretability are context and requirements dependent, and that users may even prefer more complex explanations (Ramon et al., 2021; Fürnkranz et al., 2020). Other recent examples of user studies are Byrne (2019) and Dodge et al. (2019), with the latter explicitly concluding that there is no one-size-fits-all approach to explaining, but that the usefulness of explanations depends on user profiles and expertise. Alternatively, Weld and Bansal (2019) argue in favor of interactive explanation systems, as users may have follow-up questions and more detailed concerns once an (initial) explanation has been provided.

Fernández-Loría et al. (2020) develop an algorithm to detect the most useful counterfactual explanations based on context and compare their approach with feature importance techniques in three case studies. The authors take adjustable feature weights into account, allowing their approach to suggest different kinds of counterfactuals depending on user requirements, and conclude that features with a large impact on model prediction may not necessarily be good for explaining individual decisions. In this way, their method can be seen as a necessary middle ground between the development of the algorithm on the one hand and user studies on the other.

It is clear that there is no consensus on which properties are most preferred by users. The context-dependent user preferences might even imply that there will never be a consensus and that counterfactual explanations should be personalized towards each user. Therefore, we argue that a good counterfactual algorithm should be able to provide multiple explanations with varying characteristics.

3 Methodology

In this section, we propose NICE: a nearest unlike neighbor-based approach to generate counterfactual explanations. NICE applies a depth-first heuristic approach that guarantees to always find a counterfactual, which is a hybrid between the nearest unlike neighbor and the instance to explain. Consider that this nearest unlike neighbor is not only a counterfactual due to the class change, but also an existing instance from the dataset used.

In the remainder of this section, we first explain the search process of our algorithm step by step. Afterward, we provide more information on the different reward functions used to guide this process.

3.1 NICE overview

Assume an m-dimensional feature space $X \subset \mathbb{R}^m$ consisting of both categorical and numerical features, a feature vector $x \in X$ with a corresponding label denoted as $y \in Y = \{-1, 1\}$ and a trained classification model f which maps $\mathbb{R}^m$ in the class score vector such that $f(x) \in [-1, 1]$ which leads to a predicted class $\hat{y}$. A counterfactual instance x_c for x_0 minimizes the distance $d(x_0, x_c)$ under the condition that $\hat{y_0} \neq \hat{y_c}$.

Our algorithm is very flexible in its distance metric as it does not need the categorical variables to be mapped in an ordinal vector. In this paper, we choose the Heterogeneous Euclidean Overlap Method (HEOM) as a distance metric (Wilson and Martinez, 1997). For each feature (F), the distance between two features values, a and b, is calculated according to Formula (1), while the total distance is simply the L1-norm of all feature distances. Furthermore,this metric guarantees that the contribution of each feature to the total distance is between 0 and 1.

$$d_F(a, b) = \begin{cases} 1 \text{ if } a \neq b \text{ for categorical F} \\ 0 \text{ if } a = b \text{ for categorical F} \\ \frac{|a-b|}{range(F)} \text{ for numerical F} \end{cases} \tag{1}$$

Algorithm 1 gives an overview of NICE and its constituting steps. The input of NICE is an instance x_0 for which we want to find a counterfactual instance, whereas the output x_c is such a counterfactual instance. Consider that upon termination of NICE, x_c will always remain a combination of x_0 and x_n, which substantially reduces our search space and consequently the run time of NICE. In Step 1 (lines 4-6), a current hybrid instance x_c is created between x_0 and x_n to keep track of the changes made to x_0 as the algorithm progresses, and a counter i is initialized for Step 3. In Step 2 (lines 8-9), NICE finds the nearest unlike neighbor x_n for x_0, for which $\hat{y_0} \neq \hat{y_n}$ and $y_n = \hat{y_n}$, and subsequently identifies the non-overlapping features. This results in a list of features (*featureList*), which can be changed to make x_c more similar to x_n. This approach is similar to that of Delaney et al. (2020), with the major difference that the latter uses time series instead of tabular data.

Before moving on with step 3, note that x_n from Algorithm 1 can already be used as a counterfactual instance and has some desirable properties. First, it is an actual instance, which makes it by definition plausible. In addition, the second condition ($y_n = \hat{y_n}$) implies that the observation is

correctly classified by f. Therefore, x_n corresponds to an area in $\mathbb{R}^m$ where the predictions of f are arguably more justified. If the classification model f is replaced by a different one h, trained on the same data, there would be a higher probability that x_n is also a counterfactual instance (Pawelczyk et al., 2020). We will refer to this version without further optimization as NICE(`none`) in the remainder of this work.

Step 3 of Algorithm 1 is part of a loop, which is repeated until the current x_c is classified differently as x_0 (line 18), in which case x_c is a valid counterfactual for x_0. In step 3 (lines 12-17), NICE identifies the best hybrid instance $x_{i,b}$ in iteration i, in which exactly one feature j from *featureList* changes its value from that in x_c to that in x_n. Once the value of feature j has been changed, the reward function is calculated (see Section 3.2 for details), and the instance $x_{i,b}$ with the highest reward function is retained as x_c. Finally, NICE checks whether the new instance x_c results in a class change compared to x_0, in which case the algorithm terminates and returns x_c as a valid counterfactual. Otherwise, NICE starts a new iteration of step 3 with the updated x_c, after removing the currently best feature b from *featureList*. The beauty of this approach is that we will always end up with an explanation, since after the last iteration there is only one candidate left, which is x_n, for which we know that it is a counterfactual instance.

Finally, we want to point out that NICE offers an anytime counterfactual solution, since it can at any time return an actual counterfactual, and hence does not need to run to completion to provide a counterfactual, as required by e.g., gradient descent approaches such as DiCE.

Algorithm 1 NICE (Nearest Instance Counterfactual Explanations)

NICE(x_0)
1: **Input:** x_0: instance for which to find counterfactual
2: **Output:** x_c: counterfactual instance for x_0
3: **Step 1: Initialization**
4: $x_{0,b} \leftarrow x_0$ //$x_{0,b}$ is the best instance from iteration 0
5: $x_c \leftarrow x_0$ //x_c is the current hybrid instance between x_0 and x_n
6: $i \leftarrow 1$ //i keeps track of the current iteration of step 3
7: **Step 2: Find nearest unlike neighbor**
8: $x_n \leftarrow$ FIND-NEAREST-UNLIKE- neighbor(x_0)
9: *featureList* $\leftarrow$ IDENTIFY-NON-OVERLAPPING-FEATURES(x_0, x_n)
10: **do**
11: **Step 3: Determine best hybrid instance**
12: $x_{i,b} \leftarrow \emptyset$, $R(x_{i,b}) \leftarrow -\infty$, $b \leftarrow \emptyset$ //Keep track of best hybrid instance per iteration i
13: **for** j **in** *featureList*:
14: $x_j \leftarrow x_c$, $x_j[j] \leftarrow x_n[j]$
15: **if** $R(x_j) > R(x_{i,b})$ **then** $x_{i,b} \leftarrow x_j$, $R(x_{i,b}) \leftarrow R(x_j)$, $b \leftarrow j$
16: $x_c \leftarrow x_{i,b}$ //Update current hybrid instance
17: *featureList* $\leftarrow$ *featureList*$\backslash\{b\}$ //Remove feature b
18: **while** $\hat{y_c} = \hat{y_0}$ //Predicted class is still the same for x_c and x_0
19: **return** x_c

To give the reader some idea of NICE's worst case asymptotic running time, we use O (big O) notation to signify the worst case behaviour. For an in-depth introduction to asymptotic running times and algorithm complexity, we refer to Cormen et al. (2009).

- Lines 4-6 take $O(1)$ time since these are variable assignments. neighbors
- Line 9: to identify non-overlapping features between two instances, we need to compare the values of each feature, which leads to $O(m)$ complexity.
- Lines 10-18 are part of a loop that is executed at worst $O(m)$ times, since in the worst case we need to consider $O(m)$ hybrid instances between x_0 and x_n. Note that since x_n is a nearest unlike neighbor of x_0, the actual number of features with a different value is likely (considerably) smaller than m, e.g., $m-10$, but from an asymptotic worst case point of view, this amounts to $O(m)$ iterations of the loop.
  - Line 12 takes $O(1)$ time since these are variable assignments.
  - Lines 13-15 constitute another loop which is repeated $O(m)$ times in the worst case, since the number of features with different values is again considered just like in the outer loop.
    - Line 14 takes $O(1)$ time since these are variable assignments.
    - Line 15 involves the calculation of $R(x_j)$ ($R(x_{i,b})$ has been calculated in a previous iteration), along with two variable assignments, which leads to $O(R(x))$ complexity.
    - The overall time complexity of the inner loop amounts to $O(m \cdot (1 + R(x))) = O(m \cdot R(x))$.
  - Line 16 takes $O(1)$ time since this is a variable assignment.
  - The same applies for line 17.
  - The overall time complexity of the outer loop amounts to $O(m \cdot (1 + m \cdot R(x) + 1 + 1)) = O(m^2 \cdot R(x))$.
- The overall worst case asymptotic running time of NICE then constitutes $O(1 + k \cdot m + m + m^2 \cdot R(x)) = O(R(x) \cdot m^2 + k \cdot m)$.

