Counterfactuals as Explanations for Monotonic Classifiers

Sarathi K[1,*,†], Shania Mitra[1,†], Deepak P[1,2] and Sutanu Chakraborti[1]

[1]*Indian Institute of Technology Madras, Chennai, 600036, India*
[2]*Queen's University Belfast, University Rd, Belfast BT7 1NN, United Kingdom*

[†]*These two authors have contributed equally*

Abstract

Recent advances in machine learning and in particular deep learning have led to models becoming increasingly complex and less interpretable. This has led to a surge in the field of explainable AI (XAI) which aims to understand and interpret predictions made by such models. One significant direction is that of generating counterfactuals that can help in providing rich causal explanations. In this work, we present a novel counterfactual generation algorithm, with an underlying monotonic constraint respecting classifier. The generated counterfactuals are realistic and the end-user can make changes to only a few features, allowing them to make amendments easily. We demonstrate the results of our algorithm and show how this technique can generate counterfactuals closer to the query with improved coverage, while incorporating domain knowledge in the form of monotonic constraints.

Keywords

XAI, Counterfactuals, Explanation, Case Based Reasoning, Monotonic Constraints

1. Introduction

In recent years, there have been quite a few interesting research papers reporting the applicability of Case Based Reasoning (CBR) in facilitating better explanations [1, 2]. An important direction in this regard is that of generating counterfactuals, which can help discover relationships between the inputs fed to a machine learner and the prediction decision made by it. This is especially useful for end users trying to understand how their current circumstances can be improved to receive the desired outcome in the future. For instance, let us say, a user applies for loan and a classifier rejects the loan (negative class). The simplest explanation that a CBR system may generate is a "factual" one, in which the system just reports a case or a set of few cases that were similar to the query, which had a similar outcome. A richer explanation takes the form of a counterfactual that can reveal more actionable information: If you asked for a slightly lower amount or your co-applicant income had been marginally higher, you would have been granted the loan. Generating interesting counterfactuals that are actually useful for

ICCBR XCBR'22: 4th Workshop on XCBR: Case-based Reasoning for the Explanation of Intelligent Systems at ICCBR-2022, September, 2022, Nancy, France

*Corresponding author.
saartyek@gmail.com (S. K); shaniamitra9@gmail.com (S. Mitra); deepaksp@acm.org (D. P); sutanuc@cse.iitm.ac.in (S. Chakraborti)
http://member.acm.org/~deepaksp (D. P); http://www.cse.iitm.ac.in/~sutanuc/ (S. Chakraborti)

CEUR Workshop Proceedings (CEUR-WS.org)

the end user is a challenging problem, and this work attempts to identify and address certain limitations of past research, and present fresh insights in this direction.

In this paper, we assume a twin-system context [1] where predictions from a machine learning (ML) or deep learning (DL) model have to be explained by a Case based Reasoner. In order to generate counterfactuals, we exploit the fact that certain attributes in the cases are naturally of the type More is Better (MIB) or Less is Better (LIB). This terminology was first introduced by McSherry et al.[3]. When buying a camera, a customer would typically prefer a model that has high optical zoom and low price. Here, optical zoom is an MIB attribute and price is an LIB attribute. In the context of generating counterfactuals in the loan domain, the chances of a loan getting accepted are higher if the applicant income is high and the loan amount is low. Thus, applicant income and loan amount are MIB and LIB attributes, respectively. We argue in this paper that it is important for the predictive ML system to respect the monotonic nature of attributes, in order to ensure that meaningful explanations are generated.

The rest of the paper is organized as follows. In Section 2, we introduce notation and present background information on counterfactual explanations. Section 3 investigates the current approaches to generate counterfactual explanations and the problems they are plagued by. Section 4 presents the formal approach and performance measures, and Section 5 elaborates the data sets and experiment results. Finally, we draw conclusions and discuss future steps in Section 6.

2. Background

In this section, we introduce notation and provide background on counterfactual explanations.
Notation: We use a case base $\mathscr{D}$ containing N cases, each of which is a vector of features $\boldsymbol{x} \in \mathbb{R}^d$ and label $y \in \{-1, +1\}$. We refer to the set of features in the case base $\mathscr{D}$ as $\mathscr{F}$. Further, $\boldsymbol{x}^j$ is used to index attribute j in case $\boldsymbol{x}$. It is important to note that we assume the positive class to be the desired outcome (e.g., loan application approved). Additionally, we also assume access to a classifier c that allows the imposition of feature-wise monotonic constraints prior to the training process.
Counterfactual Explanations: Given a query $\boldsymbol{q} \in \mathbb{R}^d$, belonging to the negative class, counterfactual explanations return a case $\boldsymbol{p} \in \mathbb{R}^d$ that is *close* to $\boldsymbol{q}$, by some pre-defined distance measure, but is predicted to be positive by the classifier c.
Nearest Like Neighbour: Given a query $\boldsymbol{q}$, a case is called the nearest-like neighbour (NLN) if it is the case closest to the $\boldsymbol{q}$ in the case base, belonging to the same class as $\boldsymbol{q}$, i.e., the negative class.
Monotonic Constraints: Domain knowledge often dictates the way in which a feature should influence the predictions of a classifier, in order for them to be plausible. These constraints guided by domain knowledge can be of two types:

1. Increasing Constraint or MIB: Imposing the increasing constraint on a feature $k \in \mathscr{F}$ implies that increasing the value of $\boldsymbol{x}^k$ would lead to a greater probability of x belonging to the positive class, i.e., the feature k is of type More is Better (MIB). Thus,

$$Pr\left(x^1, x^2, \ldots, x^k, \ldots, x^{d-1}, x^d\right) \leq Pr\left(x^1, x^2, \ldots, x^k + \delta, \ldots, x^{d-1}, x^d\right)$$

where $Pr(\boldsymbol{x})$ gives the probability of case $\boldsymbol{x}$ belonging to the positive class and $\delta \geq 0$. Thus, to get closest to the boundary of the two classes, we would need to find the lowest possible value of $\boldsymbol{x}^k$ for which the predicted class is positive.

2. Decreasing Constraint or LIB: Imposing the decreasing constraint on a feature $k \in \mathscr{F}$ implies that increasing the value of $\boldsymbol{x}^k$ would lead to a lower probability of x belonging to the positive class, i.e., the feature k is of type Less is Better (LIB). Thus,

$$Pr\left(x^1, x^2, \ldots, x^k, \ldots, x^{d-1}, x^d\right) \geq Pr\left(x^1, x^2, \ldots, x^k + \delta, \ldots, x^{d-1}, x^d\right)$$

where $\delta \geq 0$ and $Pr(\boldsymbol{x})$ is as defined above. To get closest to the boundary of the two classes, we would need to find the greatest possible value of $\boldsymbol{x}^k$ for which the predicted class is positive.

Distance Function: To find the distance between $\boldsymbol{x}$ and $\boldsymbol{x'}$ with respect to a single, real-valued or order-enumerated attribute $\boldsymbol{k}$, we define **feature distance** fd as the absolute difference between the two attribute values $\boldsymbol{x}^k$ and $\boldsymbol{x'}^k$ normalized by the difference between the maximum and minimum values of the attribute in the case base.

$$fd\left(\boldsymbol{x}^k, \boldsymbol{x'}^k\right) = \frac{\left|\boldsymbol{x}^k - \boldsymbol{x'}^k\right|}{\max^k - \min^k}$$

where, $fd\left(\boldsymbol{x}^k, \boldsymbol{x'}^k\right)$ refers to the feature distance between $\boldsymbol{x}$ and $\boldsymbol{x'}$ for attribute k, $\max^k = \max\left(\boldsymbol{x}^k \forall \boldsymbol{x} \in \mathscr{D}\right)$ and $\min^k = \min\left(\boldsymbol{x}^k \; \forall \boldsymbol{x} \in \mathscr{D}\right)$. Nominal attributes are handled in the usual way [4]. The search space for counterfactual explanations in $\mathbb{R}^d$ is restricted in each of the d dimensions by the minimum and maximum values of the corresponding attribute in the case base. Since the goal of counterfactual explanations is to be as close as possible to the query, looking beyond this region would prove to be wasteful. Due to this restriction on the dimensions, the value of feature distance for every attribute is upper-bounded by 1 and lower-bounded by 0, i.e., $0 \leq fd\left(\boldsymbol{x}^k, \boldsymbol{x'}^k\right) \leq 1 \; \forall k \in \mathscr{F}$. On extending this measure to all features, we obtain a combined feature distance which is equivalent to the Manhattan distance measure normalized using min-max scaling.

$$fd(\boldsymbol{x}, \boldsymbol{x'}) = \sum_{k \in \mathscr{F}} \frac{\left|\boldsymbol{x}^k - \boldsymbol{x'}^k\right|}{\max^k - \min^k}$$

where, $fd(\boldsymbol{x}, \boldsymbol{x'})$ is the combined feature distance between $\boldsymbol{x}, \boldsymbol{x'} \in \mathbb{R}^d$ and $\mathscr{F}$ is the set of features in the case base such that $|\mathscr{F}| = d$. The closer the feature distance is to 0, the better the counterfactual. Using the combined feature distance, we can define the similarity between two instances $\boldsymbol{x}, \boldsymbol{x'} \in \mathbb{R}^d$ as,

$$sim(\boldsymbol{x}, \boldsymbol{x'}) = 1 - \frac{fd(\boldsymbol{x}, \boldsymbol{x'})}{d}$$

where d is the number of features in the case base. The similarity measure is also upper-bounded by 1 and lower-bounded by 0. The closer the similarity is to 1, the better the counterfactual. While accounting for varied ranges across features, the combined feature distance (and similarity)

produces sparse solutions and accurately depicts the absolute change that would have to be made to the features of $\boldsymbol{x}$ to reach $\boldsymbol{x'}$ [5].