An example of how NICE works on instances with six features is shown in Figure 2. We start by selecting the nearest unlike neighbor x_n from the training set, for which holds: $\hat{y}_0 \neq \hat{y}_n$ and $y_n = \hat{y}_n$. The top two rows of Figure 2 show two data instances x_0 and x_n, each with six features. The black squares represent the feature values for which both instances overlap. The white and gray squares respectively represent the feature values of x_0 and x_n for the remaining features.

Next, iteration 1 shows the three hybrid instances that can be created with the first application of steps 2 and 3 from Algorithm 1, in which exactly one non-overlapping feature is replaced with the corresponding value from x_n. x_2 uses the value of the second feature of x_n, x_3 uses the value of the third feature, and x_5 uses the value of the fifth feature. For each of these new hybrid instances we calculate the outcome of a reward function $R(x)$, which will be discussed in Section 3.2. The instance with the highest value for $R(x)$, in this case x_5, has the most desirable properties. We then check if this instance is predicted as the opposite class of x_0. If so, we have our counterfactual explanation and stop the search. In our example, this is not the case and we continue our search with x_5 as the new x_c.

In the next iteration, we check the non-overlapping features of the new x_c and x_n. Again, we create all possible new combinations where one feature of x_c is replaced by the feature value of x_n. At this point, the candidate with the highest reward function (x_2) is predicted as a different class, so now we have found a counterfactual explanation.

3.2 Reward Functions

We use three reward functions, each of which measures the effect of a perturbation on the score per unit of sparsity, proximity, or plausibility. These three reward functions amount to three different objectives for NICE and can hence be used to accommodate different user requirements. Users can then e.g., choose which of these three types of counterfactuals they prefer based on the outcomes of the three versions of NICE. In the remainder of the paper, we will call these three algorithm versions: NICE(`spars`), NICE(`prox`) and NICE(`plaus`). The worst case time complexity of each of these three versions of NICE is displayed in Table 3, and discussed in more detail below.

3.2.1 Sparsity

Sparsity happens to be the most straightforward property to optimize with our approach. By simply selecting the perturbation that has the highest prediction score at each iteration, we effectively optimize for sparsity, as shown in reward function (2).

$$R(x) = \hat{y} \cdot \frac{f(x_{i-1,b}) - f(x)}{sparsity(x_{i-1,b}, x)} = \hat{y} \cdot (f(x_{i-1,b}) - f(x)) \tag{2}$$

This function compares the score and sparsity of each candidate x with that of the best candidate $x_{i-1,R_{max}}$ from the previous iteration with a factor $\hat{y}$ added to ensure a correct sign of our reward function for both classes. The sparsity difference between these instances is by definition one because we exactly add one extra feature to the explanation candidate each iteration. This allows us to remove the denominator from the formula. We then end up with a reward function which measures this score difference. This results in a theoretical range of [-2,2]for the sparsity reward function. Note that this is exactly the same as the one implicitly used by SEDC (Fernández-Loría et al., 2020). The difference is that we replace the feature values with those of x_n, while SEDC uses the mean or mode of these features.

The time complexity of function (2) depends on $f(x)$, which determines the classification of instance x based on the classification model f used. It is crucial to consider that any classification model is trained offline and that when $f(x)$ is calculated as part of NICE, we only classify an instance x according to the trained model f, but we do not retrain the model itself. This also means that the time complexity of the sparsity function (2) is $O(f(x))$. The resulting worst-case time complexity for NICE as a whole is shown in Table 3.

3.2.2 Proximity

Proximity refers to the distance from the original data point x_0 to x_c. In the reward function below, we have replaced the sparsity measure of function (2) with a proximity measure.

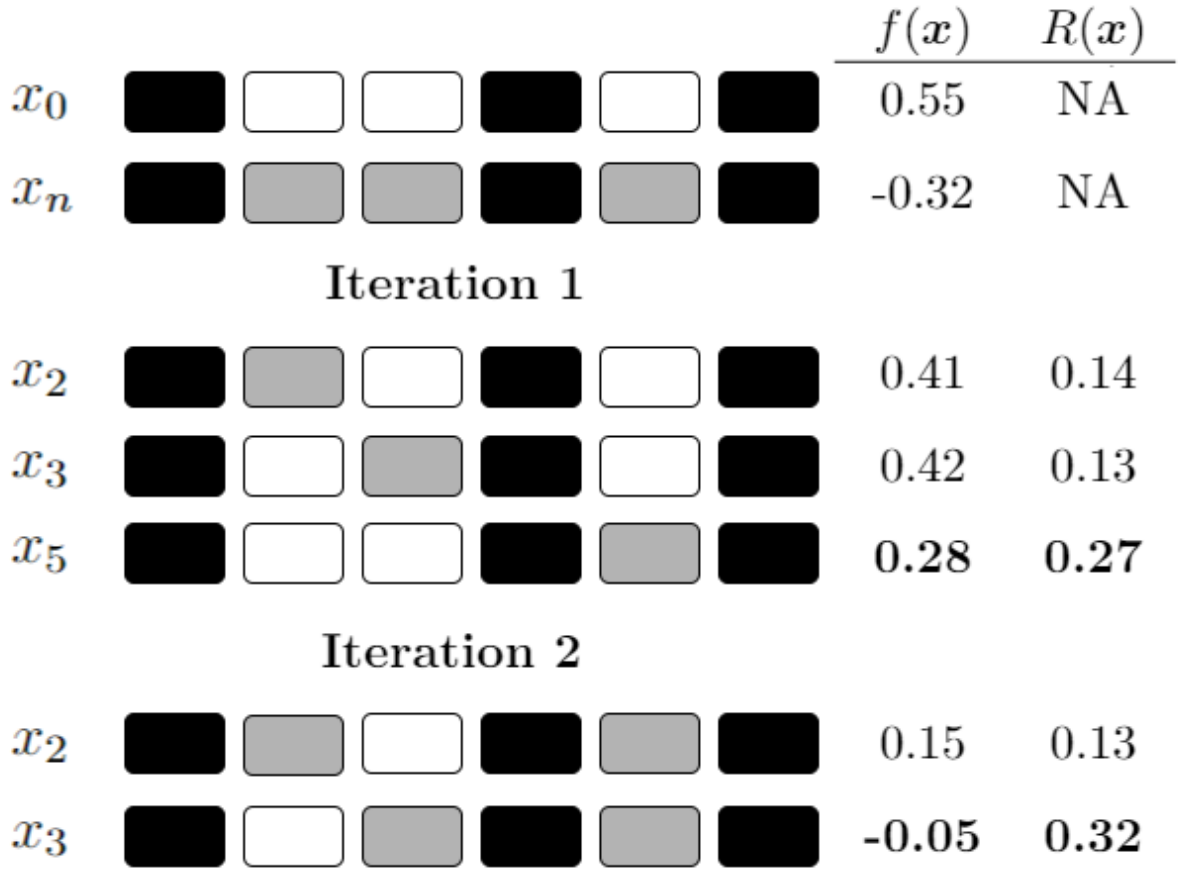


Figure 2: Example of counterfactual explanations for loan approval.

Reward function	**Worst case time complexity NICE**
Sparsity	$O(f(x) \cdot m^2 + k \cdot m)$
Proximity	$O(m^3 + f(x) \cdot m^2 + k \cdot m)$
Plausibility	$O((f(x) + g(x)) \cdot m^2 + k \cdot m)$

Table 3: Overview of NICE's worst case time complexity for each reward function. k is the number of instances, m the number of features, $f(x)$ is the classification of an instance x by the employed classification model, and $g(x)$ is the use of the AE for an instance x. The expressions have been rewritten such that each expression can be seen as a polynomial function in terms of m.

$$R(x) = \hat{y} \cdot \frac{f(x_{i-1,b}) - f(x)}{d(x_0, x) - d(x_0, x_{i-1,b})} \tag{3}$$

This function effectively calculates the decrease in prediction score per unit of distance. Sparsity and proximity often go hand in hand and (as our results show) both optimization methods often lead to the same explanation. Each iteration we move further from x_0, which makes the theoretical bounds of denominator $]0, +\infty[$. Furthermore, we know from reward function 2 that the bounds of our nominator are $[-2, 2]$, which results in a theoretical bound of $]-\infty, \infty[$ for the proximity reward function.

The time complexity of function (3) depends on $f(x)$ and on the distance function d. Whereas the former once again depends on the classification model used, the latter is $O(m)$ since we have to loop over all features to compute the distance between two instances. As a result, the worst case time complexity of the proximity function is $O(m + f(x))$. The total worst-case time complexity for NICE with proximity, after some rearranging of terms, is shown in Table 3.

3.2.3 Plausibility

We use the AE reconstruction error as a proxy for plausibility, which makes the employed plausibility function very similar to the metrics IM1 and IM2 used for interpretability by Van Looveren and Klaise (2021). An AE uses a neural network to project an instance onto a latent space and then tries to reconstruct this instance (Kramer, 1991). The error represents how successful the instance is reconstructed. When we train an AE on our training data, we can use the reconstruction error of an instance to measure how similar it is to this data. A higher (lower) error represents a data point farther from (closer to) the data manifold.

$$R(x) = \hat{y} \cdot \frac{f(x_{i-1,b}) - f(x)}{(AE_{error}(x_{i-1,b}) - AE_{error}(x))^{-1}} \tag{4}$$

This reward function behaves differently from the two previous ones. First, unlike the sparsity and proximity difference, the AE error difference can be negative and has theoretical bounds of $]-\infty, \infty[$. This is because it is possible that the hybrid instance has an AE error that is larger than the AE error of any of the two real observations x_0 and x_n. Most likely, the AE error is even larger for both those instances because it is not a real observation from the dataset. As a result, reward function 4 has theoretical bounds of $]-\infty, \infty[$. Second, the plausibility function depends on the AE in terms of complexity. However, the AE is trained offline and as part of the plausibility function we only compute the trained AE output given an instance x. To avoid an in-depth discussion of Artificial Neural Networks, of which an AE is a type, we limit ourselves to stating the AE time complexity as $O(g(x))$, with $g(x)$ referring to the AE. This results in a worst-case time complexity of $O(f(x) + g(x))$ for the plausibility reward function. Table 3 contains the total worst-case time complexity of NICE with plausibility.

Despite these downsides, the results show that it is still a valid optimization strategy. Also note that reward function (2) for sparsity is part of the plausibility reward function (4). Therefore we

are also optimizing for sparsity and the resulting explanation will be a balance between these two properties.

4 Experiments

4.1 Test design

We test NICE on datasets retrieved from the Penn Machine Learning Benchmark (PMLB) (Olson et al., 2017), from which we make a selection based on three criteria. First, the dataset must contain at least 500 instances. Second, the dataset must be a binary classification task and finally, one of our classifiers must get an AUC of at least 0.6 on the test set. This results in the 40 datasets which are shown in Table 4. Specifically, the columns of the table contain the name of the dataset, the number of instances included, and the number of features for each instance. The latter is furthermore split into two additional columns, which show the number of categorical and numerical features respectively. Consider that there is variation between the different datasets both in the number of types of features, and in the proportion of both feature types (e.g., mostly categorical, only numerical). Furthermore, for each dataset the class imbalance, i.e., the ratio of the positively and negatively classified instances, is also included.

For each dataset we start by creating a test set which contains 20% (with a minimum of 200 instances) of the data. The rest of the dataset is used as a training set to train both a Random Forest classifier (RF) and an Artificial Neural Network (ANN). The AUC values obtained for both classifiers are shown in the final two columns of Table 4. The hyperparameters of both models are trained using a five-fold cross-validation[4]. Finally, counterfactual explanations are generated with all algorithms for a random sample of 200 instances from the test set. In total 200 counterfactual explanations are generated with 10 counterfactual algorithms for 2 classifiers on 40 datasets, resulting in a sample of 160,000 explanations. For each counterfactual, we calculate diverse metrics to assess their quality.

The distribution of our results is not normal, so we need a non-parametric test for our statistical analysis. We follow Demšar (2006) by using a Friedman-rank test (Friedman, 1937, 1940). We do this by ranking all algorithms for each observation, giving a rank of 1 to the best performing explanation algorithm and a rank of 10 to the worst. If no counterfactual is available, we give this algorithm the worst rank for this observation. If there is a tie, we use the lowest rank for all tied algorithms. We then report the averages over all datasets of these ranks and submit them to the Friedman test. If this test rejects the null hypothesis of indifferent rank means, we calculate the critical difference using a Nemenyi test (Nemenyi, 1962). If the difference in average ranks between two algorithms is greater than this critical difference, we conclude that they are significantly different. In Tables 5, 6, 7 and 8 (discussed in the following subsections), the best values are marked in bold, as are values not significantly different from the best ones (5% confidence level)[5]. This use of ranks solves two problems. First, it makes the comparison possible over observations from

[4]See Appendix A.2 for more details about the hyperparameter tuning.

[5]All metrics are compared over the same number of observations and algorithms, causing the critical difference to always be 0.151.

different datasets and second, it allows us to work with data points for which not all algorithms are able to generate a counterfactual explanation.

All experiments are run on an Amazon EC2 C4.8xlarge instance where each counterfactual explanation is generated on a single 2.9 GHz Intel Xeon E5-2666 v3 Processor.

4.2 Algorithmic requirements

First, we check if all reviewed algorithms meet the algorithmic requirements of Sokol and Flach (2020). These properties are often overlooked, but are actually very important, as they determine whether algorithms can be implemented in real-world applications. The three main properties are access, coverage and computation time (Sokol and Flach, 2020). We go over each of them in the following paragraphs based on the results from Tables 5 and 6[6].

The required **access** of all these algorithms is the same. Each needs access to the training data and the scoring output of the classification model.

Coverage refers to the percentage of instances for which a valid counterfactual explanation is found. The reported coverages in Tables 5 and 6 show the strength of NICE's design. By construction, all versions of NICE have 100% coverage. In reality, anything less than perfect coverage is often not accepted. However, of the other algorithms, only WIT, which is very similar to NICE(`none`), has a perfect coverage. For both the ANN and RF, GeCo is the worst algorithm with an average coverage of only 49.4% and 39.4% respectively, while the other benchmark algorithms have a coverage between 62.7% (CBR) and 78.4% (SEDC) for the ANN, and between 48.3% (CBR) and 72.0% (SEDC) for the RF. Consider that SEDC always has perfect coverage on one of both classes (recall that we are working with binary datasets). SEDC replaces feature values with their mean or mode, and after the maximum number of replacements it ends with an instance consisting of only means or modes. If we are looking for a counterfactual instance that belongs to the predicted class of this instance, it will therefore always be found. In some applications such as fraud detection, credit scoring and clinical healthcare, we are mostly interested in explanations for one class. If this matches the class with perfect coverage for SEDC, it is a valid option. The other algorithms with imperfect coverage are unpredictable in the instances for which no explanation can be created, which makes them unusable in many real-world applications where the algorithm should be able to generate an explanation for all predictions.

Tables 5 and 6 also show the **computation time** required to generate a counterfactual explanation. We show both the average time in milliseconds (CPU time) and the average rank. First, we notice that there is a big difference in CPU times between explanations generated for an ANN and an RF. As seen in Table 4, making a prediction with an RF is much more computationally expensive than with an ANN. When comparing the different algorithms we notice that WIT and NICE(`none`), the two algorithms that use a nearest unlike neighbor without further optimization as a counterfactual explanation, are the fastest. The three versions of NICE with further optimization have reasonable optimization times which are all below 2.09 seconds on average. As expected, NICE(`spars`) is the

[6]Whereas these and other tables contain a summary of our results across the different datasets of Table 4, a detailed overview of all results per dataset can be found in the online appendix (https://github.com/ADMAntwerp/NICE_experiments).

Name	**#Inst.**	**#Feat.**	**#Cat. feat.**	**#Num. feat.**	**Class imbalance**	**AUC (ANN)**	**AUC (RF)**
adult	48,842	14	5	9	0.761	0.903	0.913
agaricus_lepiota	8154	22	21	1	0.481	1.000	1.000
australian	690	14	7	7	0.445	0.905	0.940
breast_w	699	9	8	1	0.345	0.991	0.997
buggyCrx	690	15	8	7	0.555	0.921	0.949
chess	3196	36	36	0	0.522	1.000	0.999
churn	5000	20	4	16	0.142	0.872	0.919
clean2	6598	168	0	168	0.154	1.000	1.000
coil2000	9822	85	84	1	0.060	0.691	0.745
credit_a	690	15	8	7	0.555	0.902	0.910
credit_g	1000	20	17	3	0.700	0.663	0.731
crx	690	15	8	7	0.445	0.859	0.941
diabetes	768	8	0	8	0.349	0.823	0.851
dis	3772	29	23	6	0.985	0.895	0.989
GAMETES1	1600	20	20	0	0.500	0.636	0.648
GAMETES2	1600	20	20	0	0.500	0.746	0.780
GAMETES3	1600	20	20	0	0.500	0.664	0.722
GAMETES4	1600	20	20	0	0.500	0.690	0.705
german	1000	20	17	3	0.700	0.718	0.758
Hill_Valley	1212	100	0	100	0.505	0.993	0.557
hypothyroid	3163	25	18	7	0.952	0.975	0.988
kr_vs_kp	3196	36	36	0	0.522	1.000	0.999
magic	19,020	10	0	10	0.352	0.922	0.937
mofn_3_7_10	1324	10	10	0	0.779	1.000	1.000
monk1	556	6	6	0	0.500	1.000	1.000
monk2	601	6	6	0	0.342	1.000	0.896
monk3	554	6	6	0	0.520	0.992	0.986
mushroom	8124	22	21	1	0.482	1.000	1.000
parity5+5	1124	10	10	0	0.504	1.000	0.674
phoneme	5404	5	0	5	0.294	0.906	0.970
pima	768	8	0	8	0.349	0.867	0.819
profb	672	9	3	6	0.333	0.633	0.676
ring	7400	20	0	20	0.505	0.990	0.992
spambase	4601	57	0	57	0.394	0.974	0.988
threeOf9	512	9	9	0	0.465	0.972	0.999
tic_tac_toe	958	9	9	0	0.653	0.997	1.000
tokyo1	959	44	2	42	0.639	0.962	0.983
twonorm	7400	20	0	20	0.500	0.996	0.997
wdbc	569	30	0	30	0.371	0.973	0.981
xd6	973	9	9	0	0.331	1.000	1.000

Table 4: Descriptive statistics and performance metrics of all binary datasets.