Actionable Features: In order to maintain feasibility in the proposed counterfactual explanations, often changes can be made only to a subset of features among the entire set $\mathscr{F}$. The largest possible subset of MIB/LIB features $\mathscr{A} \subseteq \mathscr{F}$, such that changes to any feature in $\mathscr{A}$ do not affect feasibility of the solution, is called the set of actionable features. For instance, features such as gender or nationality of a user cannot be modified in order to get a loan approved and hence, these cannot be included in the set of actionable features.

Difference and Match Features: The set of difference features $\mathscr{L}$ [1] between two cases $\boldsymbol{x}, \boldsymbol{x'} \in \mathbb{R}^d$ comprises the features whose feature distances are greater than or equal to some arbitrary threshold t. The threshold can be set according to the needs of the application. This is explained further in Section 4.1. The number of difference features ($|\mathscr{L}|$) is given by d_f and the maximum number of allowed difference features is given by v such that, $d_f \leq v$.

Likewise, the set of match features $\overline{\mathscr{L}}$ consists of the features whose feature distances are lower than the threshold t. The number of match features ($|\overline{\mathscr{L}}|$) is given by d_m.

$$\mathscr{L} = \{k : \ fd(\boldsymbol{x}, \boldsymbol{x'}) \geq t\}, \qquad \overline{\mathscr{L}} = \mathscr{F} - \mathscr{L} \implies d = d_m + d_f$$

Given a query $\boldsymbol{q}$ and its corresponding counterfactual explanation $\boldsymbol{p}$, the features that have to be changed to go from $\boldsymbol{q}$ to $\boldsymbol{p}$ are given by $\mathscr{L}$.

3. Related Work

Counterfactual generation algorithms involve exploration of the space around the query $\boldsymbol{q}$ to discover points that are close to $\boldsymbol{q}$ but have a different class label. For this, a variety of approaches have been proposed. These may be classified based on the search strategy, access to training data and access to the machine learning model [6]. The most commonly adopted approaches make use of convex optimization methodologies or custom heuristic rules. Convex optimization procedures need a differentiable scoring function because they incorporate the computation of gradients [7]. However, these algorithms often need to solve the optimization problem once for every generated counterfactual for each query, making it very expensive to generate a set of diverse counterfactuals for every query. There are only a few approaches that are able to generate a sizeable number of diverse counterfactuals, by some measure of diversity, for each input query [8, 9, 10].

Perturbation-based approaches attempt to meaningfully disturb the input until a point of the opposite class is obtained [9, 11]. One challenge in such approaches is to contain the number of counterfactuals that may be generated thereby – small increments in a numeric feature, for example, can result in numerous counterfactuals, only a handful of which may truly be worth considering. This problem of over-generation of potentially superfluous counterfactuals is referred to as the problem of prolixity, and may be addressed, for instance, by restricting attention to a case that is closest to the query case but has the opposite class label (the nearest unlike neighbour). A second problem is that of sparsity, which relates to the requirement that the generated counterfactual must make changes to the fewest possible features. A counterfactual that makes changes to four or more features in the query may not be usable and is unlikely

to have substantial actionable information content. The third problem is that of ensuring plausibility of the generated counterfactuals; in other words, the counterfactuals should not suggest changes to feature values that are unrealistic, or those that violate underlying domain constraints. Suggesting that doubling one's salary or changing one's gender would fetch her a loan, for instance, is an example of an implausible counterfactual [1]. A final challenge is that of ensuring diversity – in other words, we need to generate multiple counterfactuals that are sufficiently diverse with respect to each other. This enhances the flexibility of the system in terms of its ability to cater to a diverse set of users.

In this work, we focus on enhancing the method proposed by Keane et al. [1], henceforth referred to as **GCF** (**G**ood **C**ounter**F**actual), which generates counterfactuals using a classical CBR approach by reusing and revising the explanation cases close to the query.

Additionally, we experiment with imposing monotonic-constraints on features such that the MIB/LIB nature of the attribute is respected by the algorithm. Failure to take into account such constraints frequently leads to implausible or counter-intuitive answers. The proposed approach (henceforth referred to as **MBC** − **M**onotonic constrained **B**ound **C**orner counterfactuals) generates a variety of counterfactuals by finding corners in the decision surface created by a machine learning model across a subset of actionable attributes. The present work aims to maintain sparsity, by strictly restricting the number of dimensions along which changes can be made, while simultaneously improving upon diversity, by providing multiple different options to the end user, and plausibility, by introducing domain-informed monotonic constraints at the classifier-level.