	Coverage (%)	CPU time (ms)	CPU Time (rank)	Spars. (rank)	Prox. (rank)	Plaus. (rank)
NICE (none)	100	3.7	**1.26**	5.17	5.1	**3.1**
NICE (spars)	100	14.8	4.32	**1.42**	**1.86**	4.07
NICE (prox)	100	20	5.31	1.67	**1.72**	4.15
NICE (plaus)	100	81.3	6.6	3.33	3.48	3.27
WIT	100	6.1	2.42	5.33	5.34	**3.11**
CBR	62.7	26.2	4.49	4.39	5.34	6.66
SEDC	78.4	45.7	3.65	3.16	4.1	5.31
DiCE	70.1	13,0783	8.27	3.67	4.85	6.61
CFproto	76.7	19,315.8	9.88	5.35	4.08	5.86
GeCo	49.4	1606.5	8.8	7.43	7.95	6.09

Table 5: Comparison of four versions of NICE and six benchmarks from the literature based on an **ANN classifier** (best values marked in bold). For all ranked metrics the null-hypothesis of indifferent rank means is rejected at a 5% significance level and the critical difference equals 0.151.

fastest followed by NICE(`prox`) and NICE(`plaus`). This is due to NICE(`prox`) (NICE(`plaus`)) needing to compute the distance (AE error) at each iteration. Of the other benchmark algorithms, CBR, SEDC and GeCo also have reasonable CPU times. On the contrary, DiCE and CFproto take on average several minutes to generate an explanation[7]. As a result, we conclude that except for CFproto and DiCE, all of these algorithms are fast enough to be used in real-time applications.

With respect to NICE, recall that the worst case time complexity (Table 3 in Section 3.2) is related to both the k and m values of the treated dataset. To demonstrate the impact of different values on CPU time and argue that even for larger values this does not lead to excessive CPU times, we analyze the time complexity of two datasets from Table 4 as examples in Appendix A.1.

Based on these findings, we can conclude that all versions of NICE and WIT have the most desirable algorithmic properties. Perfect coverage ensures that these algorithms can be used in applications where explanations are required by law, such as under the Fair Credit Reporting Act United States Congress (1970). All versions of NICE also have an efficient run time. This is a must for high-stakes decision making under time pressure like fraud detection, credit scoring and clinical healthcare. The detailed results[8] show that all versions of NICE scale well with the number of features. Even for the largest dataset with over 162 features, the slowest version (NICE(`plaus`)) has an average run time of 17.4 seconds and a maximum of 23.5 seconds. This makes NICE useful in domains where predictions are made at high frequency and scalability is a priority. Taking into account all algorithmic requirements, we conclude that NICE and WIT are the best options for any generic classification model. The other algorithms are useful in specific situations.

[7]The speed of CFproto and DiCE could be improved if access was given to the gradients of the ANN and RF (Van Looveren and Klaise, 2021). But to level the playing field we used the model-agnostic version of both these algorithms in all our experiments, since all other algorithms are model-agnostic.

[8]See online appendix on GitHub: https://github.com/ADMAntwerp/NICE_experiments

	Coverage (%)	CPU time (ms)	CPU Time (rank)	Spars. (rank)	Prox. (rank)	Plaus. (rank)
NICE (`none`)	100	61.8	**1.01**	4.73	4.56	2.91
NICE (`spars`)	100	573.4	4.5	**1.31**	**1.63**	4.1
NICE (`prox`)	100	925.8	5.79	1.87	**1.63**	4.13
NICE (`plaus`)	100	2090.6	6.8	3.15	3.16	**2.88**
WIT	100	95.2	2.04	4.87	4.88	**2.93**
CBR	48.3	320.6	4.09	5.1	5.85	6.67
SEDC	72	1499.6	4.5	3.34	4.24	5.16
DiCE	63.3	2,016,937	7.81	4.11	5.5	6.43
CFproto	68.2	799,339.6	9.71	5.51	4.52	5.97
GeCo	39.4	6478.1	8.73	7.65	7.89	6.5

Table 6: Comparison of four versions of NICE and six benchmarks from the literature based on an **RF classifier** (best values marked in bold). For all ranked metrics the null-hypothesis of indifferent rank means is rejected at a 5% significance level and the critical difference equals 0.151.

4.3 Explanation requirements

4.3.1 Overview results

We compare the four versions of NICE with the six selected benchmark algorithms on three counterfactual properties: sparsity, proximity and plausibility (shown as "Spars. (rank)", "Prox. (rank)" and "Plaus. (rank)" respectively in both Tables 5 and 6).

First, we look at **sparsity**. The main aim of SEDC, CBR and NICE(`spars`) is to generate the most sparse counterfactual possible. The latter clearly appears to be the winner here, since for both classification models it has by far the lowest average rank (1.42 for ANN and 1.31 for RF). GeCo, CFproto and CBR are the algorithms that perform the worst, while DiCE, SEDC, WIT and NICE(`plaus`) have an intermediate performance. Consider that NICE(`prox`) also obtains excellent sparsity results, something which we come back to in Section 4.3.3 and which we already hinted at in Section 3.2.3.

Next, we consider **proximity**. All algorithms have some sort of proximity constraint, be it in direct form or induced by a sparsity constraint. The best performing algorithms in terms of proximity are NICE(`prox`) and NICE(`spars`) for both classification models. NICE(`prox`) performs slightly better but the difference is not significant. However, both algorithms are significantly better in terms of proximity compared to all other algorithms. NICE(`plaus`) also performs significantly better than the six benchmark algorithms. NICE(`none`) does considerably worse, and is also outperformed by SEDC, DiCE and CFproto, which shows how challenging it is to find an explanation with low proximity in the instances of the training set. Related to the benchmark algorithms, it is worth noting that CFproto performs considerably better for proximity than it did for sparsity, whereas the opposite holds for DiCE.

Finally, we examine the **plausibility** of all counterfactual explanations. Recall that both NICE(`plaus`)'s and CFproto's main goal is to generate explanations which lie close to the data manifold. For an

ANN, NICE(`none`) and WIT are better than the other algorithms, but there is no significant difference between them, whereas for an RF both are additionally tied with NICE(`plaus`). The comparatively good plausibility results for both NICE(`none`) and WIT are due to both techniques resulting in an actual instance, a nearest unlike neighbor, whereas for the other algorithms there is no such guarantee. Furthermore, NICE(`none`) only looks for counterfactual instances among correctly classified observations of the training set, which avoids areas of uncertainty in the feature space. As we have seen before, this comes with a cost of proximity and a benefit in computing time, while also allowing for a good plausibility score.

4.3.2 Impact performance metrics

We previously compared all algorithms on three main requirements for the explanations: sparsity, proximity and plausibility. Whereas sparsity is well defined, there are alternatives for proximity and plausibility.

For proximity we compare four different distance metrics. The L1 distance with standardization for each feature (L1(stand.)) which is used in NICE, the L1 distance with normalization (L1 norm.) used in WIT, the L2 distance with standardization (L2 stand.) used in DiCE and GeCo and the L2 distance with normalization (L2 norm.).

Plausibility is less straightforward to measure. The AE error is a good proxy but will bias the comparison in favor of NICE(`plaus`), as it is the only algorithm that optimizes for this metric. Therefore we have added four more metrics to measure plausibility: the average distance to the 5 nearest neighbors (5NN), IM1 and IM2 (Van Looveren and Klaise, 2021) and a measure taken from Pawelczyk et al. (2020) which we call cross-model robustness. It is argued that if counterfactual instances respect the data manifold, they are less vulnerable to classification model uncertainty or changes over time (Pawelczyk et al., 2020). We can measure this by checking the percentage of instances for which an explanation is also valid for another classification model trained on the same data. We use two classification models in our experiments, so we can easily check whether an explanation for one model is also an explanation for the other.

The results for all 10 algorithms given these alternative proximity and plausibility metrics are shown in Tables 7 and 8. The metrics previously used in Section 4.3.1 are marked with an asterisk (*). In general, the results of both tables are in line with those of Section 4.3.1: NICE(`spars`) and NICE(`prox`) have the best results irrespective of the proximity metric used, while NICE(`none`) and WIT have the best results for plausibility, though NICE(`plaus`) is again a close third, especially for the RF classifier. Notice that a small difference occurs between the two L1 distances for NICE(`none`) and WIT where each scores significantly better in their own proximity metric. For plausibility the difference between NICE(`none`) and NICE(`plaus`) becomes more pronounced, since for all alternative plausibility measurements, NICE(`none`) is significantly better than NICE(`plaus`). This again emphasizes that using actual instances from the dataset (i.e., nearest unlike neighbors) is a valid solution when only plausibility is preferred.

To gain more insight into how similar these different metrics are, we also calculate the correlations of the ranks over all algorithms and for both classifiers. Table 9 contains the correlation values for sparsity and the different proximity metrics, whereas Table 10 holds the correlation values for

the different plausibility metrics. Once again, the proximity and plausibility metrics that NICE optimizes have been marked with an asterisk.

From Table 9 we conclude that the different proximity metrics are highly correlated (all values $>$ 0.900), which implies that our choice for a specific proximity metric (L1(stand.)) did not have a large impact. Furthermore, also sparsity appears to be (highly) correlated with the proximity metrics, which means that optimizing for either sparsity or proximity is beneficial for both. However, note that the correlations with sparsity are somewhat lower than those between the different proximity metrics.

In Table 10 we see a positive correlation between the different plausibility metrics as well, though it is smaller than between the proximity metrics. From this we conclude that for plausibility it may still be worth to consider multiple metrics, to at least have an idea of the differences in performance, even though there is a medium positive correlation between the metrics.

	L1 norm.*	**L1 stand.**	**L2 norm.**	**L2 stand.**	**AE Loss***	**IM1**	**IM2**	**5NN**	**CMR (%)**
NICE(`none`)	5.1	5.37	4.83	5.1	**3.1**	**3.49**	**2.46**	**1.92**	**94.3**
NICE(`spars`)	**1.86**	**1.87**	**1.89**	**1.96**	4.07	4.31	5.28	3.86	68.3
NICE(`prox`)	**1.72**	**1.8**	**1.76**	**1.88**	4.15	4.38	5.41	3.92	68.7
NICE(`plaus`)	3.48	3.63	3.3	3.48	3.27	3.89	3.75	2.76	81.3
WIT	5.34	4.96	5.09	4.63	**3.11**	**3.57**	**2.54**	**1.96**	85.6
CBR	5.34	5.19	5.7	5.57	6.66	6.06	6.4	6.45	43.4
SEDC	4.1	3.88	4.28	4.01	5.31	5.69	6	5.82	45.6
DiCE	4.85	5.21	5.34	5.78	6.61	4.62	4.55	6.92	54.9
CFproto	4.08	4	3.83	3.71	5.86	6.49	6.56	6.02	43.5
GeCo	7.95	7.9	7.79	7.7	6.09	5.76	5.39	5.76	47.6

Table 7: Average ranks of four versions of NICE and six benchmarks from the literature for different proximity and plausibility metrics based on an **ANN classifier**. CMR stands for Cross-Model robustness. Standardized (stand.) and normalized (norm.) refers to how the numerical features are scaled before calculating the respective distance metric. For all ranked metrics the null-hypothesis of indifferent rank means is rejected at a 5% significance level and the critical difference equals 0.151. Best ranks and those that are not significant different from it are marked in bold. For CM robustness, the best value is marked in bold.

	L1 norm.*	L1 stand.	L2 norm.	L2 stand.	AE Loss*	IM1	IM2	5NN	CMR (%)
NICE(none)	4.56	4.85	4.39	4.65	**2.91**	**3.49**	**2.39**	**1.77**	**80.9**
NICE(spars)	**1.63**	**1.69**	**1.70**	**1.81**	4.10	4.15	5.17	3.90	64.4
NICE(prox)	**1.63**	**1.76**	**1.66**	**1.82**	4.13	4.21	5.20	3.88	62.8
NICE(plaus)	3.16	3.35	3.06	3.27	**2.88**	3.94	3.55	2.54	69.6
WIT	4.88	4.48	4.69	4.25	**2.93**	**3.51**	**2.43**	**1.81**	78.7
CBR	5.85	5.75	6.01	5.91	6.67	6.33	6.55	6.49	40.4
SEDC	4.24	4.08	4.39	4.24	5.16	5.44	5.84	5.60	50.3
DiCE	5.50	5.57	5.80	5.91	6.43	4.14	4.31	6.76	76.0
CFproto	4.52	4.47	4.36	4.28	5.97	6.18	6.38	6.10	44.7
GeCo	7.89	7.85	7.80	7.71	6.50	6.33	5.94	6.22	42.2

Table 8: Experimental results for four versions of NICE and six benchmarks from the literature for different proximity and plausibility metrics based on an **RF classifier**. CMR stands for Cross-Model robustness and is expressed in percentage. For all other metrics, the average rank is shown. Standardized (stand.) and normalized (norm.) refers to how the numerical features are scaled before calculating the respective distance metric. For all ranked metrics the null-hypothesis of indifferent rank means is rejected at a 5% significance level and the critical difference equals 0.151. Best ranks and those that are not significant different from it are marked in bold. For CM robustness, the best value is marked in bold.

	Spars.	L1 norm.	L1 stand.*	L2 norm.	L2 stand.
Spars.		0.851	0.850	0.819	0.807
L1 norm.	0.832		0.969	0.975	0.943
L1 stand.*	0.826	0.953		0.946	0.968
L2 norm.	0.782	0.955	0.909		0.955
L2 stand.	0.764	0.904	0.949	0.929	

Table 9: Correlations of ranks between sparsity and different proximity metrics over all algorithms. Values below (above) the main diagonal are for the ANN (RF).

	AE loss*	IM1	IM2	5NN
AE loss*		0.510	0.587	0.748
IM1	0.452		0.596	0.523
IM2	0.527	0.555		0.660
5NN	0.715	0.497	0.600	

Table 10: Correlations of ranks between different plausibility metrics over all algorithms. Values below (above) the main diagonal are for the ANN (RF).

4.3.3 Trade-offs between counterfactual properties

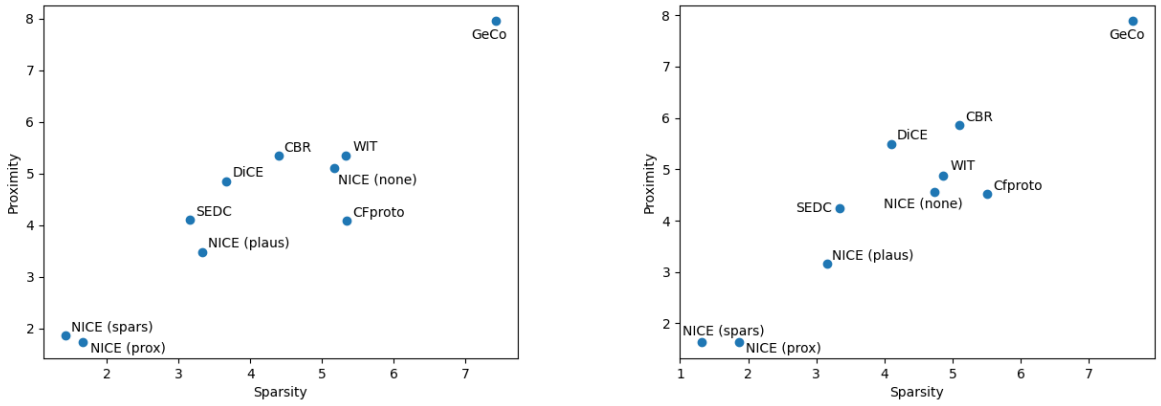
In Figures 3-5 we compare all variants of NICE with the benchmarks by showing the average ranks of each algorithm for each pairwise combination of the counterfactual properties, and this for both classifiers. In this way, we determine whether some algorithms are dominated by others for a specific pair of counterfactual properties or not. Consider that an algorithm is dominated if its average

rank is higher than or equal to the average rank of another for the two properties displayed.

- Sparsity-proximity (Figure 3): NICE(`spars`) and NICE(`prox`) dominate all others, as they have a lower average rank for sparsity and proximity than all other algorithms. Graphically this means all others are situated "to the right of" and "higher than" NICE(`spars`) and NICE(`prox`). Of the latter two, neither dominates the other since NICE(`spars`) does better for sparsity and NICE(`prox`) better for proximity, though the difference in proximity rank is very small, especially for the RF classifier, and is furthermore not significant at a 5% confidence level (see Tables 5 and 6).
- Sparsity-plausibility (Figure 4): NICE(`spars`), NICE(`plaus`) and WIT are not dominated by any other, hence, out of the 10 algorithms under study these three constitute an efficient frontier, in which case any of these are valid options depending on a user's preference for either better sparsity or plausibility. All other algorithms are dominated by at least one of these three.
- Proximity-plausibility (Figure 5): the four variants of NICE dominate all other algorithms, although we should distinguish between, on the one hand NICE(`spars`) and NICE(`prox`), which have very good results for proximity but less good for plausibility, and on the other hand NICE(`plaus`) and NICE(`none`), for which the opposite holds. Although the differences between NICE(`spars`) and NICE(`prox`) are again small, the four versions of NICE constitute an efficient frontier.

In general, we can conclude that Figures 3-5 demonstrate the edge NICE has over existing approaches from the literature, though which version of NICE performs best depends on the combination of counterfactual properties. We notice that between the versions of NICE there is a clear trade-off between plausibility on the one hand and proximity or sparsity on the other hand. NICE(`none`) generates very plausible explanations at the cost of proximity and sparsity, while this is the other way around for NICE(`prox`) and NICE(`spars`). NICE(`plaus`) seems to offer a middle ground.