Working of the GCF Algorithm: In the GCF algorithm [1], the cases $\boldsymbol{x}$ and $\boldsymbol{x'}$ closest to the query $\boldsymbol{q}$ and belonging to opposite classes are identified, where $\boldsymbol{x}$ and $\boldsymbol{q}$ belong to the negative class while $\boldsymbol{x'}$ belongs to the positive class. The values of difference features of $\boldsymbol{x}$ and $\boldsymbol{x'}$ are copied to $\boldsymbol{p}$ from $\boldsymbol{x'}$, i.e., $\mathscr{L}_{x,x'} = \mathscr{L}_{p,x'}$. The values of the remaining features, i.e., the match features, are copied from $\boldsymbol{q}$ onto counterfactual $\boldsymbol{p}$, i.e., $\boldsymbol{p}^k = \boldsymbol{q}^k \; \forall k \in \overline{\mathscr{L}}$. Upon predicting the class of $\boldsymbol{p}$ using the underlying ML model, if we obtain the desired positive class, counterfactual $\boldsymbol{p}$, which is a combination of $\boldsymbol{q}$ and $\boldsymbol{x'}$, is output as the final result. If not, an additional adaptation step is performed where the values of the difference features in $\boldsymbol{p}$ obtained from $\boldsymbol{q}$ are perturbed until the desired class is obtained.

4. Proposed Methodology

The MBC algorithm uses corners in the decision surface of a monotonicity-respecting classifier to identify the changes to be made to the query $\boldsymbol{q}$ in order to flip its class. In this section, we formalize the approach used in this work.

4.1. Approach

The goal of counterfactual explanations is to return a case $\boldsymbol{p} \in \mathbb{R}^d$ close to the query $\boldsymbol{q} \in \mathbb{R}^d$ such that the changes to be made to the features of $\boldsymbol{q}$ to flip its class label are minimal. In order to do so, we take each of the possible pairs of cases in the case base (i.e., each of the $\binom{N}{2}$ pairs) and evaluate the feature distance $fd(x^j, x'^j) \forall j \in F \wedge \forall x, x' \in \mathscr{D}$. If the feature distances of a pair,

for all except a maximum of three features, (depending on the number of allowed difference features v) are below a threshold known as tolerance level, the pair is added to a separate case base known as the **Explanation Case Base** XC [1] and is referred to as an explanation case. If the tolerance level is set to a low value, the number of explanation cases obtained is low, since it is difficult to find cases that have the same or very close values for all except a maximum of three features in the case base. If the tolerance level is set to a high value, however, the result becomes unreliable since it is based on the argument that all else equal, only the difference features contribute to the flip in class. However, if the match features vary widely, they may contribute significantly to the decision made by the classifier. In this work, the tolerance is chosen to be 2%. This value can be changed depending upon the requirement.

However, even after allowing some tolerance while computing match features, the number of explanation cases often turns out to be low due to cases in the case base being spread out in space. In case the number of case pairs in XC is less than 1% of the total number of possible pairs, we eliminate features in order of increasing feature importance, such that the most important features remain. The feature importance values are obtained from the underlying classifier c. This procedure is repeated until we obtain at least 1% of the $\binom{N}{2}$ possible pairs. The value of 1% has been chosen arbitrarily and can be changed according to the requirement of the user. The features in the set of eliminated features $\mathscr{E}$ are no longer considered during the computation of feature distance or similarity. Since the number of features under consideration reduces, the number of features required to be matched, until the criteria for an explanation case is satisfied, also reduces, i.e., $\mathscr{L} + \overline{\mathscr{L}} = \mathscr{F} - \mathscr{E}$. This allows the number of explanation cases detected to increase. When a query $\boldsymbol{q}$ is input, the explanation cases from XC containing the nearest-like neighbour are retrieved. Let the pairs be of the form (XC_A, XC_B) such that XC_A belongs to the negative class, and is the nearest like neighbour to $\boldsymbol{q}$, while XC_B belongs to the positive class. The difference features of XC_A and XC_B are chosen to be the difference features of the query $\boldsymbol{q}$ and the counterfactual $\boldsymbol{p}$. This is done because the presence of the explanation case pair (XC_A, XC_B) *close* to $\boldsymbol{q}$ tells us that by keeping the match features (approximately) constant and varying only the difference features, we are able to flip the prediction of the classifier, implying that the chosen difference features are able to influence the decision of the classifier, in turn, indicating the existence of a counterfactual. Similar to the *GCF* algorithm, the values of the match features are directly copied into the counterfactual vector $\boldsymbol{p}$ from the query $\boldsymbol{q}$. However, instead of copying the values of the difference-features from XC_B to $\boldsymbol{p}$, like in the *GCF* algorithm, the *MBC* algorithm replaces the values with those of the corners in d_f-dimensional decision surface of the classifier c. This decision surface could be visualized as one obtained when the d_f difference feature dimensions are systematically varied and the output of the classifier c is plotted at each point in this d_f-dimensional space.