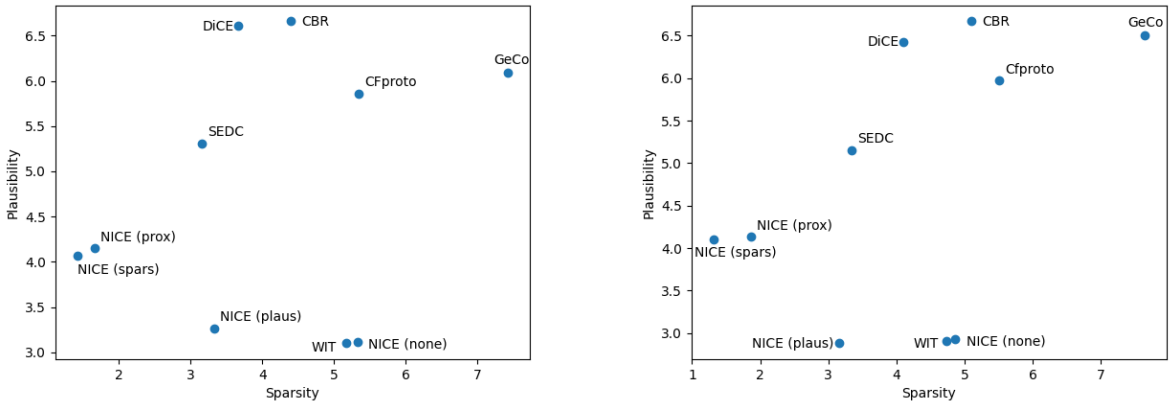
In Table 11 we show an example of how the counterfactual explanations differ for each version of NICE. The example uses the adult dataset, where we want to predict if a person's income is above or below \$50,000 a year based on demographic properties. In our example, the instance to explain (x_0) is classified as having an income below this threshold. Both NICE(`spars`) and NICE(`prox`) provide the same counterfactual explanation with a sparsity of 1 by suggesting to only increase capital-gains with \$5,178. NICE(`plaus`) suggests an additional increase in age with two years. It is very likely that people with higher capital-gains are also older in the adult dataset, and therefore NICE(`plaus`) suggests an additional change that brings the counterfactual instance closer to the data-manifold. Finally, the result of NICE(`none`) represents a real instance from the dataset and suggests an additional reduction of the number of working-hours per week of 10.



(a) ANN classifier.

(b) RF classifier.

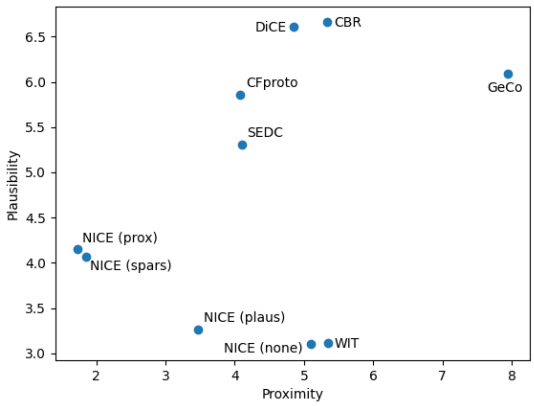
Figure 3: Comparison of algorithms for both proximity and sparsity average rankings (lower is better).



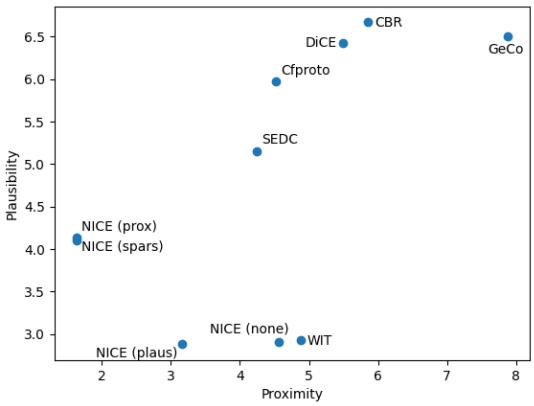
(a) ANN classifier.

(b) RF classifier.

Figure 4: Comparison of algorithms for both sparsity and plausibility average rankings (lower is better).



(a) ANN classifier.



(b) F classifier

.

Figure 5: Comparison of algorithms for both proximity and plausibility average rankings (lower is better).

	x_0	**NICE**(none)	**NICE**(spars)	**NICE**(prox)	**NICE**(plaus)
capital-gain ($)	0	**5178**	**5178**	**5178**	**5178**
age	52	**54**	52	52	**54**
hours-per-week	60	**50**	60	60	60
workclass	Private	Private	Private	Private	Private
marital-status	Divorced	Divorced	Divorced	Divorced	Divorced
relationship	Unmarried	Unmarried	Unmarried	Unmarried	Unmarried
race	White	White	White	White	White
sex	Female	Female	Female	Female	Female
education	Bachelors	Bachelors	Bachelors	Bachelors	Bachelors
education-num	9	9	9	9	9
occupation	Sales	Sales	Sales	Sales	Sales
capital-loss ($)	0	0	0	0	0
native-country	US	US	US	US	US

Table 11: Example of a counterfactual instances for each version of NICE for one instance x_0 of the adult dataset. The suggested changes for each counterfactual instance are marked in bold.

5 Conclusions & future research avenues

5.1 Conclusions

In this paper, we have introduced NICE, a new state-of-the-art algorithm for generating counterfactual explanations for tabular data. NICE is able to simultaneously achieve 100% coverage, model-agnosticism and fast counterfactual generation for different types of classification models, thereby making it suitable for real-world applications, where these algorithmic properties are expected. Specifically, NICE starts from a nearest unlike neighbor, an existing instance correctly classified as belonging to the opposite class and subsequently includes feature values from this

instance in the instance to be explained, one feature at a time, until a class change occurs.

In the extensive computational experiments, we have shown that NICE outperforms existing counterfactual algorithms from the literature for sparsity, proximity and plausibility objectives. Furthermore, when we consider alternative proximity and plausibility metrics from the literature, to determine how (un)similar these are, NICE still obtains the best results. We also looked into trade-offs between the counterfactual properties (sparsity, proximity and plausibility) and conclude that the version of NICE which performs best depends on the combination of counterfactual properties under consideration. That being said, our results show that a strong correlation exists between sparsity and proximity, which means that the most important trade-off occurs between sparsity and proximity on the one hand and plausibility on the other hand. Based on the explanations provided, users can choose which of these three types of counterfactuals they prefer.

5.2 Future research avenues

Though the goal of our paper is to propose a new algorithm, we should not ignore the importance of user expectations. Although some studies argue in favor of sparse explanations, others conclude that user expertise is paramount in determining what explanations should look like (Section 2.4). More specifically, future research should test algorithms such as NICE in real-world settings, and use feedback from all stakeholders in the predictive decision making process. A multidisciplinary approach with data scientists, domain experts and end users is needed to further improve the properties of counterfactual explanations, and see for which applications they are most valuable. The results of such user studies can subsequently help to decide which version of NICE (or other algorithm, or even which type of explanation) would be the most suited for which application.

Another future research avenue concerns the data types used, since it would be interesting to test if NICE's approach could be extended to different data types such as behavioral, textual or image data. However, different data types bring about different levels of interpretability for a person (Guidotti et al., 2018), different data properties, different types of AI techniques to be used, and due to this, also different treatment in the generation and evaluation of counterfactual methods. For example, image data is typically highly dimensional, which implies that these cannot be iterated over in a reasonable time (step 3 of Algorithm 1). One way around this could be to group pixels from images into meaningful sections (Vermeire et al., 2022). For textual and behavioral data, adding features is often a more radical change than removing features. To make NICE compatible with this data, perturbations should arguably only be limited to features which are already present in the instance to explain.

Regarding the type of datasets, we have restricted ourselves to binary ones, in which the class prediction only has two options, i.e., belonging to the class we are interested in or not. However, our approach can be generalized to multi-class classification with a small change to the reward functions, which we discuss in Appendix A.3. This approach would also result in a perfect coverage, however further experiments have to be done to check if these explanations also have desirable counterfactual properties.

Finally, in our experiments we only compare with other counterfactual algorithms, though one could argue that a broader comparison with other XAI methods is warranted. However, as shown

by Fernández-Loría et al. (2020) in their comparison of counterfactuals with feature importance explanations, features with a large impact on *prediction* may not necessarily be relevant for the *explanation* of specific decisions. Therefore, the results of feature importance methods may be misleading. Despite the above, it would be worthwhile to compare counterfactual algorithms with other XAI methods, including feature importance methods, if only to become aware of pitfalls.

Acknowledgments

We would like to thank the Flemish Research Council (FWO, Grant G0G2721N) for financial support.

References

Barocas, S., Selbst, A. D., and Raghavan, M. (2020). The hidden assumptions behind counterfactual explanations and principal reasons. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 80–89.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115.

Byrne, R. M. J. (2019). Counterfactuals in explainable artificial intelligence (XAI): Evidence from human reasoning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6276–6282. International Joint Conferences on Artificial Intelligence Organization.

Callahan, A. and Shah, N. H. (2017). Chapter 19 - machine learning in healthcare. In Sheikh, A., Cresswell, K. M., Wright, A., and Bates, D. W., editors, *Key Advances in Clinical Informatics*, pages 279–291. Academic Press.

Chen, C., Li, O., Tao, C., Barnett, A. J., Su, J., and Rudin, C. (2019). *This Looks like That: Deep Learning for Interpretable Image Recognition*. Curran Associates Inc., Red Hook, NY, USA.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2009). *Introduction to Algorithms - Third Edition*. The MIT Press.

Dandl, S., Molnar, C., Binder, M., and Bischl, B. (2020). Multi-objective counterfactual explanations. In *International Conference on Parallel Problem Solving from Nature*, pages 448–469. Springer.

de Oliveira, R. M. B. and Martens, D. (2021). A framework and benchmarking study for counterfactual generating methods on tabular data. *Applied Sciences*, 11(16).

Delaney, E., Greene, D., and Keane, M. T. (2020). Instance-based counterfactual explanations for time series classification. *arXiv*, abs/2009.13211.

Delaney, E., Greene, D., and Keane, M. T. (2021). Uncertainty estimation and out-of-distribution detection for counterfactual explanations: Pitfalls and solutions. *arXiv preprint arXiv:2107.09734*.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30.

Dhurandhar, A., Chen, P.-Y., Luss, R., Tu, C.-C., Ting, P., Shanmugam, K., and Das, P. (2018). Explanations based on the missing: Towards contrastive explanations with pertinent negatives. *Advances in neural information processing systems*, 31:592–603.

Dhurandhar, A., Pedapati, T., Balakrishnan, A., Chen, P.-Y., Shanmugam, K., and Puri, R. (2019). Model agnostic contrastive explanations for structured data. *arXiv*, abs/1906.00117.

Digiampietri, L. A., Roman, N. T., Meira, L. A., Filho, J. J., Ferreira, C. D., Kondo, A. A., Constantino, E. R., Rezende, R. C., Brandao, B. C., Ribeiro, H. S., et al. (2008). Uses of artificial intelligence in the Brazilian customs fraud detection system. In *Proceedings of the 2008 international conference on digital government research*, pages 181–187.

Dodge, J., Liao, Q. V., Zhang, Y., Bellamy, R. K., and Dugan, C. (2019). Explaining models: an empirical study of how explanations impact fairness judgment. In *Proceedings of the 24th international conference on intelligent user interfaces*, pages 275–285.

Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv*, abs/1702.08608.

Edwards, B. J., Williams, J. J., Gentner, D., and Lombrozo, T. (2019). Explanation recruits comparison in a category-learning task. *Cognition*, 185:21–38.

European Parliament (2016). Regulation (EU) 2016/679 of the European parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation).

Fernández-Loría, C., Provost, F. J., and Han, X. (2020). Explaining data-driven decisions made by AI systems: The counterfactual approach. *arXiv*, abs/2001.07417.

Förster, M., Hühn, P., Klier, M., and Kluge, K. (2021). Capturing users' reality: A novel approach to generate coherent counterfactual explanations. In *Proceedings of the 54th Hawaii International Conference on System Sciences*, page 1274.

Förster, M., Klier, M., Kluge, K., and Sigler, I. (2020). Fostering human agency: A process for the design of user-centric XAI systems. In *ICIS 2020 Proceedings*.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701.

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92.

Fürnkranz, J., Kliegr, T., and Paulheim, H. (2020). On cognitive preferences and the plausibility of rule-based models. *Machine Learning*, 109(4):853–898.

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D. (2018). A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5).

Huang, Z., Dong, W., Bath, P., Ji, L., and Duan, H. (2015). On mining latent treatment patterns from electronic medical records. *Data mining and knowledge discovery*, 29(4):914–949.

Joshi, S., Koyejo, O., Vijitbenjaronk, W., Kim, B., and Ghosh, J. (2019). Towards realistic individual recourse and actionable explanations in black-box decision making systems. *arXiv*, abs:1907.09615.

Kanamori, K., Takagi, T., Kobayashi, K., and Arimura, H. (2020). Dace: Distribution-aware counterfactual explanation by mixed-integer linear optimization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, Christian Bessiere (Ed.). International Joint Conferences on Artificial Intelligence Organization*, pages 2855–2862.

Karimi, A.-H., Barthe, G., Balle, B., and Valera, I. (2020a). Model-agnostic counterfactual explanations for consequential decisions. In *International Conference on Artificial Intelligence and Statistics*, pages 895–905. PMLR.

Karimi, A.-H., Barthe, G., Schölkopf, B., and Valera, I. (2020b). A survey of algorithmic recourse: definitions, formulations, solutions, and prospects. *arXiv*, abs/2010.04050.

Keane, M., Kenny, E., Delaney, E., and Smyth, B. (2021). If only we had better counterfactual explanations: Five key deficits to rectify in the evaluation of counterfactual xai techniques. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4466–4474.

Keane, M. T. and 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*, page 163–178. Springer-Verlag.

Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., and sayres, R. (2018). Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2668–2677. PMLR.

Kment, B. (2006). Counterfactuals and explanation. *Mind*, 115(458):261–310.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243.

Langer, M., Oster, D., Speith, T., Hermanns, H., Kästner, L., Schmidt, E., Sesing, A., and Baum, K. (2021). What do we want from explainable artificial intelligence (xai)? – a stakeholder perspective on xai and a conceptual model guiding interdisciplinary xai research. *Artificial Intelligence*, 296:103473.

Lapuschkin, S., Wäldchen, S., Binder, A., Montavon, G., Samek, W., and Müller, K.-R. (2019). Unmasking clever hans predictors and assessing what machines really learn. *Nature Communications*, 10:1–8.

Laugel, T., Lesot, M.-J., Marsala, C., Renard, X., and Detyniecki, M. (2017). Inverse classification for comparison-based interpretability in machine learning. *arXiv*, abs/1712.08443.

Laugel, T., Lesot, M.-J., Marsala, C., Renard, X., and Detyniecki, M. (2018). Comparison-based inverse classification for interpretability in machine learning. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 100–111. Springer.

Lessmann, S., Baesens, B., Seow, H.-V., and Thomas, L. C. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research*, 247(1):124–136.

Lewis, D. (2013). *Counterfactuals*. John Wiley & Sons.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA. Curran Associates Inc.

Mahajan, D., Tan, C., and Sharma, A. (2019). Preserving causal constraints in counterfactual explanations for machine learning classifiers. *arXiv*, abs/1912.03277.

Martens, D. and Provost, F. (2014). Explaining data-driven document classifications. *MIS Quarterly*, 38(1):73–100.

Medin, D. L., Wattenmaker, W. D., and Hampson, S. E. (1987). Family resemblance, conceptual cohesiveness, and category construction. *Cognitive Psychology*, 19(2):242–279.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1 – 38.

Molnar, C. (2019). *Interpretable machine learning. A Guide for Making Black Box Models Explainable*.

Mothilal, R. K., Mahajan, D., Tan, C., and Sharma, A. (2021). *Towards Unifying Feature Attribution and Counterfactual Explanations: Different Means to the Same End*, page 652–663. Association for Computing Machinery, New York, NY, USA.

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

Nemenyi, P. (1962). Distribution-free multiple comparisons. In *Biometrics*, volume 18, page 263. International Biometric Soc 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210.

Ngai, E. W., Hu, Y., Wong, Y. H., Chen, Y., and Sun, X. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision support systems*, 50(3):559–569.

Nugent, C. and Cunningham, P. (2005). A case-based explanation system for black-box systems. *Artif. Intell. Rev.*, 24(2):163–178.

Nugent, C., Doyle, D., and Cunningham, P. (2009). Gaining insight through case-based explanation. *Journal of Intelligent Information Systems*, 32:267–295.

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., and Moore, J. H. (2017). Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData mining*, 10(1):1–13.

Pawelczyk, M., Broelemann, K., and Kasneci, G. (2020). On counterfactual explanations under predictive multiplicity. In *Conference on Uncertainty in Artificial Intelligence*, pages 809–818. PMLR.

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., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Ramon, Y., Martens, D., Provost, F., and Evgeniou, T. (2020). A comparison of instance-level counterfactual explanation algorithms for behavioral and textual data: SEDC, LIME-C and SHAP-C. *Advances in Data Analysis and Classification*, pages 1–19.

Ramon, Y., Vermeire, T., Toubia, O., Martens, D., and Evgeniou, T. (2021). Understanding consumer preferences for explanations generated by XAI algorithms. *arXiv*, abs/2107.02624.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA. Association for Computing Machinery.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2018). Anchors: High-precision model-agnostic explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Ruben, D.-H. (2015). *Explaining explanation*. Routledge.

Schleich, M., Geng, Z., Zhang, Y., and Suciu, D. (2021). GeCo: Quality counterfactual explanations in real time. *Proceedings of the VLDB Endowment*, 14(9):1681–1693.