Ignoring the d_m match-feature dimensions, which are held constant throughout this exploration, the d_f dimensional decision surface of the classifier c is monotonic in nature due to the explicit MIB/LIB constraints the classifier c is met with. Although the present work is applicable to different types of classifiers, it is of special significance in case of tree-based classifiers (such as XGBoost [12], used in this work), due to the axis-aligned nature of the decision surface [13] produced which leads to the formations of corners (either 2D or 3D, depending on the value of v) on the decision boundary. Among these corners, the skyline corners [14] are identified as potential counterfactual candidates. Corners are chosen over other points in the decision

boundary as they are optimal in terms of the MIB/LIB constraints. For example, consider the point A in Fig. 1. It is on the decision boundary, however, it is not at a corner. It can naturally be seen that on moving along the *Applicant Income* direction until CF_2 is reached, increasingly better counterfactuals are obtained, since, *Applicant Income*, which is an MIB dimension, along with distance to query reduce progressively. The same argument would apply to internal corners, such as B due to which only skyline corners are seen as counterfactual candidates.

To locate skyline corners at the boundary of the decision surface in $\mathbb{R}^{d_f}$, a 2×2 grid in case of $d_f = 2$ and a $2 \times 2 \times 2$ grid in case of $d_f = 3$ is systematically slid across the d_f dimensions of the search space. A skyline corner is said to be obtained when a single positive-class instance is captured in one of the corners of the grid. The end-points of each dimension of the search space are given by the maximum and minimum values of the corresponding attribute in the case base. In addition to these corner counterfactuals having $d_f \in \{2, 3\}$, we also return counterfactuals in every difference-feature dimension having $d_f = 1$. For this, in each of the d_f dimensions, the point in the boundary, lying on the axis, where the class flips is located using binary search and output as a counterfactual. The top-5 counterfactuals, located using the aforementioned procedure, closest to the query in terms of the normalized Manhattan distance (i.e., feature distance) are returned as the output counterfactuals. This allows the user to choose convenient dimensions to make changes or to trade-off between changes in multiple dimensions as per their choice. This solves the diversity problem. As an example of the working of the method, in Fig. 1, CF_1 is an example of a counterfactual located using binary search along the *Applicant Income* dimension. Along the *Co-applicant Income* dimension, both end points of the search space have a negative class label, hence, the search is abandoned, and no counterfactual is returned along that dimension. Further, CF_2, CF_3 and CF_4 are the returned corner counterfactuals.

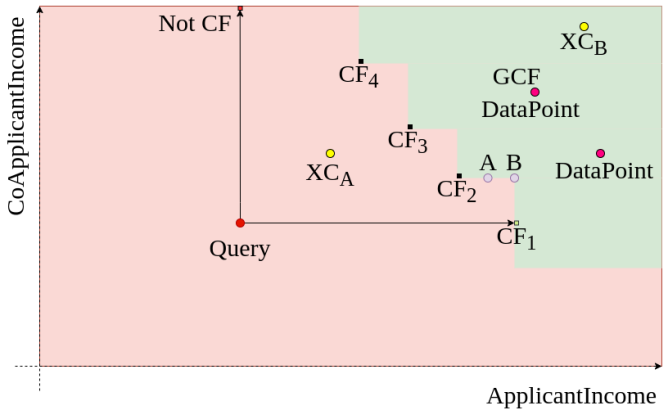

Figure 1: An Example of the Counterfactual Explanations Returned by the MBC Algorithm

4.2. Variants of the MBC and GCF Algorithms

To demonstrate the performance of the MBC algorithm, it is compared to the GCF algorithm using a variety of measures listed in Section 4.3. To test each algorithm, we consider 2 cases – $\nu = 2$ and $\nu = 3$. In case of $\nu = 2$ counterfactuals with $d_f \in \{1, 2\}$ are returned while in case of

$v = 3$ counterfactuals with $d_f \in \{1, 3\}$ are returned. Some counterfactuals with $d_f = 2$ may also be returned in case of $v = 3$, depending on the number of difference features in the explanation case closest to the query. We do not go beyond 3 difference features due to the human memory constraints [1]. The algorithm, however, could easily be extended to higher dimensions. Each of the two algorithms are subject to these two cases and are labelled as MBC_{2d}, MBC_{3d}, GCF_{2d} and GCF_{3d} respectively.

Further, to demonstrate the importance of monotonic constraints experimentally, both the algorithms, each having two variants, are evaluated with and without these constraints on the underlying classifier *c*. It is important to note that the variant without the monotonic constraints uses the same classifier as the one with constraints. The only difference is that, in one case, the MIB/LIB constraints are not fed to the classifier prior to the training procedure, and thus, it can learn any arbitrary decision surface. This, however, leads to counter-intuitive results, as we will see in Section 4.3.

Thus, all experiments are carried out on 4 variants of each of the two algorithms, namely, X_{2d}, X_{3d}, $X_{mono-2d}$ and $X_{mono-3d}$, where X = *MBC*, *GCF* and the subscript *mono* refers to the use of monotonic constraints on *c*.

4.3. Importance of Monotonicity-respecting Classifier

Most modern day classifiers, unless explicitly pre-programmed to do so, are unable to learn monotonicity patterns in the attributes of the data. For example, if we consider the case of a loan approval system, attributes such as applicant income are commonly known to be MIB, while attributes such as loan amount are LIB, i.e., decreasing the loan amount increases the chances of loan approval. In such a case the decision boundary of the classifier, unless fed with monotonic constraints, often turns out to be counter-intuitive. For example, in Fig. 2 we observe the decision boundary for a query for which applicant income and co-applicant income are being varied. In both Fig. 2 (a) and (b) we observe that, for the same loan amount, as applicant income increases from $N1$ to $N2$, the loan which was rejected at $N2$ is approved at $N2$ since $N2 > N1$. However, in the non-monotonic case (Fig. 2(a)) we can see that, all other features being the same, for the same co-applicant income, a loan with applicant income $N3$ is rejected but $N2$ is approved even though $N3 > N2$ and all other features are held constant. This does not match our expectation. Since the quality of counterfactuals produced greatly depends on the classifier, it is very important that the classifier learns the general rules for loan application acceptance rather than overfit itself to a small case base.

4.4. Performance Measures

In this section, we highlight the performance measures used to compare the variants of the algorithms listed in Section 4.2. It must be noted that the *MBC* algorithm returns multiple counterfactuals (up to 5) while the *GCF* algorithm returns only 1. To make the comparison fair, all evaluation measures for the variants of *MBC* are calculated using the output counterfactual $\boldsymbol{p}$ closest to the query $\boldsymbol{q}$. The evaluation measures include similarity to query (Sim_{query}) which measures the similarity of the output counterfactual to the query and similarity to data (Sim_{data}) measures the similarity of the output counterfactual to the nearest case. It is representative of

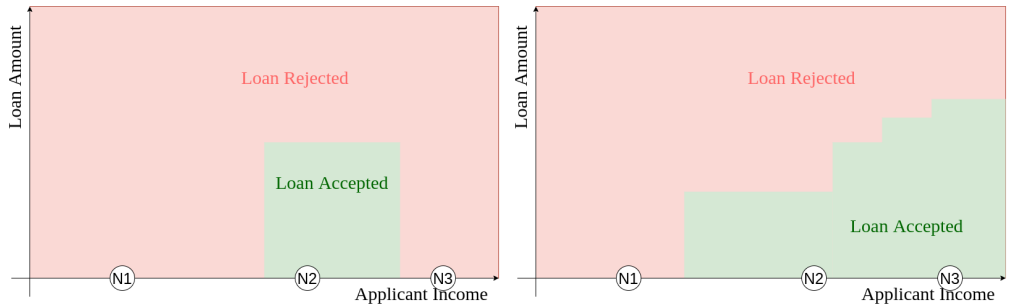

Figure 2: An Example Motivating the Need for Monotonicity-Respecting Classifiers (a) Non-Montonic Classifier (b) Monotonic Classifer

how close to real data the returned counterfactual is. Additionally, we also measure coverage, which measures the proportion of queries for which the algorithm in question is able to provide counterfactuals. A higher value of coverage is desired since it indicates that the algorithm is able to provide counterfactuals to a larger number of queries. It is given by,

$$Coverage = \frac{\sum_{q \in queries} \begin{cases} 1, & \text{if } q \text{ has a counterfactual} \\ 0 & \text{otherwise} \end{cases}}{Total\ Number\ of\ Queries}$$

5. Results

5.1. Loan Approval Data Set

The Loan approval data set, consisting of 614 instances and 12 attributes, computes loan eligibility based on customer details such as gender, marital status, education, loan amount, credit history, applicant income, co-applicant income, etc. Among these, features such as gender, marital status and education are excluded from the set of actionable features (even though education could be an MIB feature) since the customer cannot alter these in order to get their loan approved. Increasing constraints (MIB) are placed on features such applicant income and co-applicant income, while features such as loan amount are subjected to decreasing constraints (LIB). A counterfactual, in this case, would tell the loan applicant what measures to take to get her loan approved. From Table 1, it can be seen that all variants of *MBC* outperform the corresponding variants of *GCF* with respect to Sim_{query} and *Coverage*. The top-performing *MBC* variant - $mono - 2d$, with respect to Sim_{query}), outperforms the top-performing *GCF* variant - $mono - 2d$ by 5.7%. As expected, *GCF* variants have a higher similarity to data (Sim_{data}) due to their usage of actual feature values from the case base. However, the top-performing *MBC* variant is only marginally. behind the top *GCF* variant in terms of Sim_{data}. The coverage of most *GCF*-variants is seen to be very low, while the *MBC* variants fair better. The top *MBC* variant has double the coverage as that of the top *GCF* variant. Additionally, we observe that each of the variants with monotonic constraints, for both *MBC* and *GCF* outperform those without with respect to each of the three measures, demonstrating the importance of these constraints in the context of the given data set. Lastly, we observe that between the cases of $v = 2$ and $v = 3$,