Sokol, K. and Flach, P. (2020). Explainability fact sheets: a framework for systematic assessment of explainable approaches. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 56–67.

United States Congress (1970). An act to amend the federal deposit insurance act to require insured banks to maintain certain records, to require that certain transactions in u.s. currency be reported to the department of the treasury, and for other purposes.

Van Looveren, A. and Klaise, J. (2021). Interpretable counterfactual explanations guided by prototypes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 650–665. Springer.

Vanhoeyveld, J., Martens, D., and Peeters, B. (2020). Value-added tax fraud detection with scalable anomaly detection techniques. *Applied Soft Computing*, 86:105895.

Verma, S., Dickerson, J., and Hines, K. (2020). Counterfactual explanations for machine learning: A review. *arXiv*, abs/2010.10596.

Vermeire, T., Brughmans, D., Goethals, S., de Oliveira, R., and Martens, D. (2022). Explainable image classification with evidence counterfactual. *Pattern Analysis and Applications*, Availabe online.

Wachter, S., Mittelstadt, B., and Russell, C. (2018). Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.*, 31:841.

Weld, D. S. and Bansal, G. (2019). The challenge of crafting intelligible intelligence. *Communications of the ACM*, 62(6):70–79.

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

Whitrow, C., Hand, D. J., Juszczak, P., Weston, D., and Adams, N. M. (2009). Transaction aggregation as a strategy for credit card fraud detection. *Data mining and knowledge discovery*, 18(1):30–55.

Wilson, D. R. and Martinez, T. R. (1997). Improved heterogeneous distance functions. *Journal of artificial intelligence research*, 6:1–34.

A Appendix

A.1 Examples time complexity

With respect to NICE, recall that the worst case time complexity (Table 3 in Section 3.2) is related to both the k and m values of the treated dataset. More specifically, let us consider two datasets from Table 4, namely "adult" where $k = 0.8 \cdot 48,842 = 39,073$ and $m = 14$ and "clean2" where $k = 6598 \cdot 0.8 = 5278$ and $m = 168$. in Tables 12 & 13. In both tables, we repeat the worst case time complexity of each of the four variations of NICE, along with the CPU times obtained for both the ANN and RF classifiers. Furthermore, we "fill in" the complexity functions based on the specific k and m values of the two datasets, which results in an Order of #"operations".

In both tables we observe that the Order of #"operations" can become quite large (we intentionally chose two datasets with some of the largest k and m values), but that the impact on the CPU times remains limited. E.g., for "adult" with the ANN, the CPU times remain below 100 milliseconds. Even with a higher value for m (ANN for "clean2") the CPU times are still quite small. The only noticeable increase comes from NICE(`plaus`), which can be attributed to the AE. For the RF classifier, we notice the CPU times are in general larger than those for the ANN, but are still below 2 seconds, with NICE(`plaus`) being the only real exception.

In summary, the CPU times remain small, even for some of these larger (in terms of k and m values) datasets. However, the frequent use of an AE for NICE(`plaus`) can have a negative impact, as can be seen in particular for "clean2" in Table 13 (the constant by which $g(x)$ is multiplied is much larger than for "adult"). Combined with the RF classifier requiring considerable more CPU time than the ANN classifier[9], we conclude that the largest CPU times occur for NICE(`plaus`) with the RF classifier, which can also be observed from Tables 5 & 6, but that NICE's CPU times remain within reasonable bounds.

A.2 Hyperparameters classification models

For both classifiers we used the scikit-learn Pedregosa et al. (2011) implementation which is sklearn.ensemble.RandomForestClassifier for an RF and sklearn.neural_network.MLPclassifier for an ANN. A five-fold cross-validation grid search is performed with the values of Table 14 where the best performing model is selected based on the ROC AUC score. For the RF the hyperparameter class_weight is set to "balanced" and all other hyperparameters are set to default. The ANN always consists of one hidden layer for which the number of neurons in the grid is relative to the size of the input layer (k) with a minimum of 2 neurons. For example for dataset clean2, the number of

[9]Recall that as part of NICE we only classify an instance x according to the trained model f but do not retrain the model itself. The latter happens offline.

input neurons is 168, which results in the following grid for the hyperparameter hidden_layer_sizes: 2, 25, 50, 76, 101, 126, 151, 176, 202, 227 and 252.

Algorithm	**Worst case time complexity**	**Order of #"operations"**	**CPU time ANN (ms)**	**CPU time RF (ms)**
NICE(`none`)	$O(k \cdot m)$	547,022	21.95	154.11
NICE(`spars`)	$O(f(x) \cdot m^2 + k \cdot m)$	$f(x)\cdot$ 196 + 547,022	29.29	1110.52
NICE(`prox`)	$O(m^3 + f(x) \cdot m^2 + k \cdot m)$	$f(x)\cdot$ 196 + 549,766	32.04	1538.50
NICE(`plaus`)	$O((f(x) + g(x)) \cdot m^2 + k \cdot m)$	$(f(x) + g(x))\cdot$ 196 + 547,022	66.61	1776.04

Table 12: Example dataset "adult" ($k = 39,073$; $m = 14$).

Algorithm	**Worst case time complexity**	**Order of #"operations"**	**CPU time ANN (ms)**	**CPU time RF (ms)**
NICE(`none`)	$O(k \cdot m)$	886,704	12.19	40.03
NICE(`spars`)	$O(f(x) \cdot m^2 + k \cdot m)$	$f(x)\cdot$ 28,224 + 886,704	22.49	446.70
NICE(`prox`)	$O(m^3 + f(x) \cdot m^2 + k \cdot m)$	$f(x)\cdot$ 28,224 + 5,628,336	18.82	1175.36
NICE(`plaus`)	$O((f(x) + g(x)) \cdot m^2 + k \cdot m)$	$(f(x) + g(x))\cdot$ 28,224 + 886,704	433.14	17,359.46

Table 13: Example dataset "clean2" ($k = 5278$; $m = 168$).

	Parameter name	Hyperparameter Values
RF	n_estimators	50, 100, 250, 500, 1000
	max_depth	1, 2, 5, 10, 25, None
ANN	hidden_layer_sizes	2 to 1.5k, step=0.15k

Table 14: Hyperparameter grid used in both cross-validations.

A.3 Multi-class Reward Functions

To apply NICE to multi-class, our reward functions need a more general definition. For binary classification we assumed two classes (-1 and 1) for which a classifier f maps $\mathbb{R}^m$ in the class score vector such that $f(x) \in [-1, 1]$. For multi-class classification, it is no longer possible to project the scores of our model in such a one-dimensional vector. Therefore, we assume a m-dimensional feature space $X \subset \mathbb{R}^m$ consisting of both categorical and numerical features, a feature vector $x \in X$ with a corresponding label denoted as $y \in Y = \{0, n\}$ and a trained classification model h that maps $\mathbb{R}^m$ in an n-dimensional class probability vector where $h_i(x)$ corresponds to the probability of x belonging to class i.

There are two options to generate multi-class counterfactual explanations Vermeire et al. (2022). First, one might be interested in a counterfactual explanation from a specific class and second, one might be interested in a counterfactual explanation from any class. We propose the following general reward function for both cases.

$$R(x) = \frac{(h_c(x_{i-1,b}) - h_o(x_{i-1,b})) - ((h_c(x) - h_o(x))}{sparsity(x_{i-1,b}, x)} \tag{5}$$

Equation (5) can be simplified as follows because the sparsity increase in every step is equal to 1.

$$R(x) = h_c(x_{i-1,b}) - h_0(x_{i-1,b}) - h_c(x) + h_0(x) \tag{6}$$

The definition of h_c and h_o is different depending on the type of counterfactual we are looking for. To find a valid counterfactual from a specific class, the probability of this class has to be higher than the probability of all other classes. In this case the counterfactual probability h_c is equal to the probability of this specific class c, and h_o is the maximum probability of all other class probabilities:

$$h_o(x) = max\{h_i(x) : i \in [0, n] \sim c\} \tag{7}$$

For the second case, where we look for a counterfactual from any class, we want any class probability to be higher than the probability of the original class. In this case we define h_o as the probability of the class for which the original instance to explain had the highest probability and h_c as the maximum probability of all other classes.

$$h_c(x) = max\{h_i(x) : i \in [0, n] \sim o\} \tag{8}$$

The proposed reward function in Equation (6) can be reduced to our reward function (2) for binary classification. To do this we have to project the probabilities of both classes into the one dimensional score vector [-1,1]by taking the following assumptions:

$$h_0(x) = \hat{y} \cdot \frac{-f(x) - 1}{2} \text{ and } h_c(x) = \hat{y} \cdot \frac{f(x) + 1}{2} \tag{9}$$

Replacing these in Equation 6 results in:

$$R(x) = \hat{y} \cdot (\frac{f(x_{i-1,b}) + 1}{2} - \frac{-f(x_{i-1,b}) - 1}{2} - \frac{f(x) + 1}{2} + \frac{-f(x) - 1}{2}) \tag{10}$$

$$R(x) = \hat{y} \cdot (f(x_{i-1,b}) - f(x)) \tag{11}$$

which is equal to our sparsity reward function (2).