	MBC_{2d}	MBC_{3d}	GCF_{2d}	GCF_{3d}	$MBC_{mono-2d}$	$MBC_{mono-3d}$	$GCF_{mono-2d}$	$GCF_{mono-3d}$
Sim_{query}	0.975	0.954	0.911	0.888	**0.981**	0.979	0.928	0.891
Sim_{data}	0.975	0.978	0.980	0.989	0.981	0.992	0.982	**0.995**
Coverage	0.215	0.344	0.086	0.118	0.252	**0.417**	0.117	0.204

Table 1
Evaluation Measure Values for the Loan Approval Data set

	MBC_{2d}	MBC_{3d}	GCF_{2d}	GCF_{3d}	$MBC_{mono-2d}$	$MBC_{mono-3d}$	$GCF_{mono-2d}$	$GCF_{mono-3d}$
Sim_{query}	0.943	0.950	0.751	0.653	0.922	**0.961**	0.757	0.646
Sim_{data}	0.930	0.949	0.908	**0.965**	0.923	0.954	0.935	**0.965**
Coverage	0.718	**0.894**	0.076	0.329	0.718	0.824	0.059	0.294

Table 2
Evaluation Measure Values for the Wine Data set

coverage and Sim_{data} increase as v increases, while Sim_{query} is seen to decrease with increase in v across all variants of both the algorithms.

5.2. Wine Data Set

The Wine data set consists of the chemical analysis of wines. It has 178 instances and 12 attributes such as content of Malic acid, ash, Flavonoids, colour intensity, hue, etc. There are three class labels corresponding to the alcohol content. Since the *MBC* and *GCF* algorithms deal with binary classification settings, the data set is converted to a binary classification case base using the one-vs-all technique. A counterfactual, in this case, would tell the wine distillery what changes to make to improve the alcohol content in the wine. Attributes such as Flavonoid content, colour intensity and Malic acid content are assumed to have a decreasing constraint (LIB) while hue is assumed to have an increasing constraint (MIB). Similar to the Loan data set case in Section 5.1, we observe, from Table 2, that *MBC* outperforms *GCF* in terms of Sim_{query} and coverage while *GCF* marginally outperforms *MBC* in terms of Sim_{data}. In this data set, however, monotonic constraints do not add as much value as the previous one. For example, in MBC_{3d}, similarity to data and query are seen to increase with the introduction of monotonic constraints, while coverage decreases. In the case of MBC_{2d} however, the addition of monotonic constraints is seen to decrease similarity to data and query, while coverage remains constant. This can be attributed to the lack of domain knowledge in the field, due to which some constraints may have been formulated incorrectly.

5.3. Employee Attrition Data Set

The Employee Attrition data set consists of 1470 employee records with 34 attributes such as gender, percentage of salary hike, job satisfaction, distance from home, monthly income, overtime hours etc. These attributes are used to predict employee attrition - whether the employee will leave the company or not. A counterfactual, in this case, would tell the company what measures to take to prevent an employee from leaving. Features such as gender and distance from home are excluded from the list of actionable attributes, since it is not possible for

	MBC_{2d}	MBC_{3d}	GCF_{2d}	GCF_{3d}	$MBC_{mono-2d}$	$MBC_{mono-3d}$	$GCF_{mono-2d}$	$GCF_{mono-3d}$
Sim_{query}	0.951	0.950	0.876	0.782	**0.963**	0.931	0.895	0.819
Sim_{data}	0.982	0.985	0.989	0.995	0.984	0.990	0.993	**0.998**
Coverage	0.581	**0.880**	0.171	0.051	0.385	0.602	0.106	0.031

Table 3
Evaluation Measure Values for the Employee Attrition Data set

the company to change the gender of an employee. An increasing constraint (MIB) is placed on features such as percentage of salary hike, job satisfaction, monthly income (for e.g. increasing the salary of an employee would reduce the chances of him/her leaving) while, features such as overtime hours are met with a decreasing constraint (LIB)(i.e., the lesser number of overtime hours an employee is subject to the more likely he/she is to stay in the company). In Table 3, we observe trends similar to the previous two case studies. The *MBC* variants outperform their corresponding *GCF* variants in terms of Sim_{query} and coverage, while *GCF* marginally outperforms *MBC* in terms of Sim_{data}. Monotonic constraints are seen to improve similarity to data in all cases except $MBC_{mono-3d}$, while they improve similarity to query for all the variants. Coverage is seen to be substantially higher among the *MBC* variants as compared to the *GCF* variants. The *MBC* variants without constraints are seen to have a higher coverage than those with the constraints.

6. Discussion

In this work, we address the issues of prolixity, sparsity and plausibility raised by Keane et al. [1] as follows:

Prolixity: Keane et al. [1] suggests the use of methods that find the minimal changes to the features of the test case that flip the prediction (i.e., the nearest unlike neighbour) to tackle the problem of prolixity. However, in this study it is shown that to produce minimal changes that flip the prediction, we do not need to rely on the nearest unlike neighbour. The present algorithm produces points that are closer to the decision boundary, thus requiring lesser change.

Diversity: In contrast to the *GCF* algorithm which returns only 1 counterfactual per query, the *MCB* algorithm returns multiple counterfactuals per query, which make trade-offs among the different dimensions to reach the desired outcome. This presents the user with a diverse choice of counterfactuals to choose from.

Sparsity: Even though many algorithms claim to make changes to only a few features, many of these counterfactuals may still involve relatively high numbers of feature-differences (e.g. > 4) [1]. Since the present algorithm chooses and changes only between 1 and 3 features at a time, it is possible to successfully deliver sparse counterfactuals.

Plausibility: Finally, we address the problem of plausibility, which states that the counterfactuals generated may not be valid cases in the domain, or they may suggest feature-changes that are difficult-to-impossible. However, this concern stands invalidated in cases where the attributes are known to be MIB or LIB. For an LIB or MIB attribute, every point between the extremes is valid by definition, since the attributes are of monotonic nature. Further, to eradicate the possibility of impossible values being returned, since we cap the values of each feature by

the corresponding minimum and maximum of the cases in the case base.

7. Conclusion

Majority of the present day counterfactual generation algorithms face the challenges of prolixity, sparsity, plausibility and diversity among others. In this work, we explore skyline corners in the decision boundary of a tree-based classifier, as counterfactuals. We also incorporate domain knowledge by compelling the underlying classifier to respect monotonic constraints such that the generated counterfactuals more plausible. Better counterfactuals are achieved with this technique, in terms of similarity to query and coverage, while closely maintaining their similarity to the existing cases. Although this is a unique way to find diverse counterfactuals, monotonicity tends to be a subjective characteristic of each data set, and concrete domain knowledge is required to represent such constraints. There is scope to find more efficient processes to obtain counterfactuals in the monotonic feature space that could be explored in the future.

References

[1] M. T. Keane, B. Smyth, Good counterfactuals and where to find them: A case-based technique for generating counterfactuals for explainable ai (xai), Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 12311 LNAI (2020) 163–178.

[2] N. Wiratunga, A. Wijekoon, I. Nkisi-Orji, K. Martin, C. Palihawadana, D. Corsar, Discern:discovering counterfactual explanations using relevance features from neighbourhoods (2021).

[3] D. Mcsherry, Similarity and compromise, in: In Proceedings of the Fifth International Conference on Case-Based Reasoning, Springer, 2003, pp. 291–305.

[4] R. Bergmann, Experience Management: Foundations, Development Methodology, and Internet-Based Applications, Springer-Verlag, Berlin, Heidelberg, 2002.

[5] D. Slack, S. Hilgard, H. Lakkaraju, S. Singh, Counterfactual explanations can be manipulated (2021).

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

[7] I. Afonichkin, Explaining machine learning models by generating counterfactuals, 2019.

[8] S. Dandl, C. Molnar, M. Binder, B. Bischl, Multi-objective counterfactual explanations, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 12269 LNCS (2020) 448–469.

[9] R. Guidotti, A. Monreale, F. Giannotti, D. Pedreschi, S. Ruggieri, F. Turini, Factual and counterfactual explanations for black box decision making, IEEE Intelligent Systems 34 (2019) 14–23.

[10] A.-H. Karimi, G. Barthe, B. Balle, I. Valera, Model-agnostic counterfactual explanations for consequential decisions (2019).

[11] B. Mittelstadt, C. Russell, S. Wachter, Explaining explanations in ai, FAT* 2019 - Proceedings of the 2019 Conference on Fairness, Accountability, and Transparency (2018) 279–288.

[12] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 785–794.

[13] A. Criminisi, J. Shotton, E. Konukoglu, Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning, Found. Trends. Comput. Graph. Vis. 7 (2012) 81–227.

[14] S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: Proceedings of the 17th International Conference on Data Engineering, IEEE Computer Society, USA, 2001, p. 421–430.